HITACHI

ノンストップデータベース

HiRDB Version 10 UAP 開発ガイド

解説・手引書

3020-6-560-B0

前書き

■ 対象製品

●適用 OS：AIX V7.2，AIX V7.3

P-1M62-35A1 HiRDB Server Version 10 10-11

P-1M62-1BA1 HiRDB/Run Time Version 10 10-11

P-1M62-1CA1 HiRDB/Developer's Kit Version 10 10-11

P-1M62-1DA1 HiRDB/Run Time Version 10(64) 10-11

P-1M62-1EA1 HiRDB/Developer's Kit Version 10(64) 10-11

P-F1M62-11A13 HiRDB Staticizer Option Version 10 10-00

P-F1M62-11A15 HiRDB Non Recover Front End Server Version 10 10-00

P-F1M62-11A16 HiRDB Advanced High Availability Version 10 10-00

P-F1M62-11A18 HiRDB Disaster Recovery Light Edition Version 10 10-00

P-F1M62-11A1A HiRDB Accelerator Version 10 10-00

●適用 OS：Red Hat Enterprise Linux 8 (64-bit x86_64)，Red Hat Enterprise Linux 9 (64-bit x86_64)，Red Hat Enterprise Linux 10 (64-bit x86_64)

P-8462-35A1 HiRDB Server Version 10 10-11

P-8462-1DA1 HiRDB/Run Time Version 10(64) 10-11

P-8462-1EA1 HiRDB/Developer's Kit Version 10(64) 10-11

P-F8462-11A13 HiRDB Staticizer Option Version 10 10-00

P-F8462-11A15 HiRDB Non Recover Front End Server Version 10 10-00

P-F8462-11A16 HiRDB Advanced High Availability Version 10 10-00

P-F8462-11A18 HiRDB Disaster Recovery Light Edition Version 10 10-00

P-F8462-11A1A HiRDB Accelerator Version 10 10-00

●適用 OS：Red Hat Enterprise Linux 8 (64-bit x86_64)，Red Hat Enterprise Linux 9 (64-bit x86_64)，Red Hat Enterprise Linux 10 (64-bit x86_64)

P-8362-1BA1 HiRDB/Run Time Version 10 10-11

P-8362-1CA1 HiRDB/Developer's Kit Version 10 10-11

P-8362-3CA1 HiRDB Developer's Suite Version 10 10-11

●適用 OS：Windows Server 2019，Windows Server 2022，Windows Server 2025，Windows 10 Pro (x64)，Windows 10 Enterprise (x64)，Windows 11

P-2962-91A4 HiRDB Server Version 10 10-11

P-2962-7PA4 HiRDB Accelerator Version 10 10-00

P-2962-7HA4 HiRDB Non Recover Front End Server Version 10 10-00

P-2962-7JA4 HiRDB Advanced High Availability Version 10 10-00

●適用 OS：Windows Server 2019，Windows Server 2022，Windows Server 2025，Windows 10，Windows 11

P-2662-11A4 HiRDB/Run Time Version 10 10-11

P-2662-12A4 HiRDB/Developer's Kit Version 10 10-11

P-2662-32A4 HiRDB Developer's Suite Version 10 10-11

●適用 OS：Windows Server 2019，Windows Server 2022，Windows Server 2025，Windows 10 Home (x64)，Windows 10 Pro (x64)，Windows 10 Enterprise (x64)，Windows 11

P-2962-11A4 HiRDB/Run Time Version 10(64) 10-11

P-2962-12A4 HiRDB/Developer's Kit Version 10(64) 10-11

これらのプログラムプロダクトのほかにもこのマニュアルをご利用になれる場合があります。詳細は「リリースノート」でご確認ください。

■ 輸出時の注意

本製品を輸出される場合には、外国為替および外国貿易法の規制ならびに米国輸出管理規則など外国の輸出関連法規をご確認の上、必要な手続きをお取りください。

なお、不明な場合は、弊社担当営業にお問い合わせください。

■ 商標類

記載の会社名、製品名などは、それぞれの会社の商標もしくは登録商標です。

■ 発行

2026 年 4 月　3020-6-560-B0

■ 著作権

All Rights Reserved. Copyright (C) 2018, 2026, Hitachi, Ltd.

変更内容

変更内容(3020-6-560-B0)　HiRDB Version 10 10-11

追加・変更内容	変更個所
pdcpp コマンドが gcc コンパイラの-finput-charset オプションに対応し，プリプロセス時の文字コード指定が正しく扱えるようになりました。 これまで，UTF-8 以外の UAP をプリプロセスする場合に-finput-charset を指定すると，gcc が出力する一時ファイルと UAP の文字コードが一致しないために，エラーになるときがありました。 このサポートによって，文字コード不一致が解消され，プリプロセスができるようになります。	表 8-2
次の機能をサポートしたことによって，セキュリティの強化，ロール機能の利便性が向上しました。 • 監査対象イベントの定義で，オブジェクト種別（ROLE），セッション種別（ROLE）を指定できるようにしました。 • CURRENT_ROLE 値関数によって，現在使用しているロール名を検索できるようにしました。 • 埋込み型 UAP，ODBC から SET ROLE 文を発行できるようにしました。	付録 G.3(41)
pdinfoget コマンドで収集する情報を拡充しました。 また，初期情報（-e オプション）だけ取得できるようになりました。 さらに，UNIX 版で，OS の情報取得を行う pdosinfoget コマンドをサポートしました。 これらによって，HiRDB での障害発生時に早期に対応できるようになります。	表 12-1
HiRDB の適用 OS に Red Hat Enterprise Linux 10 を追加しました。	－

単なる誤字・脱字などはお断りなく訂正しました。

変更内容(3020-6-560-A0)　HiRDB Version 10 10-10

追加・変更内容
表のアクセス権限を一括で管理できるロール機能をサポートしました。これによって，権限の管理や移行が容易になるだけでなく，アクセス制御の一貫性が向上し，セキュリティの強化にもつながります。
ODBC 関数の SQLPrimaryKeys，SQLForeignKeys をサポートしました。これによって，A5:SQLMk2 等のデータベース開発支援ツールと連携し，ER 図のリバースエンジニアリングなどの機能が利用できます。
定期的にパケットを送信する機能では，クライアント環境定義 PDSWATCHTIME と PDSWAITTIME の両方の監視時間をリセットしていましたが，PDSWATCHTIME の監視時間だけをリセットする機能をサポートしました。これによって，コネクションプーリング環境下で，プーリング中に PDSWATCHTIME の時間監視によって切断されることなく，PDSWATCHTIME 及び PDSWAITTIME を適用できます。

追加・変更内容
SQL トレース機能の SQL トレース情報をメモリ上に保持し，エラー発生時にファイルへ出力する方式でのパラメタトレースの取得に対応しました。これによって，パラメタトレースを取得しやすくなり，エラー発生時のトラブルシュートが容易になります。
SELinux を enforcing モードにした環境で HiRDB を構築できるようにしました。これによって，セキュリティ要件の高いシステムでも HiRDB を使用できます。
HiRDB の適用 OS に次の OS を追加しました。 • Windows Server 2025

変更内容(3020-6-560-90)　HiRDB Version 10 10-09

追加・変更内容
SQL トレース機能で，SQL トレース情報をメモリ上に退避しエラー発生時にファイル出力する方式をサポートしました。従来の方式と比べて，SQL トレース取得時の性能への影響を低減しています。 これによって，SQL トレースを取得しやすくなり，エラー発生時のトラブルシュートが容易になります。
Type4 JDBC ドライバの次のメソッドで文字列型をサポートしました。 • ResultSet クラス，及び CallableStatement クラスの getBytes • PreparedStatement クラス，及び CallableStatement クラスの setBytes
ODBC ドライバで発生したエラーの情報を ODBC エラーログファイルへ出力する機能をサポートしました。 これによって，エラー発生時のトラブルシュートが容易になります。
データベースの内容を複製した参照専用のデータベースを構築できるようにしました。 これによって，アプリケーション側で更新用 DB と参照用 DB（リードレプリカ）を使い分け，負荷分散及びシステム全体のスループット向上につながります。
JDBC ドライバの DatabaseMetaData インタフェースに CURRENT_ROLE を追加しました。

単なる誤字・脱字などはお断りなく訂正しました。

変更内容(3020-6-560-80)　HiRDB Version 10 10-08

追加・変更内容
クライアントとサーバ間のネットワーク上で送受信するデータを暗号化する機能をサポートしました。 これによって，パケットスニファリングなどで悪意のあるユーザが不正にデータを参照することを防止できます。
JDBC 4.1，JDBC 4.2 及び JDBC 4.3 に対応した Type4 JDBC ドライバをサポートしました。 これによって，JDBC4.3 で追加された以下の機能を使用したアプリケーション開発や JDBC4.3 を前提とした製品と連携できるようになります。 • Connection インタフェース，ResultSet インタフェース及び Statement インタフェースでの try-with-resources 文 • DatabaseMetaData インタフェースでの擬似列，自動生成キーの返却有無，LOB の最大論理サイズ，REF CURSOR のサポート有無及び Sharding サポート有無を返す機能 • Connection インタフェースでのスキーマを設定する機能 • Statement インタフェースでの ResultSet クローズ時に Statement クローズを可能とする機能

追加・変更内容
• Statement インタフェースでの大きい更新カウント（2147483648 以上）を扱う機能 • Statement インタフェースでの SQL 識別子を判断又は返却する機能
UNIX 版同様に Windows 版でも，クライアント–サーバ間の接続で新たな接続方式をサポートしました。 これによって，クライアント側ファイアウォールでポート開放が不要となるほか，NAPT が設定されたネットワーク環境で接続できます。

はじめに

このマニュアルは，次の項目について説明したものです。

- プログラムプロダクト　ノンストップデータベース　HiRDB Version 10 のデータベース言語である SQL を使用して，ユーザアプリケーションプログラムを開発するための基礎技術
- HiRDB クライアントの環境設定

なお，ここに記載されていない前提情報については，マニュアル「HiRDB Version 10 解説」を参照してください。

■ 対象読者

HiRDB Version 10（以降，**HiRDB** と表記します）で UAP を作成する方，UAP を実行する方（HiRDB クライアントを使用する方），およびシステム管理者を対象にしています。

このマニュアルは次に示す知識があることを前提に説明しています。

- Windows のシステム管理の基礎的な知識（Windows 版の場合）
- UNIX または Linux のシステム管理の基礎的な知識（UNIX 版の場合）
- SQL の基礎的な知識
- C 言語のプログラミング，COBOL 言語のプログラミング，または Java のプログラミングの知識

なお，このマニュアルは，マニュアル「HiRDB Version 10 解説」を前提としていますので，あらかじめお読みいただくことをお勧めします。

■ パス名の表記

- パス名の区切りは「¥」で表記しています。UNIX 版 HiRDB を使用している場合はマニュアル中の「¥」を「/」に置き換えてください。ただし，Windows 版と UNIX 版でパス名が異なる場合は，それぞれのパス名を表記しています。
- HiRDB 運用ディレクトリのパスを%PDDIR%と表記します。ただし，Windows 版と UNIX 版でパス名が異なるため，それぞれを表記する場合，UNIX 版は$PDDIR と表記します。例を次に示します。

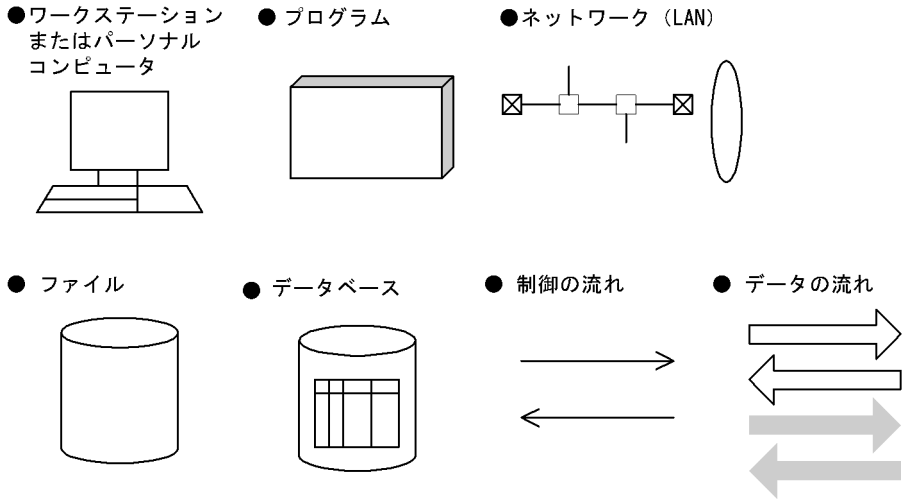
  Windows 版：%PDDIR%¥CLIENT¥UTL¥

  UNIX 版：$PDDIR/client/lib/

- Windows のインストールディレクトリのパスを%windir%と表記します。

■ 図中で使用している記号

このマニュアルの図中で使用している記号を次のように定義します。



■ このマニュアルで使用している記号

形式および説明で使用している記号を次に示します。ここで説明する文法記述記号は，説明のための記号なので実際には記述しないでください。

記号	意　　味
{　　}	この記号で囲まれている複数の項目の中から一つを選択することを示します。 （例） PDDLKPRIO={96 ｜ 64 ｜ 32} これは，PDDLKPRIO オペランドの指定値として 96，64 および 32 から選択できることを示します。
[　　]	この記号で囲まれた項目は省略できることを示します。 （例） PDHOST=システムマネジャのホスト名[，予備系システムマネジャのホスト名] これは，予備系システムを省略できることを示します。
｜（ストローク）	記号{　　}で囲まれた複数の項目を一つずつの項目に区切ることを示します。 （例） PDXAMODE={0 ｜ 1} これは，PDXAMODE オペランドの設定値として 0，および 1 の項目に区切ることを示します。
（下線）	記号{　　}で囲まれた複数の項目の中の一つにだけ使用され，記号内の項目をすべて省略したとき，システムが設定する標準値を示します。

記号	意　　味
	（例） PDDFLNVAL={USE ｜ NOUSE} これは，PDDFLNVAL オペランドを省略したときに PDDFLNVAL=NOUSE と仮定されることを示します。
～	この記号の後にユーザ指定値の属性を示します。
《　》	ユーザが指定しなかった場合に仮定される値を示します。
<　>	ユーザ指定値の構文要素を示します。
((　))	ユーザ指定値の指定範囲を示します。

■ このマニュアルで使用している構文要素記号

このマニュアルで使用している構文要素記号を次に示します。

構文要素記号	意　　味
<英字>	アルファベット（A～Z，a～z）と下線（_）
<英字記号>	アルファベット（A～Z，a～z）と＃，@，￥
<英数字>	英字と数字（0～9）
<英数字記号>	英字記号と数字
<符号なし整数>	数字
<符号なし 10 進数>※1	数字（0～9 の並び）ピリオド（.）数字（0～9 の並び）
<識別子>※2	先頭がアルファベットの英数字列
<文字列>	任意の文字の配列
<記号名称>	先頭が英字記号の英数字記号 UNIX 版の場合，￥は使用できません。
<パス名>※3	UNIX 版の場合：/，英数字，ピリオド（.），＃，および@ Windows 版の場合：￥，英数字，ピリオド（.），空白，丸括弧，＃，および@

注

すべて半角文字を使用してください。また，英字の大文字と小文字は区別されます。さらに，パス名は使用している OS に依存します。

注※1

ピリオドの前の数字がすべて 0 の場合，ピリオドより前の 0 を省略できます。また，ピリオドの後ろの数字がすべて 0 の場合，ピリオド以降を省略できます。

例１：0.008 → .008

例２：15.000 → 15

注※2

RD エリア名の場合は，先頭が英字記号で始まる英数字記号，下線（_），ハイフン（-），および空白となります。また，RD エリア名に空白が含まれる場合は，引用符（"）で囲んでください。

ホスト名の場合は，アルファベット（A～Z，a～z），数字，ピリオド（.），ハイフン（-），下線（_），および@で構成される文字列となります。

注※3

パス名に空白，または丸括弧を含む場合は，前後を引用符（"）で囲んでください。

なお，Windows 版の場合，コロン（：）をドライブ名に使用できます。

■ HiRDB のデータベース言語の出典

このマニュアルで記述する HiRDB のデータベース言語仕様は，次に示す規格を基に日立製作所独自の解釈と仕様を追加したものです。原開発者に謝意を表するとともに，仕様の出典を示します。

- JIS X 3005 規格群　データベース言語 SQL
- ISO/IEC 9075　Information technology - Database languages - SQL -

注

JIS：日本工業規格（Japanese Industrial Standard）

ISO：国際標準化機構（International Organization for Standardization）

IEC：国際電気標準会議（International Electrotechnical Commission）

目次

前書き　2
変更内容　4
はじめに　7

1　概要　32
1.1　UAP の開発の流れ　33
1.2　UAP の特長　34
1.2.1　UAP の形式　34
1.2.2　HiRDB で使用できる SQL 一覧　35
1.3　HiRDB とのインタフェース　36
1.4　UAP の動作環境　37
1.5　HiRDB サーバ，HiRDB/Run Time および HiRDB/Developer's Kit の機能差　42

2　データベースの操作　44
2.1　データベースのデータ表現　45
2.1.1　リレーショナルデータベースの表　45
2.1.2　オブジェクトリレーショナルデータベースの表　47
2.2　カーソルの利用　49
2.2.1　カーソルを使用した検索及び使用しない検索　49
2.3　データの検索　52
2.3.1　1 個の表からの検索　52
2.3.2　複数の表からの検索　55
2.3.3　FIX 属性の表の検索　56
2.4　データの更新　58
2.4.1　カーソルを使用した更新　58
2.4.2　条件指定による更新　59
2.4.3　FIX 属性の表の更新　60
2.4.4　繰返し列がある表の更新　61
2.5　データの削除　63
2.5.1　カーソルを使用した削除　63
2.5.2　条件指定による削除　64
2.5.3　表の全行削除　65
2.6　データの挿入　67
2.6.1　列単位の挿入　67
2.6.2　FIX 属性の表への行の挿入　68

2.6.3 繰返し列がある表への行の挿入 68
2.7 特定データの探索 70
2.7.1 特定の範囲内のデータの探索 70
2.7.2 特定の文字パターンの探索 73
2.7.3 ナル値でないデータの探索 74
2.7.4 複数の条件を満たすデータの探索 75
2.7.5 論理述語を使用した検索 76
2.7.6 構造化繰返し述語を使用した検索 76
2.7.7 副問合せを使用した検索 77
2.8 データの演算 81
2.8.1 数値データの四則演算 81
2.8.2 日付，時刻データの演算 81
2.9 データの加工 83
2.9.1 データのグループ分け 83
2.9.2 データの並べ替え 84
2.9.3 重複したデータの排除 84
2.10 表の外結合 86
2.11 ビュー表の定義と操作 89
2.11.1 ビュー表の定義例及び操作例 89
2.12 抽象データ型を含む表のデータ操作 95
2.12.1 SGMLTEXT 型の場合 95
2.12.2 XML 型の場合 102
2.12.3 ユーザが定義する抽象データ型の場合 106

3 UAP の設計 111
3.1 UAP 中での SQL の基本構成 112
3.1.1 UAP 中での SQL の基本構成の説明 112
3.2 UAP の記述 118
3.2.1 UAP の記述言語 118
3.2.2 インタフェース領域 118
3.2.3 整合性制約 119
3.2.4 SQL を使用した検索方法の分類 120
3.2.5 静的 SQL と動的 SQL 121
3.2.6 マルチスレッドを使用した UAP の SQL の実行 128
3.3 トランザクション制御 132
3.3.1 HiRDB システムへの接続と切り離し 132
3.3.2 トランザクションの開始と終了 132
3.3.3 同期点の設定とロールバックの設定 132
3.3.4 OLTP 環境での UAP のトランザクション管理 133

3.3.5 トランザクションの移行 135
3.4 排他制御 137
3.4.1 排他制御の単位 137
3.4.2 排他制御のモード 138
3.4.3 排他の期間 162
3.4.4 デッドロックと回避策 162
3.4.5 無排他条件判定 172
3.4.6 インデクスキー値無排他 174
3.4.7 コミットしていない削除データの排他制御 177
3.4.8 残存エントリによる排他待ちの回避（行識別子の再利用抑止） 182
3.4.9 残存エントリによる排他待ちの回避（ユニークチェック用排他使用） 189
3.4.10 UAP でできる排他制御 192
3.4.11 SQL 文の種類とインデクスの種別による排他制御の順序 194
3.4.12 行に掛かる排他制御の順序 204
3.4.13 インデクスキー値の排他資源の作成方法 212
3.4.14 複数トランザクションで検索と更新をする場合 213
3.5 カーソルの効果 216
3.5.1 カーソルを使用して表を操作するときの留意事項 216
3.5.2 FOR UPDATE 句と FOR READ ONLY 句の使い分け 218
3.5.3 カーソル宣言と排他の関係 219
3.5.4 ホールダブルカーソル 224
3.5.5 カーソルの使用例 228
3.6 SQL のエラーの判定と処置 231
3.6.1 エラーの判定 231
3.6.2 エラーの自動判定 234

4 性能向上，操作性向上に関する UAP の設計 236
4.1 インデクス 237
4.1.1 インデクスの提案 237
4.1.2 未使用インデクスの調査 240
4.1.3 インデクス検索時の留意事項 242
4.2 表に対する操作 250
4.2.1 FIX 属性の表 250
4.2.2 採番業務で使用する表 250
4.2.3 文字集合を使用した表 257
4.3 ストアドプロシジャ，ストアドファンクション 261
4.3.1 ストアドプロシジャの定義 261
4.3.2 ストアドファンクションの定義 270
4.3.3 ストアドファンクションを定義，又は削除するときの注意事項 279

4.4 トリガ 281
4.5 SQL の最適化 282
4.5.1 SQL 最適化モード 283
4.5.2 最適化方法の種類 295
4.5.3 SQL の最適化の指定方法 296
4.5.4 フロータブルサーバの割り当て方法（HiRDB/パラレルサーバ限定） 297
4.5.5 グループ分け処理方式（HiRDB/パラレルサーバ限定） 303
4.5.6 結合方式 307
4.5.7 検索方式 317
4.5.8 外への参照のない副問合せの実行方式 323
4.5.9 外への参照のある副問合せの実行方式 328
4.5.10 ハッシュジョイン，副問合せのハッシュ実行を適用する場合の準備 331
4.5.11 探索高速化条件の導出 337
4.6 データ保証レベル 344
4.6.1 データ保証レベルの指定方法 344
4.6.2 データ保証レベルの種類 345
4.6.3 データ保証レベルを指定した場合の検索結果の例 346
4.7 ブロック転送機能 348
4.7.1 機能概要 348
4.7.2 使用方法 348
4.7.3 サーバ，クライアント間の通信バッファサイズの指定 349
4.7.4 1 回の通信で転送する行数 349
4.7.5 注意事項 350
4.8 配列を使用した機能 352
4.8.1 配列を使用した FETCH 機能 352
4.8.2 配列を使用した INSERT 機能 359
4.8.3 配列を使用した UPDATE 機能 369
4.8.4 配列を使用した DELETE 機能 372
4.9 グループ分け高速化機能 375
4.9.1 概要 375
4.9.2 適用条件 375
4.9.3 指定方法 376
4.9.4 チューニング方法 376
4.10 複数接続機能 378
4.10.1 機能概要 378
4.10.2 処理概要 379
4.10.3 コーディング例 384
4.10.4 規則 389
4.11 絞込み検索 390

4.11.1 絞込み検索とは 390
4.11.2 絞込み検索をするためには 390
4.11.3 リストを使用した検索 391
4.11.4 リストを使用するトランザクションでロールバックが発生した場合の処置 393
4.11.5 HiRDB の起動と停止時のリストの自動削除 393
4.11.6 リスト使用時の注意事項 394
4.12 BLOB データのファイル出力機能 397
4.12.1 BLOB データのファイル出力機能とは 397
4.12.2 適用基準 398
4.12.3 指定方法 398
4.12.4 BLOB データのファイル出力機能を使用する場合の留意点 398
4.12.5 BLOB データのファイル出力機能を使用した例 399
4.13 BLOB データ，BINARY データの部分的な更新・検索 401
4.13.1 BLOB データ，BINARY データの部分的な更新・検索とは 401
4.13.2 使用例 401
4.13.3 BLOB データ，BINARY データの部分的な更新・検索を行う場合の留意点 403
4.14 先頭から n 行の検索結果を取得する機能 405
4.14.1 概要 405
4.14.2 留意事項 405
4.14.3 アクセスパスの確認方法 406
4.15 自動再接続機能 407
4.15.1 適用基準 407
4.15.2 再接続する契機 407
4.15.3 自動再接続での CONNECT 処理 412
4.15.4 留意事項 412
4.16 位置付け子機能 413
4.16.1 位置付け子機能とは 413
4.16.2 適用基準 414
4.16.3 使用方法 414
4.16.4 使用例 414
4.16.5 留意事項 416
4.17 総ヒット件数返却機能 417
4.17.1 機能概要 417
4.17.2 使用例 417
4.17.3 留意事項 417
4.18 RD エリア名を指定した検索，更新，又は削除 419
4.18.1 機能概要 419
4.18.2 使用例 419
4.18.3 留意事項 420

4.19 自動採番機能 421
4.19.1 順序数生成子とは 421
4.19.2 順序数生成子の定義 421
4.19.3 順序数生成子の削除 424
4.19.4 順序数生成子が生成する順序番号の取得 424
4.19.5 使用例 424
4.19.6 留意事項 425
4.20 プラグインインデクスでの他インデクス絞り込み結果の利用 431
4.20.1 概要 431
4.20.2 使用方法 431
4.20.3 留意事項 431
4.20.4 使用例 432

5 オブジェクトリレーショナルデータベースをアクセスする UAP 作成時の注意事項 434
5.1 抽象データ型及びユーザ定義関数を使用する場合の注意事項 435
5.1.1 埋込み変数のデータ型 435
5.1.2 定数のデータ型 435
5.2 プラグイン提供関数の制限 437
5.2.1 プラグイン提供関数間の値の受け渡しに関する制限 437
5.2.2 プラグイン提供関数の実行方法に関する制限 439
5.2.3 受渡し値を格納したリストについての注意事項 442

6 クライアントの環境設定 445
6.1 HiRDB クライアントの種類 446
6.2 HiRDB クライアントの環境設定手順 447
6.3 HiRDB クライアントのインストール及びアンインストール 448
6.3.1 UNIX クライアントでのインストール 448
6.3.2 Windows クライアントでの GUI によるインストール 448
6.3.3 UNIX クライアントでのアンインストール 451
6.3.4 Windows クライアントでの GUI によるアンインストール 451
6.3.5 Windows クライアントでのサイレントインストール及びアンインストール 451
6.4 HiRDB クライアントのディレクトリ及びファイル構成 460
6.4.1 UNIX クライアントのディレクトリ及びファイル構成 460
6.4.2 Windows クライアントのディレクトリ及びファイル構成 473
6.5 hosts ファイルの設定 491
6.5.1 hosts ファイルの設定方法 491
6.6 クライアント環境定義（環境変数の設定） 492
6.6.1 クライアント環境定義の設定形式 492
6.6.2 OLTP 下の X/Open に従った API を使用した UAP をクライアントとする場合の指定方法 502

6.6.3 クライアント環境定義の一覧 515
6.6.4 クライアント環境定義の設定内容 526
6.6.5 HiRDB サーバと接続するための環境変数と接続形態との関係 668
6.7 環境変数のグループ登録 671
6.7.1 UNIX 環境の場合 671
6.7.2 Windows 環境の場合（レジストリ登録） 672
6.7.3 Windows 環境の場合（ファイル登録） 679
6.8 クライアントライブラリのメモリ容量見積もり 680
6.8.1 クライアントライブラリのメモリ容量の見積もり方法 680
6.9 HiRDB クライアントの SELinux セキュリティコンテキスト設定及び解除 688
6.9.1 前提条件 688
6.9.2 SELinux セキュリティコンテキストを設定する方法 689
6.9.3 SELinux セキュリティコンテキストを解除する方法 689

7 UAP の作成 691
7.1 埋込み型 UAP の概要 692
7.1.1 UAP の基本構成 692
7.1.2 UAP の構成要素 692
7.2 C 言語による UAP の作成 694
7.2.1 記述規則 694
7.2.2 プログラム例題 701
7.3 COBOL 言語による UAP の作成 727
7.3.1 記述規則 727
7.3.2 プログラム例題 733
7.4 C++言語による UAP の作成 758
7.4.1 記述規則 758
7.5 OOCOBOL 言語による UAP の作成 759
7.5.1 記述規則 759
7.6 64 ビットモードでの UAP の作成 760
7.6.1 64 ビットモード対応の UAP で使用できる言語，及び機能 760
7.6.2 SQL 連絡領域の違い 760
7.6.3 SQL 記述領域の違い 760
7.6.4 SQL のデータ型と C 言語のデータ記述の違い 761
7.6.5 表分割ハッシュ関数を使用した UAP の作成方法の違い（Linux 版限定） 762
7.6.6 32 ビットモードから 64 ビットモードへの HiRDB クライアントへの移行 762

8 UAP 実行前の準備 763
8.1 UAP の実行手順 764
8.1.1 C 言語で作成した UAP の実行手順 764

8.1.2 COBOL 言語で作成した UAP の実行手順 765
8.2 プリプロセス 767
8.2.1 プリプロセスの概要 767
8.2.2 UNIX 環境でのプリプロセス 768
8.2.3 Windows 環境でのプリプロセス 785
8.2.4 プリプロセサ宣言文の有効化 799
8.2.5 埋込み SQL 宣言節の不要化 801
8.2.6 ポインタでの埋込み変数指定 802
8.2.7 構造体の参照 804
8.2.8 プリプロセサの/E2，/E3 オプションを指定した場合のポインタ，構造体，及び構造体修飾の使用可否 806
8.2.9 COBOL 言語で複数接続機能を使用する場合の注意事項 808
8.2.10 バージョンによるプリプロセスオプションの変更点 810
8.2.11 COBOL のビッグエンディアンオプションへの対応 811
8.3 コンパイルとリンケージ 816
8.3.1 コンパイル，リンケージ時に指定するライブラリ 816
8.3.2 UNIX 環境でのコンパイルとリンケージ 821
8.3.3 Windows 環境でのコンパイルとリンケージ 826
8.3.4 複数接続機能を使用する場合のコンパイルとリンケージ 829
8.4 注意事項 832
8.4.1 UAP 実行時の注意事項 832
8.4.2 X/Open に従った API（TX_関数）を使用した UAP の実行 836
8.4.3 COBOL2002 の Unicode 機能を使用した UAP の実行 844
8.4.4 XDM/RD と UNIFY2000 で作成した UAP の移行性 845
8.4.5 HiRDB をバージョンアップした場合に必要な作業 846
8.4.6 バージョン，リビジョンによるクライアント環境定義及びプリプロセスオプションの変更点 846
8.4.7 Red Hat Enterprise Linux 10，Oracle Linux 10 以降での UAP 作成 853

9 Java ストアドプロシジャ，Java ストアドファンクション 855

9.1 概要 856
9.2 外部 Java ストアドルーチンの作成から実行までの各作業 859
9.2.1 外部 Java ストアドルーチンの作成 859
9.2.2 JAR ファイルの新規登録 861
9.2.3 外部 Java ストアドルーチンの定義 862
9.2.4 外部 Java ストアドルーチンの実行 863
9.3 外部 Java ストアドルーチンのプログラム例 865
9.3.1 プログラム例 865
9.3.2 HiRDB が提供する外部 Java ストアドルーチンのサンプル 867
9.4 Java プログラム作成時の注意事項 879
9.4.1 Type2 JDBC ドライバ又は Type4 JDBC ドライバの使用 879

9.4.2 実行できないメソッド 879
9.4.3 パッケージ，クラス，及びメソッドの定義 880
9.4.4 パラメタ入出力モードのマッピング（Java ストアドプロシジャ限定） 881
9.4.5 結果集合返却機能（Java ストアドプロシジャ限定） 882
9.4.6 Java ストアドプロシジャ中のコネクション 886
9.4.7 結果集合の解放 887
9.5 テスト，デバッグ時の注意事項 888
9.5.1 Java ストアドプロシジャ用の Java プログラムの場合 888
9.5.2 Java ストアドファンクション用の Java プログラムの場合 889
9.6 JAR ファイル作成時の注意事項 891
9.6.1 Class ファイルを統合する場合 891
9.6.2 Java ファイルを統合する場合 892

10 C ストアドプロシジャ，C ストアドファンクション 894
10.1 概要 895
10.2 外部 C ストアドルーチンの作成から実行までの各作業 896
10.2.1 外部 C ストアドルーチンの作成 898
10.2.2 C ライブラリファイルの新規登録 902
10.2.3 外部 C ストアドルーチンの定義 903
10.2.4 外部 C ストアドルーチンの実行 904
10.3 外部 C ストアドルーチンのプログラム例 905
10.4 C プログラム作成時の制限事項 907

11 UAP の障害対策 909
11.1 トラブルシュート 910
11.1.1 SQL トレース機能 910
11.1.2 クライアントエラーログ機能 949
11.1.3 拡張 SQL エラー情報出力機能 952
11.1.4 UAP 統計レポート機能 963
11.1.5 コマンドトレース機能 995
11.1.6 SQL トレース動的取得機能 997
11.1.7 再接続トレース機能 999
11.1.8 HiRDB SQL Tuning Advisor 用アクセスパス情報ファイル 1001
11.2 UAP 障害の回復 1006
11.2.1 UAP 障害時の回復方法 1006

12 UAP からのコマンド実行 1008
12.1 概要 1009
12.2 COMMAND EXECUTE からコマンドを実行するための準備 1010
12.2.1 HiRDB/シングルサーバの場合 1010

12.2.2 HiRDB/パラレルサーバの場合 1012
12.3 コマンドの実行可否 1015

13 ODBC 対応アプリケーションプログラムからの HiRDB アクセス 1021
13.1 ODBC 対応アプリケーションプログラム 1022
13.2 ODBC2.0 ドライバのインストール 1023
13.3 ODBC3.5 ドライバのインストールと環境変数の設定 1026
13.3.1 インストール 1026
13.3.2 UNIX 版 HiRDB ODBC3.5 ドライバ固有の ODBC 環境定義 1031
13.3.3 環境変数の設定（Windows 版の場合） 1032
13.3.4 ODBC3.5 ドライバのバージョン情報の確認方法 1032
13.3.5 接続情報の優先順位 1032
13.3.6 文字コード変換機能の設定 1034
13.4 HiRDB が提供する ODBC 関数 1037
13.5 ODBC 関数のデータ型と HiRDB のデータ型との対応 1041
13.5.1 ODBC 関数と HiRDB とのデータ型の対応 1041
13.6 ODBC 関数の各属性の指定可否 1045
13.6.1 SQLSetConnectAttr 1045
13.6.2 SQLGetConnectAttr 1046
13.6.3 SQLSetDescField 1047
13.6.4 SQLGetDescField 1048
13.6.5 SQLSetEnvAttr 1050
13.6.6 SQLGetEnvAttr 1050
13.6.7 SQLSetStmtAttr 1051
13.6.8 SQLGetStmtAttr 1052
13.6.9 SQLGetInfo 1054
13.6.10 SQLColAttribute 1063
13.7 HiRDB ODBC3.5 ドライバの接続文字列 1065
13.7.1 認可識別子及びパスワードの小文字指定について 1065
13.8 ODBC 関数の非同期実行 1067
13.8.1 ODBC 関数の非同期実行とは 1067
13.8.2 ODBC 関数の非同期実行の手順 1067
13.8.3 ODBC 関数の非同期実行のキャンセル 1069
13.8.4 コーディング例 1070
13.9 カーソルライブラリの設定 1071
13.10 ファイル DSN について 1072
13.11 Unicode の UAP の実行 1073
13.11.1 Unicode の UAP で使用できる ODBC 関数 1073
13.11.2 注意事項 1074

13.12 チューニング，トラブルシュート 1076
13.12.1 ODBC エラーログ 1076
13.12.2 複数行検索をする UAP で性能が出ない場合 1079
13.12.3 頻繁に CONNECT，DISCONNECT をする UAP の場合 1079
13.12.4 HiRDB に要求される SQL 文を確認したい場合 1079
13.12.5 その他 1080
13.13 ODBC 経由で HiRDB をアクセスする場合に使用できない機能について 1081
13.14 UNIX 版 HiRDB ODBC3.5 ドライバを使用する場合の留意事項 1082
13.15 .NET Framework Data Provider for ODBC による SQL 文の自動生成 1084
13.16 ADO 及び ADO.NET から ODBC を経由して HiRDB サーバにアクセスする場合の留意事項 1085
13.17 ODBC3.5 ドライバが返却する SQLSTATE 1087
13.18 Python アプリケーションからの HiRDB アクセス 1097
13.18.1 サポート対象 1097
13.18.2 データ型 1097
13.18.3 使用できない機能 1100

14 OLE DB 対応アプリケーションプログラムからの HiRDB アクセス 1102
14.1 概要 1103
14.1.1 OLE DB の概要 1103
14.2 接続インタフェース 1104
14.2.1 レジストリ情報 1104
14.2.2 接続プロパティ 1106
14.3 スキーマ情報 1107
14.4 データ型の対応 1109
14.5 障害対策 1110
14.5.1 トラブルシュート機能 1110
14.6 留意事項 1111
14.6.1 ADO でのカーソルについて 1111

15 ADO.NET 対応アプリケーションプログラムからの HiRDB アクセス 1112
15.1 概要 1113
15.1.1 HiRDB データプロバイダ for .NET Framework 1113
15.1.2 HiRDB データプロバイダ for .NET Framework の前提プログラム 1113
15.1.3 HiRDB データプロバイダ for .NET Framework と対応する ADO.NET のバージョン 1117
15.2 HiRDB データプロバイダ for .NET Framework のインストール 1119
15.2.1 インストール手順 1119
15.2.2 インストールされるファイル 1119
15.2.3 バージョン情報の確認 1120
15.3 HiRDB データプロバイダ for .NET Framework のクラス一覧 1121

15.4 HiRDB データプロバイダ for .NET Framework のメンバー一覧 1122
15.4.1 HiRDBCommand のメンバー一覧 1122
15.4.2 HiRDBCommandBuilder のメンバー一覧 1123
15.4.3 HiRDBConnection のメンバー一覧 1125
15.4.4 HiRDBDataAdapter のメンバー一覧 1126
15.4.5 HiRDBDataReader のメンバー一覧 1127
15.4.6 HiRDBException のメンバー一覧 1129
15.4.7 HiRDBParameter のメンバー一覧 1130
15.4.8 HiRDBParameterCollection のメンバー一覧 1132
15.4.9 HiRDBProviderFactory のメンバー一覧 1134
15.4.10 HiRDBRowUpdatedEventArgs のメンバー一覧 1134
15.4.11 HiRDBRowUpdatingEventArgs のメンバー一覧 1135
15.4.12 HiRDBTransaction メンバー一覧 1135
15.5 HiRDB データプロバイダ for .NET Framework のインタフェース 1137
15.5.1 HiRDBCommand 1137
15.5.2 HiRDBCommandBuilder 1141
15.5.3 HiRDBConnection 1147
15.5.4 HiRDBDataAdapter 1153
15.5.5 HiRDBDataReader 1154
15.5.6 HiRDBException 1166
15.5.7 HiRDBParameter 1166
15.5.8 HiRDBParameterCollection 1172
15.5.9 HiRDBProviderFactory 1179
15.5.10 HiRDBRowUpdatedEventArgs 1181
15.5.11 HiRDBRowUpdatingEventArgs 1182
15.5.12 HiRDBTransaction 1182
15.6 HiRDB データプロバイダ for .NET Framework の留意事項 1185
15.6.1 グローバルアセンブリキャッシュへの配置 1185
15.6.2 各メソッド，プロパティについての留意事項 1187
15.6.3 UAP 開発時のプラットフォームターゲットの設定 1189
15.7 HiRDB データプロバイダ for .NET Framework のデータ型 1191
15.7.1 DbType プロパティと HiRDBType プロパティ 1191
15.7.2 UAP で使用するデータ型とアクセサ 1192
15.7.3 HiRDB データプロバイダ for .NET Framework の型変換 1193
15.8 接続プーリング機能 1200
15.8.1 接続プーリング機能とは 1200
15.8.2 使用方法 1200
15.8.3 HiRDB データプロバイダ for .NET Framework の接続プーリングにおける接続回復機能 1200
15.9 DbProviderFactory を使用したプロバイダに依存しないコード 1203

15.9.1 DbProviderFactory を使用したプロバイダに依存しないコードの作成方法 1203
15.10 HiRDB データプロバイダ for .NET Framework のトラブルシュート機能 1206
15.10.1 メソッドトレースの取得方法 1206
15.10.2 メソッドトレースの出力規則 1206
15.10.3 メソッドトレース情報の見方 1206
15.10.4 メソッドトレースファイルのバックアップの取得 1209
15.11 HiRDB データプロバイダ for .NET Framework を使用した UAP 例 1210
15.11.1 データベースへの接続 1210
15.11.2 SQL 文の実行 1212
15.11.3 トランザクションの実行 1213
15.11.4 検索文の実行 1216
15.11.5 配列を使用した INSERT 機能の実行 1217
15.11.6 繰返し列の実行 1218
15.11.7 SQL 文のエラー判定とエラー情報の取得 1220
15.12 パラメタのデータ長及びデータタイプの仮定 1222
15.12.1 省略できるプロパティ 1222
15.12.2 HiRDB データプロバイダ for .NET Framework が仮定するデータタイプ 1223
15.12.3 HiRDB データプロバイダ for .NET Framework が仮定するデータ長 1224
15.12.4 プロパティの明示的な指定の有無と HiRDB データプロバイダ for .NET Framework による仮定可否 1225
15.12.5 仮定を適用するプロパティの初期化 1227
15.12.6 仮定値の取得 1227
15.12.7 注意事項 1228
15.13 構成ファイル 1229
15.13.1 適用契機 1229
15.13.2 サポートする構成ファイル 1229
15.13.3 HiRDB データプロバイダ for .NET Framework がサポートする指定可能なキー 1229
15.13.4 指定方法 1230
15.13.5 指定値の優先順位 1230
15.13.6 指定例 1231
15.13.7 例外 1231
15.13.8 .NET Framework の仕様に関連する注意事項 1231
15.14 接続情報の優先順位 1233

16 Type2 JDBC ドライバ 1235

16.1 インストールと環境設定 1236
16.1.1 インストール 1236
16.1.2 環境設定 1236
16.1.3 メソッドの略記について 1237
16.2 JDBC1.0 機能 1238

16.2.1 Driver クラス 1238
16.2.2 Connection クラス 1246
16.2.3 Statement クラス 1246
16.2.4 PreparedStatement クラス 1247
16.2.5 CallableStatement クラス 1248
16.2.6 ResultSet クラス 1249
16.2.7 ResultSetMetaData クラス 1249
16.2.8 DatabaseMetaData クラス 1251
16.2.9 SQLWarning クラス 1252
16.3 JDBC2.0 基本機能 1254
16.3.1 結果セットの拡張 1254
16.3.2 バッチ更新 1256
16.3.3 追加されたデータ型 1260
16.4 JDBC2.0 Optional Package 1268
16.4.1 DataSource と JNDI を使用した DB 接続 1268
16.4.2 接続プール 1271
16.4.3 分散トランザクション 1272
16.5 JAR ファイルアクセス機能 1275
16.5.1 クラス名 1275
16.5.2 メソッド名 1275
16.6 Array クラス 1278
16.6.1 getArray 1278
16.6.2 getResultSet 1278
16.7 繰返し列を?パラメタにしたときの値の指定方法 1280
16.7.1 Array インタフェースを実装したクラスのオブジェクトの指定方法 1280
16.7.2 配列オブジェクトを setObject メソッドで指定する方法 1281
16.7.3 繰返し列の要素と?パラメタに指定したオブジェクトとの関係 1281
16.7.4 繰返し列の途中の要素にナル値を指定する方法 1281
16.8 HiRDB JDBC ドライバの提供機能 1282
16.8.1 提供クラス 1282
16.8.2 setBlockUpdate 1282
16.8.3 getBlockUpdate 1283
16.9 BLOB 型を使用する場合の注意事項 1285
16.9.1 各メソッドの処理内容と注意事項 1285
16.9.2 ?パラメタでの指定方法 1286
16.10 システムプロパティの設定 1287
16.10.1 配列機能の設定 1287
16.10.2 SQL の検索項目，又は?パラメタの最大数の設定 1288
16.11 接続情報設定／取得インタフェース 1291

16.11.1 setDescription 1292
16.11.2 getDescription 1294
16.11.3 setDBHostName 1294
16.11.4 getDBHostName 1295
16.11.5 setEncodeLang 1296
16.11.6 getEncodeLang 1297
16.11.7 setUser 1298
16.11.8 getUser 1299
16.11.9 setPassword 1299
16.11.10 getPassword 1300
16.11.11 setXAOpenString 1301
16.11.12 getXAOpenString 1302
16.11.13 setXACloseString 1303
16.11.14 getXACloseString 1303
16.11.15 setRMID 1304
16.11.16 getRMID 1305
16.11.17 setXAThreadMode 1305
16.11.18 getXAThreadMode 1306
16.11.19 setCommit_Behavior 1307
16.11.20 getCommit_Behavior 1309
16.11.21 setBlockUpdate 1309
16.11.22 getBlockUpdate 1310
16.11.23 setLONGVARBINARY_Access 1311
16.11.24 getLONGVARBINARY_Access 1312
16.11.25 setSQLInNum 1313
16.11.26 getSQLInNum 1314
16.11.27 setSQLOutNum 1314
16.11.28 getSQLOutNum 1315
16.11.29 setSQLWarningLevel 1316
16.11.30 getSQLWarningLevel 1317
16.11.31 setClear_Env 1317
16.11.32 getClear_Env 1318
16.12 データ型，文字コード 1320
16.12.1 データ型 1320
16.12.2 文字コード変換機能 1321
16.13 制限事項があるクラスとメソッド 1323
16.13.1 Driver クラス 1323
16.13.2 Connection クラス 1323
16.13.3 Statement クラス 1324

16.13.4 PreparedStatement クラス 1324
16.13.5 CallableStatement クラス 1325
16.13.6 ResultSet クラス 1325
16.13.7 ResultSetMetaData クラス 1327
16.13.8 DatabaseMetaData クラス 1327
16.13.9 Blob クラス 1333
16.13.10 Array クラス 1333

17 Type4 JDBC ドライバ 1335
17.1 インストールと環境設定 1336
17.1.1 前提条件 1336
17.1.2 インストール 1336
17.1.3 環境設定 1337
17.1.4 メソッドの略記について 1338
17.2 DriverManager クラスによる DB 接続 1339
17.2.1 Driver クラスの登録 1339
17.2.2 getConnection メソッドによる HiRDB への接続 1340
17.3 DataSource と JNDI を使用した DB 接続 1364
17.3.1 DataSource と JNDI を使用した DB 接続の手順 1364
17.4 JDBC1.2 コア API 1368
17.4.1 Driver インタフェース 1368
17.4.2 Connection インタフェース 1374
17.4.3 Statement インタフェース 1404
17.4.4 PreparedStatement インタフェース 1442
17.4.5 CallableStatement インタフェース 1476
17.4.6 ResultSet インタフェース 1547
17.4.7 DatabaseMetaData インタフェース 1629
17.4.8 ResultSetMetaData インタフェース 1768
17.4.9 Blob インタフェース 1784
17.4.10 Array インタフェース 1790
17.4.11 SQLException インタフェース 1796
17.4.12 SQLWarning インタフェース 1796
17.4.13 サポートしていないインタフェース 1798
17.5 JDBC2.1 コア API 1799
17.5.1 結果セットの拡張 1799
17.5.2 バッチ更新 1800
17.5.3 追加されたデータ型 1805
17.5.4 サポートしていないインタフェース 1806
17.6 JDBC2.0 Optional Package 1807

17.6.1 DataSource インタフェース 1807
17.6.2 ConnectionPoolDataSource インタフェース 1812
17.6.3 PooledConnection インタフェース 1816
17.6.4 XAConnection インタフェース 1820
17.6.5 XADataSource インタフェース 1821
17.6.6 XAResource インタフェース 1823
17.6.7 XAException インタフェース 1824
17.6.8 サポートしていないインタフェース 1824
17.7 JDBC3.0 API 1825
17.7.1 ParameterMetaData インタフェース 1825
17.7.2 サポートしていないインタフェース 1833
17.8 JDBC4.0 API 1834
17.8.1 JDBC4.0 API での追加機能 1834
17.8.2 Wrapper インタフェース 1838
17.8.3 SQLException 拡張機能 1840
17.8.4 サポートしていないインタフェース 1843
17.9 JDBC4.3API 1844
17.9.1 JDBC4.3 API での追加機能 1844
17.9.2 サポートしていないインタフェース 1845
17.10 システムプロパティの設定 1846
17.10.1 ステートメントに関する接続情報の設定 1846
17.10.2 互換維持に関する接続情報の設定 1847
17.10.3 DatabaseMetaData クラスのメソッドの動作に関する接続情報の設定 1848
17.11 接続情報設定／取得インタフェース 1850
17.11.1 setDescription 1852
17.11.2 getDescription 1854
17.11.3 setDBHostName 1855
17.11.4 getDBHostName 1855
17.11.5 setJDBC_IF_TRC 1856
17.11.6 getJDBC_IF_TRC 1857
17.11.7 setTRC_NO 1858
17.11.8 getTRC_NO 1859
17.11.9 setUapName 1859
17.11.10 getUapName 1860
17.11.11 setUser 1861
17.11.12 getUser 1862
17.11.13 setPassword 1863
17.11.14 getPassword 1864
17.11.15 setXAOpenString 1865

17.11.16 getXAOpenString 1866
17.11.17 setXACloseString 1867
17.11.18 getXACloseString 1867
17.11.19 setLONGVARBINARY_Access 1868
17.11.20 getLONGVARBINARY_Access 1869
17.11.21 setSQLInNum 1870
17.11.22 getSQLInNum 1871
17.11.23 setSQLOutNum 1871
17.11.24 getSQLOutNum 1872
17.11.25 setSQLWarningLevel 1873
17.11.26 getSQLWarningLevel 1874
17.11.27 setXALocalCommitMode 1875
17.11.28 getXALocalCommitMode 1876
17.11.29 setSQLWarningIgnore 1877
17.11.30 getSQLWarningIgnore 1877
17.11.31 setHiRDBCursorMode 1878
17.11.32 getHiRDBCursorMode 1879
17.11.33 setNotErrorOccurred 1880
17.11.34 getNotErrorOccurred 1881
17.11.35 setEnvironmentVariables 1882
17.11.36 getEnvironmentVariables 1883
17.11.37 setEncodeLang 1884
17.11.38 getEncodeLang 1885
17.11.39 setMaxBinarySize 1886
17.11.40 getMaxBinarySize 1887
17.11.41 setStatementCommitBehavior 1888
17.11.42 getStatementCommitBehavior 1889
17.11.43 setLONGVARBINARY_AccessSize 1890
17.11.44 getLONGVARBINARY_AccessSize 1891
17.11.45 setLONGVARBINARY_TruncError 1892
17.11.46 getLONGVARBINARY_TruncError 1893
17.11.47 setStatementCloseBehavior 1893
17.11.48 getStatementCloseBehavior 1894
17.11.49 setHiRDBINI 1895
17.11.50 getHiRDBINI 1896
17.11.51 setBatchExceptionBehavior 1897
17.11.52 getBatchExceptionBehavior 1897
17.11.53 setUpdateCountBehavior 1898
17.11.54 getUpdateCountBehavior 1899

17.12 データ型 1901
17.12.1 SQL データ型のマッピング 1901
17.12.2 検索データ取得時のマッピング 1902
17.12.3 ？パラメタ設定時のマッピング 1905
17.12.4 TIME 型，DATE 型，及び TIMESTAMP 型列のデータ変換処理 1909
17.12.5 オーバフローの扱い 1911
17.13 文字コード変換機能 1920
17.14 指定できるクライアント環境定義 1921
17.15 接続情報の優先順位 1934
17.15.1 接続情報の優先順位一覧 1934
17.15.2 そのほかのクライアント環境定義の優先順位 1939
17.16 Type2 JDBC ドライバからの移行 1941
17.17 DABroker for Java からの移行 1944
17.17.1 DABroker for Java 互換機能に関するシステムプロパティ 1944
17.17.2 Type4 JDBC ドライバと互換性のない項目 1946
17.18 JDBC インタフェースメソッドトレース 1948
17.18.1 取得するための設定 1948
17.18.2 取得規則 1948
17.18.3 出力例 1949
17.19 Exception トレースログ 1951
17.19.1 取得するメソッド，及び取得するための設定 1951
17.19.2 出力形式 1961
17.19.3 出力例と解析方法 1967
17.19.4 必要となるメモリ所要量及びファイルサイズ 1971
17.19.5 注意事項 1973
17.20 不正電文トレース 1975
17.20.1 不正電文トレースを取得するための設定 1975
17.20.2 ファイル名 1976
17.20.3 必要となるファイルサイズ 1976
17.21 JDBC ドライバを使用した UAP 例 1977
17.22 JDBC ドライバを使用した場合のメモリの容量見積もり 1979
17.22.1 Connection オブジェクトの容量見積もり 1979
17.22.2 Statement オブジェクトの容量見積もり 1981
17.22.3 PreparedStatement オブジェクトの容量見積もり 1982
17.22.4 CallableStatement オブジェクトの容量見積もり 1982
17.22.5 ResultSet オブジェクトの容量見積もり 1983
17.22.6 トレースオブジェクトの容量見積もり 1985

18 SQLJ 1989

18.1 概要 1990
18.1.1 SQLJ とは 1990
18.1.2 環境設定 1991
18.2 SQLJ トランスレータ 1993
18.3 UAP の記述規則 1994
18.3.1 名標の付け方の規則 1994
18.3.2 SQL の記述規則 1994
18.3.3 SQLJ で使用できる SQL 文 1998
18.3.4 HiRDB のデータ型と SQLJ のデータ型の対応 2001
18.3.5 出力変数の設定（ネイティブインタフェース版限定） 2002
18.3.6 カーソル宣言時のデータ型の使用（ネイティブインタフェース版限定） 2004
18.3.7 HiRDB サーバとの接続，切り離しの記述 2004
18.3.8 カーソルによる検索の記述 2009
18.3.9 動的結果セットの受け取り 2013
18.3.10 JDBC との相互運用 2014
18.3.11 UAP の作成と実行 2017
18.3.12 スタンダードインタフェース版からネイティブインタフェース版への移行 2020
18.3.13 UAP 開発時の注意事項 2022
18.4 ネイティブランタイム 2023
18.4.1 パッケージの構成 2023
18.4.2 ネイティブランタイムの公開クラス一覧 2023
18.4.3 クラス仕様 2024
18.4.4 ネイティブインタフェースを使用したコーディング例 2029

付録 2034

付録 A SQL 連絡領域 2035
付録 A.1 SQL 連絡領域の構成と内容 2035
付録 A.2 SQL 連絡領域の展開 2039
付録 B SQL 記述領域 2042
付録 B.1 SQL 記述領域の構成と内容 2042
付録 B.2 SQL 記述領域の展開 2050
付録 C 列名記述領域 2062
付録 C.1 列名記述領域の構成と内容 2062
付録 C.2 列名記述領域の展開 2064
付録 D 型名記述領域 2066
付録 D.1 型名記述領域の構成 2066
付録 D.2 型名記述領域の内容 2066
付録 D.3 型名記述領域の展開 2067

付録 E　文字集合名記述領域　2069
付録 E.1　文字集合名記述領域の構成と内容　2069
付録 E.2　文字集合名記述領域の展開　2075
付録 F　SQL のデータ型とデータ記述　2079
付録 F.1　SQL のデータ型と C 言語のデータ記述　2079
付録 F.2　SQL のデータ型と COBOL 言語のデータ記述　2097
付録 G　データディクショナリ表の検索　2116
付録 G.1　GUI 版 HiRDB SQL Executer によるデータディクショナリ表の参照　2116
付録 G.2　操作系 SQL によるデータディクショナリ表の参照　2121
付録 G.3　ディクショナリ表の詳細　2126
付録 H　HiRDB が提供する関数　2205
付録 H.1　表分割ハッシュ関数　2205
付録 H.2　空白変換関数　2228
付録 H.3　DECIMAL 型符号正規化関数　2233
付録 H.4　文字コード種別設定関数　2235
付録 I　エスケープ句で指定できるスカラ関数　2238
付録 J　文字集合を使用した場合の文字コード変換規則　2243
付録 J.1　シフト JIS 漢字コードを EBCDIK に変換する場合　2243
付録 J.2　EBCDIK をシフト JIS 漢字コードに変換する場合　2244
付録 K　HiRDB SQL Tuning Advisor の環境設定　2246
付録 L　HiRDB の最大値・最小値　2250
付録 M　UAP のサンプル一覧　2252
付録 N　メインフレームからのマイグレーション　2254
付録 N.1　次の行取り出しで排他を解除する機能　2254

索引　2257

1 概要

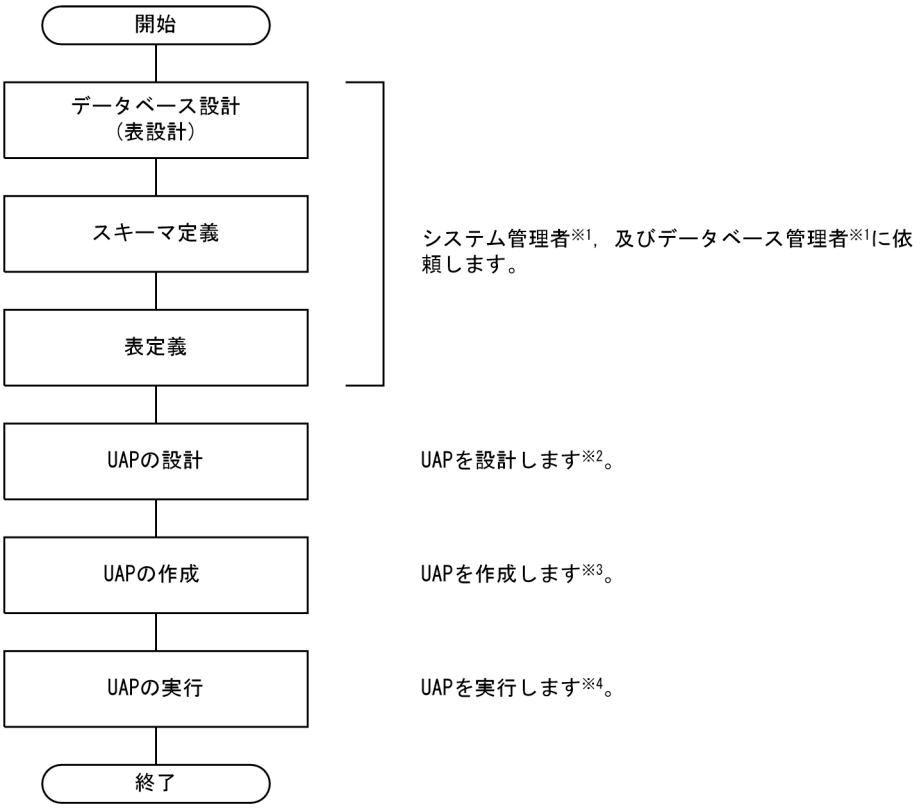
この章では，UAP を開発するときの作業の流れ，UAP の特長，及び HiRDB システムとのインタフェースについて説明します。

1.1　UAP の開発の流れ

UAP を開発する準備として，業務などで使用しているデータをデータベース化するために業務内容を分析します。分析した結果を基に全体的なデータベースの規模を検討し，UAP の概略を決定します。

UAP の開発作業とこのマニュアルの構成との関係を次の図に示します。

図 1–1　UAP の開発作業とこのマニュアルの構成との関係



注※1

作業の内容については，マニュアル「HiRDB システム導入・設計ガイド」を参照してください。

注※2

詳細は，「データベースの操作」～「オブジェクトリレーショナルデータベースをアクセスする UAP 作成時の注意事項」を参照してください。

注※3

詳細は，「クライアントの環境設定」及び「UAP の作成」を参照してください。

注※4

詳細は，「クライアントの環境設定」，「UAP 実行前の準備」，及び「C ストアドプロシジャ，C ストアドファンクション」を参照してください。

1.2　UAP の特長

1.2.1　UAP の形式

HiRDB のデータベースを操作する UAP の形式は，埋込み型です。

埋込み型とは，高級言語で記述されたソースプログラムの中に SQL と呼ばれるデータベース言語を直接記述する方式です。埋込み型の UAP は，データベース操作（SQL）を含めて一つのプログラムとして記述できるため，プログラムの解析が容易になります。

また，UAP 中には ODBC 関数も指定でき，Java（SQLJ）での UAP の作成もできます。

(1)　ソースプログラムの記述

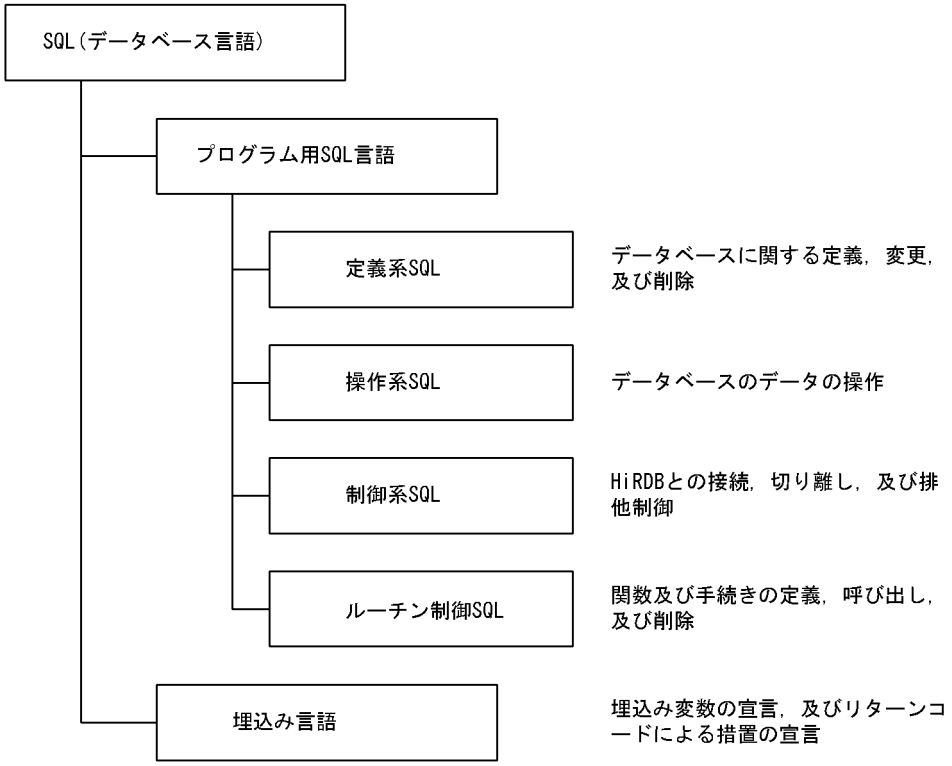
埋込み型 UAP に使用できる高級言語を次に示します。

- C 言語
- C++言語
- COBOL 言語
- OOCOBOL 言語

(2)　SQL の記述

SQL は，データベースの定義，操作，運用，及び制御の指示を記述するためのデータベース言語です。高級言語で記述したソースプログラムの中に埋め込んで使用できます。SQL の機能体系を次の図に示します。

図 1–2　SQL の機能体系



なお，プログラム用 SQL 言語の種類と機能概略については，「HiRDB で使用できる SQL 一覧」を参照してください。また，埋込み言語の詳細については，マニュアル「HiRDB SQL リファレンス」を参照してください。

1.2.2　HiRDB で使用できる SQL 一覧

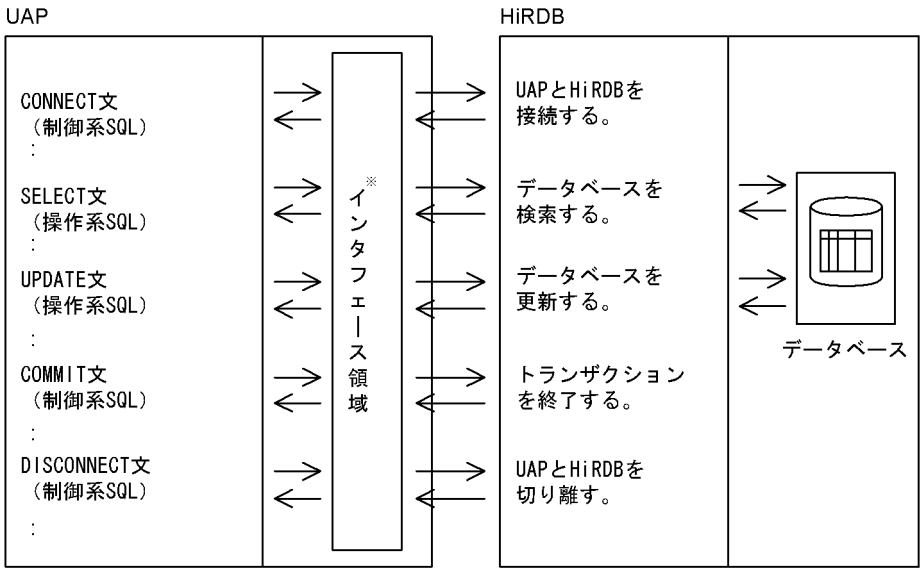
HiRDB で使用できる SQL については，マニュアル「HiRDB SQL リファレンス」の「SQL 一覧」を参照してください。

1.3　HiRDB とのインタフェース

HiRDB のデータベースは，UAP を作成して操作します。UAP は SQL 文を発行することで，インタフェース領域を使用して HiRDB との情報のやり取りをします。

UAP と HiRDB とのインタフェースを次の図に示します。

図 1–3　UAP と HiRDB とのインタフェース



注※

インタフェース領域については，「インタフェース領域」を参照してください。

1.4　UAP の動作環境

HiRDB は，サーバ–クライアント型のネットワーク環境で動作します。UAP を発行する要求元をクライアントといい，その要求を受けるシステムをサーバといいます。また，サーバとなるシステムを HiRDB サーバといいます。

クライアントの運用形態には，次に示す 7 つがあります。なお，これらの運用形態はそれぞれ混在して動作させることもできます。

- サーバマシンとは別のマシンをクライアントとする運用形態
- HiRDB サーバと同一のサーバマシンでクライアントを実行する運用形態
- OLTP 下の UAP をクライアントとする運用形態
- ODBC※1対応のアプリケーションプログラムをクライアントとする運用形態
- OLE DB※2対応のアプリケーションプログラムをクライアントとする運用形態
- ADO.NET 対応のアプリケーションプログラムをクライアントとする運用形態
- Java（JDBC 対応）のアプリケーションプログラムをクライアントとする運用形態

注※1

ODBC は，米国 Microsoft Corp.が提唱するデータベースアクセス機構です。ODBC 対応のアプリケーションプログラムから HiRDB をアクセスする方法については，「ODBC 対応アプリケーションプログラムからの HiRDB アクセス」を参照してください。

注※2

OLE DB は，ODBC と同様に広範囲なデータソースにアクセスするための API です。さらに，ODBC とは異なり，SQL データ以外のデータアクセスに適したインタフェースも定義されています。OLE DB 対応のアプリケーションプログラムから HiRDB をアクセスする方法については，「OLE DB 対応アプリケーションプログラムからの HiRDB アクセス」を参照してください。

クライアントの運用形態を次の図に示します。

- サーバマシンとは別のマシンをクライアントとする運用形態
- HiRDB サーバと同一のサーバマシンでクライアントを実行する運用形態
- OLTP 下の UAP をクライアントとする運用形態
- ODBC 対応のアプリケーションプログラムをクライアントとする運用形態
- OLE DB 対応のアプリケーションプログラムをクライアントとする運用形態
- ADO.NET 対応のアプリケーションプログラムをクライアントとする運用形態
- Java（JDBC 対応）のアプリケーションプログラムをクライアントとする運用形態

なお，UAP を開発する HiRDB/Developer's Kit と，UAP を実行する HiRDB/Developer's Kit のプラットフォームは，同じにしてください。

図 1–4　サーバマシンとは別のマシンをクライアントとする運用形態

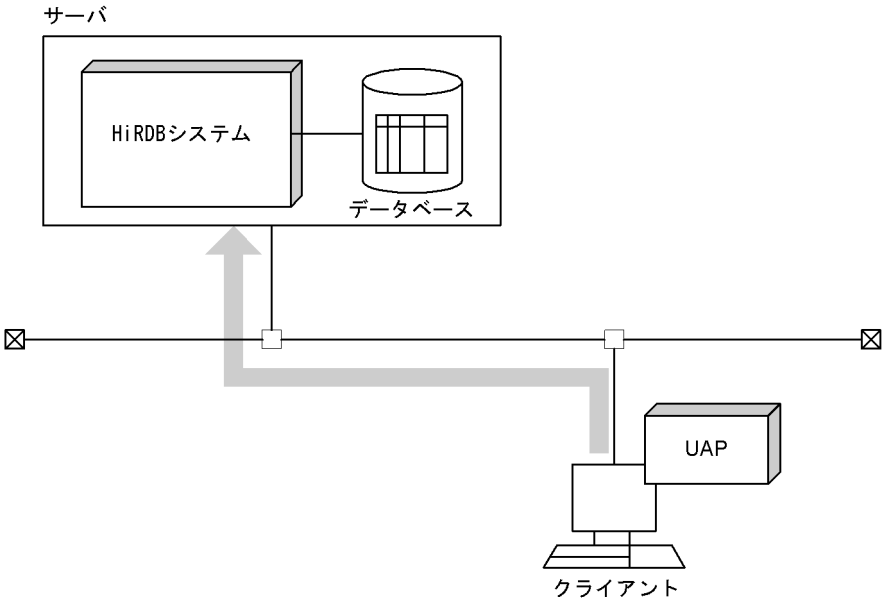


図 1–5　HiRDB サーバと同一のサーバマシンでクライアントを実行する運用形態

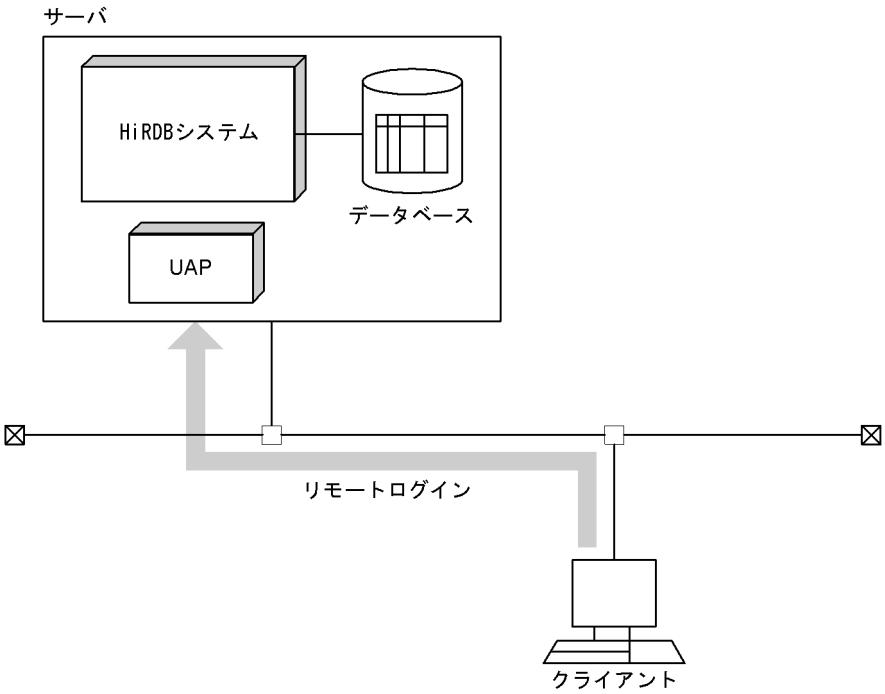


図 1–6　OLTP 下の UAP をクライアントとする運用形態

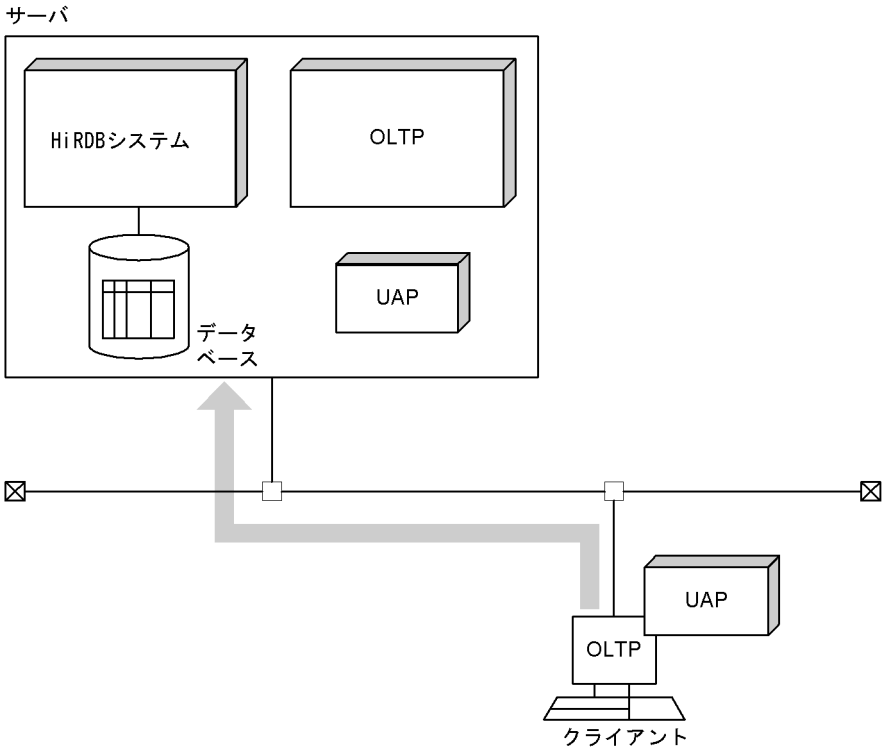
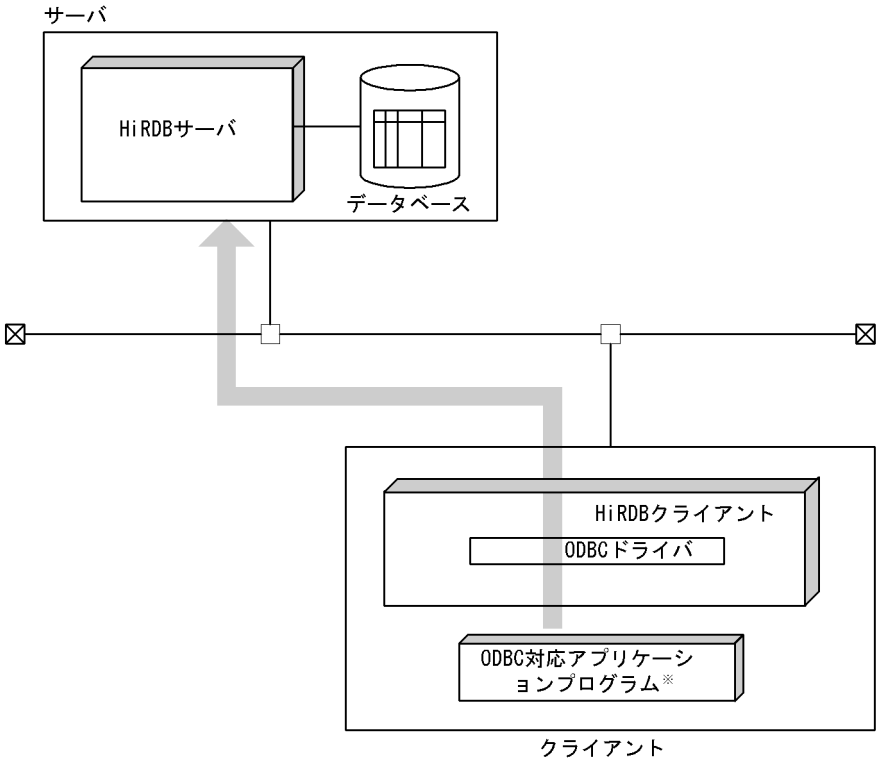
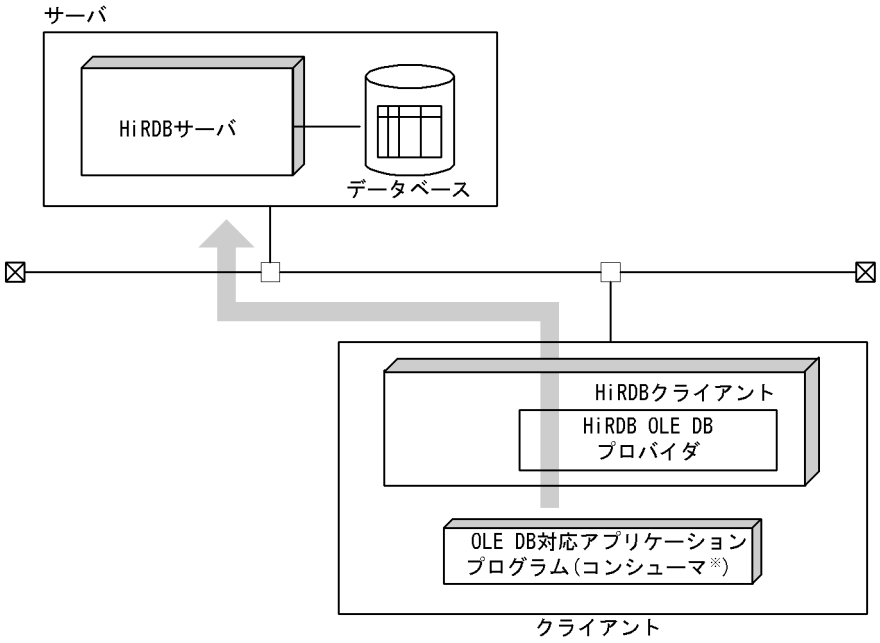


図 1–7　ODBC 対応のアプリケーションプログラムをクライアントとする運用形態



注※　ODBC対応のUAPも含まれます。

図 1–8　OLE DB 対応のアプリケーションプログラムをクライアントとする運用形態



注※　コンシューマとは，OLE DBのメソッドとインタフェースを呼び出すプログラムのことをいいます。

図 1–9　ADO.NET 対応のアプリケーションプログラムをクライアントとする運用形態

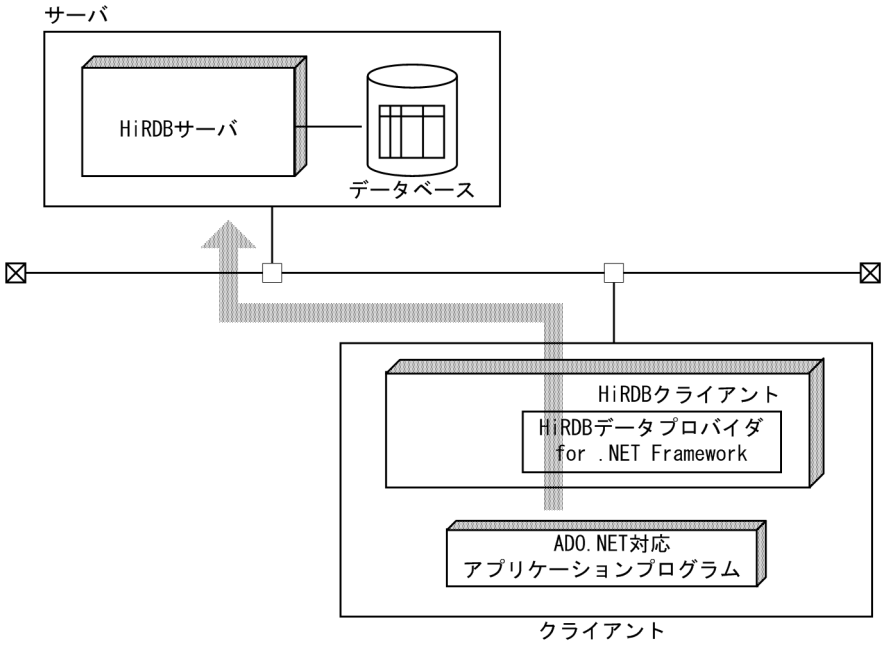
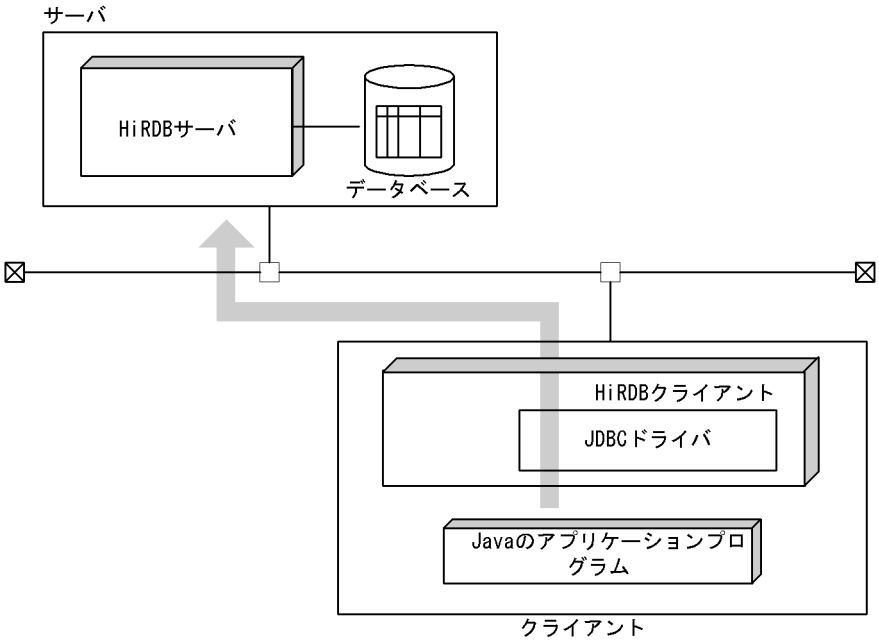


図 1–10　Java（JDBC 対応）のアプリケーションプログラムをクライアントとする運用形態



1.5 HiRDB サーバ，HiRDB/Run Time および HiRDB/Developer's Kit の機能差

HiRDB サーバには一部のクライアント機能が含まれています。

HiRDB/Run Time 及び HiRDB/Developer's Kit に同梱されるクライアント機能と HiRDB サーバとの差について次の表に示します。

表 1–1　UNIX 系 HiRDB 製品でのクライアント機能同梱状況

クライアント機能	HiRDB サーバ	HiRDB/Run Time	HiRDB/Developer's Kit
クライアントライブラリ	○	○	○
SQL プリプロセサ	○	×	○
ODBC ドライバ※	○	○	○
JDBC ドライバ	○	○	○
SQLJ	○	○	○
XML 変換ライブラリ	×	○	○
コマンド・ユティリティ	○	○	○
サンプル UAP	○	×	○

(凡例)

○：同梱している。

×：同梱していない。

注※

Linux 版だけでサポートしています。

表 1–2　Windows 系 HiRDB 製品でのクライアント機能同梱状況

クライアント機能	HiRDB サーバ	HiRDB/Run Time	HiRDB/Developer's Kit
クライアントライブラリ	○	○	○
SQL プリプロセサ	○	×	○
ODBC ドライバ	×	○	○
HiRDB OLE DB プロバイダ	×	○	○
HiRDB データプロバイダ for .NET Framework	×	○	○
JDBC ドライバ	○	○	○
SQLJ	○	○	○
XML 変換ライブラリ	×	○	○

クライアント機能	HiRDB サーバ	HiRDB/Run Time	HiRDB/Developer's Kit
コマンド・ユティリティ	○	○	○
サンプル UAP	○	×	○

(凡例)

○：同梱している。

×：同梱していない。

2 データベースの操作

この章では，データベースのデータ表現，及びデータベースの基本的な操作例について説明します。

なお，説明の中で使用している SQL は，文法に従って作成した中から説明に必要な部分だけを抜粋したものです。SQL の詳細については，マニュアル「HiRDB SQL リファレンス」を参照してください。

2.1　データベースのデータ表現

2.1.1　リレーショナルデータベースの表

HiRDB のデータベースはリレーショナルデータベースで，論理的な構造は表として表現されます。ここでは，表について説明します。

(1)　表の基本構成

リレーショナルデータベースは，論理的には表形式をしています。

表の縦方向を列と呼び，横方向を行と呼びます。各行の同じ列には，同一属性（データ型）のデータを格納します。表は行の集合で，一つの行は一つの検索単位になります。また，各列には名称（列名）が付いていて，データベースを操作するときに使用します。

表の基本構成例を次の図に示します。

図 2–1　表の基本構成例

ZAIKO（在庫表）

CHAR(4)	NCHAR(10)	NCHAR(5)	DECIMAL	INTEGER
商品コード (SCODE)	商品名 (SNAME)	色 (COL)	単価 (TANKA)	在庫量 (ZSURYO)
101L	ブラウス	青	3500	62
101M	ブラウス	白	3500	85
201M	ポロシャツ	白	3640	29
202M	ポロシャツ	赤	3640	67
302S	スカート	白	5110	65
353L	スカート	赤	4760	18
353M	スカート	緑	4760	56
411M	セーター	青	8400	12
412M	セーター	赤	8400	22
591L	ソックス	赤	250	300
591M	ソックス	青	250	90
591S	ソックス	白	250	280

データ型
列名
行
列

(2)　繰返し列を使用した表

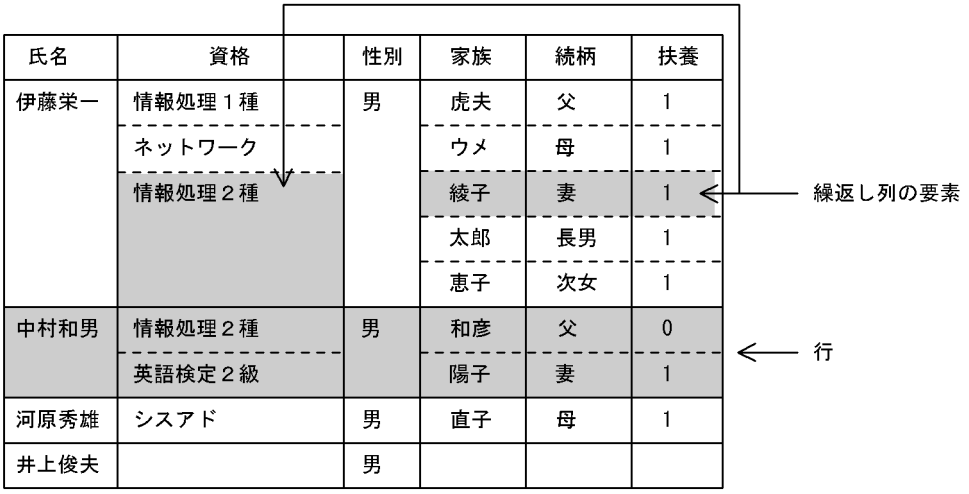
繰返し列とは，複数の要素から構成される列のことをいいます。繰返し列を使用すると，次のようなメリットがあります。

- 複数の表の結合が不要になります。
- 重複する情報が少なくなるため，ディスク容量を削減できます。
- 関連データ（繰返しデータ）が近くに格納されるため，別の表にするよりアクセス性能が良いです。

繰返し列がある場合の表の構成例を次の図に示します。

図 2–2　繰返し列がある場合の表の構成例

社員表



注　空白の箇所は，ナル値を表します。

(3)　ビュー表

実際の表（以降，実表と呼びます）を基に，利用者が操作する範囲を限定した仮想の表を設定できます。この仮想の表のことをビュー表といいます。次に示す場合にビュー表を定義すると操作する範囲が限定され，操作が簡単になります。

- 表の特定の列だけを検索する。
- 表の列の順序を入れ替える。
- 表の特定の行だけを検索する。

ビュー表は，表の特定の列や行を見るために定義しますが，実表と同様の検索ができます。また，ビュー表を使用することで，操作する範囲が限定されるので細かな機密保護ができます。

実表に対して定義したビュー表の例を次の図に示します。

なお，ビュー表の定義，及び操作の方法については，「ビュー表の定義と操作」を参照してください。

図 2–3　実表に対して定義したビュー表の例

実表

JUTYU（受注表）

DNO	TCODE	SCODE	JSURYO	JDATE	JTIME
伝票番号	得意先コード	商品コード	受注量	受付日付	受付時刻
026551	TT002	101M	10	1995-06-14	09:23:11
026552	TT002	591M	25	1995-06-14	09:23:11
026553	TH001	353M	8	1995-06-14	10:10:55
026554	TK001	411M	6	1995-06-14	10:15:47
026555	TA001	591M	30	1995-06-14	10:15:47
026556	TT002	202M	10	1995-06-14	11:48:09
026557	TZ001	411M	5	1995-06-14	13:02:00
026558	TZ001	412M	4	1995-06-14	13:02:00
026559	TH001	591M	80	1995-06-14	14:04:16
026560	TT001	591L	10	1995-06-14	15:31:20

ビュー表

DNO	SCODE	JSURYO
伝票番号	商品コード	受注量
026551	101M	10
026552	591M	25
026553	353M	8
026554	411M	6
026555	591M	30
026556	202M	10
026557	411M	5
026558	412M	4
026559	591M	80
026560	591L	10

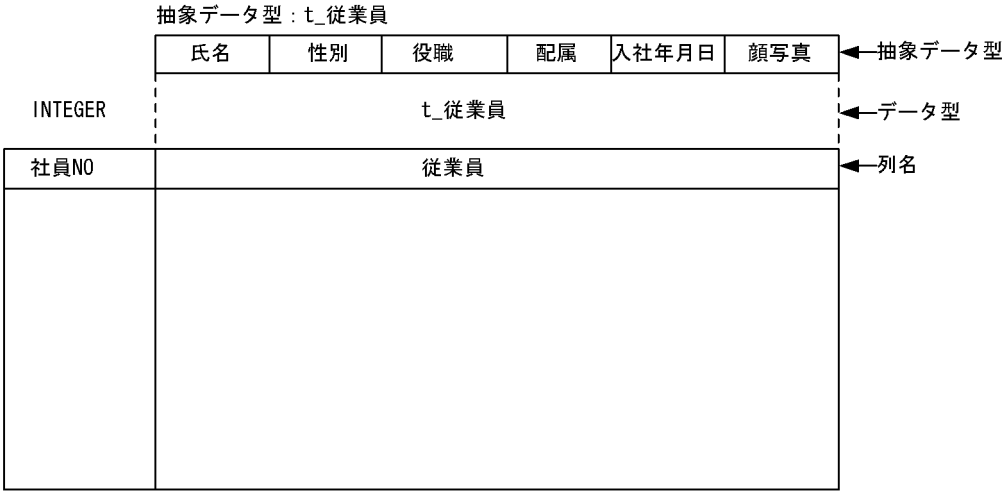
2.1.2　オブジェクトリレーショナルデータベースの表

HiRDB のデータベースは，オブジェクトリレーショナルデータベースとして定義することもできます。オブジェクトリレーショナルデータベースの表は，表の列に抽象データ型を定義することで作成できます。

抽象データ型がある表の基本構成例を次の図に示します。

図 2–4　抽象データ型がある表の基本構成例

表：社員表



2.2　カーソルの利用

2.2.1　カーソルを使用した検索及び使用しない検索

表の検索結果は一般には複数行にわたります。UAP で複数行の検索結果を 1 行ずつ取り出すために最新の取り出し位置を保持するのがカーソルです。

ここでは，カーソルを使用した検索と，カーソルを使用した検索行の更新について説明します。

なお，カーソルを使用するときに考慮する内容については，「カーソルの効果」を参照してください。

(1)　カーソルを使用した検索

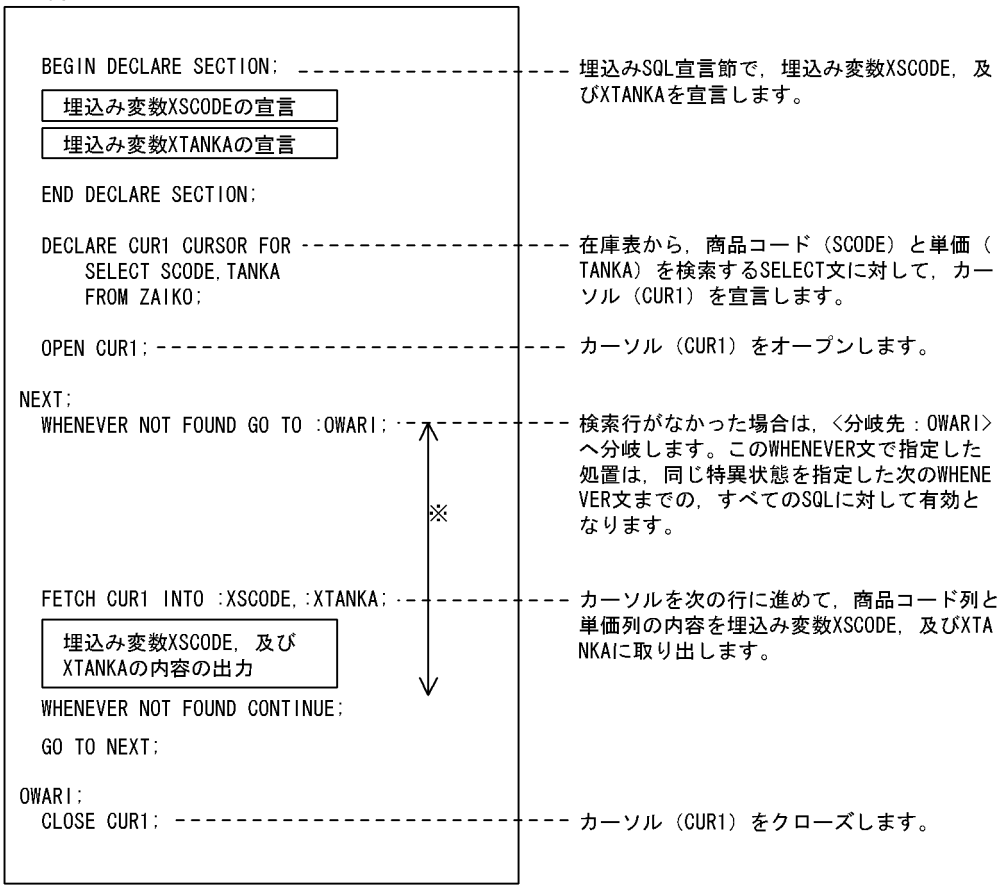
表の検索結果が 2 行以上になる場合や，SQL の文字列を PREPARE 文で前処理して動的に検索する場合，カーソルを使用して検索します。

検索結果が 1 行以下の場合，カーソルを使用しないで 1 行 SELECT 文で検索することもできます。

なお，PREPARE 文，及び 1 行 SELECT 文については，マニュアル「HiRDB SQL リファレンス」を参照してください。

カーソルを使用して複数行を検索するときの例として，在庫表から各商品の品番と単価を検索する UAP を次に示します。

<UAPの例>



注　SQL先頭子，及びSQL終了子は省略しています。
注※　WHENEVER NOT FOUND GO TO :OWARI;の有効範囲です。

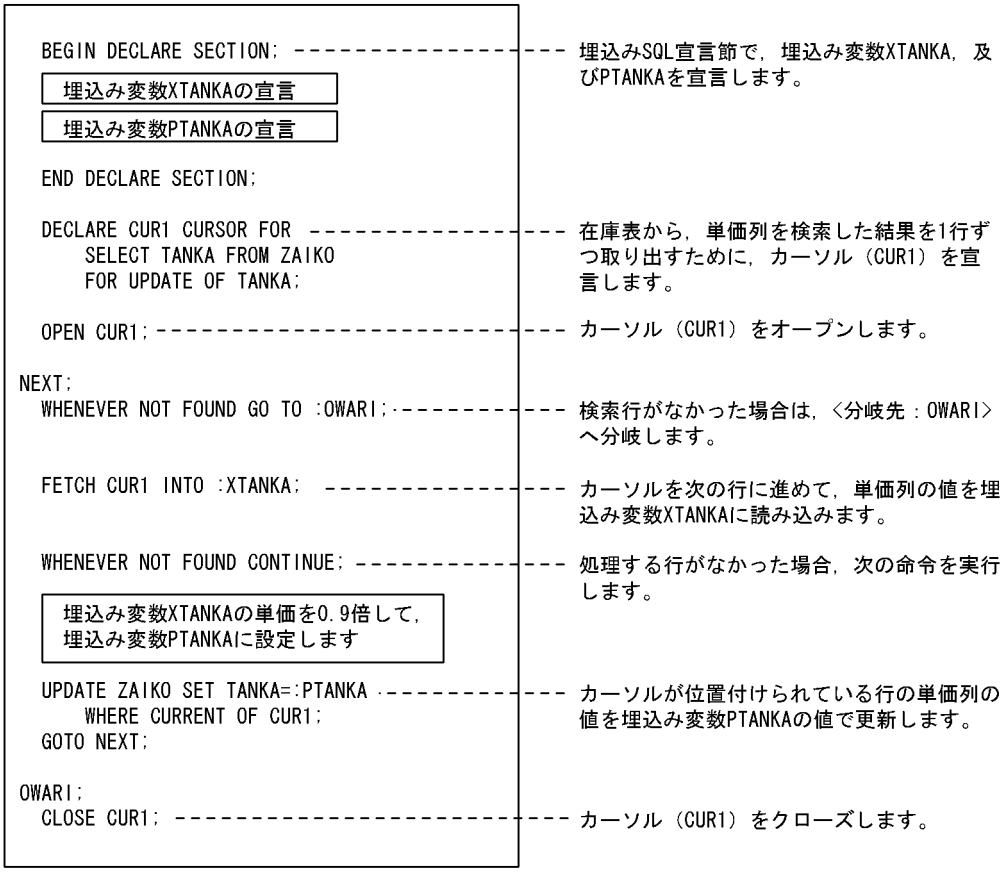
(2)　カーソルを使用した検索行の更新

複数の検索行を更新する場合，カーソルを使用して 1 行ずつ検索しながら更新します。

なお，検索結果が 1 行以下の場合，1 行 SELECT 文で検索して更新できますが，カーソルを使用する方が処理効率は良くなります。

カーソルを使用して検索行を更新するときの例として，在庫表の各商品の単価を 1 割下げる（0.9 倍する）UAP を次に示します。

〈UAPの例〉

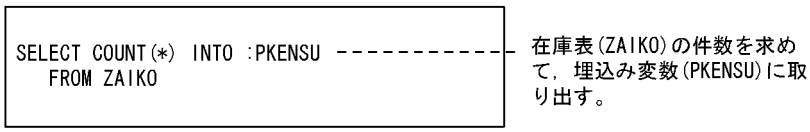


注　SQL先頭子，及びSQL終了子は省略しています。

(3)　カーソルを使用しない検索（1 行検索）

カーソルを使用しないで検索するときの例として，在庫表の件数を求めて埲込み変数に取り出す UAP を次に示します。

＜UAP例＞



2.3　データの検索

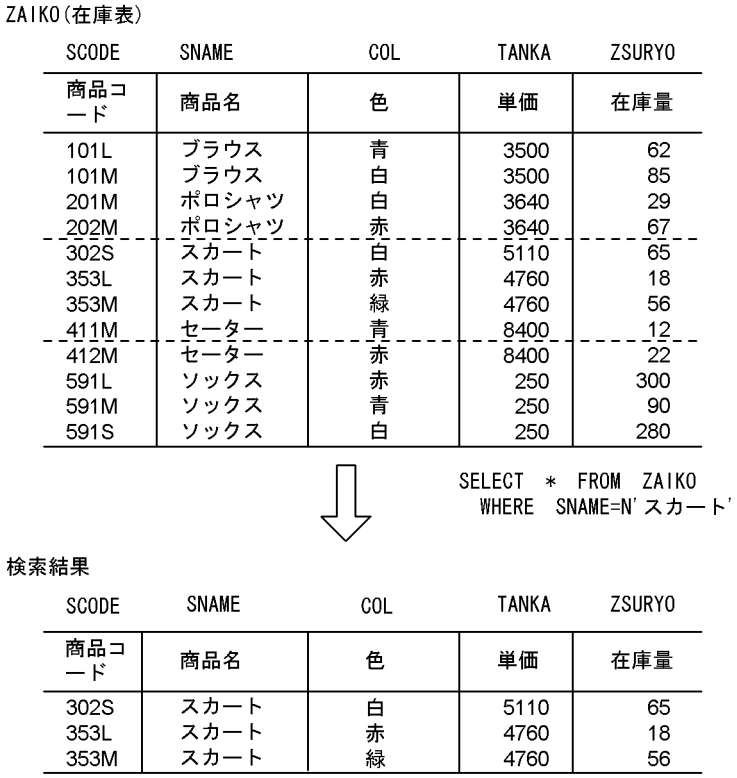
表を構成する行の中から，ある列に対して指定した条件を満たす行を選び出すことを検索といいます。2個以上の表を特定の列の値でつなぎ合わせて検索し，1組の検索結果を取り出すこともできます。

ここでは，表の検索について説明します。

2.3.1　1個の表からの検索

1個の表からの検索例として，在庫表の中から商品名が"スカート"の行だけをSELECT文で検索するときの例を次の図に示します。

図 2–5　1個の表からの検索

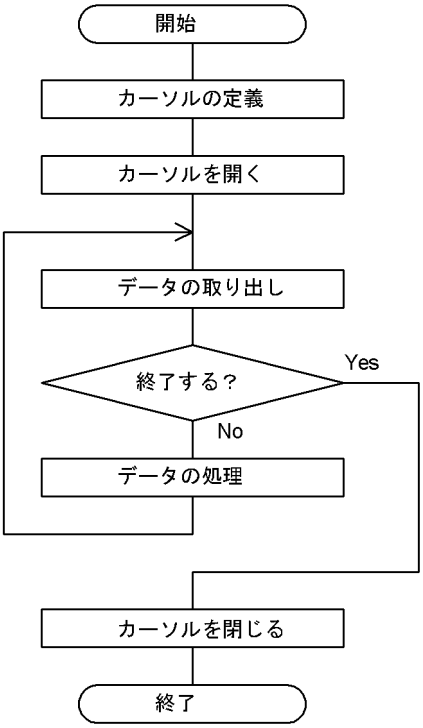


表の検索結果は表形式になり，処理要求したUAPに渡されます。

検索結果の表をUAPから参照するためには，カーソルを使用します。カーソルには，検索結果の表の特定の行を指す機能があるので，UAPはカーソルの指している行の内容を読み出して加工できます。

検索結果の表に対するUAPからのデータの処理手順を次の図に示します。

図 2–6　検索結果の表に対する UAP からのデータの処理手順



次に図「検索結果の表に対する UAP からのデータの処理手順」で示した処理手順の各ステップを説明します。

1. カーソルの定義

   カーソルを使用するために，カーソルの名称，そのカーソルを使用して検索する表の名称，及び探索条件を定義します。例えば，図「1 個の表からの検索」に示す例に当てはめて考えた場合，カーソル名称を「CUR1」として，在庫表から"スカート"だけを探索するときは，次のように定義します。

```
DECLARE CUR1 CURSOR FOR
SELECT SNAME,COL,TANKA FROM ZAIKO
  WHERE SNAME=N'スカート'
```

2. カーソルを開く

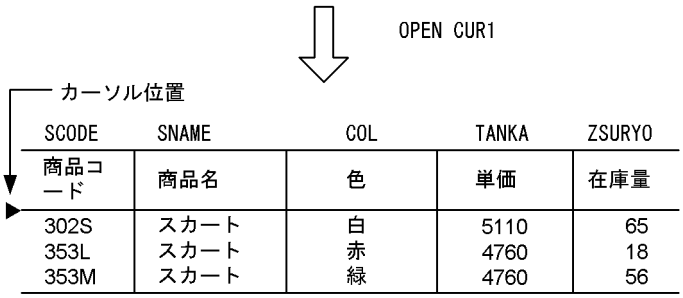
   カーソルを開くと，定義した条件に従って検索結果を取り出せる状態になります。検索結果は，システム内に表形式で格納され，カーソルを閉じるまで有効です。

   カーソルを開くには，次のように記述します。

```
OPEN CUR1
```

   カーソルを開いた直後は，表の第 1 列の第 1 行目の一つ上にカーソルがあります。カーソル名称が「CUR1」で，在庫表から"スカート"を検索する条件にしたとき，カーソルを開いた直後の状態を次の図に示します。

図 2–7　カーソルを開いた直後の状態

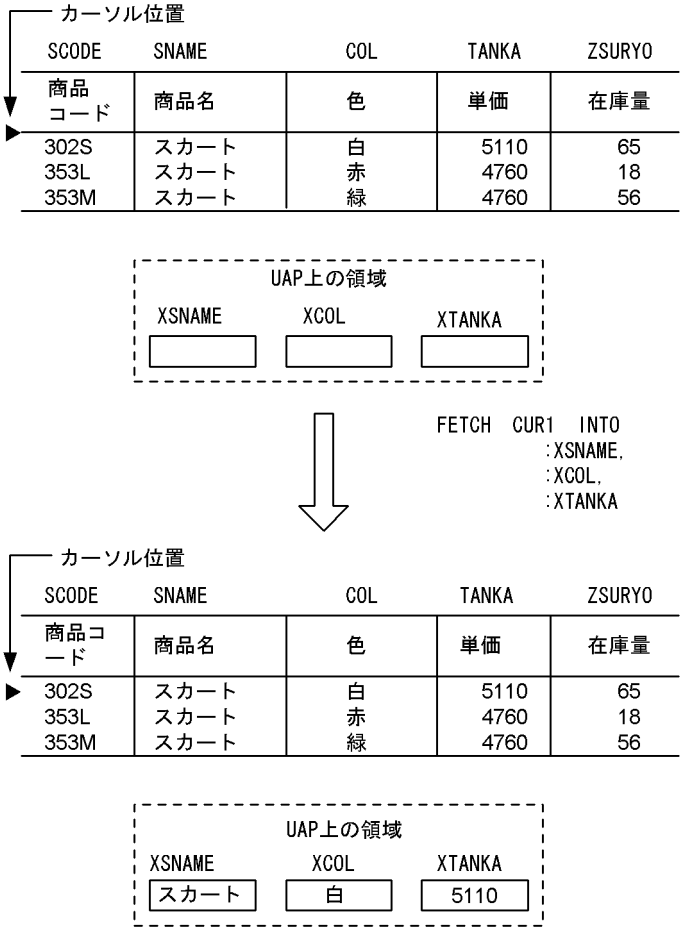


3. データの取り出し

FETCH 文を使用し，カーソルを開いた状態の位置から 1 行進めて，その行の内容を UAP 内の指定した領域に格納します。

カーソルを開いた直後の状態から，検索した内容が UAP の領域に格納される例を次の図に示します。

図 2–8　検索した内容を取り出して UAP の領域に格納する例



4. データの処理

UAP 上の領域に格納されたデータの内容を必要に応じて出力します。

5. カーソルを閉じる

検索結果に対する UAP のデータ処理が完了したらカーソルを閉じます。

カーソルを閉じると，システム内に保存されていた検索結果の表は消去されます。カーソルを閉じるには，次のように記述します。

```
CLOSE CUR1
```

2.3.2　複数の表からの検索

2 個以上の表から検索するには，SELECT 文の FROM 句を使用します。複数の表から一つの結果を得る例として，在庫量が 60 未満でかつ，受注量が 30 未満の商品の伝票番号と商品名の表を作成するときの例を次の図に示します。

図 2-9　2 個の表からの検索例

ZAIKO（在庫表）

SCODE	SNAME	COL	TANKA	ZSURYO
商品コード	商品名	色	単価	在庫量
101L	ブラウス	青	3500	62
101M	ブラウス	白	3500	85
201M	ポロシャツ	白	3640	29
202M	ポロシャツ	赤	3640	67
302S	スカート	白	5110	65
353L	スカート	赤	4760	18
353M	スカート	緑	4760	56
411M	セーター	青	8400	12
412M	セーター	赤	8400	22
591L	ソックス	赤	250	300
591M	ソックス	青	250	90
591S	ソックス	白	250	280

JUTYU（受注表）

DNO	TCODE	SCODE	JSURYO	･･･
伝票番号	得意先コード	商品コード	受注量	･･･
026551	TT002	101M	10	･･･
026552	TT002	591M	25	
026553	TH001	353M	8	
026554	TK001	411M	6	
026555	TA001	591M	30	･･･
026556	TT002	202M	10	
026557	TZ001	411M	5	
026558	TZ001	412M	4	
026559	TH001	591M	80	･･･
026560	TT001	591L	10	

```
SELECT DNO,SNAME FROM ZAIKO,JUTYU
   WHERE ZAIKO.SCODE=JUTYU.SCODE
   AND ZAIKO.ZSURYO<60
   AND JUTYU.JSURYO<30
```

検索結果

DNO	SNAME
伝票番号	商品名
026553	スカート
026554	セーター
026557	セーター
026558	セーター

2.3.3　FIX 属性の表の検索

FIX 属性の表を検索する場合，1 行全体を一つの固定長レコードとして検索できます。行単位の検索は，SELECT 文の選択句に ROW を指定します。

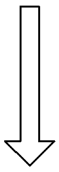
1 行単位で検索すると，列ごとに検索するオーバヘッドがなくなるため，アクセス性能が向上します。

行単位の検索例として，カーソル（CUR1）を使用し，在庫表から商品名が"ポロシャツ"だけを検索するとき，埋込み変数（：XROW）に取り出す例を次の図に示します。

図 2-10　行単位の検索例

ZAIKO（在庫表）

SCODE	SNAME	COL	TANKA	ZSURYO
商品コード	商品名	色	単価	在庫量
101L	ブラウス	青	3500	62
101M	ブラウス	白	3500	85
201M	ポロシャツ	白	3640	29
202M	ポロシャツ	赤	3640	67
302S	スカート	白	5110	65
353L	スカート	赤	4760	18
353M	スカート	緑	4760	56
411M	セーター	青	8400	12
412M	セーター	赤	8400	22
591L	ソックス	赤	250	300
591M	ソックス	青	250	90
591S	ソックス	白	250	280



```
DECLARE CUR1 CURSOR FOR SELECT
  ROW FROM ZAIKO
  WHERE SNAME='ポロシャツ'

OPEN CUR1
```

SCODE	SNAME	COL	TANKA	ZSURYO
商品コード	商品名	色	単価	在庫量
201M	ポロシャツ	白	3640	29
202M	ポロシャツ	青	3640	67

:XROW



```
FETCH CUR1 INTO :XROW
```

:XROW

201M	ポロシャツ	白	3640	29

2.4　データの更新

表中の情報の更新には，次に示す三つの方法があります。

- カーソルが指している行を更新する。
- 条件を満たす行だけ更新する。
- 行単位で更新する。

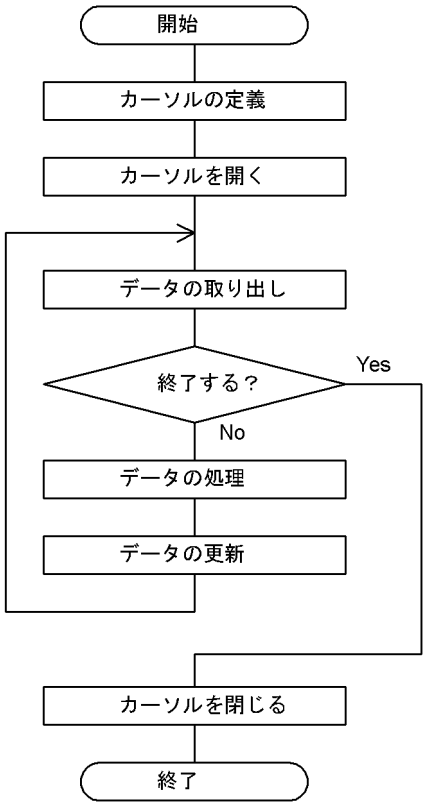
なお，次に示す列の値は更新できません。

- 表がフレキシブルハッシュ分割以外で横分割されている場合，分割するキーとなる列
- 改竄防止表の更新可能列以外の列
- FIX 表に列を追加するための予備列

2.4.1　カーソルを使用した更新

複数の検索行を更新する場合，カーソルを使用して 1 行ずつ検索しながら更新します。カーソルを使用した表の更新の処理手順を次の図に示します。

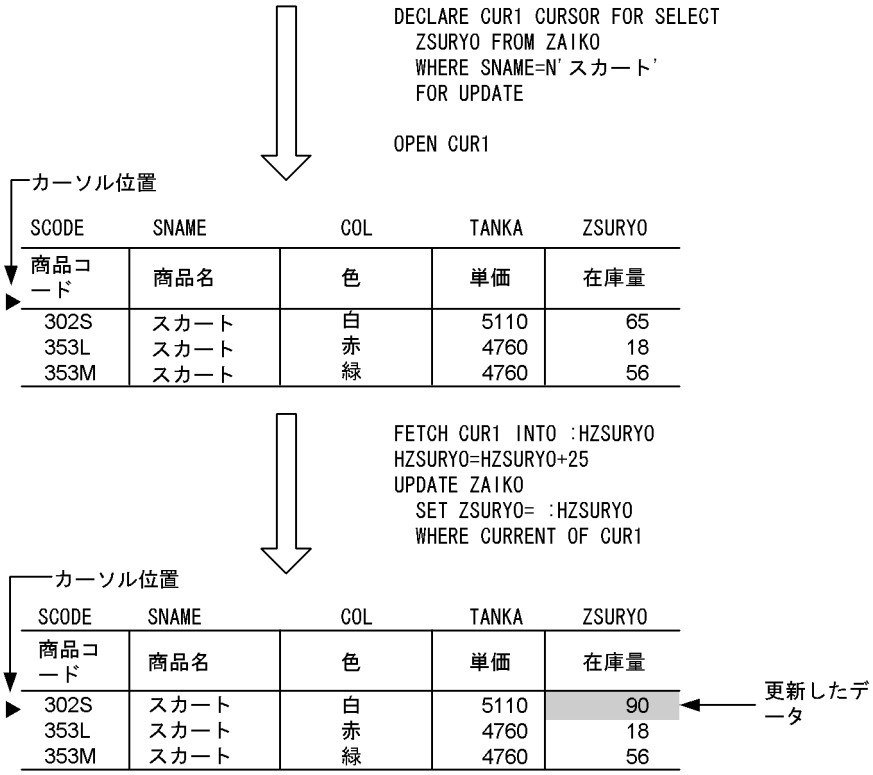
図 2–11　表の更新の処理手順



図「表の更新の処理手順」で示した処理手順の各ステップは，データの更新を除くと基本的に図「検索結果の表に対する UAP からのデータの処理手順」と同じです。

カーソルを使用してデータを更新する例を次の図に示します。なお，既にデータの取り出しまで終わっているものとします。

図 2–12　カーソルを使用して表を更新する例



2.4.2　条件指定による更新

データを更新するとき条件を指定すると，条件を満たすすべての行が更新されます。条件指定による更新は，UPDATE 文の WHERE 句で指定します。

なお，表がキーレンジ分割されている場合，分割するキーとなる列の値は更新できません。

条件指定による更新の例として，在庫表から商品コードが"411M"の数量を"20"に更新する例を次の図に示します。

図 2–13　条件指定による更新の例

ZAIKO（在庫表）

SCODE	SNAME	COL	TANKA	ZSURYO
商品コード	商品名	色	単価	在庫量
101L	ブラウス	青	3500	62
101M	ブラウス	白	3500	85
201M	ポロシャツ	白	3640	29
202M	ポロシャツ	赤	3640	67
302S	スカート	白	5110	65
353L	スカート	赤	4760	18
353M	スカート	緑	4760	56
411M	セーター	青	8400	12
412M	セーター	赤	8400	22
591L	ソックス	赤	250	300
591M	ソックス	青	250	90
591S	ソックス	白	250	280

```
UPDATE ZAIKO
  SET ZSURYO=20
  WHERE SCODE='411M'
```

更新結果

SCODE	SNAME	COL	TANKA	ZSURYO	
商品コード	商品名	色	単価	在庫量	
：	：	：	：	：	
411M	セーター	青	8400	20	← 更新したデータ
412M	セーター	赤	8400	22	
591L	ソックス	赤	250	300	
：	：	：	：	：	

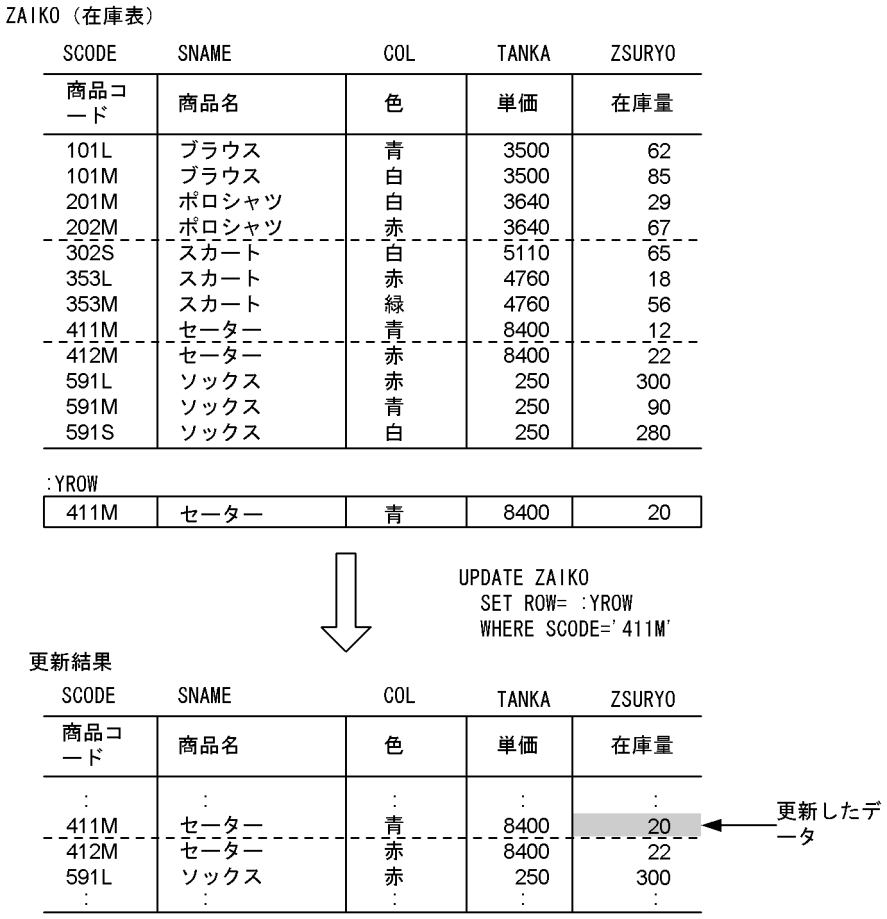
2.4.3　FIX 属性の表の更新

FIX 属性の表を更新する場合，1 行全体を一つの固定長データとして更新できます。行単位の更新は，UPDATE 文の SET 句に ROW を指定します。

1 行単位で更新すると，列ごとに更新するオーバヘッドがなくなるため，アクセス性能が向上します。

行単位の更新例として，在庫表の商品コードが"411M"の数量を"12"から"20"に更新するとき，更新する値を埋込み変数（：YROW）に指定する例を次の図に示します。

図 2–14　行単位の更新例



2.4.4　繰返し列がある表の更新

繰返し列がある表を更新する場合，次の三つの更新方法があります。

- 既存の要素を更新する方法（SET 句）
- 新たに要素を追加する方法（ADD 句）
- 既存の要素を削除する方法（DELETE 句）

繰返し列がある表を更新する場合は，繰返し列名［{添字｜＊}］で更新する繰返し列の要素を指定します。添字は，要素の位置です。

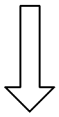
ここでは，新たに要素を追加する方法について説明します。

繰返し列がある表の更新例として，社員表の氏名が"中村和男"の資格に，要素"データベース"を追加する例を次の図に示します。

図 2-15　繰返し列がある表の更新例

社員表

氏名	資格	性別	家族	続柄	扶養
伊藤栄一	情報処理１種	男	虎夫	父	1
	ネットワーク		ウメ	母	1
	情報処理２種		綾子	妻	1
			太郎	長男	1
			恵子	次女	1
中村和男	情報処理２種	男	和彦	父	0
	英語検定２級		陽子	妻	1
河原秀雄	シスアド	男	直子	母	1
井上俊夫		男			



```
UPDATE 社員表
  ADD 資格[*]=ARRAY[N'データベース']
  WHERE 氏名=N'中村和男'
```

更新結果

氏名	資格	性別	家族	続柄	扶養
伊藤栄一	情報処理１種	男	虎夫	父	1
	ネットワーク		ウメ	母	1
	情報処理２種		綾子	妻	1
			太郎	長男	1
			恵子	次女	1
中村和男	情報処理２種	男	和彦	父	0
	英語検定２級		陽子	妻	1
	データベース				
河原秀雄	シスアド	男	直子	母	1
井上俊夫		男			

更新したデータ（追加した要素）

2.5　データの削除

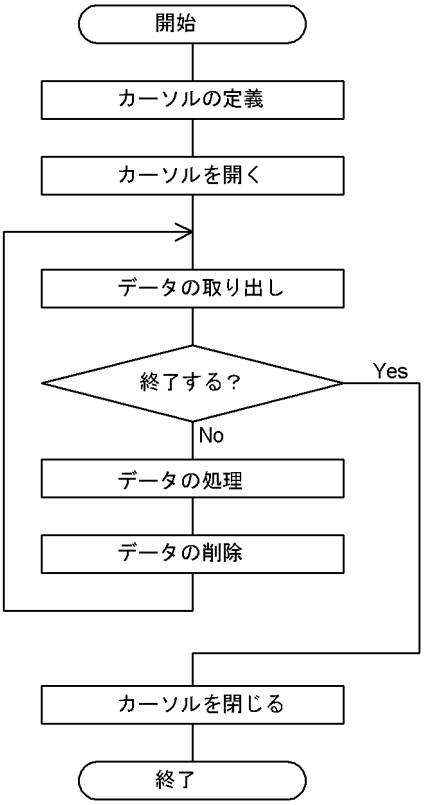
表中の情報の削除には，次に示す三つの方法があります。

- カーソルが指している行を削除する。
- 条件を満たす行だけ削除する。
- 表のすべての行を削除する。

2.5.1　カーソルを使用した削除

表中の行を削除する場合，カーソルを使用して 1 行ずつ内容を確認しながら削除します。カーソルを使用した行の削除の処理手順を次の図に示します。

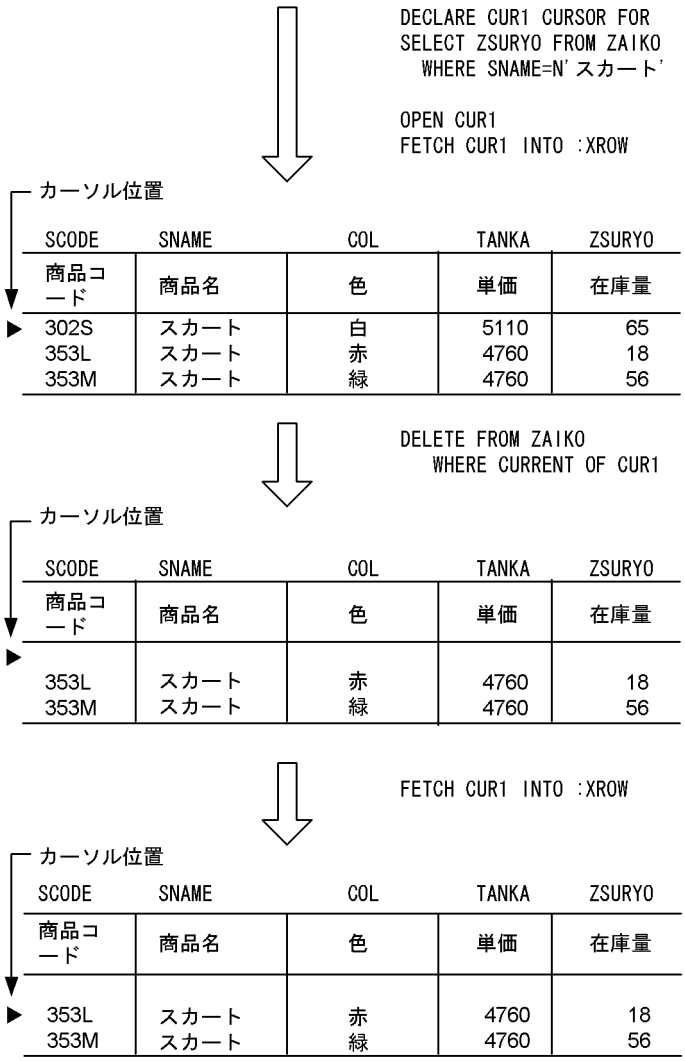
図 2–16　行の削除の処理手順



図「行の削除の処理手順」で示した処理手順の各ステップは，データの削除を除くと基本的に図「検索結果の表に対する UAP からのデータの処理手順」と同じです。

カーソルを使用してデータを 1 行ずつ削除する例を次の図に示します。なお，既にデータの取り出しまで終わっているものとします。

図 2–17　カーソルを使用して行を削除する例



2.5.2　条件指定による削除

データを削除するとき条件を指定すると，条件を満たすすべての行が削除されます。条件指定による削除は，DELETE 文の WHERE 句で条件を指定して行います。

条件指定による削除の例として，在庫表から商品名が"スカート"のデータだけを削除する例を次の図に示します。

図 2–18　条件指定による削除の例

ZAIKO（在庫表）

SCODE	SNAME	COL	TANKA	ZSURYO
商品コード	商品名	色	単価	在庫量
101L	ブラウス	青	3500	62
101M	ブラウス	白	3500	85
201M	ポロシャツ	白	3640	29
202M	ポロシャツ	赤	3640	67
302S	スカート	白	5110	65
353L	スカート	赤	4760	18
353M	スカート	緑	4760	56
411M	セーター	青	8400	12
412M	セーター	赤	8400	22
591L	ソックス	赤	250	300
591M	ソックス	青	250	90
591S	ソックス	白	250	280



```
DELETE FROM ZAIKO
  WHERE SNAME=N'スカート'
```

削除結果

SCODE	SNAME	COL	TANKA	ZSURYO
商品コード	商品名	色	単価	在庫量
101L	ブラウス	青	3500	62
101M	ブラウス	白	3500	85
201M	ポロシャツ	白	3640	29
202M	ポロシャツ	赤	3640	67
411M	セーター	青	8400	12
412M	セーター	赤	8400	22
591L	ソックス	赤	250	300
591M	ソックス	青	250	90
591S	ソックス	白	250	280

2.5.3　表の全行削除

データの削除の対象となる表が実表の場合，表中のすべての行を一括して削除することもできます。表の全行を一括して削除するには，PURGE TABLE 文を使用します。表の全行を一括して削除すると，DELETE 文の WHERE 句を省略して（条件を指定しないで）削除するよりも処理性能に優れています。

なお，OLTP 環境下の X/Open に従ったアプリケーションプログラムの場合は，PURGE TABLE 文を実行できません。

表の全行削除の例として，在庫表の全データを削除する例を次の図に示します。

図 2–19　表の全行削除の例

ZAIKO（在庫表）

SCODE	SNAME	COL	TANKA	ZSURYO
商品コード	商品名	色	単価	在庫量
101L	ブラウス	青	3500	62
101M	ブラウス	白	3500	85
201M	ポロシャツ	白	3640	29
202M	ポロシャツ	赤	3640	67
302S	スカート	白	5110	65
353L	スカート	赤	4760	18
353M	スカート	緑	4760	56
411M	セーター	青	8400	12
412M	セーター	赤	8400	22
591L	ソックス	赤	250	300
591M	ソックス	青	250	90
591S	ソックス	白	250	280



PURGE TABLE ZAIKO

削除結果

SCODE	SNAME	COL	TANKA	ZSURYO
商品コード	商品名	色	単価	在庫量

2.6　データの挿入

表中への行の挿入には，次に示す二つの方法があります。

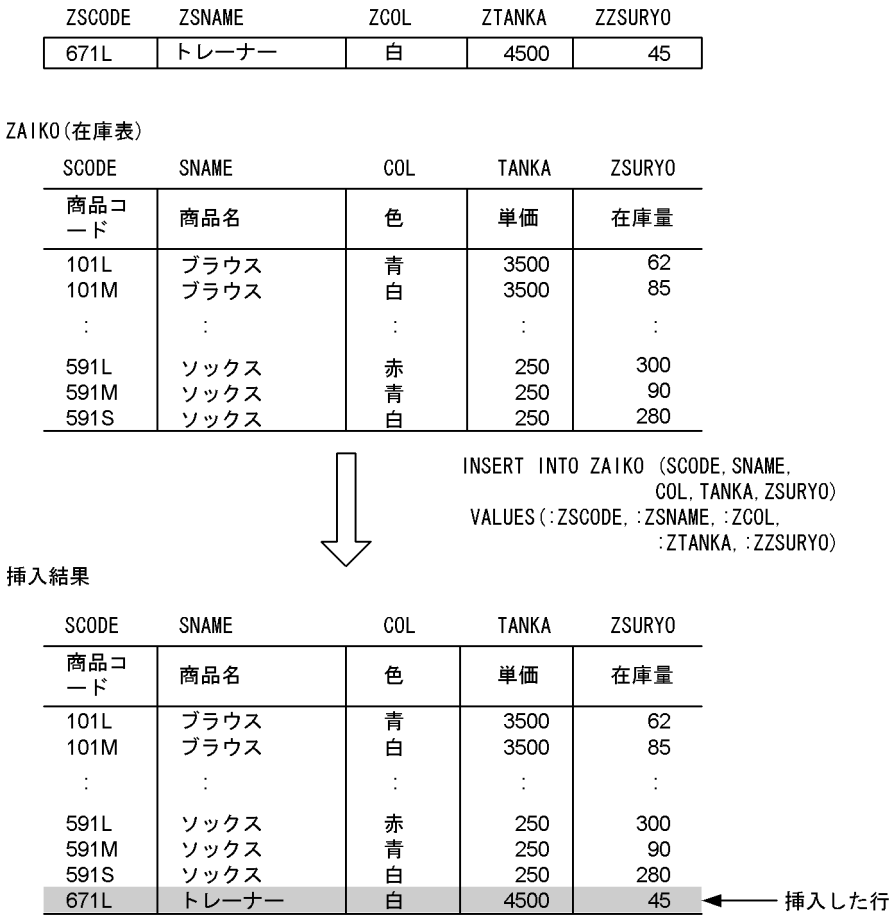
- 列単位で行を挿入する。
- 行単位で行を挿入する。

2.6.1　列単位の挿入

表に行を挿入するとき，値を直接指定して一つの行を挿入できます。行の挿入は，INSERT 文を使用します。

列単位の挿入の例として，埋込み変数（：ZSCODE～：ZZSURYO）に設定されている値を在庫表の各列に挿入する例を次の図に示します。

図 2–20　列単位の行の挿入例



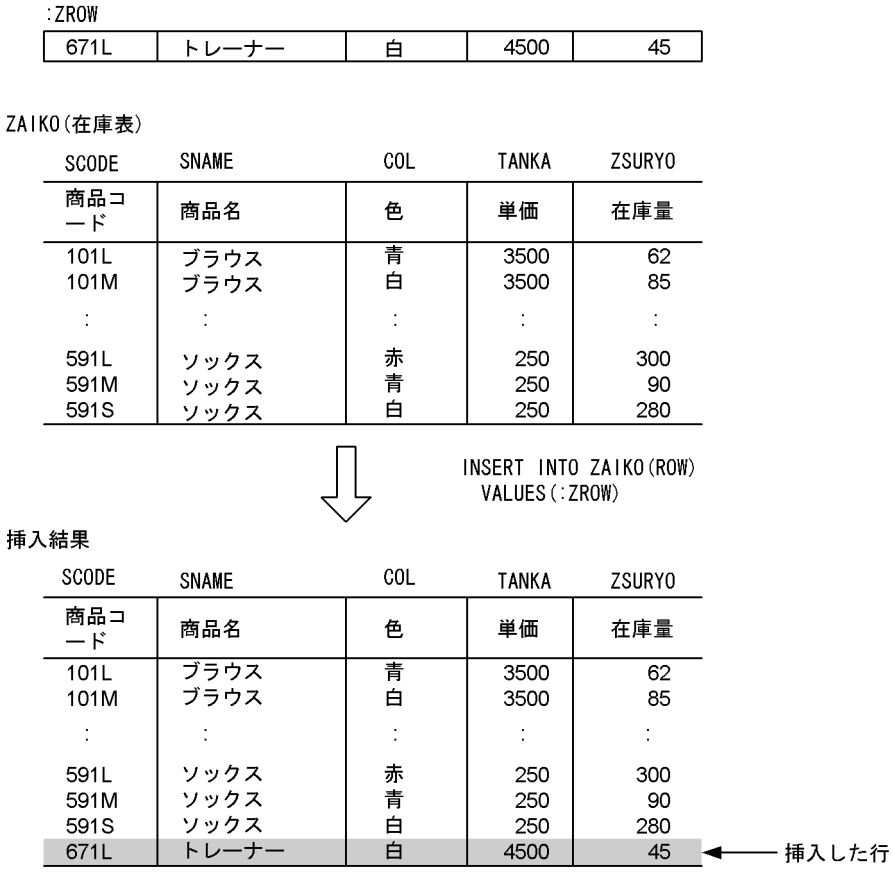
2.6.2　FIX 属性の表への行の挿入

FIX 属性の表に行を挿入する場合，1 行全体を一つの固定長データとして行単位で行を挿入できます。行単位で行を挿入するには，INSERT 文に ROW を指定します。

なお，行単位の挿入は，実表のときだけ実行できます。

行単位の挿入例として，埋込み変数（：ZROW）に設定した値を 1 行分まとめて在庫表に挿入する例を次の図に示します。

図 2–21　行単位の行の挿入例



2.6.3　繰返し列がある表への行の挿入

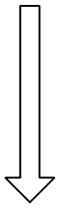
繰返し列がある表に行を挿入する場合，挿入値に ARRAY[要素の値〔,要素の値〕…]を指定します。

繰返し列がある表への行の挿入例として，社員表に行を追加する例を次の図に示します。

図 2-22　繰返し列がある表への行の挿入例

社員表

氏名	資格	性別	家族	続柄	扶養
伊藤栄一	情報処理２種	男	虎夫	父	1
	ネットワーク		ウメ	母	1
	情報処理２種		綾子	妻	1
			太郎	長男	1
			恵子	次女	1
中村和男	情報処理２種	男	和彦	父	0
	英語検定２級		陽子	妻	1
河原秀雄	シスアド	男	直子	母	1
井上俊夫		男			



```
INSERT INTO 社員表
  VALUES(N'平尾英子',
         ARRAY[N'情報処理１種',N'情報処理２種'],
         N'女',
         ARRAY[N'真樹',N'健一'],
         ARRAY[N'夫',N'長男'],
         ARRAY[1,1])
```

挿入結果

氏名	資格	性別	家族	続柄	扶養	
伊藤栄一	情報処理２種	男	虎夫	父	1	
	ネットワーク		ウメ	母	1	
	情報処理２種		綾子	妻	1	
			太郎	長男	1	
			恵子	次女	1	
中村和男	情報処理２種	男	和彦	父	0	
	英語検定２級		陽子	妻	1	
河原秀雄	シスアド	男	直子	母	1	
井上俊夫		男				
平尾英子	情報処理1種	女	真樹	夫	1	← 挿入した行
	情報処理2種		健一	長男	1	

2.7　特定データの探索

表中のデータを条件付きで操作するには，探索条件を指定します。探索条件とは行の選択条件のことで，論理演算子を使用して複数の条件を組み合わせることもできます。表中のデータの探索には，次に示す四つの方法があります。

- 特定の範囲内のデータを探索する。
- 特定の文字パターンを探索する。
- ナル値でないデータを探索する。
- 複数の条件を満たすデータを探索する。

2.7.1　特定の範囲内のデータの探索

特定の範囲を指定して行を操作するには，比較述語，BETWEEN 述語，IN 述語のどれかを条件に合わせて使用します。

(1)　比較述語

探索条件として，等価，又は大小比較をするときに比較述語を使用します。

比較述語を使用したデータの探索例として，在庫表から在庫量が 50 以下の商品の商品コード，及び商品名を検索する例を次の図に示します。

図 2–23　比較述語を使用したデータの探索例

ZAIKO（在庫表）

SCODE	SNAME	COL	TANKA	ZSURYO
商品コード	商品名	色	単価	在庫量
101L	ブラウス	青	3500	62
101M	ブラウス	白	3500	85
201M	ポロシャツ	白	3640	29
202M	ポロシャツ	赤	3640	67
302S	スカート	白	5110	65
353L	スカート	赤	4760	18
353M	スカート	緑	4760	56
411M	セーター	青	8400	12
412M	セーター	赤	8400	22
591L	ソックス	赤	250	300
591M	ソックス	青	250	90
591S	ソックス	白	250	280



```
SELECT SCODE,SNAME FROM ZAIKO
  WHERE ZSURYO<=50
```

探索結果

SCODE	SNAME
商品コード	商品名
201M	ポロシャツ
353L	スカート
411M	セーター
412M	セーター

(2)　BETWEEN 述語

探索条件として，一定の範囲内のデータだけを取り出すときに BETWEEN 述語を使用します。

BETWEEN 述語を使用したデータの探索例として，在庫表から在庫量が 200 以上 300 以下の商品の商品コード，及び商品名を検索する例を次の図に示します。

図 2–24　BETWEEN 述語を使用したデータの探索例

ZAIKO（在庫表）

SCODE	SNAME	COL	TANKA	ZSURYO
商品コード	商品名	色	単価	在庫量
101L	ブラウス	青	3500	62
101M	ブラウス	白	3500	85
201M	ポロシャツ	白	3640	29
202M	ポロシャツ	赤	3640	67
302S	スカート	白	5110	65
353L	スカート	赤	4760	18
353M	スカート	緑	4760	56
411M	セーター	青	8400	12
412M	セーター	赤	8400	22
591L	ソックス	赤	250	300
591M	ソックス	青	250	90
591S	ソックス	白	250	280



```
SELECT SCODE,SNAME FROM ZAIKO
  WHERE ZSURYO BETWEEN 200 AND 300
```

探索結果

SCODE	SNAME
商品コード	商品名
591L	ソックス
591S	ソックス

(3)　IN 述語

探索条件として，指定した複数の値と一致するデータだけを取り出すときに IN 述語を使用します。

IN 述語を使用したデータの探索例として，在庫表から単価が"3640"，又は"4760"の商品の商品コード，及び商品名を検索する例を次の図に示します。

図 2–25　IN 述語を使用したデータの探索例

ZAIKO（在庫表）

SCODE	SNAME	COL	TANKA	ZSURYO
商品コード	商品名	色	単価	在庫量
101L	ブラウス	青	3500	62
101M	ブラウス	白	3500	85
201M	ポロシャツ	白	3640	29
202M	ポロシャツ	赤	3640	67
302S	スカート	白	5110	65
353L	スカート	赤	4760	18
353M	スカート	緑	4760	56
411M	セーター	青	8400	12
412M	セーター	赤	8400	22
591L	ソックス	赤	250	300
591M	ソックス	青	250	90
591S	ソックス	白	250	280



```
SELECT SCODE,SNAME FROM ZAIKO
  WHERE TANKA IN(3640,4760)
```

探索結果

SCODE	SNAME
商品コード	商品名
201M	ポロシャツ
202M	ポロシャツ
353L	スカート
353M	スカート

2.7.2　特定の文字パターンの探索

特定の文字パターンを含む列がある行を操作するには，LIKE 述語を使用します。

LIKE 述語を使用したデータの探索例として，受注表から得意先コードの 2 文字目が"T"の伝票番号，商品コード，及び受注量を検索する例を次の図に示します。

図 2–26　LIKE 述語を使用したデータの探索例

JUTYU（受注表）

DNO	TCODE	SCODE	JSURYO	JDATE	JTIME
伝票番号	得意先コード	商品コード	受注量	受付日付	受付時刻
026551	TT002	101M	10	1995-06-14	09:23:11
026552	TT002	591M	25	1995-06-14	09:23:11
026553	TH001	353M	8	1995-06-14	10:10:55
026554	TK001	411M	6	1995-06-14	10:15:47
026555	TA001	591M	30	1995-06-14	10:15:47
026556	TT002	202M	10	1995-06-14	11:48:09
026557	TZ001	411M	5	1995-06-14	13:02:00
026558	TZ001	412M	4	1995-06-14	13:02:00
026559	TH001	591M	80	1995-06-14	14:04:16
026560	TT001	591L	10	1995-06-14	15:31:20



```
SELECT DNO,SCODE,JSURYO
    FROM JUTYU
    WHERE TCODE LIKE'_T%'
```

探索結果

DNO	SCODE	JSURYO
伝票番号	商品コード	受注量
026551	101M	10
026552	591M	25
026556	202M	10
026560	591L	10

2.7.3　ナル値でないデータの探索

表の列中にナル値が含まれていない行を操作するには，NULL 述語に NOT を組み合わせて使用します。

なお，NULL 述語に NOT を組み合わせない場合，ナル値が含まれている行が操作の対象になります。

NULL 述語と NOT を組み合わせたデータの探索例として，受注表から得意先コードが未設定（ナル値）ではない伝票の伝票番号，商品コード，及び受注量を検索する例を次の図に示します。

図 2–27　NULL 述語と NOT を組み合わせて使用したデータの探索例

JUTYU（受注表）

DNO	TCODE	SCODE	JSURYO	JDATE	JTIME
伝票番号	得意先コード	商品コード	受注量	受付日付	受付時刻
026551	TT002	101M	10	1995-06-14	09:23:11
026552	TT002	591M	25	1995-06-14	09:23:11
026553	TH001	353M	8	1995-06-14	10:10:55
026554	TK001	411M	6	1995-06-14	10:15:47
026555	TA001	591M	30	1995-06-14	10:15:47
026556	TT002	202M	10	1995-06-14	11:48:09
026557	TZ001	411M	5	1995-06-14	13:02:00
026558	TZ001	412M	4	1995-06-14	13:02:00
026559	TH001	591M	80	1995-06-14	14:04:16
026560	TT001	591L	10	1995-06-14	15:31:20



```
SELECT DNO,SCODE,JSURYO FROM JUTYU
  WHERE TCODE IS NOT NULL
```

探索結果

DNO	SCODE	JSURYO
伝票番号	商品コード	受注量
026551	101M	10
026552	591M	25
026553	353M	8
026554	411M	6
026555	591M	30
026556	202M	10
026557	411M	5
026558	412M	4
026559	591M	80
026560	591L	10

2.7.4　複数の条件を満たすデータの探索

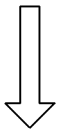
複数の条件を組み合わせて，該当するデータを含む行を操作するには，論理演算子（AND，OR，NOT）を使用します。

複数の条件を満たすデータの探索例として，在庫表から商品名が"ブラウス"，又は"ポロシャツ"で，在庫量が 50 以上の商品の商品コード，及び在庫量を検索する例を次の図に示します。

図 2-28　複数の条件を満たすデータの探索例

ZAIKO（在庫表）

SCODE	SNAME	COL	TANKA	ZSURYO
商品コード	商品名	色	単価	在庫量
101L	ブラウス	青	3500	62
101M	ブラウス	白	3500	85
201M	ポロシャツ	白	3640	29
202M	ポロシャツ	赤	3640	67
302S	スカート	白	5110	65
353L	スカート	赤	4760	18
353M	スカート	緑	4760	56
411M	セーター	青	8400	12
412M	セーター	赤	8400	22
591L	ソックス	赤	250	300
591M	ソックス	青	250	90
591S	ソックス	白	250	280



```
SELECT SCODE,ZSURYO FROM ZAIKO
  WHERE (SNAME=N'ブラウス'
      OR SNAME=N'ポロシャツ')
      AND ZSURYO>=50
```

探索結果

SCODE	ZSURYO
商品コード	在庫量
101L	62
101M	85
202M	67

2.7.5　論理述語を使用した検索

抽象データ型で定義した関数や，ユーザ定義関数の結果が論理値（TRUE，FALSE，又は UNKNOWN）となる場合，真偽を判定するために論理述語を使用します。論理述語を使用したデータの探索例については，「抽象データ型を含む表のデータ操作」を参照してください。

2.7.6　構造化繰返し述語を使用した検索

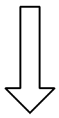
繰返し列がある表の，複数の繰返し列に対して条件を指定して検索する場合，構造化繰返し述語を使用します。

構造化繰返し述語を使用した検索例として，社員表から父を扶養している社員を検索する例を次の図に示します。

図 2–29　構造化繰返し述語を使用した検索例

社員表

氏名	資格	性別	家族	続柄	扶養
伊藤栄一	情報処理１種	男	虎夫	父	1
	ネットワーク		ウメ	母	1
	情報処理２種		綾子	妻	1
			太郎	長男	1
			恵子	次女	1
中村和男	情報処理２種	男	和彦	父	0
	英語検定２級		陽子	妻	1
河原秀雄	シスアド	男	直子	母	1
井上俊夫		男			



```
SELECT 氏名 FROM 社員表
  WHERE ARRAY(続柄,扶養)
          [ANY](続柄=N'父' AND 扶養=1)
```

探索結果

氏名
伊藤栄一

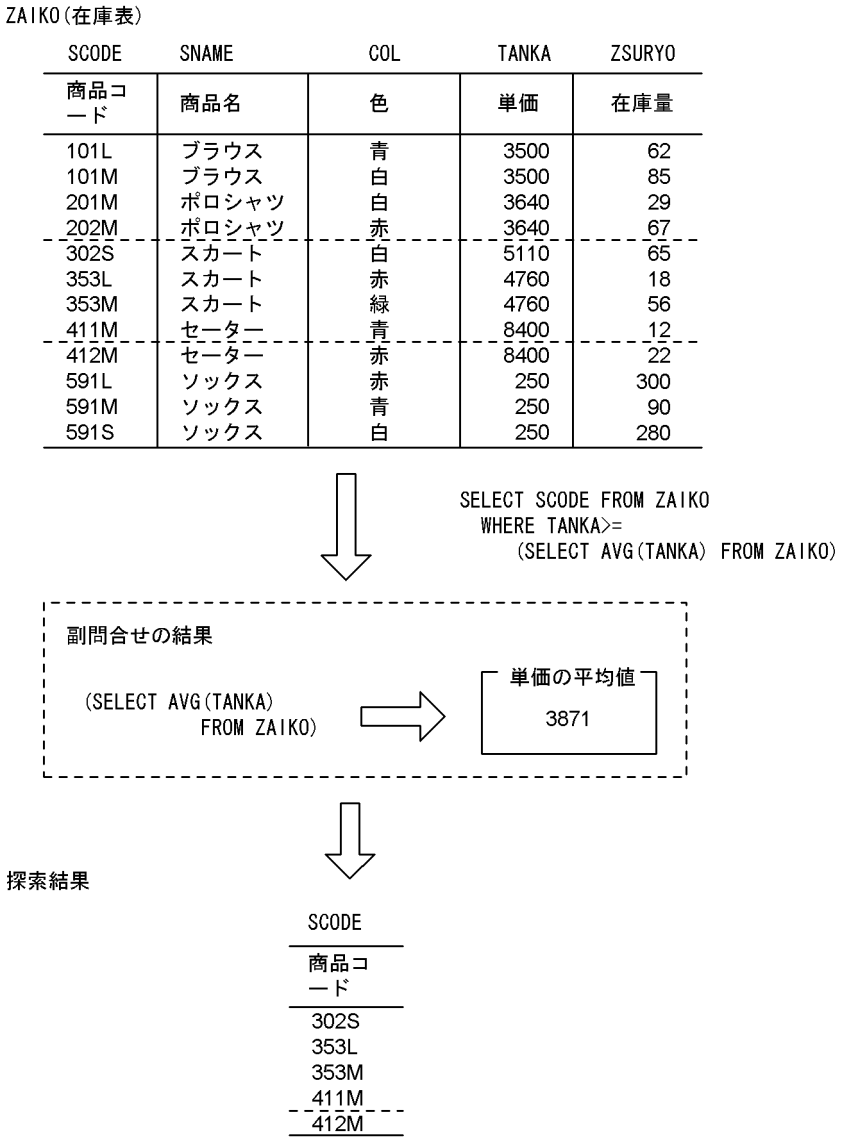
注　続柄，扶養の列には，続柄，扶養で構成される複数列インデクスが定義されている必要があります。

2.7.7　副問合せを使用した検索

問合せ結果の値を検索条件の中で指定することで，問合せを構造的に表現します。副問合せでは，データベースに対して，より複雑な問合せを読みやすくできます。

副問合せを使用した検索例として，在庫表から単価が平均値以上の商品の商品コードを検索する例を次の図に示します。

図 2-30　副問合せを使用した検索例



(1)　限定述語を使用した副問合せ

副問合せの結果が，指定した比較条件を満たしているかどうか判定し，副問合せの結果の範囲を更に絞り込むとき，限定述語を使用します。

限定述語を使用した副問合せの例として，在庫表から"ブラウス"（商品コードに関係なく）のどの在庫量よりも多く在庫がある商品の商品コードと商品名を検索する例を次の図に示します。

図 2–31　限定述語を使用した副問合せの検索例

ZAIKO（在庫表）

SCODE	SNAME	COL	TANKA	ZSURYO
商品コード	商品名	色	単価	在庫量
101L	ブラウス	青	3500	62
101M	ブラウス	白	3500	85
201M	ポロシャツ	白	3640	29
202M	ポロシャツ	赤	3640	67
302S	スカート	白	5110	65
353L	スカート	赤	4760	18
353M	スカート	緑	4760	56
411M	セーター	青	8400	12
412M	セーター	赤	8400	22
591L	ソックス	赤	250	300
591M	ソックス	青	250	90
591S	ソックス	白	250	280

```
SELECT SCODE,SNAME FROM ZAIKO
  WHERE ZSURYO>ALL
        (SELECT ZSURYO FROM ZAIKO
           WHERE SNAME=N'ブラウス')
```

副問合せの結果

```
(SELECT ZSURYO FROM ZAIKO
  WHERE SNAME=N'ブラウス'
```

ブラウスの在庫量

85
62

探索結果

SCODE	SNAME
商品コード	商品名
591L	ソックス
591M	ソックス
591S	ソックス

(2)　EXISTS 述語を使用した副問合せ

副問合せの結果が空集合でないかどうか判定するとき，EXISTS 述語を使用します。

EXISTS 述語を使用した副問合せの検索例として，在庫表と受注表から受注のない商品を検索する例を次の図に示します。

図 2-32　EXISTS 述語を使用した副問合せの検索例

ZAIKO(在庫表)

SCODE	SNAME	COL	TANKA	ZSURYO
商品コード	商品名	色	単価	在庫量
101L	ブラウス	青	3500	62
101M	ブラウス	白	3500	85
201M	ポロシャツ	白	3640	29
202M	ポロシャツ	赤	3640	67
302S	スカート	白	5110	65
353L	スカート	赤	4760	18
353M	スカート	緑	4760	56
411M	セーター	青	8400	12
412M	セーター	赤	8400	22
591L	ソックス	赤	250	300
591M	ソックス	青	250	90
591S	ソックス	白	250	280

JUTYU(受注表)

DNO	TCODE	SCODE	JSURYO	・・・
伝票番号	得意先コード	商品コード	受注量	・・・
026551	TT002	101M	10	・・・
026552	TT002	591M	25	
026553	TH001	353M	8	
026554	TK001	411M	6	
026555	TA001	591M	30	・・・
026556	TT002	202M	10	
026557	TZ001	411M	5	
026558	TZ001	412M	4	
026559	TH001	591M	80	・・・
026560	TT001	591L	10	

```
SELECT * FROM ZAIKO
  WHERE NOT EXISTS
      (SELECT * FROM JUTYU
         WHERE SCODE=ZAIKO.SCODE)
```

副問合せの結果

```
(SELECT * FROM JUTYU
  WHERE SCODE=ZAIKO.SCODE)
```

商品コード
101M
202M
353M
411M
412M
591L
591M

探索結果

SCODE	SNAME	COL	TANKA	ZSURYO
商品コード	商品名	色	単価	在庫量
101L	ブラウス	青	3500	62
201M	ポロシャツ	白	3640	29
302S	スカート	白	5110	65
353L	スカート	赤	4760	18
591S	ソックス	白	250	280

2.8　データの演算

表中の列の数値や日時を検索して，演算した結果を取り出すことができます。

表中のデータの演算には，次に示す二つがあります。

- 数値データを四則演算する。
- 日付，又は時刻データを演算する。

2.8.1　数値データの四則演算

指定した列の数値を基に，四則演算をして検索結果を取り出せます。

四則演算には，加算，減算，乗算，及び除算があります。

数値データの四則演算の例として，在庫表から商品名が"ソックス"の単価，及び在庫量から売上げ見込みを演算し，商品コード，及び演算結果（百円単位）を取り出す例を次の図に示します。

図 2–33　数値データの演算の例

ZAIKO（在庫表）

SCODE	SNAME	COL	TANKA	ZSURYO
商品コード	商品名	色	単価	在庫量
101L	ブラウス	青	3500	62
101M	ブラウス	白	3500	85
201M	ポロシャツ	白	3640	29
202M	ポロシャツ	赤	3640	67
302S	スカート	白	5110	65
353L	スカート	赤	4760	18
353M	スカート	緑	4760	56
411M	セーター	青	8400	12
412M	セーター	赤	800	22
591L	ソックス	赤	250	300
591M	ソックス	青	250	90
591S	ソックス	白	250	280

```
SELECT SCODE,TANKA*ZSURYO/100,
                N'百円' FROM ZAIKO
  WHERE SNAME=N'ソックス'
```

演算結果

SCODE	TANKA*ZSURYO/100
商品コード	売り上げ見込み
591L	750百円
591M	225百円
591S	700百円

2.8.2　日付，時刻データの演算

表中の日付データ（時刻データを含む）を演算して，期間を限定した検索結果を取り出すことができます。

日付データ，又は時刻データの演算にはスカラ関数を使用します。日付データには日付演算を使用し，時刻データには時刻演算を使用します。

時刻データの演算の例として，受注表から"12：00：00"より前に受注された伝票の伝票番号，商品コード，及び受注量を取り出す例を次の図に示します。

図 2–34　日付データの演算の例

JUTYU（受注表）

DNO	TCODE	SCODE	JSURYO	JDATE	JTIME
伝票番号	得意先コード	商品コード	受注量	受付日付	受付時刻
026551	TT002	101M	10	1995-06-14	09:23:11
026552	TT002	591M	25	1995-06-14	09:23:11
026553	TH001	353M	8	1995-06-14	10:10:55
026554	TK001	411M	6	1995-06-14	10:15:47
026555	TA001	591M	30	1995-06-14	10:15:47
026556	TT002	202M	10	1995-06-14	11:48:09
026557	TZ001	411M	5	1995-06-14	13:02:00
026558	TZ001	412M	4	1995-06-14	13:02:00
026559	TH001	591M	80	1995-06-14	14:04:16
026560	TT001	591L	10	1995-06-14	15:31:20



```
SELECT DNO,SCODE,JSURYO
    FROM JUTYU
  WHERE JTIME<TIME('12:00:00')
```

演算結果

DNO	SCODE	JSURYO
伝票番号	商品コード	受注量
026551	101M	10
026552	591M	25
026553	353M	8
026554	411M	6
026555	591M	30
026556	202M	10

2.9　データの加工

表中のデータを取り出すとき，グループ分けや，昇順，又は降順に並べ替えるなどデータの加工ができます。表中のデータの加工には，次に示す三つがあります。

- データをグループ分けする。
- データを昇順，又は降順に並べ替える。
- 重複したデータを排除する。

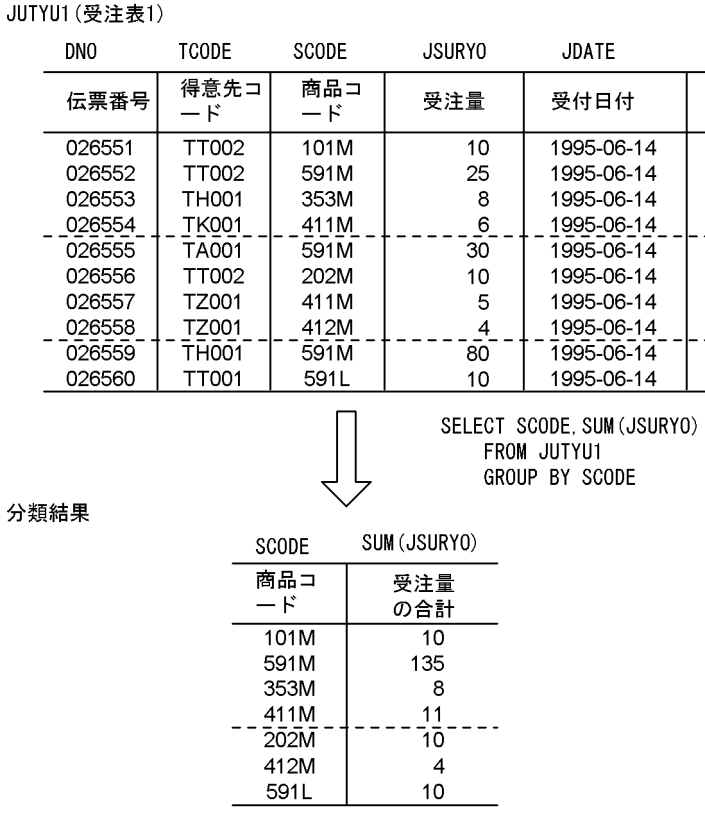
2.9.1　データのグループ分け

指定した列の中に同一の値が複数あるとき，各値をグループごとに分類して検索結果を取り出すことができます。グループ分けして検索結果を取り出すには，GROUP BY 句を使用します。

また，各グループの平均値，合計値，最大値，最小値，及び行数を求めるには，それぞれ AVG，SUM，MAX，MIN，及び COUNT の集合関数を使用します。

データのグループ分けの例として，受注表 1 から商品コードごとに分類し，受注量の合計を取り出す例を次の図に示します。

図 2–35　データのグループ分けの例



2.9.2　データの並べ替え

表中の列を指定し，昇順，又は降順にデータをソート（並べ替え）して検索結果を取り出すことができます。

データの並べ替えの例として，受注表から伝票番号，商品コード，及び発注量を検索し，商品コードを昇順にソートする例を次の図に示します。

図 2–36　データの並べ替えの例

JUTYU（受注表）

DNO	TCODE	SCODE	JSURYO	JDATE	JTIME
伝票番号	得意先コード	商品コード	受注量	受付日付	受付時刻
026551	TT002	101M	10	1995-06-14	09:23:11
026552	TT002	591M	25	1995-06-14	09:23:11
026553	TH001	353M	8	1995-06-14	10:10:55
026554	TK001	411M	6	1995-06-14	10:15:47
026555	TA001	591M	30	1995-06-14	10:15:47
026556	TT002	202M	10	1995-06-14	11:48:09
026557	TZ001	411M	5	1995-06-14	13:02:00
026558	TZ001	412M	4	1995-06-14	13:02:00
026559	TH001	591M	80	1995-06-14	14:04:16
026560	TT001	591L	10	1995-06-14	15:31:20

```
SELECT DNO, SCODE, JSURYO
    FROM JUTYU
    ORDER BY SCODE
```

探索結果

DNO	SCODE	JSURYO
伝票番号	商品コード	受注量
026551	101M	10
026556	202M	10
026553	353M	8
026554	411M	6
026557	411M	5
026558	412M	4
026560	591L	10
026552	591M	25
026555	591M	30
026559	591M	80

2.9.3　重複したデータの排除

2 個以上の表を操作するとき，重複するデータを取り除いた検索結果を取り出すことができます。重複するデータを取り除くには，UNION，又は DISTINCT を使用します。

重複するデータの排除の例として，二つの受注表から受注量が 10 以上の商品の商品コードを検索し，重複したデータを取り除く例を次の図に示します。

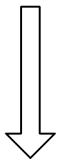
図 2-37　重複するデータの排除の例

JUTYU1（受注表1）

DNO	TCODE	SCODE	JSURYO	JDATE	JTIME
伝票番号	得意先コード	商品コード	受注量	受付日付	受付時刻
026551	TT002	101M	10	1995-06-14	09:23:11
026552	TT002	591M	25	1995-06-14	09:23:11
026553	TH001	353M	8	1995-06-14	10:10:55
026554	TK001	411M	6	1995-06-14	10:15:47
026555	TA001	591M	30	1995-06-14	10:15:47
026556	TT002	202M	10	1995-06-14	11:48:09

JUTYU2（受注表2）

DNO	TCODE	SCODE	JSURYO	JDATE	JTIME
伝票番号	得意先コード	商品コード	受注量	受付日付	受付時刻
026557	TZ001	411M	5	1995-06-14	13:02:00
026558	TZ001	412M	4	1995-06-14	13:02:00
026559	TH001	591M	80	1995-06-14	14:04:16
026560	TT001	591L	10	1995-06-14	15:31:20



```
SELECT SCODE FROM JUTYU1
  WHERE JSURYO>=10
  UNION
SELECT SCODE FROM JUTYU2
  WHERE JSURYO>=10
```

探索結果

SCODE
商品コード
101M
591M
202M
591L

2.10　表の外結合

全体の情報を持つ外表と部分的な情報を持つ内表とを結合し，通常の結合（内結合）で得られる情報以外に，外表に関するすべての行の情報が必要な場合，アウタジョイン（表の外結合）によって検索結果を取り出すことができます。アウタジョインでは，結合条件を満たしていない場合，内表の列にはナル値が補われるので，欠損値を含んだ結合ができます。

アウタジョインの例として，在庫表と受注表を外結合し，在庫量が 100 未満の商品の商品コード，商品名，色，及び伝票番号を検索する例を次の図に示します。

図 2-38　アウタジョインの例

ZAIKO（在庫表）

SCODE	SNAME	COL	TANKA	ZSURYO
商品コード	商品名	色	単価	在庫量
101L	ブラウス	青	3500	62
101M	ブラウス	白	3500	85
201M	ポロシャツ	白	3640	29
202M	ポロシャツ	赤	3640	67
302S	スカート	白	5110	65
353L	スカート	赤	4760	18
353M	スカート	緑	4760	56
411M	セーター	青	8400	12
412M	セーター	赤	8400	22
591L	ソックス	赤	250	300
591M	ソックス	青	250	90
591S	ソックス	白	250	280

JUTYU（受注表）

DNO	TCODE	SCODE	JSURYO	･･･
伝票番号	得意先コード	商品コード	受注量	･･･
026551	TT002	101M	10	･･･
026552	TT002	591M	25	･･･
026553	TH001	353M	8	･･･
026554	TK001	411M	6	･･･
026555	TA001	591M	30	･･･
026556	TT002	202M	10	･･･
026557	TZ001	411M	5	･･･
026558	TZ001	412M	4	･･･
026559	TH001	591M	80	･･･
026560	TT001	591L	10	･･･

```
SELECT ZAIKO.SCODE,SNAME,COL,DNO
       FROM ZAIKO LEFT OUTER JOIN JUTYU
            ON ZAIKO.SCODE=JUTYU.SCODE
  WHERE ZSURYO<100
```

検索結果

SCODE	SNAME	COL	DNO
商品コード	商品名	色	伝票番号
101L	ブラウス	青	
101M	ブラウス	白	026551
201M	ポロシャツ	白	
202M	ポロシャツ	赤	026556
302S	スカート	白	
353L	スカート	赤	
353M	スカート	緑	026553
411M	セーター	青	026554
411M	セーター	青	026557
412M	セーター	赤	026558
591M	ソックス	青	026552
591M	ソックス	青	026555
591M	ソックス	青	026559

注　受注のない商品の伝票番号はナル値になります。

3 表以上のアウタジョインの例として，在庫表，受注表，先月受注表を外結合し，全商品の商品名，単価と，単価が 5,000 円以上の商品の今月の受注量と，先月の受注量を検索する例を次の図に示します。

図 2-39　3 表以上のアウタジョインの例

ZAIKO（在庫表）

SCODE	SNAME	COL	TANKA	ZSURYO
商品コード	商品名	色	単価	在庫量
101L	ブラウス	青	3500	62
101M	ブラウス	白	3500	85
201M	ポロシャツ	白	3640	29
202M	ポロシャツ	赤	3640	67
302S	スカート	白	5110	65
353L	スカート	赤	4760	18
353M	スカート	緑	4760	56
411M	セーター	青	8400	12
412M	セーター	赤	8400	22
591L	ソックス	赤	250	300
591M	ソックス	青	250	90
591S	ソックス	白	250	280

JUTYU（受注表）

···	SCODE	JSURYO	···
···	商品コード	受注量	···
···	101M	10	···
···	591M	25	···
···	353M	8	···
···	411M	6	···
···	591M	30	···
···	202M	10	···
···	411M	5	···
···	412M	4	···
···	591M	80	···
···	591L	10	···

SENJUTYU（先月受注表）

···	SCODE	SENRYO	···
···	商品コード	先月受注量	···
···	101M	10	···
···	591M	20	···
···	591L	40	···
···	353L	6	···
···	591M	30	···
···	202M	20	···
···	302S	8	···
···	412M	4	···

```
SELECT A.SCODE,A.SNAME,A.TANKA,B.JSURYO,C.SENRYO
    FROM ZAIKO A LEFT OUTER JOIN JUTYU B ON A.SCODE=B.SCODE
                AND A.TANKA>=5000
            LEFT OUTER JOIN SENJUTYU C ON A.SCODE=C.SCODE
              AND A.TANKA>=5000
```

検索結果

SCODE	SNAME	TANKA	JSURYO	SENRYO
商品コード	商品名	単価	受注量	先月受注量
101L	ブラウス	3500		
101M	ブラウス	3500		
201M	ポロシャツ	3640		
202M	ポロシャツ	3640		
302S	スカート	5110		8
353L	スカート	4760		
353M	スカート	4760		
411M	セーター	8400	6	
411M	セーター	8400	5	
412M	セーター	8400	4	4
591L	ソックス	250		
591M	ソックス	250		
591S	ソックス	250		

注　受注がないか，又は単価が5,000より小さい商品の受注量（JSRYO，SENRYO）は，ナル値になります。

2.11　ビュー表の定義と操作

2.11.1　ビュー表の定義例及び操作例

表から特定の列や行を見るためにビュー表を定義すると，操作する範囲を限定できます。

ここでは，次の図に示す在庫表と売上表を使用してビュー表の定義，及び操作について説明します。

図 2–40　ビュー表の操作の説明で使用する表

ZAIKO(在庫表)

SCODE	SNAME	COL	TANKA	ZSURYO
商品コード	商品名	色	単価	在庫量
101L	ブラウス	青	3500	62
101M	ブラウス	白	3500	85
201M	ポロシャツ	白	3640	29
202M	ポロシャツ	赤	3640	67
302S	スカート	白	5110	65
353L	スカート	赤	4760	18
353M	スカート	緑	4760	56
411M	セーター	青	8400	12
412M	セーター	赤	8400	22
591L	ソックス	赤	250	300
591M	ソックス	青	250	90
591S	ソックス	白	250	280

URIAGE(売上表)

SCODE	SHITEN	USURYO	URIAGE
商品コード	支店名	売上数量	売上高
101M	関西支店	5	17500
202M	九州支店	10	36400
202M	関西支店	3	10920
302S	中部支店	5	25550
411M	東北支店	2	16800
591M	北海道支店	8	2000

(1)　ビュー表の定義

ビュー表の定義例として次の五つを示します。

- 検索する列を限定したビュー表の定義
- 探索条件を使用したビュー表の定義
- 読み込み専用のビュー表の定義
- 重複排除したビュー表の定義
- ビュー表からビュー表の定義

(a)　検索する列を限定したビュー表の定義

検索する列を限定するビュー表の定義例として，在庫表を基に色以外の列を検索できるビュー表（V1）として定義する例を次の図に示します。

図 2–41　検索する列を限定するビュー表の定義の例

ZAIKO（在庫表）

SCODE	SNAME	COL	TANKA	ZSURYO
商品コード	商品名	色	単価	在庫量
101L	ブラウス	青	3500	62
101M	ブラウス	白	3500	85
201M	ポロシャツ	白	3640	29
202M	ポロシャツ	赤	3640	67
302S	スカート	白	5110	65
353L	スカート	赤	4760	18
353M	スカート	緑	4760	56
411M	セーター	青	8400	12
412M	セーター	赤	8400	22
591L	ソックス	赤	250	300
591M	ソックス	青	250	90
591S	ソックス	白	250	280

```
CREATE VIEW V1(SCODE, SNAME, TANKA, ZSURYO)
  AS SELECT SCODE, SNAME, TANKA, ZSURYO
  FROM ZAIKO
```

V1

SCODE	SNAME	TANKA	ZSURYO
商品コード	商品名	単価	在庫量
101L	ブラウス	3500	62
101M	ブラウス	3500	85
201M	ポロシャツ	3640	29
202M	ポロシャツ	3640	67
302S	スカート	5110	65
353L	スカート	4760	18
353M	スカート	4760	56
411M	セーター	8400	12
412M	セーター	8400	22
591L	ソックス	250	300
591M	ソックス	250	90
591S	ソックス	250	280

(b)　探索条件を使用したビュー表の定義

探索条件を使用したビュー表の定義例として，在庫表と売上表を基に支店ごとの売上数量が 10 未満の商品名を求める問合せをビュー表（V2）として定義する例を次の図に示します。

図 2-42　探索条件を使用したビュー表の定義の例

ZAIKO(在庫表)

SCODE	SNAME	COL	TAANKA	ZSURYO
商品コード	商品名	色	単価	在庫量
101L	ブラウス	青	3500	62
101M	ブラウス	白	3500	85
201M	ポロシャツ	白	3640	29
202M	ポロシャツ	赤	3640	67
302S	スカート	白	5110	65
353L	スカート	赤	4760	18
353M	スカート	緑	4760	56
411M	セーター	青	8400	12
412M	セーター	赤	8400	22
591L	ソックス	赤	250	300
591M	ソックス	青	250	90
591S	ソックス	白	250	280

URIAGE(売上表)

SCODE	SHITEN	USURYO	URIAGE
商品コード	支店名	売上数量	売上高
101M	関西支店	5	17500
202M	九州支店	10	36400
202M	関西支店	3	10920
302S	中部支店	5	25550
411M	東北支店	2	16800
591M	北海道支店	8	2000

```
CREATE VIEW V2
  AS SELECT SNAME,SHITEN,URIAGE FROM ZAIKO,URIAGE
  WHERE ZAIKO.SCODE=URIAGE.SCODE AND USURYO<10
```

V2

SNAME	SHITEN	URIAGE
商品名	支店名	売上高
ブラウス	関西支店	17500
ポロシャツ	関西支店	10920
スカート	中部支店	25550
セーター	東北支店	16800
ソックス	北海道支店	2000

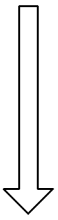
(c)　読み込み専用のビュー表の定義

読み込み専用のビュー表の定義例として，在庫表を基に商品名ごとに平均単価より高い商品の商品コード，商品名，単価，及び在庫量を求める問合せを読み込み専用のビュー表（V3）として定義する例を次の図に示します。

図 2-43　読み込み専用のビュー表の定義の例

ZAIKO（在庫表）

SCODE	SNAME	COL	TANKA	ZSURYO
商品コード	商品名	色	単価	在庫量
101L	ブラウス	青	3500	62
101M	ブラウス	白	3500	85
201M	ポロシャツ	白	3640	29
202M	ポロシャツ	赤	3640	67
302S	スカート	白	5110	65
353L	スカート	赤	4760	18
353M	スカート	緑	4760	56
411M	セーター	青	8400	12
412M	セーター	赤	8400	22
591L	ソックス	赤	250	300
591M	ソックス	青	250	90
591S	ソックス	白	250	280



```
CREATE READ ONLY VIEW V3
  AS SELECT SCODE,SNAME,TANKA,ZSURYO
  FROM ZAIKO X
  WHERE TANKA>(SELECT AVG(TANKA)
               FROM ZAIKO Y)
```

V3

SCODE	SNAME	TANKA	ZSURYO
商品コード	商品名	単価	在庫量
302S	スカート	5110	65
353L	スカート	4760	18
353M	スカート	4760	56
411M	セーター	8400	12
412M	セーター	8400	22

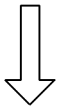
(d)　重複を排除したビュー表の定義

重複を排除したビュー表の定義例として，在庫表を基に商品名，及び単価を重複排除したビュー表（V4）として定義する例を次の図に示します。

図 2-44　重複を排除したビュー表の定義の例

ZAIKO（在庫表）

SCODE	SNAME	COL	TANKA	ZSURYO
商品コード	商品名	色	単価	在庫量
101L	ブラウス	青	3500	62
101M	ブラウス	白	3500	85
201M	ポロシャツ	白	3640	29
202M	ポロシャツ	赤	3640	67
302S	スカート	白	5110	65
353L	スカート	赤	4760	18
353M	スカート	緑	4760	56
411M	セーター	青	8400	12
412M	セーター	赤	8400	22
591L	ソックス	赤	250	300
591M	ソックス	青	250	90
591S	ソックス	白	250	280



```
CREATE VIEW V4(SNAME,TANKA)
  AS SELECT DISTINCT SNAME,TANKA
  FROM ZAIKO
```

V4

SNAME	TANKA
商品名	単価
ブラウス	3500
ポロシャツ	3640
スカート	5110
スカート	4760
セーター	8400
ソックス	250

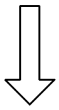
(e)　ビュー表からビュー表の定義

ビュー表からビュー表の定義例として，（a）で定義したビュー表（V1）から，商品名が"スカート"の行を求める問合せをビュー表（V5）として定義する例を次の図に示します。

図 2-45　ビュー表からビュー表の定義の例

V1

SCODE	SNAME	TANKA	ZSURYO
商品コード	商品名	単価	在庫量
101L	ブラウス	3500	62
101M	ブラウス	3500	85
201M	ポロシャツ	3640	29
202M	ポロシャツ	3640	67
302S	スカート	5110	65
353L	スカート	4760	18
353M	スカート	4760	56
411M	セーター	8400	12
412M	セーター	8400	22
591L	ソックス	250	300
591M	ソックス	250	90
591S	ソックス	250	280



```
CREATE VIEW V5
  AS SELECT * FROM V1
  WHERE SNAME=N'スカート'
```

V5

SCODE	SNAME	TANKA	ZSURYO
商品コード	商品名	単価	在庫量
302S	スカート	5110	65
353L	スカート	4760	18
353M	スカート	4760	56

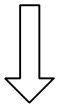
(2)　ビュー表の操作

ビュー表の操作例として，「探索条件を使用したビュー表の定義」で定義したビュー表（V2）から，売上高が最高の商品の商品名，支店名，及び売上高を検索（副問合せを指定した SQL 文中でビュー表を指定）する例を次の図に示します。

図 2-46　ビュー表の操作例

V2

SNAME	SHITEN	URIAGE
商品名	支店名	売上高
ブラウス	関西支店	17500
ポロシャツ	関西支店	10920
スカート	中部支店	25550
セーター	東北支店	16800
ソックス	北海道支店	2000



```
SELECT * FROM V2
  WHERE URIAGE=
      (SELECT MAX(URIAGE) FROM V2)
```

検索結果

SNAME	SHITEN	URIAGE
商品名	支店名	売上高
スカート	中部支店	25550

2.12　抽象データ型を含む表のデータ操作

抽象データ型がある表に対して操作する場合，関数又はコンポネント指定を用います。関数には，抽象データ型を定義したときに自動的に作成されるコンストラクタ関数（又はデフォルトコンストラクタ関数），及びユーザが任意に定義したユーザ定義型関数があります。また，コンポネント指定は，抽象データ型を構成する属性に対して操作するものです。

なお，抽象データ型には，プラグインが提供するものと，ユーザが定義するものがあります。プラグインが提供する抽象データ型として，ここでは SGMLTEXT 型と XML 型を使用した場合の例について説明します。

2.12.1　SGMLTEXT 型の場合

ここでは，全文検索プラグイン（HiRDB Text Search Plug-in）を使用した例について説明します。HiRDB Text Search Plug-in が提供する抽象データ型関数を次に示します。なお，プラグインが提供する抽象データ型関数については，各プラグインマニュアルを参照してください。

関数名	説明
SGMLTEXT	SGML 文書登録
contains	構造指定検索
contains_with_score，score	スコア検索

この項では，薬品の取扱い説明書を SGML 文書で管理する例について説明します。

例題で使用している表は，マニュアル「HiRDB システム導入・設計ガイド」のデータベースの作成（プラグインが提供する抽象データ型を含む表の場合）で定義している表を利用しています。

(1)　検索

(a)　SGMLTEXT 型の場合の検索例（その 1）

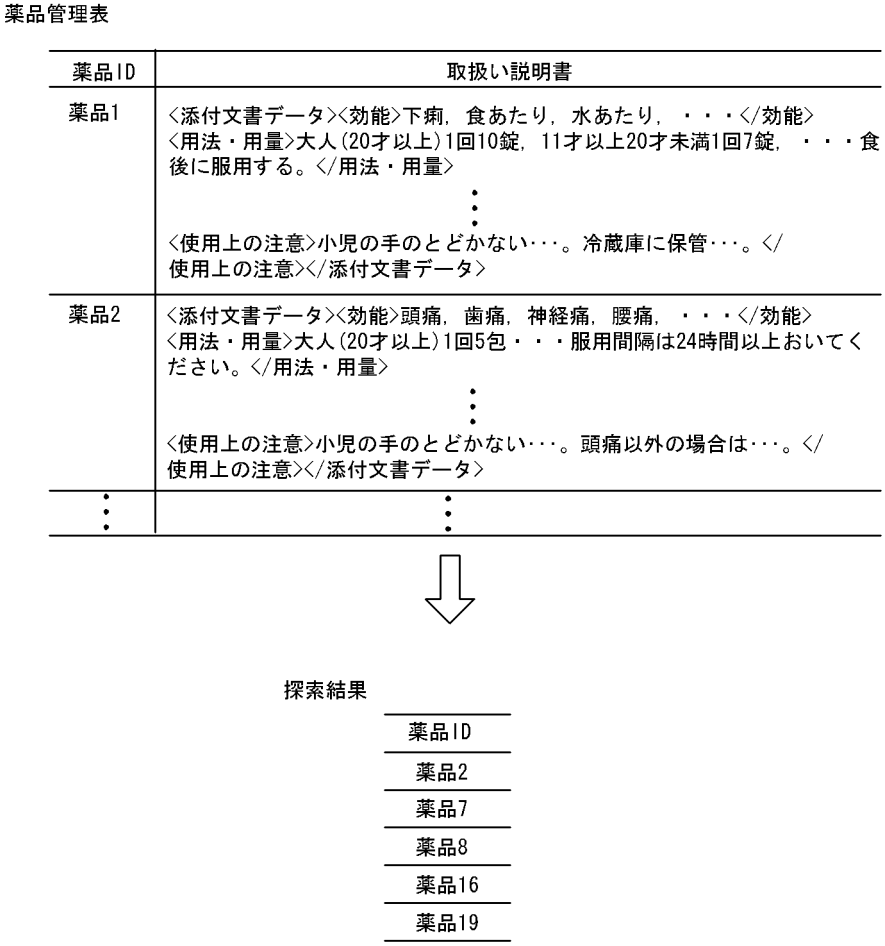
SGMLTEXT 型の場合の検索例として，頭痛に効く薬品を調べる例を次の図に示します。検索をする SQL 文は，次のように記述できます。

```
SELECT 薬品ID FROM 薬品管理表
  WHERE contains(取扱い説明書,'添付文書データ[効能{"頭痛"}]')
    IS TRUE
```

[説明]

この例では，抽象データ型関数 contains を使用して，列「取扱い説明書」の効能という構造部分に"頭痛"の文字列を含んでいる薬品を検索しています。

図 2–47　SGMLTEXT 型の場合の検索例（その 1）



(b)　SGMLTEXT 型の場合の検索例（その 2）

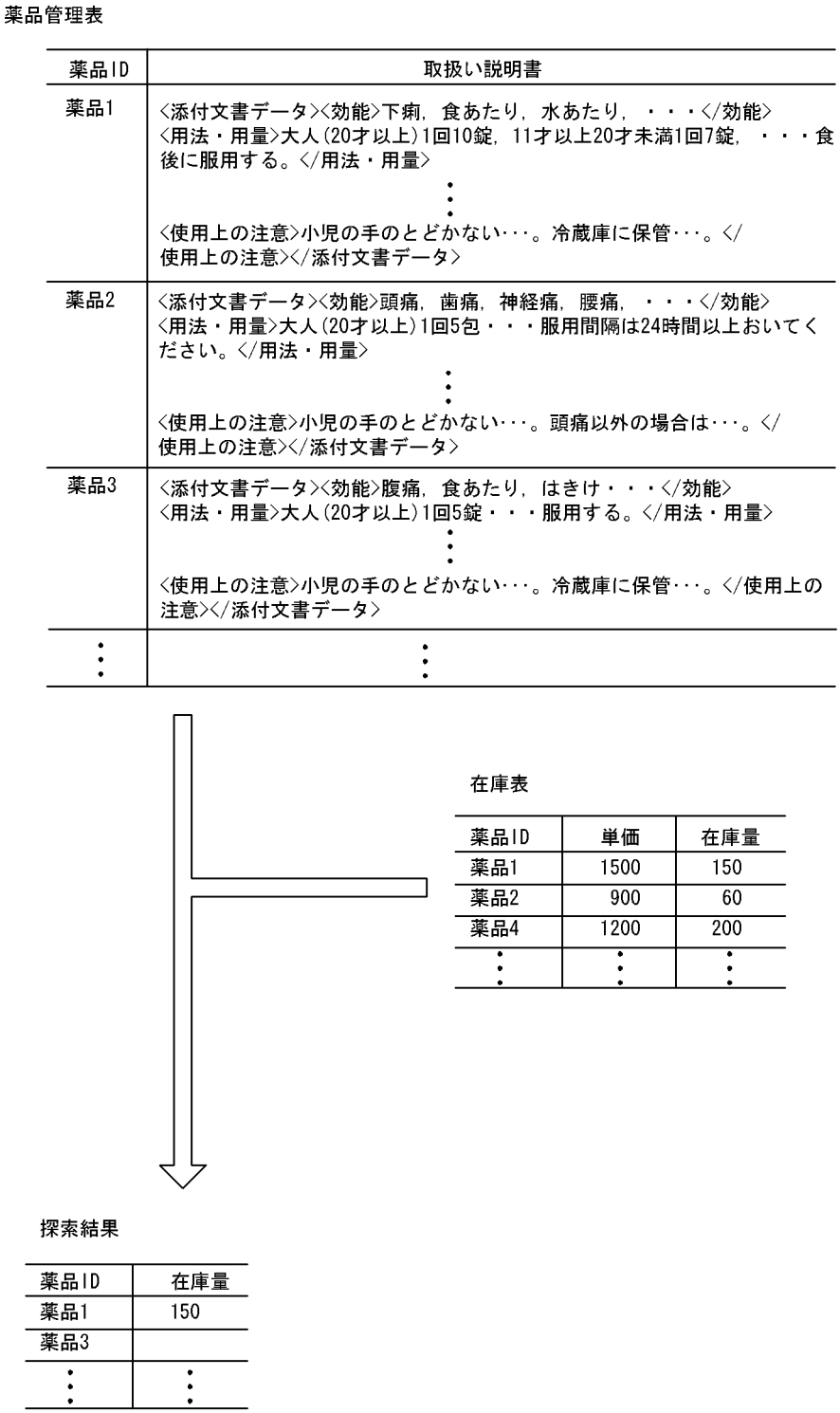
SGMLTEXT 型の場合の検索例として，食あたりに効く薬品の薬品 ID と在庫量を求める例を次の図に示します。検索をする SQL 文は，次のように記述できます。

```
SELECT 薬品管理表.薬品ID,在庫量
  FROM 薬品管理表 LEFT OUTER JOIN 在庫表
    ON 薬品管理表.薬品ID=在庫表.薬品ID
    WHERE contains(取扱い説明書,'添付文書データ[効能{"食あたり"}]')
        IS TRUE
```

[説明]

この例では，薬品管理表と在庫表を外結合して検索しています。抽象データ型関数 contains を使用して，列「取扱い説明書」の効能という構造部分に"食あたり"の文字列を含んでいる薬品 ID を検索して，その薬品 ID の在庫量を求めています。

図 2–48　SGMLTEXT 型の場合の検索例（その 2）



(2)　更新

SGMLTEXT 型の場合の更新例として，薬品 2 の取扱い説明書を更新する例を次の図に示します。更新をする SQL 文は，次のように記述できます。

```
UPDATE 薬品管理表 SET 取扱い説明書 = SGMLTEXT(:sgml AS BLOB(1M))
  WHERE 薬品ID = '薬品2'
```

[説明]

この例では，抽象データ型関数 SGMLTEXT を使用して，薬品 2 の取扱い説明書のデータを更新しています。

なお，UPDATE 文の前に，あらかじめ次の BLOB 型の埋込み変数「sgml」を定義しているものとします。

```
EXEC SQL BEGIN DECLARE SECTION;     1.
   SQL TYPE IS BLOB(300K) sgml;     1.
EXEC SQL END DECLARE SECTION;       1.
strcpy(sgml.sgml_data,char_ptr_pointing_to_a_sgml_text);         2.
sgml.sgml_length = strlen(char_ptr_pointing_to_a_sgml_text);     3.

 [説明]
```

1. BLOB 型の埋込み変数「sgml」を定義します。
2. 埋込み変数「sgml」に，更新する新しいデータを格納します。
3. 作成した BLOB データの属性値 sgml_length を，格納したデータの長さにセットします。

図 2–49　SGMLTEXT 型の場合の更新例

薬品管理表

薬品ID	取扱い説明書
薬品1	<添付文書データ><効能>下痢，食あたり，水あたり，・・・</効能> <用法・用量>大人(20才以上)1回10錠，11才以上20才未満1回7錠，・・・食後に服用する。</用法・用量> ⋮ <使用上の注意>小児の手のとどかない･･･。冷蔵庫に保管･･･。</使用上の注意></添付文書データ>
薬品2	<添付文書データ><効能>頭痛，歯痛，神経痛，腰痛，・・・</効能> <用法・用量>大人(20才以上)1回5包・・・服用間隔は24時間以上おいてください。</用法・用量> ⋮ <使用上の注意>小児の手のとどかない･･･。頭痛以外の場合は･･･。</使用上の注意></添付文書データ>
⋮	⋮



更新結果

薬品ID	取扱い説明書
薬品1	<添付文書データ><効能>下痢，食あたり，水あたり，・・・</効能> <用法・用量>大人(20才以上)1回10錠，11才以上20才未満1回7錠，・・・食後に服用する。</用法・用量> ⋮ <使用上の注意>小児の手のとどかない･･･。冷蔵庫に保管･･･。</使用上の注意></添付文書データ>
薬品2	<添付文書データ><効能>胃痛，胸やけ，飲み過ぎ，・・・</効能> <用法・用量>次の量を1日5回服用してください。20才以上　1包，11～19才　1/3包，　・・・</用法・用量> ⋮ <使用上の注意>お子さまの手のとどかない・・・</使用上の注意></添付文書データ> ← 更新したデータ
⋮	⋮

(3)　削除

SGMLTEXT 型の場合の削除例として，薬品 2 を削除する例を次の図に示します。行を削除する SQL 文は，次のように記述できます。

```
DELETE FROM 薬品管理表
  WHERE 薬品ID = '薬品2'
```

[説明]

この例では，薬品管理表から薬品 2 の行を削除しています。

図 2–50　SGMLTEXT 型の場合の削除例

薬品管理表

薬品ID	取扱い説明書
薬品1	<添付文書データ><効能>下痢，食あたり，水あたり，・・・</効能> <用法・用量>大人(20才以上)1回10錠，11才以上20才未満1回7錠，・・・食後に服用する。</用法・用量> ⋮ <使用上の注意>小児の手のとどかない･･･。冷蔵庫に保管･･･。</使用上の注意></添付文書データ>
薬品2	<添付文書データ><効能>頭痛，歯痛，神経痛，腰痛，・・・</効能> <用法・用量>大人(20才以上)1回5包・・・服用間隔は24時間以上おいてください。</用法・用量> ⋮ <使用上の注意>小児の手のとどかない･･･。頭痛以外の場合は･･･。</使用上の注意></添付文書データ>
⋮	⋮

⇩

削除結果

薬品ID	取扱い説明書
薬品1	<添付文書データ><効能>下痢，食あたり，水あたり，・・・</効能> <用法・用量>大人(20才以上)1回10錠，11才以上20才未満1回7錠，・・・食後に服用する。</用法・用量> ⋮ <使用上の注意>小児の手のとどかない･･･。冷蔵庫に保管･･･。</使用上の注意></添付文書データ>
薬品3	<添付文書データ><効能>かぜ，かぜと思われる症状，・・・</効能> <用法・用量>1日5回，食後なるべく5分以内に　･･･　1回の量は，成人(20才以上)は30錠，10才～19才は15錠です。</用法・用量> ⋮ <使用上の注意>日の当たらないところに･･･。服用後は車等の･･･。</使用上の注意></添付文書データ>
⋮	⋮

(4)　挿入

SGMLTEXT 型の場合の挿入例として，薬品 25 の行を挿入する例を次の図に示します。行を挿入する SQL 文は，次のように記述できます。

```
INSERT INTO 薬品管理表(薬品ID,取扱い説明書)
  VALUES(薬品25,SGMLTEXT(:sgml AS BLOB(1M)))
```

[説明]

この例では，抽象データ型関数 SGMLTEXT を使用して，薬品管理表に薬品 25 の行を追加しています。

なお，INSERT 文の前に，あらかじめ次の BLOB 型の埋込み変数「sgml」を定義しているものとします。

```
EXEC SQL BEGIN DECLARE SECTION;     1.
   SQL TYPE IS BLOB(300K) sgml;     1.
EXEC SQL END DECLARE SECTION;       1.
strcpy(sgml.sgml_data,char_ptr_pointing_to_a_sgml_text);         2.
sgml.sgml_length = strlen(char_ptr_pointing_to_a_sgml_text);     3.

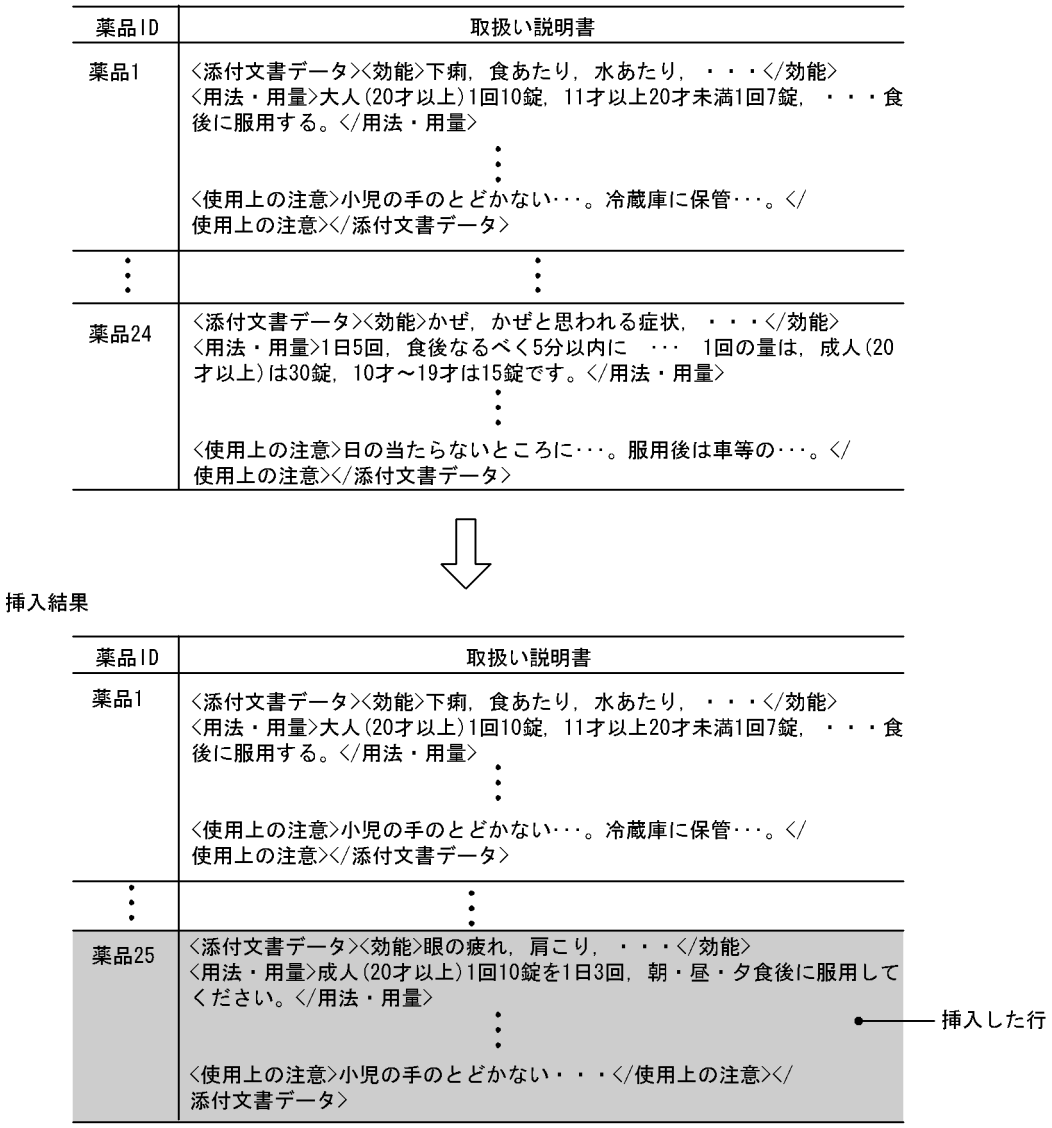
 [説明]
```

1. BLOB 型の埋込み変数「sgml」を定義します。

2. 埋込み変数「sgml」に，挿入するデータを格納します。

3. 作成した BLOB データの属性値 sgml_length を，格納したデータの長さにセットします。

図 2–51　SGMLTEXT 型の場合の挿入例



2.12.2　XML 型の場合

ここでは，HiRDB XML Extension を使用した例について説明します。HiRDB XML Extension が提供する抽象データ型関数については，マニュアル「HiRDB SQL リファレンス」を参照してください。

この項では，書籍情報を XML 文書で管理する例について説明します。

例題で使用している表は，マニュアル「HiRDB システム導入・設計ガイド」のデータベースの作成（プラグインが提供する抽象データ型を含む表の場合）で定義している表を利用しています。

(1)　検索

(a)　XML 型の場合の検索例（その１）

XML 型の場合の検索例として，書籍 ID が 126513592 である書籍情報を VARCHAR 型の値として取り出す例を次の図に示します。検索をする SQL 文は，次のように記述できます。

```
SELECT 書籍ID, XMLSERIALIZE(書籍情報 AS VARCHAR(32000))
    FROM 書籍管理表
    WHERE 書籍ID = 126513592
```

図 2–52　XML 型の場合の検索例（その 1）

書籍管理表

書籍ID	書籍情報
452469630	<書籍情報 書籍ID="452469630"> 　<カテゴリ>データベース</カテゴリ> 　<タイトル>リレーショナルデータベース解説</タイトル> 　<説明>リレーショナルモデルの概念から，それに基づくRDBMSの仕組みを解説している。</説明> </書籍情報>
126513592	<書籍情報 書籍ID="126513592"> 　<カテゴリ>データベース</カテゴリ> 　<タイトル>SQL徹底解説</タイトル> 　<説明>最新の標準SQL言語について詳細に解説している。</説明> </書籍情報>
940123531	<書籍情報 書籍ID="940123531"> 　<カテゴリ>ネットワーク</カテゴリ> 　<タイトル>図解TCP/IP</タイトル> 　<説明>TCP/IPプロトコルについて図を用いながら分かりやすく説明している。</説明> </書籍情報>
⋮	⋮

検索結果

書籍ID	書籍情報
126513592	<書籍情報 書籍ID="126513592"> 　<カテゴリ>データベース</カテゴリ> 　<タイトル>SQL徹底解説</タイトル> 　<説明>最新の標準SQL言語について詳細に解説している。</説明> </書籍情報>

(b)　XML 型の場合の検索例（その 2）

XQuery 式による評価結果の取り出しをします。XML 型の場合の検索例として，カテゴリが"データベース"である書籍のタイトルを取り出す例を次の図に示します。検索をする SQL 文は，次のように記述できます。

```
SELECT 書籍ID,
       XMLSERIALIZE(
         XMLQUERY('/書籍情報/タイトル'
                  PASSING BY VALUE 書籍情報
                  RETURNING SEQUENCE EMPTY ON EMPTY)
         AS VARCHAR(32000))
  FROM 書籍管理表
  WHERE XMLEXISTS('/書籍情報[カテゴリ="データベース"]'
                  PASSING BY VALUE 書籍情報)
```

[説明]

XMLQUERY 関数を使用して，XQuery 式を評価した結果を取り出します。XQuery 式の評価結果が空のシーケンスである行を出力しないように，XMLEXISTS 述語を使用します。

図 2–53　XML 型の場合の検索例（その 2）

書籍管理表

書籍ID	書籍情報
452469630	<書籍情報 書籍ID="452469630"> 　<カテゴリ>データベース</カテゴリ> 　<タイトル>リレーショナルデータベース解説</タイトル> 　<説明>リレーショナルモデルの概念から，それに基づくRDBMSの仕組みを解説している。</説明> </書籍情報>
126513592	<書籍情報 書籍ID="126513592"> 　<カテゴリ>データベース</カテゴリ> 　<タイトル>SQL徹底解説</タイトル> 　<説明>最新の標準SQL言語について詳細に解説している。</説明> </書籍情報>
940123531	<書籍情報 書籍ID="940123531"> 　<カテゴリ>ネットワーク</カテゴリ> 　<タイトル>図解TCP/IP</タイトル> 　<説明>TCP/IPプロトコルについて図を用いながら分かりやすく説明している。</説明> </書籍情報>
⋮	⋮

検索結果

書籍ID	書籍情報
452469630	<タイトル>リレーショナルデータベース解説</タイトル>
126513592	<タイトル>SQL徹底解説</タイトル>

(c)　XML 型の場合の検索例（その 3）

XML 型の値を一つの XML 型の値として出力します。XML 型の場合の検索例として，カテゴリが"データベース"である書籍のタイトルを結合して取り出す例を次の図に示します。検索をする SQL 文は，次のように記述できます。

```
SELECT XMLSERIALIZE(
         XMLAGG(
           XMLQUERY('/書籍情報/タイトル'
                     PASSING BY VALUE 書籍情報
                     RETURNING SEQUENCE EMPTY ON EMPTY)
         )
         AS VARCHAR(32000))
       FROM 書籍管理表
       WHERE XMLEXISTS('/書籍情報[カテゴリ="データベース"]'
                        PASSING BY VALUE 書籍情報)
```

[説明]

各行の XML 型の値を一つの XML 型の値として出力するには，XMLAGG 集合関数を使用します。

図 2–54　XML 型の場合の検索例（その 3）

書籍管理表

書籍ID	書籍情報
452469630	<書籍情報 書籍ID="452469630"> 　<カテゴリ>データベース</カテゴリ> 　<タイトル>リレーショナルデータベース解説</タイトル> 　<説明>リレーショナルモデルの概念から，それに基づくRDBMSの仕組みを解説している。</説明> </書籍情報>
126513592	<書籍情報 書籍ID="126513592"> 　<カテゴリ>データベース</カテゴリ> 　<タイトル>SQL徹底解説</タイトル> 　<説明>最新の標準SQL言語について詳細に解説している。</説明> </書籍情報>
940123531	<書籍情報 書籍ID="940123531"> 　<カテゴリ>ネットワーク</カテゴリ> 　<タイトル>図解TCP/IP</タイトル> 　<説明>TCP/IPプロトコルについて図を用いながら分かりやすく説明している。</説明> </書籍情報>
⋮	⋮

検索結果

書籍情報
<タイトル>リレーショナルデータベース解説</タイトル> <タイトル>SQL徹底解説</タイトル>

(d)　XML 型の場合の検索例（その 4）

XML 型の値を一つの XML 型の値として，その値に対して XQuery 式を評価します。XML 型の場合の検索例として，タイトルが"SQL 徹底解説"の書籍情報とカテゴリが同じ書籍の書籍情報を取り出す例を次の図に示します。検索をする SQL 文は，次のように記述できます。

```
SELECT
  XMLSERIALIZE(
    XMLQUERY(
      '$BOOKS/書籍情報[カテゴリ=$BOOKS/書籍情報[タイトル="SQL徹底解説"]/カテゴリ]'
      PASSING BY VALUE XMLAGG(書籍情報) AS BOOKS
      RETURNING SEQUENCE EMPTY ON EMPTY))
    AS VARCHAR(32000))
  FROM 書籍管理表
```

図 2-55　XML 型の場合の検索例（その 4）

書籍管理表

書籍ID	書籍情報
452469630	<書籍情報 書籍ID="452469630"> 　<カテゴリ>データベース</カテゴリ> 　<タイトル>リレーショナルデータベース解説</タイトル> 　<説明>リレーショナルモデルの概念から，それに基づくRDBMSの仕組みを解説している。</説明> </書籍情報>
126513592	<書籍情報 書籍ID="126513592"> 　<カテゴリ>データベース</カテゴリ> 　<タイトル>SQL徹底解説</タイトル> 　<説明>最新の標準SQL言語について詳細に解説している。</説明> </書籍情報>
940123531	<書籍情報 書籍ID="940123531"> 　<カテゴリ>ネットワーク</カテゴリ> 　<タイトル>図解TCP/IP</タイトル> 　<説明>TCP/IPプロトコルについて図を用いながら分かりやすく説明している。</説明> </書籍情報>
⋮	⋮

検索結果

書籍情報
<書籍情報 書籍ID="126513592"> 　<カテゴリ>データベース</カテゴリ> 　<タイトル>SQL徹底解説</タイトル> 　<説明>最新の標準SQL言語について詳細に解説している。</説明> </書籍情報> <書籍情報 書籍ID="452469630"> 　<カテゴリ>データベース</カテゴリ> 　<タイトル>リレーショナルデータベース解説</タイトル> 　<説明>リレーショナルモデルの概念から，それに基づくRDBMSの仕組みを解説している。</説明> </書籍情報>

(e)　XML 型の場合の検索例（その 5）

部分構造インデクスを使用した検索をします。書籍情報中のカテゴリ要素を VARCHAR 型としてキーにするインデクスの定義例を次に示します。

部分構造インデクスの定義例

```
CREATE INDEX INDX1 ON 書籍管理表(書籍情報)
  IN (RDAREA02) KEY FROM '/書籍情報/カテゴリ' AS VARCHAR(100)
```

このインデクスを利用することで，次の SQL の場合は行の絞り込みの処理時間が削減できます。書籍管理表から，カテゴリが"ネットワーク"である書籍情報を取り出す例を次に示します。

部分構造インデクスを使用した検索例

```
SELECT 書籍ID,
       XMLSERIALIZE(書籍情報 AS VARCHAR(32000))
  FROM 書籍管理表
  WHERE XMLEXISTS('/書籍情報[カテゴリ="ネットワーク"]'
                  PASSING BY VALUE 書籍情報)
```

(f)　XML 型の場合の検索例（その 6）

XML 型全文検索用インデクスを使用した検索をします。書籍情報列に対する XML 型全文検索用インデクスの定義例を次に示します。

XML 型全文検索用インデクスの定義例

```
CREATE INDEX INDX1
  USING TYPE IXXML ON 書籍管理表(書籍情報)
  IN (LOBAREA01)
```

このインデクスを利用することで，次の SQL の場合は行の絞り込みの処理時間が削減できます。書籍管理表から，説明に"RDBMS"を含む書籍情報を取り出す例を次に示します。

XML 型全文検索用インデクスを使用した検索例

```
SELECT 書籍ID,
       XMLSERIALIZE(書籍情報 AS VARCHAR(32000))
  FROM 書籍管理表
  WHERE XMLEXISTS('/書籍情報/説明/text()[contains( . ,"RDBMS")]'
                  PASSING BY VALUE 書籍情報)
```

2.12.3　ユーザが定義する抽象データ型の場合

ここでは，ユーザが定義する抽象データ型を含む表を操作する例について説明します。

なお，例題で使用している表は，マニュアル「HiRDB システム導入・設計ガイド」のデータベースの作成（ユーザが定義する抽象データ型を含む表の場合）で定義している表を利用しています。

(1)　抽象データ型がある表の検索

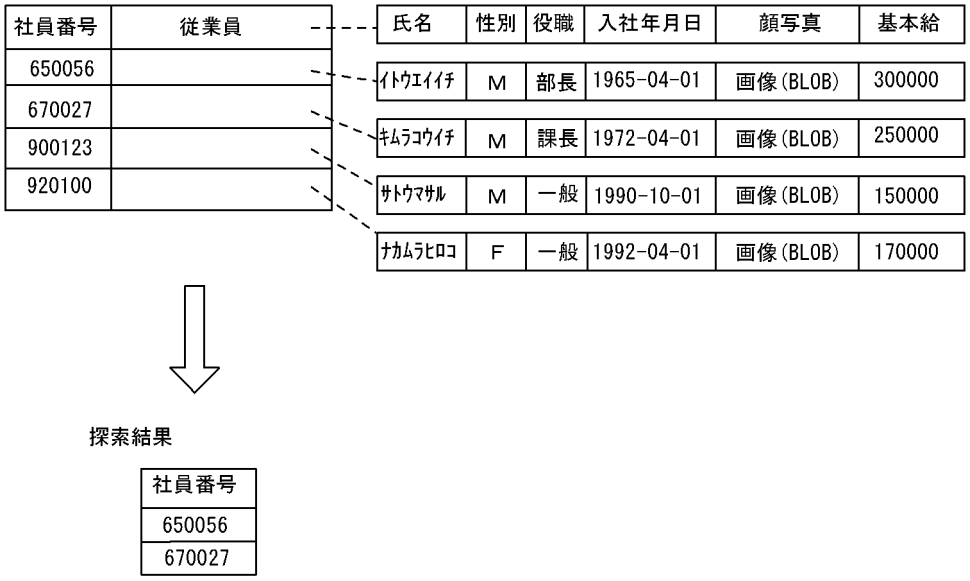
抽象データ型がある表の検索例として，勤続年数が 20 年以上の社員を求める例を次の図に示します。検索をする SQL 文は，次のように記述できます。

```
SELECT 社員番号
    FROM 社員表
    WHERE 勤続年数(従業員)>=20
```

[説明]

この例では，ユーザ定義関数「勤続年数」を使用して，勤続年数が 20 年以上の社員の社員番号を検索しています。ユーザ定義関数「勤続年数」の引数は，従業員です。

図 2–56　抽象データ型がある表の検索例



(2)　抽象データ型がある表の更新

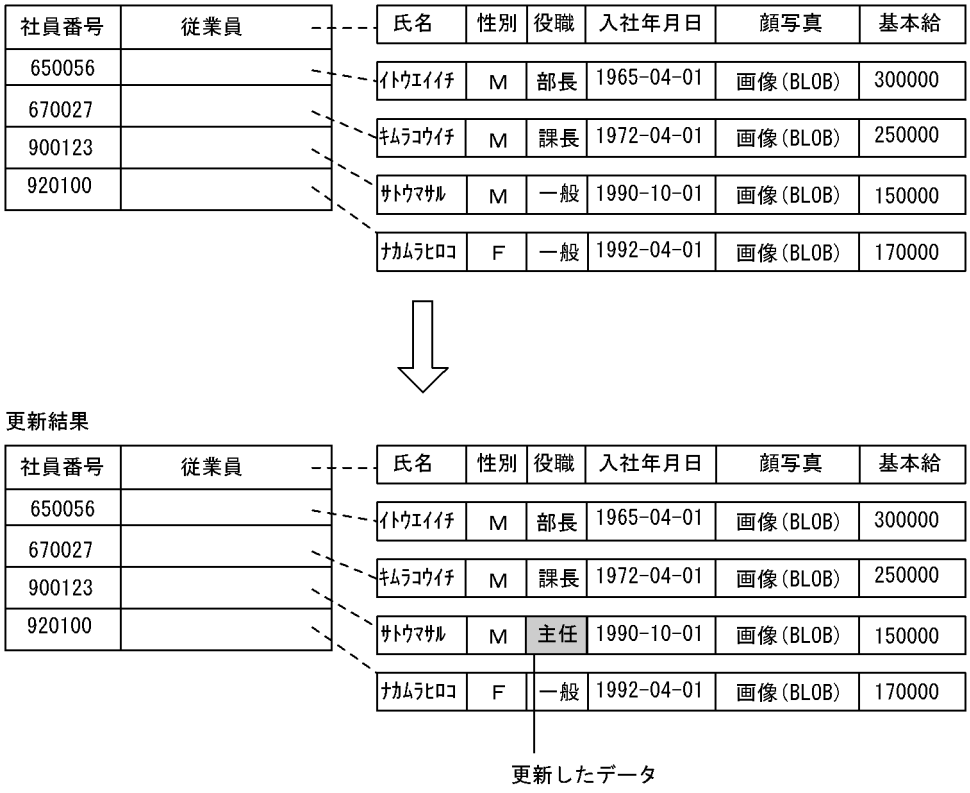
抽象データ型がある表の更新例として，社員番号が 900123 の社員の役職を主任に更新する例を次の図に示します。更新をする SQL 文は，次のように記述できます。

```
UPDATE 社員表
    SET 従業員..役職='主任'
    WHERE 社員番号='900123'
```

[説明]

この例では，社員番号が 900123 の社員の，列「従業員」の属性「役職」を主任に更新しています。抽象データ型の属性を指定する場合は，コンポネント指定を使用します。この例の場合，従業員..役職がコンポネント指定です。

図 2-57　抽象データ型がある表の更新例



(3)　抽象データ型がある表の行の削除

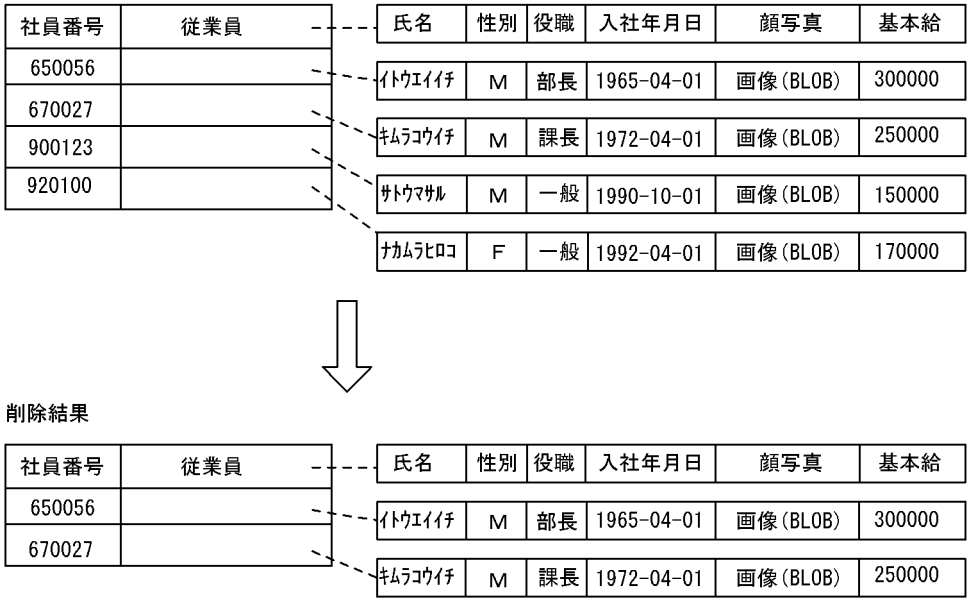
抽象データ型がある表の行の削除例として，役職が一般の社員のデータを削除する例を次の図に示します。行を削除する SQL 文は，次のように記述できます。

```
DELETE FROM 社員表
    WHERE 従業員..役職='一般'
```

[説明]

この例では，列「従業員」の属性「役職」が一般の社員の行を削除しています。抽象データ型の属性を指定する場合は，コンポネント指定を使用します。この例の場合，従業員..役職がコンポネント指定です。

図 2-58　抽象データ型がある表の行の削除例



(4)　抽象データ型がある表への行の挿入

抽象データ型がある表への行の挿入例として，社員表に行を挿入する例を次の図に示します。行を挿入する SQL 文は，次のように記述できます。

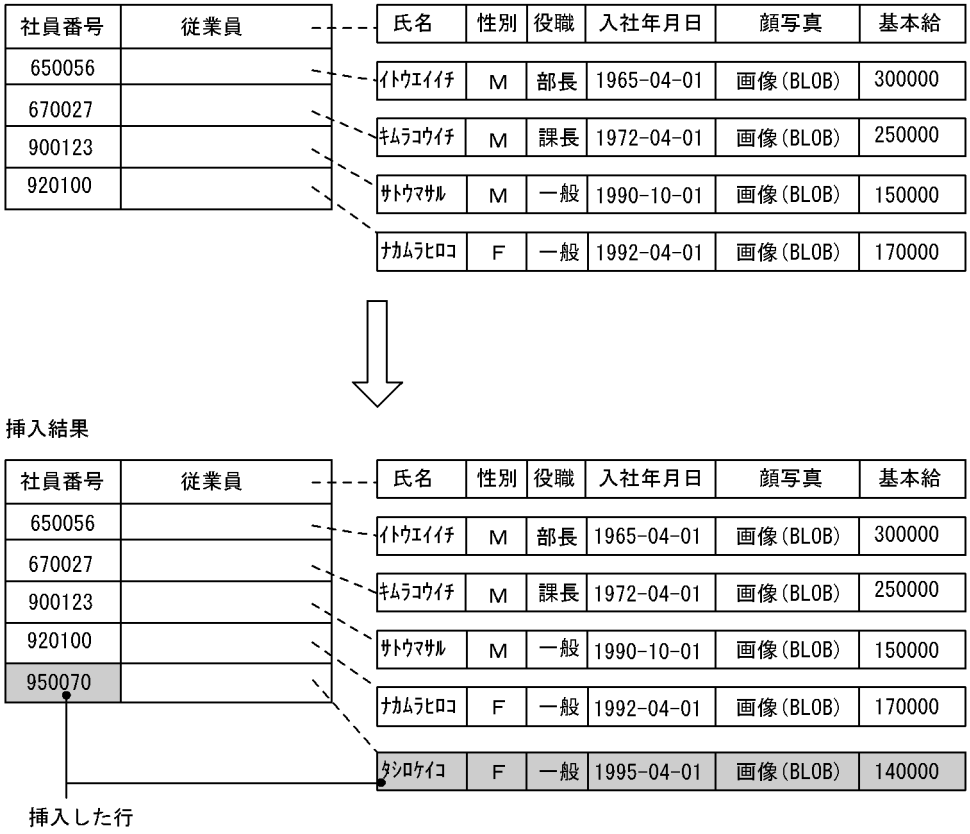
```
INSERT INTO 社員表
    VALUES ('950070',t_従業員('タシロケイコ',
                              'F',
                              '一般',
                              '1995-04-01',
                              :x顔写真 AS BLOB,
                              140000
                              )
            )
```

[説明]

この例では，抽象データ型定義時に定義したコンストラクタ関数 t_従業員を使用して，社員表に社員番号 950070 の行を挿入しています。

なお，:x 顔写真は BLOB 型の埋込み変数で，顔写真の画像が設定されているものとします。

図 2–59　抽象データ型がある表への行の挿入例



3 UAP の設計

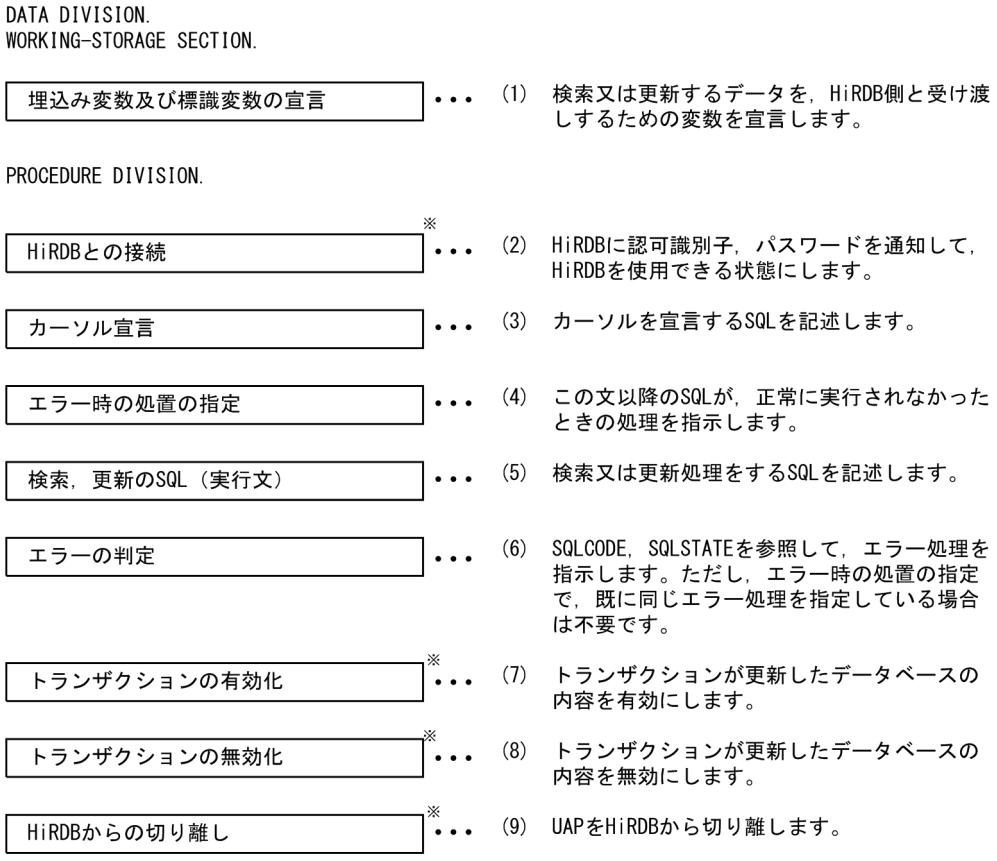
この章では，UAP を設計するときに考慮する基本的事項について説明します。

3.1　UAP 中での SQL の基本構成

3.1.1　UAP 中での SQL の基本構成の説明

UAP 中での SQL の基本構成を次の図に示します。なお，ここでは UAP を COBOL 言語で記述する場合について説明します。

図 3–1　UAP 中での SQL の基本構成



注　括弧付きの番号は，それぞれ以降の説明の番号と対応しています。
注※　必要に応じて，「エラー時の処置の指定」又は「エラーの判定」によるエラー処理を指定してください。ただし，「エラー時の処置の指定」によって，「トランザクションの無効化」が無限ループしないように注意してください。

(1)　埋込み変数及び標識変数の宣言

SQL で検索したデータを UAP 側で受け取ったり，逆に UAP 側のデータを SQL で表に挿入したりするには，両方の言語間の橋渡しをする変数が必要になります。このために，埋込み変数を使用します。また，ナル値を含むデータを受け渡す必要のあるときには，埋込み変数と併せて標識変数を使用します。

埋込み変数及び標識変数の宣言例を次に示します。

なお，SQL 中での埋込み変数及び標識変数の指定方法については，「検索，更新の SQL（実行文）」を参照してください。

```
EXEC SQL
  BEGIN DECLARE SECTION  ..........1
END-EXEC.
77 XUSERID    PIC X(7). ...........2
77 XPSWD      PIC X(7). ...........2
77 XSCODE     PIC X(4). ...........2
77 XSNAME     PIC N(8). ...........2
77 XGRYO      PIC S9(9) COMP. .....2
77 IGRYO      PIC S9(4) COMP. .....3
EXEC SQL
  END DECLARE SECTION  ............4
END-EXEC.
```

[説明]

1. 埋込み変数の宣言の始まりを示します。
2. 埋込み変数を宣言します。SQL と UAP 側でデータを受け渡しするときには，あらかじめ決められた規則に従って記述します。SQL のデータ型とデータ記述については，「SQL のデータ型とデータ記述」を参照してください。
3. 埋込み変数 xgryo に対する標識変数を宣言します。なお，BLOB 型の埋込み変数に対する標識変数の場合は，PIC S9(9) COMP.となります。
4. 埋込み変数の宣言の終わりを示します。

ナル値の既定値設定機能を使用している場合，検索結果がナル値のときはナル値の代わりに既定値（数データの場合は 0，文字データの場合は空白）を埋込み変数で受け取ることができます。この場合，既定値とナル値とを区別しなくていいときは，標識変数を使用する必要がなくなります。ナル値の既定値設定機能については，マニュアル「HiRDB SQL リファレンス」を参照してください。

(2)　HiRDB との接続

HiRDB にユーザの認可識別子及びパスワードを通知して，UAP が HiRDB を使用できる状態にします。これを HiRDB との接続といいます。HiRDB との接続方法を次に示します。

```
EXEC SQL
  CONNECT :XUSERID IDENTIFIED BY :XPSWD
END-EXEC.
```

[説明]

埋込み変数（:XUSERID）に格納された認可識別子，及び埋込み変数（:XPSWD）に格納されたパスワードで HiRDB と接続します。

(3)　カーソル宣言

UAP で複数行の検索結果を 1 行ずつ取り出すために，DECLARE CURSOR でカーソルを宣言します。宣言したカーソルを使用して，データの検索，更新，及び削除をします。また，カーソルをオープンする場合は OPEN 文，検索結果を取り出しカーソルを次の行へ進める場合は FETCH 文，カーソルをクローズする場合は CLOSE 文を使用します。

カーソル宣言中には，探索条件中の条件値として，埋込み変数及び標識変数を指定できます。これらを指定した場合，そのカーソルに対する OPEN 文の実行時に，それらの変数中の値が HiRDB に渡されます。

カーソルの詳細については，「カーソルの効果」を参照してください。

カーソルの宣言方法を次に示します。

```
EXEC SQL
  DECLARE CR1 CURSOR FOR SELECT SCODE,SNAME,GRYO FROM ZAIKO
END-EXEC.
```

［説明］

ZAIKO 表から SCODE，SNAME，GRYO を 1 行ずつ取り出すために，カーソル CR1 を宣言します。

(4)　エラー時の処置の指定

SQL 文の前に WHENEVER 文を指定しておくと，エラーが発生したかどうかを自動的に判定できます。WHENEVER 文の指定方法を次に示します。

(a)　エラーが発生した場合

```
EXEC SQL
  WHENEVER SQLERROR GO TO エラー処理
END-EXEC.
```

［説明］

WHENEVER SQLERROR：

エラーが発生したときの処置を宣言します。

GO TO エラー処理：

エラーが発生したとき，指定した節名又は段落名（エラー処理）へ処理を移します。この処理内から SQL 連絡領域を参照すれば，リターンコードと関連する情報を確認できます。

(b)　検索する行がなくなった場合

```
EXEC SQL
  WHENEVER NOT FOUND GO TO 検索終了処理
END-EXEC.
```

［説明］

WHENEVER NOT FOUND：

検索する行がなくなったときの処置を宣言します。

GO TO 検索終了処理：

検索する行がなくなったとき，指定した節名又は段落名（検索終了処理）へ処理を移します。

(c)　WHENEVER 文の有効範囲

WHENEVER 文は次に同じ種類の WHENEVER 文が現れるまで，その間にあるすべての SQL に対して有効となります。WHENEVER 文の有効範囲については，マニュアル「HiRDB SQL リファレンス」を参照してください。

(5)　検索，更新の SQL（実行文）

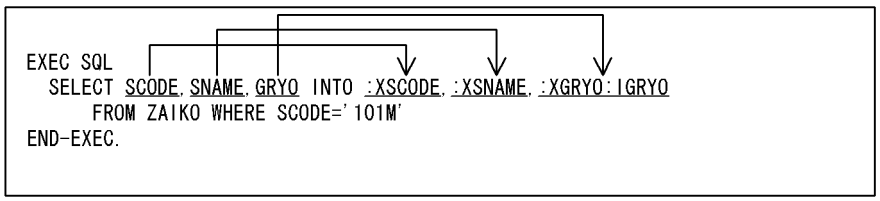
データを検索，更新，挿入，又は削除するための SQL 文を記述します。各 SQL 文の記述方法については，「データベースの操作」を参照してください。

ここでは，埋込み変数及び標識変数の使用方法について説明します。

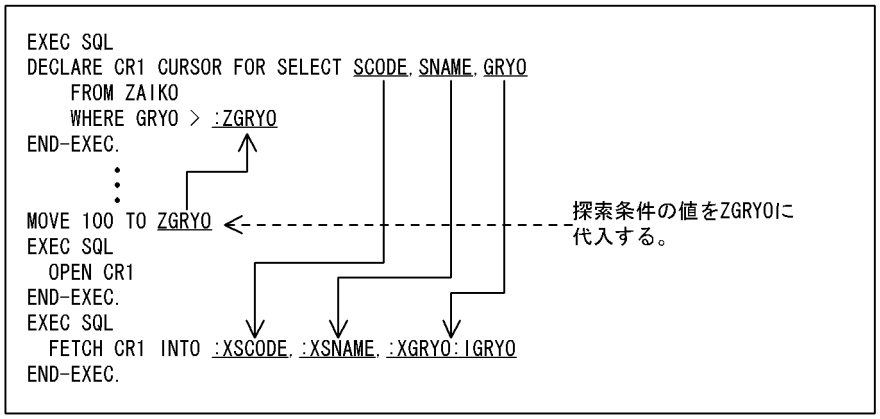
(a)　1 行 SELECT 文又は FETCH 文に記述する場合

1 行 SELECT 文又は FETCH 文の INTO 句に，埋込み変数及び標識変数を指定します。このとき，各変数の先頭にはコロンを付けます。標識変数は対応する埋込み変数に続けて指定します。例を次に示します。

＜1行SELECT文の場合＞



＜FETCH文の場合＞



INTO 句に指定した埋込み変数は，SELECT 文の列の並びに指定した列名の順に対応しており，この順序に従って埋込み変数に検索結果が格納されます。

検索結果にナル値を含む場合は標識変数に負の値が格納されるので，この値を参照してナル値かどうかを判断します。このとき，埋込み変数の値は不定となります。また，標識変数の値が 0 のときは，ナル値以外の値を受け取ったことを示し，値が正のときは，ナル値以外の文字列データを受け取ったが，埋込み変数の領域長が不足していたため，右端が切り捨てられたことを示します。

また，探索条件の値に埋込み変数を指定しておけば，SQL の実行時に探索条件の値を与えることができます。

(b)　UPDATE 文及び INSERT 文に記述する場合

UPDATE 文の SET 句及び INSERT 文の VALUES 句に埋込み変数及び標識変数を指定します。このとき，各変数の先頭にはコロンを付けます。標識変数は対応する埋込み変数に続けて指定します。例を次に示します。

＜ UPDATE 文の場合＞

```
EXEC SQL
  UPDATE ZAIKO SET GRYO=:XGRYO:IGRYO WHERE SCODE=:XSCODE
END-EXEC.
```

＜ INSERT 文の場合＞

```
EXEC SQL
  INSERT INTO ZAIKO VALUES(:XSCODE,:XSNAME,:XCOL,:XTANKA
                           ,:XGRYO:IGRYO,:XMAXRYO,:XSAFRYO)
END-EXEC.
```

UPDATE 文又は INSERT 文によって表にナル値を設定する場合は，その SQL を実行する前にあらかじめ標識変数に負の値を設定しておきます。このとき，埋込み変数には何も設定しなくてよいです。また，ナル値以外の値を渡す場合は，標識変数の値を 0 又は正にしておきます。

(6)　エラーの判定

SQL の実行中にエラーが発生した場合，SQLCODE，SQLSTATE を参照すると，HiRDB から返されるリターンコードが分かります。そのリターンコードを利用して，その後の処理をどうするか指示します。ただし，「エラー時の処置の指定」で，既に同じ内容のエラー処理を指定している場合は，ここで指示する必要はありません。

DECLARE CURSOR のような宣言文の直後に，エラーの判定はしないでください。エラーの判定をすると，不正な SQLCODE を参照して，HiRDB が誤動作してしまいます。

エラーの判定については，「エラーの判定」を参照してください。

(7)　トランザクションの有効化

トランザクション内で更新処理をした場合，更新したデータベースの内容を有効にし，トランザクションを正常終了させます。

トランザクションを有効にする SQL 文を次に示します。

```
EXEC SQL
  COMMIT
END-EXEC.
```

[説明]

トランザクションを有効にします。また，トランザクションの有効化後に UAP を HiRDB から切り離したい場合には，RELEASE 指定の COMMIT 文を実行します。RELEASE を指定した場合，DISCONNECT 文を実行しなくてもよいです。

(8) トランザクションの無効化

トランザクション内で更新したデータベースの内容を無効にし，トランザクションを終了させます。トランザクション内での更新処理が不正だった場合など，データベースの更新を取り消したいときに指定します。

トランザクションを無効にする SQL 文を次に示します。

```
EXEC SQL
  ROLLBACK
END-EXEC.
```

[説明]

トランザクションを無効にします。また，トランザクション終了後に UAP を HiRDB から切り離したい場合には，RELEASE 指定の ROLLBACK 文を実行します。RELEASE を指定した場合，DISCONNECT 文を実行しなくてもよいです。

(9) HiRDB からの切り離し

トランザクションを正常終了させて，UAP を HiRDB から切り離します。DISCONNECT 文は，RELEASE 指定の COMMIT 文を実行したときと同じになります。

トランザクションを正常終了させて，UAP を HiRDB から切り離す SQL 文を次に示します。

```
EXEC SQL
  DISCONNECT
END-EXEC.
```

[説明]

トランザクションを正常終了させて，UAP を HiRDB から切り離します。また，トランザクションを取り消した後に UAP を HiRDB から切り離す場合には，RELEASE 指定の ROLLBACK 文を実行します。なお，DISCONNECT 文，COMMIT 文（RELEASE 指定），及び ROLLBACK 文（RELEASE 指定）のどれも実行しないで UAP が終了した場合，ROLLBACK 文（RELEASE 指定）が自動的に実行され，実行中のトランザクションは無効となるので注意してください。

3.2　UAP の記述

ここでは，UAP を作成するときに考慮する必要がある基本的な内容について説明します。

3.2.1　UAP の記述言語

UAP の形式は，高級言語で記述されたソースプログラム中に直接 SQL を記述する埋込み型です。

HiRDB で使用できる UAP の記述言語を次の表に示します。

表 3–1　UAP の記述言語

動作環境	記　述　言　語
AIX	• C 言語 • C++言語 • COBOL 言語（COBOL85 及び COBOL2002）
Linux	• C 言語（gcc） • C++言語（GCC） • COBOL 言語（COBOL85 及び COBOL2002） • OOCOBOL 言語（OOCOBOL）※
Windows	• C 言語（Microsoft Visual C++） • C++言語（Microsoft Visual C++） • COBOL 言語（COBOL85 及び COBOL2002） • OOCOBOL 言語（OOCOBOL）※

注※

　複数接続機能は使用できません。

なお，埋込み型の UAP は，そのままではコンパイル，及びリンケージができません。SQL プリプロセサを実行し，ポストソースプログラムに変換してからコンパイル，及びリンケージをしてください。プリプロセス，コンパイル，及びリンケージについては，「UAP 実行前の準備」を参照してください。

3.2.2　インタフェース領域

インタフェース領域は，HiRDB と UAP との間で情報をやり取りするために使用します。インタフェース領域の種類と使用目的を次の表に示します。

表 3–2　インタフェース領域の種類と使用目的

領域名	使用目的	言語	
		C	COBOL
SQL 連絡領域	SQL の実行結果の詳細情報を取得します。	○※1	○※1
SQL 記述領域	• UAP 実行時に動的に決定した入力変数の情報をシステムに通知します。 • UAP を動的に実行するために前処理した SQL の検索項目の情報を受け取ります。 • 列名記述領域を指定します。	△	△
列名記述領域	UAP を動的に実行するために前処理した SQL の検索項目の情報を受け取ります。	△	△
型名記述領域	ユーザ定義型のデータ型名を受け取ります。	△	△
文字集合名記述領域	• UAP 実行時に動的に決定した入力変数の文字集合名をシステムに通知します。 • UAP を動的に実行するために前処理した，SQL の検索項目の文字集合名を受け取ります。	△	△
埋込み変数	埋込み型 UAP の SQL 中に指定して値の受け渡しをします。	△	△
標識変数	埋込み型 UAP の SQL 中に指定して値の受け渡しをします。	△	△
パラメタ	UAP を動的に実行するために前処理する SQL に対して，UAP から値を渡します。	△	△※2

（凡例）○：必要　　△：任意

注※1

SQL プリプロセサを実行すると UAP 中に展開されるので，宣言は不要です。SQL プリプロセサの実行については，「プリプロセス」を参照してください。

注※2

?パラメタの代わりに埋込み変数，及び標識変数を使用します。

SQL 連絡領域の詳細は「SQL 連絡領域」を，SQL 記述領域の詳細は「SQL 記述領域」を参照してください。また，埋込み変数，標識領域，及び?パラメタの詳細については，マニュアル「HiRDB SQL リファレンス」を参照してください。

3.2.3　整合性制約

HiRDB には，データベースのデータが正しい状態であることを保証するため，次に示す二つの整合性制約があります。

- 非ナル値制約

- 一意性制約

(1)　非ナル値制約

指定した列の値に，ナル値を許さない制約のことを非ナル値制約といいます。

非ナル値制約は，CREATE TABLE の NOT NULL オペランドで指定します。非ナル値制約を指定した列に対しては，常に値が定まっていることが要求されるため，ナル値を与えようようとすると制約違反となります。制約違反の場合，データベースを更新できないので，非ナル値制約を指定した列には，ナル値を与えないようにする必要があります。

(2)　一意性制約

指定した列の値がすべての行で一意であり，列中での重複を許さない制約のことを一意性制約といいます。

一意性制約は，次に示す列に指定できます。

(a)　クラスタキーとして定義する列

CREATE TABLE の UNIQUE オペランドで指定します。

クラスタキーの指定については，マニュアル「HiRDB システム導入・設計ガイド」を参照してください。

(b)　インデクスを定義する列

CREATE INDEX の UNIQUE オペランドで指定します。

CREATE TABLE，及び CREATE INDEX の指定については，マニュアル「HiRDB SQL リファレンス」を参照してください。

3.2.4　SQL を使用した検索方法の分類

SQL を使用して表を検索する場合，大別すると SQL の静的実行と動的実行に分類できます。SQL を使用したときの UAP からの検索方法の分類を次の表に示します。

表 3–3　SQL を使用した UAP からの検索方法の分類

検索方法		問合せ内容を指定する SQL
埋込み型 UAP	静的 SQL	1 行 SELECT 文
		カーソル宣言
	動的 SQL	1 行 SELECT 文
		動的 SELECT 文

(1)　1 行 SELECT 文

表から 1 行だけ検索結果を取り出すには，1 行 SELECT 文を使用します。

1 行 SELECT 文を使用すると，カーソルを使用する必要がないため，一つの SQL で検索できます。

1 行 SELECT 文は，次の場合に使用すると有効です。また，UAP 実行時に組み立てた 1 行 SELECT 文を動的に実行することもできます。

- 検索結果が 1 行以下になることが明確な場合
- グループ分け（GROUP BY 句）をしないで集合関数を使用する場合

なお，1 行だけの検索の場合でも，検索した行を更新したり，削除したりするときは，カーソルを使用する方が処理効率が良いので，場合によって使い分ける必要があります。

(2)　カーソル宣言

検索結果が複数行になる場合，UAP では一度に受け取れないため，カーソルを使用して 1 行ずつ取り出します。カーソルの宣言から検索の終了までの流れは，次に示すとおりです。

1. DECLARE CURSOR でカーソル宣言をします。
2. OPEN 文を実行すると，カーソルが開いて使用できる状態になります。
3. FETCH 文を実行すると，カーソルが検索結果の 1 行目を指すので，埋込み変数，又はパラメタ（FETCH 文の INTO 句で指定した）を使用して検索結果を取り出します。
4. 再度 FETCH 文を実行すると，カーソルが次の行へ進むので，1 行ずつ検索結果を取り出します。
5. 検索する行がなくなるまで 4. の操作を繰り返します。
6. 検索が終了したら，CLOSE 文を実行してカーソルを閉じます。

(3)　動的 SELECT 文

SQL の動的実行によって検索結果を複数行取り出すには，動的 SELECT 文を使用します。動的 SELECT 文で検索結果を取り出すには，あらかじめカーソル宣言をしておくか，又は ALLOCATE CURSOR 文でカーソルを割り当てておく必要があります。カーソル宣言，又はカーソル割り当てをしておくと，UAP 実行時に組み立てた SQL 文を PREPARE 文で前処理した後は，通常のカーソルを使用した検索と同様の操作ができます。

3.2.5　静的 SQL と動的 SQL

UAP を作成するとき，プログラム中に SQL を記述する方法を静的 SQL といいます。これに対し，UAP を作成するとき，プログラム中に SQL を記述しないで，UAP 実行時に SQL の文字列を組み立てる方法を動的 SQL といいます。

静的 SQL と動的 SQL とでは，実行時の特徴が異なりますので，UAP を作成する前に十分検討する必要があります。

(1)　実行時の相違点

静的 SQL と動的 SQL の実行時の特徴を次の表に示します。

表 3–4　静的 SQL と動的 SQL の実行時の特徴

種　類	長　　所	短　所
静的 SQL	同一の UAP を繰り返し実行する場合，一度実行した SQL 文は実行形式に変換され，共用メモリ上で再利用できるため，処理効率が良い。	UAP 中に SQL を埋め込むので，探索条件の変更が限定される。
動的 SQL	実行時に SQL の文字列を組み立てるため，探索条件を変更するときの自由度が大きい。	実行するたびに SQL を解析して実行形式に変換するため，処理効率が悪い。※

注※

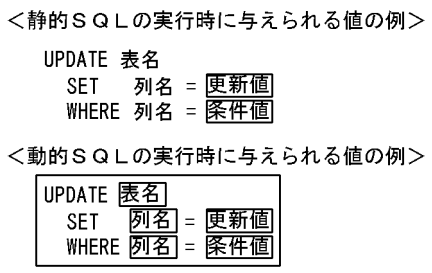
同じ文字列の SQL を複数回実行するようなケースでは，処理効率は良くなります。

(2)　実行時に指定できる値

静的 SQL の実行では，実行時に挿入値，更新値，及び探索条件の値を変更できます。また，動的 SQL の実行では，静的 SQL の実行時に変更できる値以外に，表名，列名，条件式など SQL の任意の部位を変更できます。

静的 SQL の場合と動的 SQL の場合とで，各々実行時に変更できる値の例を次の図に示します。なお，枠で囲んである部分が値を変更できる箇所です。

図 3–2　SQL の実行時に与えられる値



(3)　動的 SQL の実行と留意点

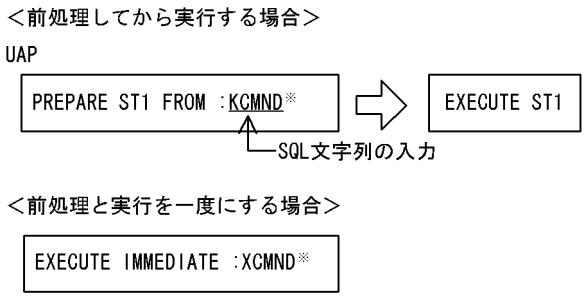
動的 SQL は，静的 SQL と比較して探索条件を変更するときの自由度は大きいが，条件を変更するたびに SQL を実行しなければならないため，あらかじめ実行時の性能（処理効率）を考慮する必要があります。

(a)　動的 SQL の前処理と実行

動的 SQL では，UAP 実行時に PREPARE 文で前処理してから実行します。実行は，前処理する SQL が動的 SELECT 文か動的 SELECT 文以外かで異なります。前処理する SQL が動的 SELECT 文の場合，OPEN 文，FETCH 文，及び CLOSE 文で実行し，前処理する SQL が動的 SELECT 文以外の場合，EXECUTE 文で実行します。また，EXECUTE IMMEDIATE 文を使用すれば，前処理と実行を一度にできます。値を変えて同じ SQL を動的に実行する場合，前処理を何度も実行するより，?パラメタを使用して，前処理を一度だけして，実行時に?パラメタに与える値を変えて実行する方が性能（処理効率）が向上します。?パラメタの詳細については，マニュアル「HiRDB SQL リファレンス」を参照してください。

動的 SQL の実行形態を次の図に示します。

図 3–3　動的 SQL の実行形態



注※

埋込み変数（:XCMND）は，埋込み SQL 宣言節で宣言しておいてください。

埋込み変数については，マニュアル「HiRDB SQL リファレンス」を参照してください。

PREPARE 文で前処理できる SQL と EXECUTE IMMEDIATE 文で前処理と実行が一度にできる SQL を次の表に示します。

表 3–5　PREPARE 文で前処理できる SQL と EXECUTE IMMEDIATE 文で前処理と実行が一度に実行できる SQL

分類	SQL 文	PREPARE	EXECUTE IMMEDIATE
操作系 SQL	ASSIGN LIST 文	○※3	○
	CALL	○※3	○
	DELETE※1	○※3	○
	準備可能動的 DELETE 文：位置付け	○	○
	DROP LIST 文	○※3	○
	INSERT	○※3	○
	PURGE TABLE	○※3	○
	1 行 SELECT※2	○※3	○

分類	SQL 文	PREPARE	EXECUTE IMMEDIATE
	動的 SELECT	○※4	×
	UPDATE※1	○※3	○
	準備可能動的 UPDATE 文：位置付け	○	○
	代入文	○※3	×
制御系 SQL	CALL COMMAND	○	○
	COMMIT	×	×
	CONNECT	×	×
	DISCONNECT	×	×
	LOCK TABLE	○※3	○
	ROLLBACK	×	×
	SET SESSION AUTHORIZATION 文	○※3	×
定義系 SQL	ALTER INDEX	○※3	○
	ALTER PROCEDURE	○※3	○
	ALTER ROUTINE	○※3	○
	ALTER TABLE	○※3	○
	ALTER TRIGGER	○※3	○
	COMMENT	○※3	○
	CREATE AUDIT	○※3	○
	CREATE CONNECTION SECURITY	○※3	○
	CREATE FUNCTION	○※3	○
	CREATE INDEX	○※3	○
	CREATE PROCEDURE	○※3	○
	CREATE ROLE	○※3	○
	CREATE SCHEMA	○※3	○
	CREATE SEQUENCE	○※3	○
	CREATE TABLE	○※3	○
	CREATE TRIGGER	○※3	○
	CREATE TYPE	○※3	○

分類	SQL 文	PREPARE	EXECUTE IMMEDIATE
	CREATE VIEW	○※3	○
	DROP AUDIT	○	○
	DROP CONNECTION SECURITY	○※3	○
	DROP DATA TYPE	○※3	○
	DROP FUNCTION	○※3	○
	DROP INDEX	○※3	○
	DROP PROCEDURE	○※3	○
	DROP ROLE	○※3	○
	DROP SCHEMA	○※3	○
	DROP SEQUENCE	○※3	○
	DROP TABLE	○※3	○
	DROP TRIGGER	○※3	○
	DROP VIEW	○※3	○
	GRANT	○※3	○
	REVOKE	○※3	○

（凡例）

○：使用できます。 ×：使用できません。

注

埋込み変数を含む SQL は，動的に実行できないので，?パラメタを使用してください。?パラメタの詳細については，マニュアル「HiRDB SQL リファレンス」を参照してください。

注※1

カーソルを使用した操作はできません。

注※2

INTO 句を含まないようにしてください。

注※3

EXECUTE 文で実行してください。

注※4

OPEN 文，FETCH 文，及び CLOSE 文で実行してください。

動的に指定された表にデータを挿入する場合の例を次の図に示します。

図 3-4　動的に指定された表にデータを挿入する例



＜説明＞

1. 表名を格納する変数(TNAME)を宣言します。
2. 入力データから変数(TNAME)を表名に読み込みます。
3. ?パラメタ個数，及び各?パラメタに対するデータ領域のデータ型，データ長，最大要素数として，2.で指定された表の列数，各列のデータ型，データ長，最大要素数をDESCRIBE文を利用して，SQL記述領域(D_AREA)に設定します。
4. 指定された表にデータを挿入するためのINSERT文を生成します。
5. XCMND中のINSERT文を前処理して，SQL文識別子(ST2)を付けます。
6. 挿入するデータがある間，行単位に挿入データの入力，データ領域への設定，EXECUTE文による実行を繰り返します。

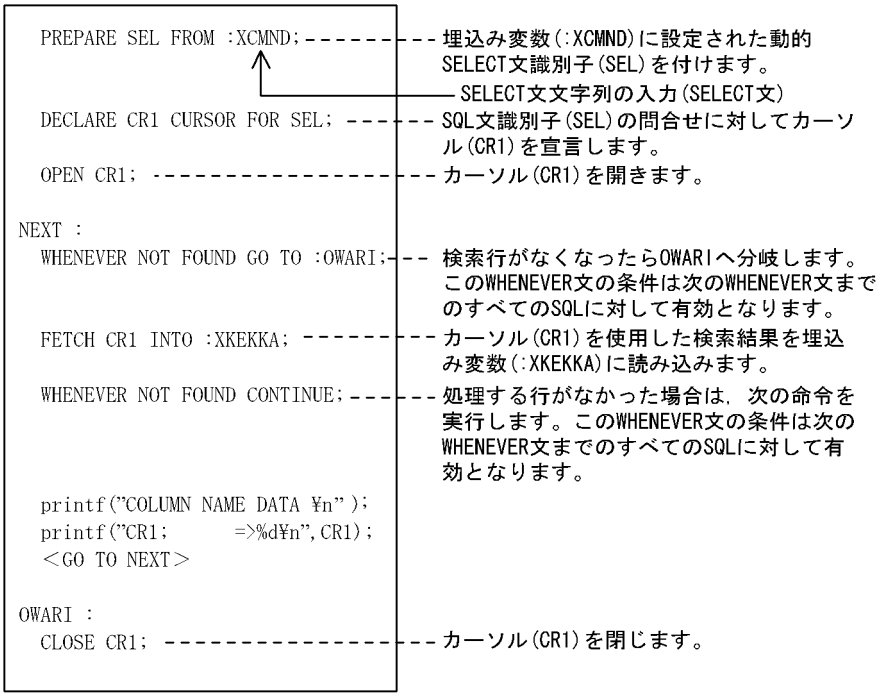
(b)　EXECUTE 文と EXECUTE IMMEDIATE 文の使い分け

EXECUTE IMMEDIATE 文は，PREPARE 文と EXECUTE 文を連続して実行するのと同じです。ただし，同じ SQL を何度も繰り返して実行する場合，EXECUTE IMMEDIATE 文でその都度前処理するより，PREPARE 文で前処理してから EXECUTE 文で繰り返して実行する方が性能（処理効率）が良くなります。

(c)　前処理する SQL が動的 SELECT 文の場合の動的実行

前処理する SQL が動的 SELECT 文か動的 SELECT 文以外かで異なります。前処理する SQL が動的 SELECT 文の場合，OPEN 文，FETCH 文，及び CLOSE 文で実行し，前処理する SQL が動的 SELECT 文以外の場合，EXECUTE 文で実行します。前処理する SQL が動的 SELECT 文の場合の動的実行例を，次の図に示します。

図 3-5　前処理する SQL が動的 SELECT 文の場合の動的実行例



注　この例では，SQL先頭子，及び終了子は省略しています。

(d)　動的 SELECT 文に対する，カーソルを使用した SQL の動的実行

動的 SELECT 文を前処理し，その動的 SELECT 文に対するカーソルを使用した SQL を，動的に実行する場合，カーソル宣言で宣言したカーソルは使用しません。この場合，前処理した動的 SQL 文に対して ALLOCATE CURSOR 文で割り当てたカーソルを使用します。動的 SELECT 文に対して，カーソルを使用した SQL を動的に実行する場合の例を次に示します。

```
PREPARE GLOBAL :SEL FROM :XCMND; ..................  埒込み変数(:XCMND)に設定された
                                                     動的SELECT文に，拡張文名
                                                     (:SEL='SEL1')を付けます。
ALLOCATE GLOBAL :CR CURSOR FOR GLOBAL :SEL; .......  拡張文名(:SEL='SEL1')が識別する
                                                     問合せに対して，カーソル
                                                     (:CR='CR1')を割り当てます。
PREPARE UPD1 FROM
  'UPDATE SET C1=? WHERE CURRENT OF GLOBAL CR1'; ..  カーソル(CR1)を使用したUPDATE文
                                                     を前処理し，SQL文識別子(UPD1)を
                                                     付けます。
OPEN GLOBAL :CR; ..................................  カーソル(:CR='CR1')を開きます。
FETCH GLOBAL :CR INTO :XKEKKA; ....................  カーソル(:CR='CR1')を使用した
                                                     検索結果を埒込み変数(:XKEKKA)に
                                                     読み込みます。
EXECUTE UPD1 USING :XDATA; ........................  前処理したSQL文識別子(UPD1)の
                                                     UPDATE文を実行します。このとき，
                                                     ?パラメタに対応する埒込み変数
                                                     (:XDATA)を指定します。
CLOSE GLOBAL :CR; .................................  カーソル(:CR='CR1')を閉じます。
```

(e)　動的 SQL の実行時に決定した情報の受け取り

UAP で動的に SQL を実行した場合，実行時に決定した情報（データ受け渡し領域の個数，属性，番地など）を HiRDB に通知するための領域として SQL 記述領域が使用されます。また，SQL 記述領域は，動的に実行するために PREPARE 文で前処理する SQL の検索項目の情報を，次のどちらかの方法で受け取れます。

- DESCRIBE 文を実行する。
- PREPARE 文実行時に，OUTPUT，及び INPUT を指定し実行する（この場合，DESCRIBE 文を実行しなくても，PREPARE 文実行時に情報の受け取りもできるため，通信回数を削減できます）。

DESCRIBE 文については，マニュアル「HiRDB SQL リファレンス」を参照してください。また，SQL 記述領域の使用例については，「SQL 記述領域」を参照してください。

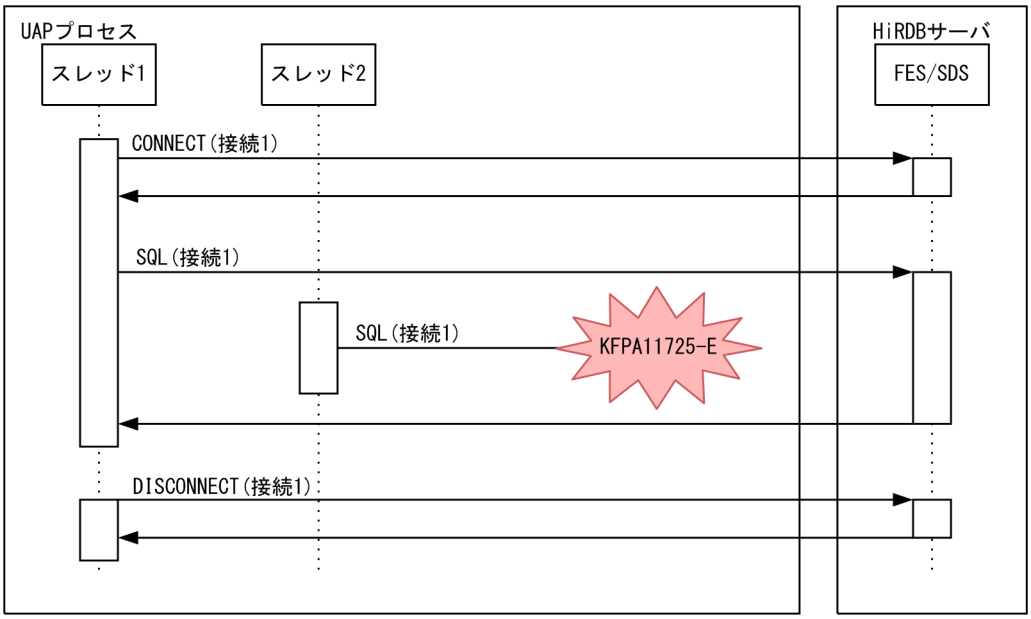
3.2.6　マルチスレッドを使用した UAP の SQL の実行

(1)　単一接続の場合（接続ハンドルを使用しない場合）

マルチスレッドを使用した UAP で SQL を実行する場合，SQL を実行中の接続に対して，別スレッドから SQL を実行することはできません。実行した場合は動作を保証しません。

SQL を実行中の接続に対して，別スレッドから SQL を実行した場合は，KFPA11725-E エラーとします。そのため，KFPA11725-E エラーが発生する場合は，SQL を実行するスレッドとの間で処理をシリアライズして，SQL を実行中の接続に対して，別スレッドから SQL を実行しないようにプログラムを修正してください。スレッド間で接続を共有する例を次の図に示します。

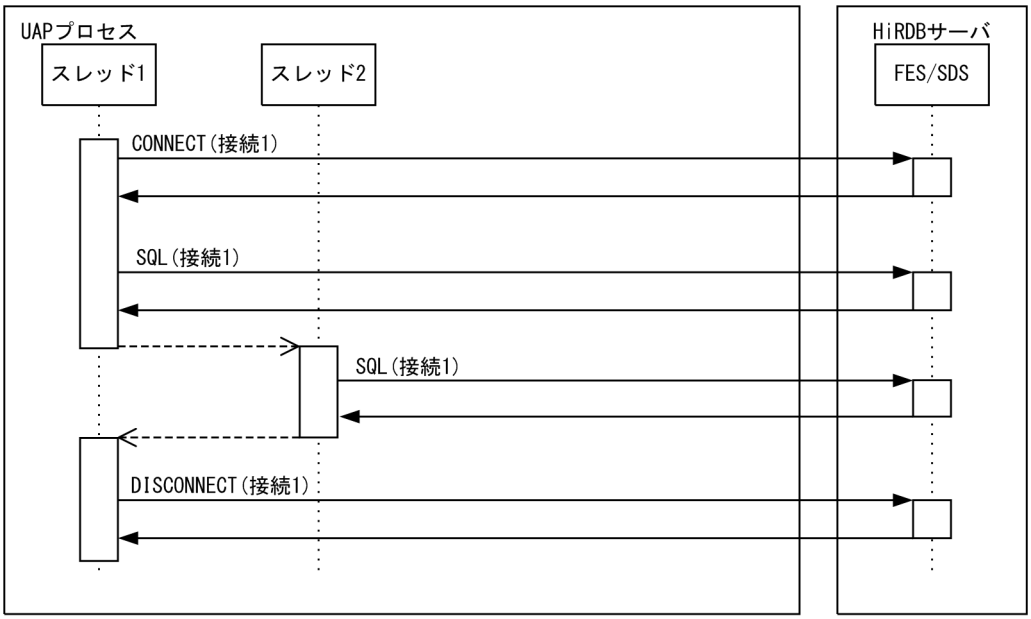
図 3-6　スレッド間で接続を共有する例（異常時）



（凡例）

：SQL実行

図 3-7　スレッド間で接続を共有する例（正常時）



（凡例）

：SQL実行

：スレッド制御

接続 1 は複数接続機能を使用していない（接続ハンドルを使用していない）ため，プロセス内で接続が共有されます。スレッド 1 の接続 1 での SQL 実行中に，スレッド 2 が接続 1 に対して SQL を実行したため，スレッド 2 の SQL は KFPA11725-E エラーとなります。このような場合，接続 1 を共有しているスレッド

の処理をシリアライズし，スレッド 1 の SQL 実行中に，スレッド 2 から SQL を実行しないようにしてください。

(2)　複数接続機能を使用する場合（接続ハンドルを使用する場合）

マルチスレッドを使用した UAP で，複数接続機能の接続ハンドルを使用する場合，SQL を実行中の接続ハンドルを使用して，別スレッドから SQL を実行することはできません。実行した場合は動作を保証しません。

SQL を実行中の接続ハンドルを使用して，別スレッドから SQL を実行した場合は，KFPA11725-E エラーとします。そのため，KFPA11725-E エラーが発生する場合は，SQL を実行するスレッドとの間で処理をシリアライズして，SQL を実行中の接続ハンドルを使用して，別スレッドから SQL を実行しないようにプログラムを修正してください。複数接続機能を使用してスレッド間で接続ハンドルを共有する例を次の図に示します。なお，複数接続機能の詳細は「複数接続機能」を参照してください。

図 3–8　スレッド間で接続ハンドルを共有する例（異常時）

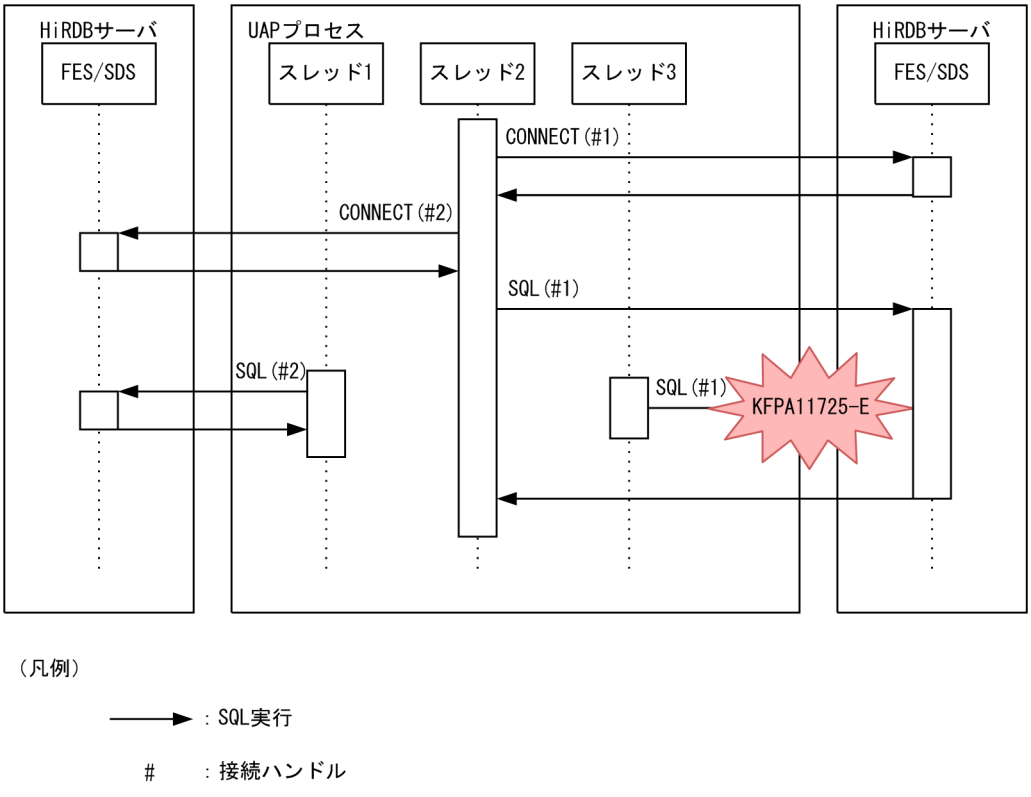
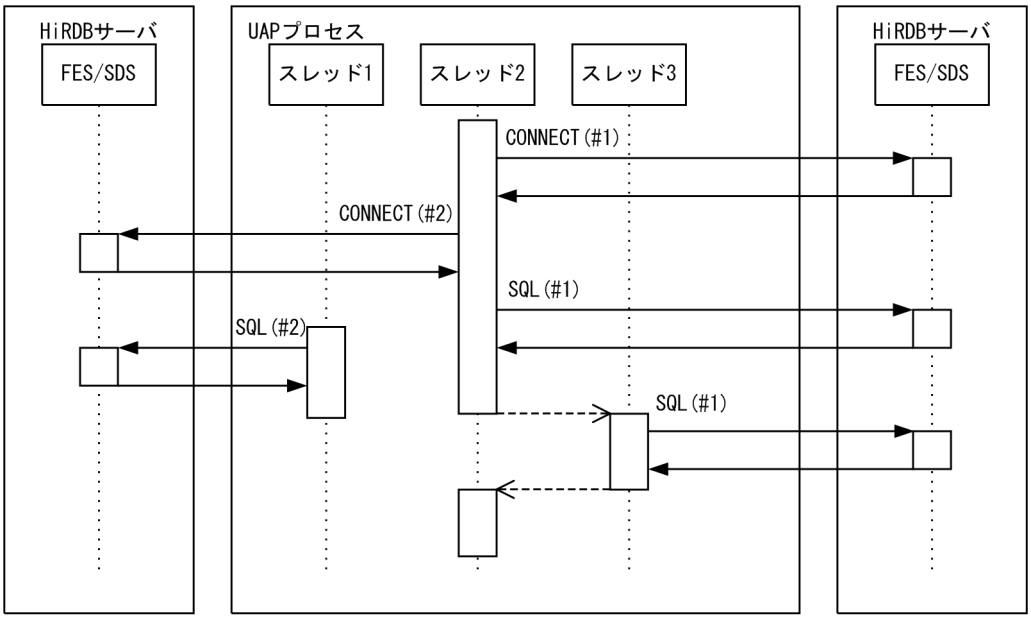


図 3–9　スレッド間で接続ハンドルを共有する例（正常時）



（凡例）

：SQL実行

：スレッド制御

#　：接続ハンドル

接続ハンドル 1 及び接続ハンドル 2 を，スレッド間で共有しています。スレッド 2 の接続ハンドル 1 での SQL 実行中に，スレッド 3 が接続ハンドル 1 を使用して SQL を実行したため，スレッド 3 の SQL は KFPA11725-E エラーとなります。このような場合，接続ハンドル 1 を共有しているスレッドとの間で処理をシリアライズして，スレッド 2 の SQL 実行中に，スレッド 3 から SQL を実行しないようにしてください。

なお，スレッド 2 の SQL 実行中に，スレッド 1 も SQL を実行していますが，スレッド 1 が使用した接続ハンドル 2 はスレッド 2 の接続ハンドルと異なるため，スレッド 1 の SQL は実行できます。

3.3　トランザクション制御

ここでは，HiRDB システムに対して UAP がトランザクションを開始する，又は終了する契機，同期点の設定とトランザクションの扱い，及びロールバックの設定について説明します。

3.3.1　HiRDB システムへの接続と切り離し

HiRDB システムへの接続は，CONNECT 文を実行したときです。

HiRDB システムとの切り離しは，DISCONNECT 文を実行したときです。

3.3.2　トランザクションの開始と終了

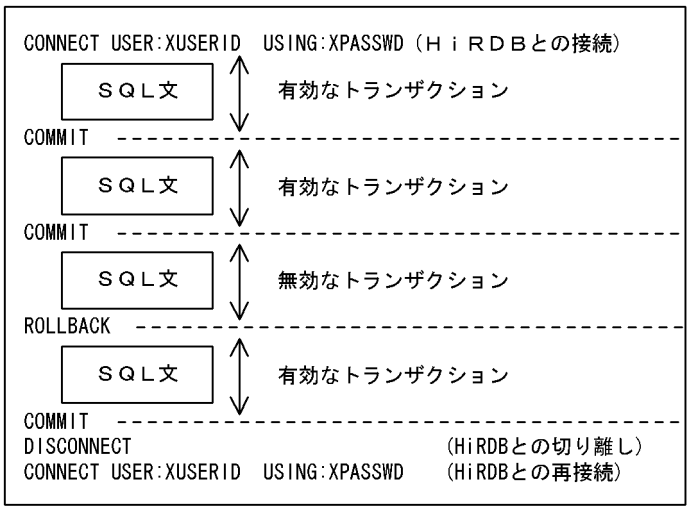
トランザクションの開始は，UAP の SQL 文を実行したときです。

トランザクションの終了は，COMMIT 文，又は ROLLBACK 文を実行したときです。

HiRDB システムと接続中は，幾つでもトランザクションの開始と終了をできます。

トランザクションの開始と終了の例を次の図に示します。

図 3–10　トランザクションの開始と終了の例



なお，HiRDB/パラレルサーバでは，SQL の処理を複数のサーバに分岐しますが，一つのトランザクションとして管理されるので，内部の分岐については考慮する必要はありません。

3.3.3　同期点の設定とロールバックの設定

同期点の設定とトランザクションの扱いを次の表に示します。

表 3–6　同期点の設定とトランザクションの扱い

同期点の設定	同期点設定の要因	トランザクションの扱い
SQL 実行による UAP での設定	COMMIT 文の実行	有効※1
	ROLLBACK 文の実行	無効※1※2
SQL 実行による HiRDB システムでの設定	定義系 SQL の実行	有効※1
	PURGE TABLE 文の実行	有効※1
	DISCONNECT 文の実行	有効※1
	SQL 実行時に処理を継続できなくなったとき	無効※3
UAP 終了による HiRDB システムでの設定	次の SQL 文を実行しないで UAP が終了した • COMMIT 文 • ROLLBACK 文 • DISCONNECT 文	無効※2

注※1

OLTP 環境で X/Open に従ったアプリケーションプログラムでは実行できません。X/Open に従ったアプリケーションプログラムで同期点，及びロールバックの設定については，「OLTP 環境での UAP のトランザクション管理」を参照してください。

注※2

トランザクションが無効になる場合，直前の同期点までが無効になります。

注※3

暗黙的ロールバックになります。暗黙的ロールバックになる主な要因は次に示すとおりです。

- デッドロック
- RD エリアのページ不足
- RD エリアの障害の検知，又は閉塞の検知

3.3.4　OLTP 環境での UAP のトランザクション管理

OLTP 環境で，トランザクションの同期点，及びロールバックを UAP から実行する場合，X/Open に従ったアプリケーションプログラムインタフェース（以降 API と略します）を使用します。

ここでは，OpenTP1 の場合を例に説明します。なお，OpenTP1 を利用する場合のプログラムの作成方法については，マニュアル「OpenTP1 プログラム作成リファレンス C 言語編」，又はマニュアル「OpenTP1 プログラム作成リファレンス COBOL 言語編」を参照してください。

1 件のトランザクションを RPC（リモートプロシジャコール）を使用して，複数の OLTP ユーザサーバプロセスで実現できます。個々のプロセスをトランザクションブランチと呼び，全体をまとめて OLTP のグローバルトランザクションと呼びます。

このような OLTP のグローバルトランザクションから HiRDB をアクセスする場合，一つのグローバルトランザクション内の複数のトランザクションブランチから HiRDB をアクセスできません。

アクセスする資源によっては，グローバルトランザクション内の先発トランザクションブランチで掛けた排他を，後発トランザクションブランチが待ってタイムアウトになったり，トランザクションブランチ間でデッドロックが発生したりすることがあります。

このような場合は，連鎖 RPC などの機能を使用して，複数の RPC を同一トランザクションブランチとして扱うようにしてください。

(1) C 言語で記述する場合

(a) トランザクションの開始

tx_begin 関数を実行します。

(b) 同期点の設定

tx_commit 関数を実行します。

(c) ロールバックの設定

tx_rollback 関数を実行します。

(2) COBOL85 言語で記述する場合

(a) トランザクションの開始

```
    DATA DIVISION.

    *include TX definitions.

    01 TX-RETURN_STATUS
       COPY TXSTATUS.

    PROCEDURE DIVISION.
    CALL ”TXBEGIN” USING TX-RETURN_STATUS.
```

(b) 同期点の設定

```
    DATA DIVISION.

    *include TX definitions.

    01 TX-RETURN_STATUS
```

```
    COPY TXSTATUS.

PROCEDURE DIVISION.
CALL "TXCOMMIT" USING TX-RETURN_STATUS.
```

(c) ロールバックの設定

```
DATA DIVISION.

*include TX definitions.

01 TX-RETURN_STATUS
    COPY TXSTATUS.

PROCEDURE DIVISION.
CALL "TXROLLBACK" USING TX-RETURN_STATUS.
```

3.3.5 トランザクションの移行

トランザクションのコミット処理を，UAP が HiRDB をアクセスしたときと異なるプロセスで実行することをトランザクションの移行といいます。

なお，ここでいう UAP とは，HiRDB XA ライブラリを使用して HiRDB に接続する UAP のことです。

トランザクションの移行機能を使用する場合，クライアント環境定義の PDXAMODE オペランドに 1 を指定してください。

PDXAMODE オペランドについては，「クライアント環境定義の設定内容」を参照してください。

(1) PDXAMODE オペランドの指定による LOCK TABLE UNTIL DISCONNECT の有効範囲

PDXAMODE の指定によって，LOCK 文の LOCK TABLE UNTIL DISCONNECT の有効範囲が変わります。

(a) PDXAMODE=0 の場合

1. AP 記述によってリソースマネジャのオープン処理をする場合
   リソースマネジャのクローズ実行時まで有効になります。

2. トランザクションごとにリソースマネジャのオープン処理をする機能がある場合
   グローバルトランザクション内で有効になります。

(b) PDXAMODE=1 の場合

1. AP 記述によってリソースマネジャのオープン処理をする場合
   - トランザクションの移行が発生しない場合

リソースマネジャのクローズ実行時まで有効になります。

- トランザクションの移行が発生する場合
  グローバルトランザクション内で有効になります。

2. トランザクションごとにリソースマネジャのオープン処理をする機能がある場合
   グローバルトランザクション内で有効になります。

OpenTP1 を利用した場合の LOCK TABLE UNTIL DISCONNECT の有効範囲を，次の表に示します。

表 3–7　OpenTP1 を利用した場合の LOCK TABLE UNTIL DISCONNECT の有効範囲

<table>
<tr><th>PDXAMODE の指定値</th><th colspan="4">OpenTP1 の指定値</th><th>LOCK TABLE UNTIL DISCONNECT の有効範囲</th></tr>
<tr><td rowspan="2">0</td><td colspan="4">trn_rm_open_close_scope=process</td><td>リソースマネジャのクローズまで有効</td></tr>
<tr><td colspan="4">trn_rm_open_close_scope=transaction</td><td>グローバルトランザクション内で有効</td></tr>
<tr><td rowspan="5">1</td><td rowspan="4">trn_rm_open_close_scope=process</td><td colspan="3">trnstring オペランドで-d オプションを指定</td><td rowspan="3">リソースマネジャのクローズまで有効</td></tr>
<tr><td rowspan="3">trnstring オペランドで-d オプションを省略</td><td colspan="2">同一の OpenTP1 システム内では単体の AP でグローバルトランザクションを構成</td></tr>
<tr><td rowspan="2">同一の OpenTP1 システム内では複数の AP でグローバルトランザクションを構成</td><td>一つの AP が HiRDB XA ライブラリとリンク</td></tr>
<tr><td>複数の AP が HiRDB XA ライブラリとリンク</td><td>グローバルトランザクション内で有効</td></tr>
<tr><td colspan="4">trn_rm_open_close_scope=transaction</td></tr>
</table>

注　-d オプションは，TP1/Server Base のバージョンが 03-03 以降で，かつ UNIX 版 HiRDB の場合に指定できます。

3.4　排他制御

複数のユーザが一つの表を同時に操作した場合，データの不整合が起きることがあるため，HiRDB システムでは自動的に排他制御をしてデータの不整合を防止しています。ここでは，排他制御の仕組みとユーザが変更できる排他制御の内容について説明します。

3.4.1　排他制御の単位

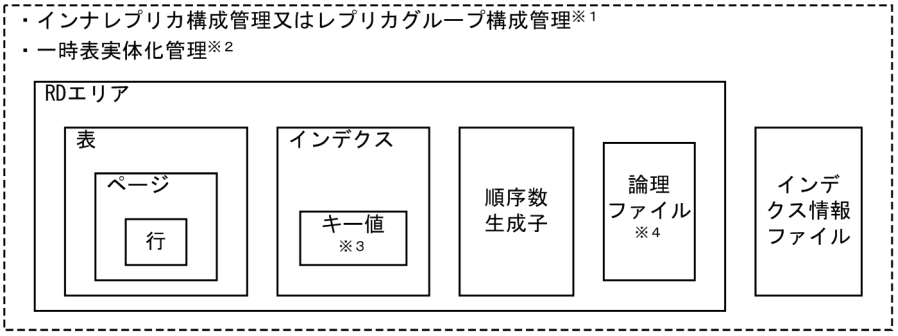
(1)　排他資源とその包含関係

HiRDB では，データベースの整合性を保つように排他制御をしています。特に，HiRDB/パラレルサーバでは，基本的に各サーバ間で資源の共有をしないので，サーバごとに閉じた排他制御をしています。

HiRDB では，排他資源という単位で排他を掛けて，不正にデータが参照されたり，更新されたりすることを防止しています。

排他資源には包含関係があるため，上位の資源で排他を掛けると，それより下位の資源には排他を掛ける必要がありません。排他資源とその包含関係を次の図に示します。

図 3–11　排他資源とその包含関係



注※1

インナレプリカ機能使用時の最上位排他資源は，インナレプリカ構成管理情報，又はレプリカグループ構成管理情報となります。

インナレプリカ構成管理情報の排他を取得できない場合に，レプリカグループ構成管理情報の排他を取得します。レプリカ RD エリアを定義していない RD エリアにアクセスする場合でも，排他が掛かります。これによって，業務実行中に通常 RD エリアにレプリカ RD エリアが定義されたり，インナレプリカグループ内の構成が変わったりすることを抑止します。

注※2

一時表の実体化時に，一時的に取得します。一時表に対する操作で取得される排他については，マニュアル「HiRDB システム導入・設計ガイド」を参照してください。

注※3

インデクスキー値無排他を適用した場合，キー値には排他を掛けません。

注※4

プラグインが使用するファイルです。

(2)　最小排他資源単位の設定

HiRDB システムが自動的にする排他制御では，表ごとに最小排他資源単位を設定できます。設定できる排他資源は，行，又はページです。

また，インデクスに対してインデクスキー値で排他制御しないようにすることもできます。これを**インデクスキー値無排他**といいます。

(a)　最小排他資源単位を行にした場合

ページ単位の排他と比較して排他資源の単位が小さいため，同時実行性が向上する反面，排他制御による処理時間やメモリ使用量が増加します。

最小排他資源単位を行にするには，CREATE TABLE，ALTER TABLE，又は LOCK 文を使用します。詳細については，マニュアル「HiRDB SQL リファレンス」を参照してください。

(b)　最小排他資源単位をページにした場合

行単位の排他と比較して排他制御による処理時間やメモリ使用量が減少する反面，同時実行性が低下します。

最小排他資源単位をページにするには，CREATE TABLE，ALTER TABLE，又は LOCK 文を使用します。詳細については，マニュアル「HiRDB SQL リファレンス」を参照してください。

(c)　インデクスキー値無排他の場合

インデクスキー値には排他を掛けないで，表に対してだけ排他を掛けます。これによって，次のような問題を回避できます。

- データ更新とインデクス検索との間のデッドロック
- 同一キーを持つデータに対するアクセスが必要もなく待たされる
- 異なるキーを持つデータに対するアクセスが必要もなく待たされる

インデクスキー値無排他については，「インデクスキー値無排他」を参照してください。

(a)～(c)はそれぞれ特徴が異なりますので，特徴を考慮した上で設定してください。

3.4.2　排他制御のモード

(1)　モードの種類

排他制御では，各々の排他資源に対して 5 種類の排他制御モードを用います。

1. **共用モード（PR モード：**Protected Retrieve）
   ほかのトランザクションには参照だけを許すモードです。
2. **排他モード（EX モード：**Exclusive）
   一つのトランザクションだけが排他資源を占有し，ほかのトランザクションには参照，追加，更新及び削除を許さないモードです。
3. **意図共用モード（SR モード：**Shared Retrieve）
   参照だけが許されているモードです。ほかのトランザクションには排他資源の参照，追加，更新及び削除を許すモードです。
4. **意図排他モード（SU モード：**Shared Update）
   参照，追加，更新及び削除が許されているモードです。ほかのトランザクションにも排他資源の参照，追加，更新及び削除を許すモードです。
5. **共用意図排他モード（PU モード：**Protected Update）
   参照，追加，更新及び削除が許されているモードです。ほかのトランザクションには参照だけを許すモードです。このモードは 1.～4.とは異なり，排他制御モードの遷移の結果として発生します。

排他制御では，排他資源の上位から下位へと順番に排他が掛けられます。排他を掛けていく途中で同一資源に対して，一つでもほかのトランザクションと同時実行できないトランザクションがある場合，そのトランザクションは待ち状態になります。また，上位から下位の資源へ排他を掛けていく途中に PR モード，又は EX モードがある場合，そのモードが掛かっている資源より下位の資源には排他が掛かりません。

同一資源に対して 2 ユーザが同じ処理をすると仮定したとき，排他制御モードの違いによって，同時に実行できたりできなかったりします。排他制御モードの違いによる 2 ユーザの同時実行性を次の表に示します。

表 3–8　排他制御モードの違いによる 2 ユーザの同時実行性

モードの種類	SR	PR	SU	PU	EX
SR	○	○	○	○	×
PR	○	○	×	×	×
SU	○	×	○	×	×
PU	○	×	×	×	×
EX	×	×	×	×	×

（凡例）
○：同時に実行できます。
×：同時に実行できません。

2 ユーザが同時に実行できない場合，一般にはトランザクションのコミット待ちになります。ほかのトランザクションのコミット待ちをしないでエラーリターンにする場合，SQL 文に WITH ROLLBACK，又は NO WAIT を指定します。

(2)　モードの遷移

同一ユーザが同一資源に対して，異なる種類の排他制御のモードで二重，三重と次々に排他を掛けていくと，モードは強い方へと遷移していきます。

一度強いモードで排他を掛けてしまうと，後から弱いモードを掛けても，弱いモードへは遷移しません。例えば，行の更新をするときに EX で排他を掛けた場合，更新した行を参照するために PR で排他を掛けたとしても，その行の排他制御のモードは EX のままです。

現在の排他制御のモードに対して，更に排他制御を掛けた場合のモードの遷移規則を次の表に示します。

表 3–9　排他制御のモードの遷移規則

後から掛けるモード	現在のモード				
	SR	PR	SU	PU	EX
SR	–	–	–	–	–
PR	PR	–	PU	–	–
SU	SU	PU	–	–	–
EX	EX	EX	EX	EX	–

（凡例）

–：モードは遷移しません。

–以外：遷移後のモードです。

(3)　モードの組み合わせ

排他制御は，SQL 文の種類，及び実行環境によって，モードの組み合わせが異なります。

SQL 文の種類，及び実行環境の違いによる排他制御のモードの組み合わせの例を次の表に示します。

- 表「排他制御のモードの組み合わせの代表例（行排他の場合）（1/2）」
- 表「排他制御のモードの組み合わせの代表例（行排他の場合）（2/2）」
- 表「排他制御のモードの組み合わせの代表例（ページ排他の場合）（1/2）」
- 表「排他制御のモードの組み合わせの代表例（ページ排他の場合）（2/2）」
- 表「排他制御のモードの組み合わせの代表例（インデクスキー値無排他の場合）（1/2）」
- 表「排他制御のモードの組み合わせの代表例（インデクスキー値無排他の場合）（2/2）」
- 表「排他制御のモードの組み合わせの代表例（検査保留状態に設定する場合）（1/2）」
- 表「排他制御のモードの組み合わせの代表例（検査保留状態に設定する場合）（2/2）」

表 3–10　排他制御のモードの組み合わせの代表例（行排他の場合）（1/2）

SQL 文			資源							
			上位← →下位							
			インナレプリカ構成管理※6	レプリカグループ構成管理※8	RD エリア				表	表（NOWAIT 検索）
					表用	インデクス用	順序数生成子用	最終 HiRDB ファイル※5		
検索	NOWAIT 指定		SR	SR	SR		–	–	–	SR
	WITH SHARE 指定		SR	SR	SR		–	–	SR	–
	WITH EXCLUSIVE 指定※1		SR	SR	SU	SR	–	–	SU	–
	FOR UPDATE 句指定※1		SR	SR	SU	SR	–	–	SU	–
	上記以外		SR	SR	SR		–	–	SR	–
更新※1※12	NEXT VALUE 式指定		SR		SU		SU	EX	SU	–
	上記以外		SR		SU		–	EX	SU	–
追加※1	NEXT VALUE 式指定		SR		SU		SU	EX	SU	–
	上記以外		SR		SU		–	EX	SU	–
削除※1			SR	SR	SU		–	–	SU	–
LOCK 文	SHARE 指定※11		SR	SR	SR	–	–	–	PR	–
	EXCLUSIVE 指定	非共用表	SR	SR	SU	–	–	–	EX	–
		共用表※11	SR	SR	EX	–	–	–	EX	–
表削除※2※13			–	–	SU		–	–	EX	
インデクス定義※13，表定義変更（主キーの追加）※13			–	–	SU		–	–	EX	–
インデクス削除※3※13，表定義変更（主キーの削除）※3※13			–	–	SR※10	SU	–	–	EX※4	EX
全行削除※2※13※14			SR	SR	SU		–	–	EX	
表定義変更（主キーの追加及び削除は除く）※13			SR※9	SR※9	SU※7		–	–	EX	
順序数生成子定義			–	–	–	–	SU	–	–	–

SQL 文	資源							
	上位← →下位							
	インナレプリカ構成管理※6	レプリカグループ構成管理※8	RD エリア				表	表（NOAIT 検索）
			表用	インデクス用	順序数生成子用	最終 HiRDB ファイル※5		
順序数生成子削除	–	–	–	–	SU	–	–	–

（凡例）

–：排他を掛けません。

–以外：排他を掛けるモードです。

注※1

UAP 実行時にデータベースの更新ログを取得しない場合，表に対して EX モードで排他を掛けてコミットまで保持します。行，及びキーに対する排他は掛けません。

なお，UAP 実行時にデータベースの更新ログを取得しない運用については，マニュアル「HiRDB システム運用ガイド」を参照してください。

注※2

表，及びインデクスのすべての使用中セグメントに対して EX モードで排他を掛けてコミットまで保持します。

注※3

インデクスのすべての使用中セグメントに対して EX モードで排他を掛けてコミットまで保持します。

注※4

プラグインインデクスの場合は EX モードで排他が掛かりますが，B-Tree インデクスでは排他は掛かりません。

注※5

RD エリアの自動増分が適用されている場合，自動増分処理の開始から終了まで，RD エリアを構成する最終 HiRDB ファイルに対して排他が掛かります。

注※6

インナレプリカ機能を使用している場合，処理対象 RD エリアがあるサーバに対して排他が掛かります。ただし，pd_inner_replica_lock_shift オペランドに Y を指定している場合は排他が掛かりません。

注※7

RD エリアの追加，及び空き領域の再利用機能の変更をする場合，排他が掛かります。

注※8

カレント DB の変更，レプリカの定義や削除，又は更新可能なオンライン再編成実行時など，インナレプリカの構成に関係する変更をした場合，処理対象となる RD エリアがあるレプリカグループに対して

排他が掛かります。pd_inner_replica_lock_shift オペランドに Y を指定している場合は常に排他が掛かります。

注※9

処理対象の RD エリアに対してアクセスする場合に排他が掛かります。

注※10

インナレプリカ機能を適用している場合に排他が掛かります。

注※11

HiRDB/パラレルサーバの場合，共用表をアクセスするときは，すべてのバックエンドサーバに対して排他が掛かります。

注※12

HiRDB/パラレルサーバの場合，共用表に対してインデクスを更新しない UPDATE 文を実行するときは，すべてのバックエンドサーバに対して排他が掛かります。

注※13

HiRDB/パラレルサーバの場合，共用表及び共用インデクスに対して実行するときは，すべてのバックエンドサーバに対して EXCLUSIVE 指定の LOCK 文相当の排他が掛かります。HiRDB/シングルサーバの場合，共用表及び共用インデクスに対して実行するときは，EXCLUSIVE 指定の LOCK 文相当の排他が掛かります。

注※14

システム定義の pd_check_pending オペランドに USE を指定，又は指定を省略した場合，ディクショナリ表（資源種別：3005，種別名：DICT）に対して一時的に EX モードで排他を掛けます。データディクショナリ用 RD エリア（資源種別：0001，種別名：RDAR）に対しては，トランザクションが終了するまで SU モードで排他を掛けます。

表 3–11　排他制御のモードの組み合わせの代表例（行排他の場合）（2/2）

SQL 文		資源						
		上位←　　→下位						
		インデクス	インデクス情報ファイル※4	順序数生成子	ページ	行	キー値	論理ファイル
検索	NOWAIT 指定	−	−	−	−	−	−	PR
	WITH SHARE 指定	−	−	−	−	PR	PR	PR
	WITH EXCLUSIVE 指定※1	−	−	−	−	EX	PR	EX
	FOR UPDATE 句指定※1	−	−	−	−	EX	PR	EX
	上記以外	−	−	−	−	PR	PR	PR
更新※1※6	NEXT VALUE 式指定	−	EX	SU	−	EX	EX	EX

SQL 文			資源						
			上位← →下位						
			インデクス	インデクス情報ファイル※4	順序数生成子	ページ	行	キー値	論理ファイル
	上記以外		–	EX	–	–	EX	EX	EX
追加※1	NEXT VALUE 式指定		–	EX	SU	–	EX	EX	EX
	上記以外		–	EX	–	–	EX	EX	EX
削除※1			–	–	–	–	EX※8	EX	EX
LOCK 文	SHARE 指定※5		–	–	–	–	–	–	–
	EXCLUSIVE 指定	非共用表	–	–	–	–	–	–	–
		共用表※5	–	–	–	–	–	–	–
表削除※2※7			–	–	–	–	–	–	–
インデクス定義※7，表定義変更（主キーの追加）※7			–	–	–	–	–	–	–
インデクス削除※3※7，表定義変更（主キーの削除）※3※7			EX	–	–	–	–	–	–
全行削除※2※7※9			–	–	–	–	–	–	–
表定義変更（主キーの追加及び削除は除く）※7			–	–	–	–	–	–	–
順序数生成子定義			–	–	EX	–	–	–	–
順序数生成子削除			–	–	EX	–	–	–	–

（凡例）

–：排他を掛けません。

–以外：排他を掛けるモードです。

注※1

UAP 実行時にデータベースの更新ログを取得しない場合，表に対して EX モードで排他を掛けてコミットまで保持します。行，及びキーに対する排他は掛けません。

なお，UAP 実行時にデータベースの更新ログを取得しない運用については，マニュアル「HiRDB システム運用ガイド」を参照してください。

注※2

表，及びインデクスのすべての使用中セグメントに対して EX モードで排他を掛けてコミットまで保持します。

注※3

インデクスのすべての使用中セグメントに対して EX モードで排他を掛けてコミットまで保持します。

注※4

プラグインインデクスの遅延一括作成機能を使用して，プラグインインデクスに対して更新をしたときに掛ける排他です。この排他は，コミットまで保持されます。

注※5

HiRDB/パラレルサーバの場合，共用表をアクセスするときは，すべてのバックエンドサーバに対して排他が掛かります。

注※6

HiRDB/パラレルサーバの場合，共用表に対してインデクスを更新しない UPDATE 文を実行するときは，すべてのバックエンドサーバに対して排他が掛かります。

注※7

HiRDB/パラレルサーバの場合，共用表及び共用インデクスに対して実行するときは，すべてのバックエンドサーバに対して EXCLUSIVE 指定の LOCK 文相当の排他が掛かります。

注※8

コミット，又はロールバックするまで EX モードで排他が掛かります。ただし，検索処理は削除行に対して排他を掛けないため，排他待ちになりません。

注※9

システム定義の pd_check_pending オペランドに USE を指定，又は指定を省略した場合，ディクショナリ表（資源種別：3005，種別名：DICT）に対して一時的に EX モードで排他を掛けます。データディクショナリ用 RD エリア（資源種別：0001，種別名：RDAR）に対しては，トランザクションが終了するまで SU モードで排他を掛けます。

表 3–12　排他制御のモードの組み合わせの代表例（ページ排他の場合）（1/2）

SQL 文		資源							
		上位←　　→下位							
		インナレプリカ構成管理※6	レプリカグループ構成管理※8	RD エリア				表	表（NOWAIT 検索）
				表用	インデクス用	順序数生成子用	最終 HiRDB ファイル※5		
検索	NOWAIT 指定	SR	SR	SR		–	–	–	SR
	WITH SHARE 指定	SR	SR	SR		–	–	SR	–
	WITH EXCLUSIVE 指定※1	SR	SR	SU	SR	–	–	SU	–

SQL 文			資源							
			上位← →下位							
			インナレプリカ構成管理※6	レプリカグループ構成管理※8	RD エリア				表	表（NOWAIT 検索）
					表用	インデクス用	順序数生成子用	最終 HiRDB ファイル※5		
	FOR UPDATE 句指定※1		SR	SR	SU	SR	–	–	SU	–
	上記以外		SR	SR	SR		–	–	SR	–
更新※1※12	NEXT VALUE 式指定		SR	SR	SU		SU	EX	SU	–
	上記以外		SR	SR	SU		–	EX	SU	–
追加※1	NEXT VALUE 式指定		SR	SR	SU		SU	EX	SU	–
	上記以外		SR	SR	SU		–	EX	SU	–
削除※1			SR	SR	SU		–	–	SU	–
LOCK 文	SHARE 指定※11		SR	SR	SR	–	–	–	PR	–
	EXCLUSIVE 指定	非共用表	SR	SR	SU	–	–	–	EX	–
		共用表※11	SR	SR	EX	–	–	–	EX	–
表削除※2※13			–	–	SU		–	–	EX	EX
インデクス定義※13，表定義変更（主キーの追加）※13			–	–	SU		–	–	EX	–
インデクス削除※3※13，表定義変更（主キーの削除）※3※13			–	–	SR※10	SU	–	–	EX※4	EX
全行削除※2※13※14			SR	SR	SU		–	–	EX	EX
表定義変更（主キーの追加及び削除は除く）※13			SR※9	SR※9	SU※7		–	–	EX	EX
順序数生成子定義			–	–	–		SU	–	–	–
順序数生成子削除			–	–	–		SU	–	–	–

（凡例）

–：排他を掛けません。

–以外：排他を掛けるモードです。

注※1

UAP 実行時にデータベースの更新ログを取得しない場合，表に対して EX モードで排他を掛けてコミットまで保持します。ページ，及びキーに対する排他は掛けません。

なお，UAP 実行時にデータベースの更新ログを取得しない運用については，マニュアル「HiRDB システム運用ガイド」を参照してください。

注※2

表，及びインデクスのすべての使用中セグメントに対して EX モードで排他を掛けてコミットまで保持します。

注※3

インデクスのすべての使用中セグメントに対して EX モードで排他を掛けてコミットまで保持します。

注※4

プラグインインデクスの場合は EX モードで排他が掛かりますが，B-Tree インデクスでは排他は掛かりません。

注※5

RD エリアの自動増分が適用されている場合，自動増分処理の開始から終了まで，RD エリアを構成する最終 HiRDB ファイルに対して排他が掛かります。

注※6

インナレプリカ機能を使用している場合，処理対象 RD エリアがあるサーバに対して排他が掛かります。ただし，pd_inner_replica_lock_shift オペランドに Y を指定している場合は排他が掛かりません。

注※7

RD エリアの追加，及び空き領域の再利用機能の変更をする場合，排他が掛かります。

注※8

カレント DB の変更，レプリカの定義や削除，又は更新可能なオンライン再編成実行時など，インナレプリカの構成に関係する変更をした場合，処理対象となる RD エリアがあるレプリカグループに対して排他が掛かります。pd_inner_replica_lock_shift オペランドに Y を指定している場合は常に排他が掛かります。

注※9

処理対象の RD エリアに対してアクセスする場合に排他が掛かります。

注※10

インナレプリカ機能を適用している場合に排他が掛かります。

注※11

HiRDB/パラレルサーバの場合，共用表をアクセスするときは，すべてのバックエンドサーバに対して排他が掛かります。

注※12

HiRDB/パラレルサーバの場合，共用表に対してインデクスを更新しない UPDATE 文を実行するときは，すべてのバックエンドサーバに対して排他が掛かります。

注※13

HiRDB/パラレルサーバの場合，共用表及び共用インデクスに対して実行するときは，すべてのバックエンドサーバに対して EXCLUSIVE 指定の LOCK 文相当の排他が掛かります。HiRDB/シングルサーバの場合，共用表及び共用インデクスに対して実行するときは，EXCLUSIVE 指定の LOCK 文相当の排他が掛かります。

注※14

システム定義の pd_check_pending オペランドに USE を指定，又は指定を省略した場合，ディクショナリ表（資源種別：3005，種別名：DICT）に対して一時的に EX モードで排他を掛けます。データディクショナリ用 RD エリア（資源種別：0001，種別名：RDAR）に対しては，トランザクションが終了するまで SU モードで排他を掛けます。

表 3–13　排他制御のモードの組み合わせの代表例（ページ排他の場合）（2/2）

SQL 文			資源						
			上位←　　→下位						
			インデクス	インデクス情報ファイル※4	順序数生成子	ページ	行	キー値	論理ファイル
検索	NOWAIT 指定		−	−	−	−	−	−	PR
	WITH SHARE 指定		−	−	−	PR	−	PR	PR
	WITH EXCLUSIVE 指定※1		−	−	−	EX	−	PR	EX
	FOR UPDATE 句指定※1		−	−	−	EX	−	PR	EX
	上記以外		−	−	−	PR	−	PR	PR
更新※1※6	NEXT VALUE 式指定		−	EX	SU	EX	−	EX	EX
	上記以外		−	EX	−	EX	−	EX	EX
追加※1	NEXT VALUE 式指定		−	EX	SU	EX	−	EX	EX
	上記以外		−	EX	−	EX	−	EX	EX
削除※1			−	−	−	EX	−	EX	EX
LOCK 文	SHARE 指定※5		−	−	−	−	−	−	−
	EXCLUSIVE 指定	非共用表	−	−	−	−	−	−	−

SQL 文	資源						
	上位← →下位						
	インデクス	インデクス情報ファイル※4	順序数生成子	ページ	行	キー値	論理ファイル
共用表※5	–	–	–	–	–	–	–
表削除※2※7	–	–	–	–	–	–	–
インデクス定義※7，表定義変更（主キーの追加）※7	–	–	–	–	–	–	–
インデクス削除※3※7，表定義変更（主キーの削除）※3※7	EX	–	–	–	–	–	–
全行削除※2※7※8	–	–	–	–	–	–	–
表定義変更（主キーの追加及び削除は除く）※7	–	–	–	–	–	–	–
順序数生成子定義	–	–	EX	–	–	–	–
順序数生成子削除	–	–	EX	–	–	–	–

（凡例）

–：排他を掛けません。

–以外：排他を掛けるモードです。

注※1

UAP 実行時にデータベースの更新ログを取得しない場合，表に対して EX モードで排他を掛けてコミットまで保持します。ページ，及びキーに対する排他は掛けません。

なお，UAP 実行時にデータベースの更新ログを取得しない運用については，マニュアル「HiRDB システム運用ガイド」を参照してください。

注※2

表，及びインデクスのすべての使用中セグメントに対して EX モードで排他を掛けてコミットまで保持します。

注※3

インデクスのすべての使用中セグメントに対して EX モードで排他を掛けてコミットまで保持します。

注※4

プラグインインデクスの遅延一括作成機能を使用して，プラグインインデクスに対して更新をしたときに掛ける排他です。この排他は，コミットまで保持されます。

注※5

HiRDB/パラレルサーバの場合，共用表をアクセスするときは，すべてのバックエンドサーバに対して排他が掛かります。

注※6

HiRDB/パラレルサーバの場合，共用表に対してインデクスを更新しない UPDATE 文を実行するときは，すべてのバックエンドサーバに対して排他が掛かります。

注※7

HiRDB/パラレルサーバの場合，共用表及び共用インデクスに対して実行するときは，すべてのバックエンドサーバに対して EXCLUSIVE 指定の LOCK 文相当の排他が掛かります。

注※8

システム定義の pd_check_pending オペランドに USE を指定，又は指定を省略した場合，ディクショナリ表（資源種別：3005，種別名：DICT）に対して一時的に EX モードで排他を掛けます。データディクショナリ用 RD エリア（資源種別：0001，種別名：RDAR）に対しては，トランザクションが終了するまで SU モードで排他を掛けます。

表 3–14　排他制御のモードの組み合わせの代表例（インデクスキー値無排他の場合）（1/2）

SQL 文		資源							
		上位←　　→下位							
		インナレプリカ構成管理※5	レプリカグループ構成管理※7	RD エリア				表	表（NOWAIT 検索）
				表用	インデクス用	順序数生成子用	最終 HiRDB ファイル※4		
検索	NOWAIT 指定	SR	SR	SR		–	–	–	SR
	WITH SHARE 指定	SR	SR	SR		–	–	SR	–
	WITH EXCLUSIVE 指定※10	SR	SR	SU	SR	–	–	SU	SU
	FOR UPDATE 句指定※10	SR	SR	SU	SR	–	–	SU	SU
	上記以外	SR	SR	SR		–	–	SR	–
更新※10※12	NEXT VALUE 式指定	SR	SR	SU		SU	EX	SU	–
	上記以外	SR	SR	SU		–	EX	SU	–
追加※10	NEXT VALUE 式指定	SR	SR	SU		SU	EX	SU	–
	上記以外	SR	SR	SU		–	EX	SU	–
削除※10		SR	SR	SU		–	–	SU	–
LOCK 文	SHARE 指定	SR	SR	SR	–	–	–	PR	–

SQL 文			資源							
			上位← →下位							
			インナレプリカ構成管理※5	レプリカグループ構成管理※7	RD エリア				表	表（NOWAIT 検索）
					表用	インデクス用	順序数生成子用	最終 HiRDB ファイル※4		
	※11									
	EXCLUSIVE 指定	非共用表	SR	SR	–	–	–	–	EX	–
		共用表※11	SR	SR	–	EX	–	–	EX	–
表削除※1※13			–	–	SU		–	–	EX	EX
インデクス定義※13，表定義変更（主キーの追加）※13			–	–	SU		–	–	EX	–
インデクス削除※2※13，表定義変更（主キーの削除）※2※13			–	–	SR※9	–	–	–	EX※3	EX
全行削除※1※13※14			SR	SR	SU		–	–	EX	EX
表定義変更（主キーの追加及び削除は除く）※13			SR※8	SR※8	SU※6		–	–	EX	EX
順序数生成子定義			–	–	–		SU	–	–	–
順序数生成子削除			–	–	–		SU	–	–	–

（凡例）

–：排他を掛けません。

–以外：排他を掛けるモードです。

注※1

表，及びインデクスのすべての使用中セグメントに対して EX モードで排他を掛けて，コミットまで保持します。

注※2

インデクスのすべての使用中セグメントに対して EX モードで排他を掛けて，コミットまで保持します。

注※3

プラグインインデクスの場合は EX モードで排他が掛かりますが，B-Tree インデクスでは排他は掛かりません。

注※4

RD エリアの自動増分が適用されている場合，自動増分処理の開始から終了まで，RD エリアを構成する最終 HiRDB ファイルに対して排他が掛かります。

注※5

インナレプリカ機能を使用している場合，処理対象 RD エリアがあるサーバに対して排他が掛かります。ただし，pd_inner_replica_lock_shift オペランドに Y を指定している場合は排他が掛かりません。

注※6

RD エリアの追加，及び空き領域の再利用機能の変更をする場合，排他が掛かります。

注※7

カレント DB の変更，レプリカの定義や削除，又は更新可能なオンライン再編成実行時など，インナレプリカの構成に関係する変更をした場合，処理対象となる RD エリアがあるレプリカグループに対して排他が掛かります。pd_inner_replica_lock_shift オペランドに Y を指定している場合は常に排他が掛かります。

注※8

処理対象の RD エリアに対してアクセスする場合に排他が掛かります。

注※9

インナレプリカ機能を適用している場合に排他が掛かります。

注※10

UAP 実行時にデータベースの更新ログを取得しない場合，表に対して EX モードで排他を掛けて，コミットまで保持します。行及びキー値には排他を掛けません。なお，UAP 実行時にデータベースの更新ログを取得しない運用については，マニュアル「HiRDB システム運用ガイド」を参照してください。

注※11

HiRDB/パラレルサーバの場合，共用表をアクセスするときは，すべてのバックエンドサーバに対して排他が掛かります。

注※12

HiRDB/パラレルサーバの場合，共用表に対してインデクスを更新しない UPDATE 文を実行するときは，すべてのバックエンドサーバに対して排他が掛かります。

注※13

HiRDB/パラレルサーバの場合，共用表及び共用インデクスに対して実行するときは，すべてのバックエンドサーバに対して EXCLUSIVE 指定の LOCK 文相当の排他が掛かります。HiRDB/シングルサーバの場合，共用表及び共用インデクスに対して実行するときは，EXCLUSIVE 指定の LOCK 文相当の排他が掛かります。

注※14

システム定義の pd_check_pending オペランドに USE を指定，又は指定を省略した場合，ディクショナリ表（資源種別：3005，種別名：DICT）に対して一時的に EX モードで排他を掛けます。データディクショナリ用 RD エリア（資源種別：0001，種別名：RDAR）に対しては，トランザクションが終了するまで SU モードで排他を掛けます。

表 3–15　排他制御のモードの組み合わせの代表例（インデクスキー値無排他の場合）（2/2）

SQL 文			資源						
			上位←　　→下位						
			インデクス	インデクス情報ファイル※3	順序数生成子	ページ	行	キー値	論理ファイル
検索	NOWAIT 指定		−	−	−	−	−	−	PR
	WITH SHARE 指定		−	−	−	−，PR ※4	PR，− ※4	−	PR
	WITH EXCLUSIVE 指定※6		−	−	−	−，EX ※4	EX，− ※4	−	EX
	FOR UPDATE 句指定※6		−	−	−	−，EX ※4	EX，− ※4	−	EX
	上記以外		−	−	−	−，PR ※4	PR，− ※4	−	PR
更新※6※8	NEXT VALUE 式指定		−	EX	SU	−，EX ※4※5	EX，− ※4※5	−	EX
	上記以外		−	EX	−	−，EX ※4※5	EX，− ※4※5	−	EX
追加※6	NEXT VALUE 式指定		−	EX	SU	−，EX ※4※5	EX，− ※4※5	−	EX
	上記以外		−	EX	−	−，EX ※4※5	EX，− ※4※5	−	EX
削除※6			−	−	−	−，EX ※4※5	EX，− ※4※5	−	EX
LOCK 文	SHARE 指定※7		−	−	−	−	−	−	−
	EXCLUSIVE 指定	非共用表	−	−	−	−	−	−	−
		共用表※7	−	−	−	−	−	−	−
表削除※1※9			−	−	−	−	−	−	−
インデクス定義※9，表定義変更（主キーの追加）※9			−	−		−	−	−	−
インデクス削除※2※9，表定義変更（主キーの削除）※2※9			EX	−	−	−	−	−	−
全行削除※1※9※10			−	−	−	−	−	−	−

SQL 文	資源						
	上位← →下位						
	インデクス	インデクス情報ファイル※3	順序数生成子	ページ	行	キー値	論理ファイル
表定義変更（主キーの追加及び削除は除く）※9	−	−	−	−	−	−	−
順序数生成子定義	−	−	EX	−	−	−	−
順序数生成子削除	−	−	EX	−	−	−	−

（凡例）

−：排他を掛けません。

−以外：排他を掛けるモードです。

注※1

表，及びインデクスのすべての使用中セグメントに対して EX モードで排他を掛けて，コミットまで保持します。

注※2

インデクスのすべての使用中セグメントに対して EX モードで排他を掛けて，コミットまで保持します。

注※3

プラグインインデクスの遅延一括作成機能を使用して，プラグインインデクスに対して更新をした場合に掛ける排他です。コミットまで保持します。

注※4

行排他の場合，資源「行」に排他を掛けて，資源「ページ」には排他を掛けません。

ページ排他の場合，資源「行」に排他を掛けないで，資源「ページ」に排他を掛けます。

注※5

ユニークインデクスが定義されている場合，ページ排他の場合でも資源「行」に排他が掛かります。

注※6

UAP 実行時にデータベースの更新ログを取得しない場合，表に対して EX モードで排他を掛けて，コミットまで保持します。行及びキー値には排他を掛けません。なお，UAP 実行時にデータベースの更新ログを取得しない運用については，マニュアル「HiRDB システム運用ガイド」を参照してください。

注※7

HiRDB/パラレルサーバの場合，共用表をアクセスするときは，すべてのバックエンドサーバに対して排他が掛かります。

注※8

HiRDB/パラレルサーバの場合，共用表に対してインデクスを更新しない UPDATE 文を実行するときは，すべてのバックエンドサーバに対して排他が掛かります。

注※9

HiRDB/パラレルサーバの場合，共用表及び共用インデクスに対して実行するときは，すべてのバックエンドサーバに対して EXCLUSIVE 指定の LOCK 文相当の排他が掛かります。

注※10

システム定義の pd_check_pending オペランドに USE を指定，又は指定を省略した場合，ディクショナリ表（資源種別：3005，種別名：DICT）に対して一時的に EX モードで排他を掛けます。データディクショナリ用 RD エリア（資源種別：0001，種別名：RDAR）に対しては，トランザクションが終了するまで SU モードで排他を掛けます。

表 3–16　排他制御のモードの組み合わせの代表例（検査保留状態に設定する場合）（1/2）

SQL 文及び ユティリティ	資源※1					
	上位←　　→下位					
	RD エリア				表	表（NOWAIT 検索）
	表用※2	インデクス用	LOB 用	最終 HiRDB ファイル		
全行削除※4	SU	–	–	–	EX	EX
表定義変更（分割格納条件変更）	SU	–	–	–	EX	EX
データベース作成ユティリティ（pdload）※3	SU	–	–	–	EX	EX
データベース再編成ユティリティ（pdrorg）※3	SU	–	–	–	EX	EX
データベース構成変更ユティリティ（pdmod）	SU	–	–	–	EX	EX
整合性チェックユティリティ（pdconstck）※3	SU	–	–	–	EX	EX
オンライン再編成の追い付き反映コマンド（pdorend）※3	SU	–	–	–	EX	EX

（凡例）

–：排他を掛けません。

–以外：排他を掛けるモードです。

注※1

参照制約，又は検査制約を定義した表に関する資源を示します。

注※2

検査保留状態に設定する RD エリアに対して排他を掛けます。

注※3

HiRDB/パラレルサーバの場合，共用表に対して実行するときは，すべてのバックエンドサーバに対して EXCLUSIVE 指定の LOCK 文相当の排他が掛かります。共用表の LOCK 文実行時に掛かる排他制御モードについては，次に示す表の LOCK 文の EXCLUSIVE 指定の行を参照してください。

- 表「排他制御のモードの組み合わせの代表例（行排他の場合）（1/2）」
- 表「排他制御のモードの組み合わせの代表例（行排他の場合）（2/2）」
- 表「排他制御のモードの組み合わせの代表例（ページ排他の場合）（1/2）」
- 表「排他制御のモードの組み合わせの代表例（ページ排他の場合）（2/2）」
- 表「排他制御のモードの組み合わせの代表例（インデクスキー値無排他の場合）（1/2）」
- 表「排他制御のモードの組み合わせの代表例（インデクスキー値無排他の場合）（2/2）」

注※4

システム定義の pd_check_pending オペランドに USE を指定，又は指定を省略した場合，ディクショナリ表（資源種別：3005，種別名：DICT）に対して一時的に EX モードで排他を掛けます。データディクショナリ用 RD エリア（資源種別：0001，種別名：RDAR）に対しては，トランザクションが終了するまで SU モードで排他を掛けます。

表 3–17　排他制御のモードの組み合わせの代表例（検査保留状態に設定する場合）（2/2）

SQL 文及びユティリティ	資源※1					
	上位←　　→下位					
	インデクス	インデクス情報ファイル	ページ	行	キー値	論理ファイル
全行削除※3	－	－	－	－	－	－
表定義変更（分割格納条件変更）	－	－	－	－	－	－
データベース作成ユティリティ（pdload）※2	－	－	－	－	－	－
データベース再編成ユティリティ（pdrorg）※2	－	－	－	－	－	－
データベース構成変更ユティリティ（pdmod）	－	－	－	－	－	－
整合性チェックユティリティ（pdconstck）※2	－	－	－	－	－	－
オンライン再編成の追い付き反映コマンド（pdorend）※2	－	－	－	－	－	－

（凡例）

－：排他を掛けません。

注※1

参照制約，又は検査制約を定義した表に関する資源を示します。

注※2

HiRDB/パラレルサーバの場合，共用表に対して実行するときは，すべてのバックエンドサーバに対して EXCLUSIVE 指定の LOCK 文相当の排他が掛かります。共用表の LOCK 文実行時に掛かる排他制御モードについては，次に示す表の LOCK 文の EXCLUSIVE 指定の行を参照してください。

- 表「排他制御のモードの組み合わせの代表例（行排他の場合）（1/2）」
- 表「排他制御のモードの組み合わせの代表例（行排他の場合）（2/2）」
- 表「排他制御のモードの組み合わせの代表例（ページ排他の場合）（1/2）」
- 表「排他制御のモードの組み合わせの代表例（ページ排他の場合）（2/2）」
- 表「排他制御のモードの組み合わせの代表例（インデクスキー値無排他の場合）（1/2）」
- 表「排他制御のモードの組み合わせの代表例（インデクスキー値無排他の場合）（2/2）」

注※3

システム定義の pd_check_pending オペランドに USE を指定，又は指定を省略した場合，ディクショナリ表（資源種別：3005，種別名：DICT）に対して一時的に EX モードで排他を掛けます。データディクショナリ用 RD エリア（資源種別：0001，種別名：RDAR）に対しては，トランザクションが終了するまで SU モードで排他を掛けます。

(4) CREATE TABLE に WITHOUT ROLLBACK オプションを指定した表の排他解除タイミング

CREATE TABLE に WITHOUT ROLLBACK オプションを指定した表の排他解除タイミングを次の表に示します。

- 表「CREATE TABLE に WITHOUT ROLLBACK オプションを指定した表の排他解除タイミング（インデクスが定義されていない場合（1/2））」
- 表「CREATE TABLE に WITHOUT ROLLBACK オプションを指定した表の排他解除タイミング（インデクスが定義されていない場合（2/2））」
- 表「CREATE TABLE に WITHOUT ROLLBACK オプションを指定した表の排他解除タイミング（インデクスが定義されている場合（1/2））」
- 表「CREATE TABLE に WITHOUT ROLLBACK オプションを指定した表の排他解除タイミング（インデクスが定義されている場合（2/2））」

表 3–18　CREATE TABLE に WITHOUT ROLLBACK オプションを指定した表の排他解除タイミング（インデクスが定義されていない場合（1/2））

SQL 文		資源						
		上位←　　→下位						
		インナレプリカ構成管理※	レプリカグループ構成管理※	RD エリア			表	表（NOWAIT 検索）
				表用	インデクス用	最終 HiRDB ファイル		
検索	NOWAIT 指定	×	×	×	−	−	−	×
	WITH SHARE 指定	×	×	×	−	−	×	−
	WITH EXCLUSIVE 指定	×	×	×	−	−	×	−
	FOR UPDATE 句指定	×	×	×	−	−	×	−
	上記以外	×	×	×	−	−	×	−
更新		×	×	×	−	−	×	−
追加		×	×	×	−	−	×	−
削除		×	×	×	−	−	×	−
LOCK 文	SHARE 指定	×	×	×	−	−	×	−
	EXCLUSIVE 指定	×	×	×	−	−	×	−
表削除		−	−	×	−	−	×	×
インデクス	定義	−	−	×	−	−	×	×
	削除	−	−	×	−	−	×	×
全行削除		×	×	×	−	−	×	×
表定義変更		−	−	−	−	−	×	×

（凡例）

−：排他が掛からない，又は該当しない（ページ排他は指定できない）ことを示します。

×：SQL 実行時に排他が解除されないことを示します。

注※

インナレプリカ機能を使用している場合，インナレプリカ構成管理に排他が掛かります。更新可能なオンライン再編成を使用している場合，インナレプリカ構成管理又はレプリカグループ構成管理に排他が掛かります。

表 3–19　CREATE TABLE に WITHOUT ROLLBACK オプションを指定した表の排他解除タイミング（インデクスが定義されていない場合（2/2））

SQL 文		資源					
		上位← →下位					
		インデクス	インデクス情報ファイル	ページ	行	キー値	論理ファイル
検索	NOWAIT 指定	−	−	−	−	−	−
	WITH SHARE 指定	−	−	−	×	−	−
	WITH EXCLUSIVE 指定	−	−	−	×	−	−
	FOR UPDATE 句指定	−	−	−	×	−	−
	上記以外	−	−	−	×	−	−
更新		−	−	−	○	−	−
追加		−	−	−	○	−	−
削除		−	−	−	○	−	−
LOCK 文	SHARE 指定	−	−	−	−	−	−
	EXCLUSIVE 指定	−	−	−	−	−	−
表削除		−	−	−	−	−	−
インデクス	定義	−	−	−	−	−	−
	削除	−	−	−	−	−	−
全行削除		−	−	−	−	−	−
表定義変更		−	−	−	−	−	−

（凡例）

−：排他が掛からない，又は該当しない（ページ排他は指定できない）ことを示します。

○：SQL 実行時に排他が解除されることを示します。

×：SQL 実行時に排他が解除されないことを示します。

表 3–20　CREATE TABLE に WITHOUT ROLLBACK オプションを指定した表の排他解除タイミング（インデクスが定義されている場合（1/2））

SQL 文		資源						
		上位←　　→下位						
		インナレプリカ構成管理※	レプリカグループ構成管理※	RD エリア			表	表（NOWAIT 検索）
				表用	インデクス用	最終 HiRDB ファイル		
検索	NOWAIT 指定	×	×	×	×	−	−	×
	WITH SHARE 指定	×	×	×	×	−	×	−
	WITH EXCLUSIVE 指定	×	×	×	×	−	×	−
	FOR UPDATE 句指定	×	×	×	×	−	×	−
	上記以外	×	×	×	×	−	×	−
更新		×	×	×	×	−	×	−
追加		×	×	×	×	−	×	−
削除		×	×	×	×	−	×	−
LOCK 文	SHARE 指定	×	×	×	×	−	×	−
	EXCLUSIVE 指定	×	×	×	×	−	×	−
表削除		−	−	×	×	−	×	×
インデクス	定義	−	−	×	×	−	×	×
	削除	−	−	×	×	−	×	×
全行削除		×	×	×	×	−	×	×
表定義変更		−	−	−	×	−	×	×

（凡例）

−：排他が掛からない，又は該当しない（ページ排他は指定できない）ことを示します。

×：SQL 実行時に排他が解除されないことを示します。

注※

インナレプリカ機能を使用している場合，インナレプリカ構成管理に排他が掛かります。更新可能なオンライン再編成を使用している場合，インナレプリカ構成管理又はレプリカグループ構成管理に排他が掛かります。

表 3–21　CREATE TABLE に WITHOUT ROLLBACK オプションを指定した表の排他解除タイミング（インデクスが定義されている場合（2/2））

SQL 文		資源					
		上位← →下位					
		インデクス	インデクス情報ファイル	ページ	行	キー値	論理ファイル
検索	NOWAIT 指定	−	−	−	−	−	−
	WITH SHARE 指定	−	−	−	×	−※1	−
	WITH EXCLUSIVE 指定	−	−	−	×	−※1	−
	FOR UPDATE 句指定	−	−	−	×	−※1	−
	上記以外	−	−	−	×	−※1	−
更新		−	−	−	○	−※2	−
追加		−	−	−	○※4	−※3	−
削除		−	−	−	○※4	−※3	−
LOCK 文	SHARE 指定	−	−	−	−	−	−
	EXCLUSIVE 指定	−	−	−	−	−	−
表削除		−	−	−	−	−	−
インデクス	定義	−	−	−	−	−	−
	削除	−	−	−	−	−	−
全行削除		−	−	−	−	−	−
表定義変更		−	−	−	−	−	−

（凡例）

−：排他が掛からない，又は該当しない（インデクス定義及びページ排他は指定できない）ことを示します。

○：SQL 実行時に排他が解除されることを示します。

×：SQL 実行時に排他が解除されないことを示します。

注※1

システム定義の pd_indexlock_mode オペランドが KEY（インデクス排他あり）の場合，処理対象のキー値を別のキー値に変更したときに排他が解除されます。

注※2

ユニークキーインデクスの場合，○となります。

注※3

システム定義の pd_indexlock_mode オペランドが KEY（インデクス排他あり）の場合，×となります。

注※4

システム定義の pd_idx_without_rollback オペランドが N の場合，×となります。

3.4.3　排他の期間

(1)　排他の開始と解放

1 トランザクションがある資源に対して排他を掛けると，一般にコミット，又はロールバックをするまで資源を占有します。例えば，ある排他資源（行，又はページ）に対して更新処理を掛けると EX モードになるため，更新中の行に対するほかのトランザクション処理は，コミット，又はロールバックをするまですべて待ち状態になります。ただし，LOCK 文で UNTIL DISCONNECT 指定がある場合，その資源の排他は，DISCONNECT 時，又はそのテーブル削除後のコミットまで保持されます。

なお，通常，行の削除処理をするとトランザクション完了まで排他は保持していますが，データベースから行がなくなるため，ほかのトランザクションの検索処理は削除中の行で排他待ちになりません。排他待ちをさせたい場合は，「コミットしていない削除データの排他制御」を参照してください。

(2)　排他時の参照

ある資源に排他制御が掛かると，通常はコミット，又はロールバックをするまで資源は解放されません。しかし，WITHOUT LOCK を指定した SQL 文の検索では，参照済みの排他資源（行又はページ）の排他は解除されます。また，論理ファイルを参照する場合を除き，WITHOUT LOCK NOWAIT を指定した SQL 文の検索では，ほかのトランザクションが表や行に EX モードで排他を掛けているときでも，排他なしと同じ状態として参照できます。ただし，pdload 及び pdrorg でアクセス中の表は参照できません。pdload 及び pdrorg については，マニュアル「HiRDB コマンドリファレンス」を参照してください。

なお，WITHOUT LOCK NOWAIT を指定した SQL 文の検索では，更新中でも参照できるため，参照時の結果と更新後の結果が必ずしも一致しないので注意が必要です。

3.4.4　デッドロックと回避策

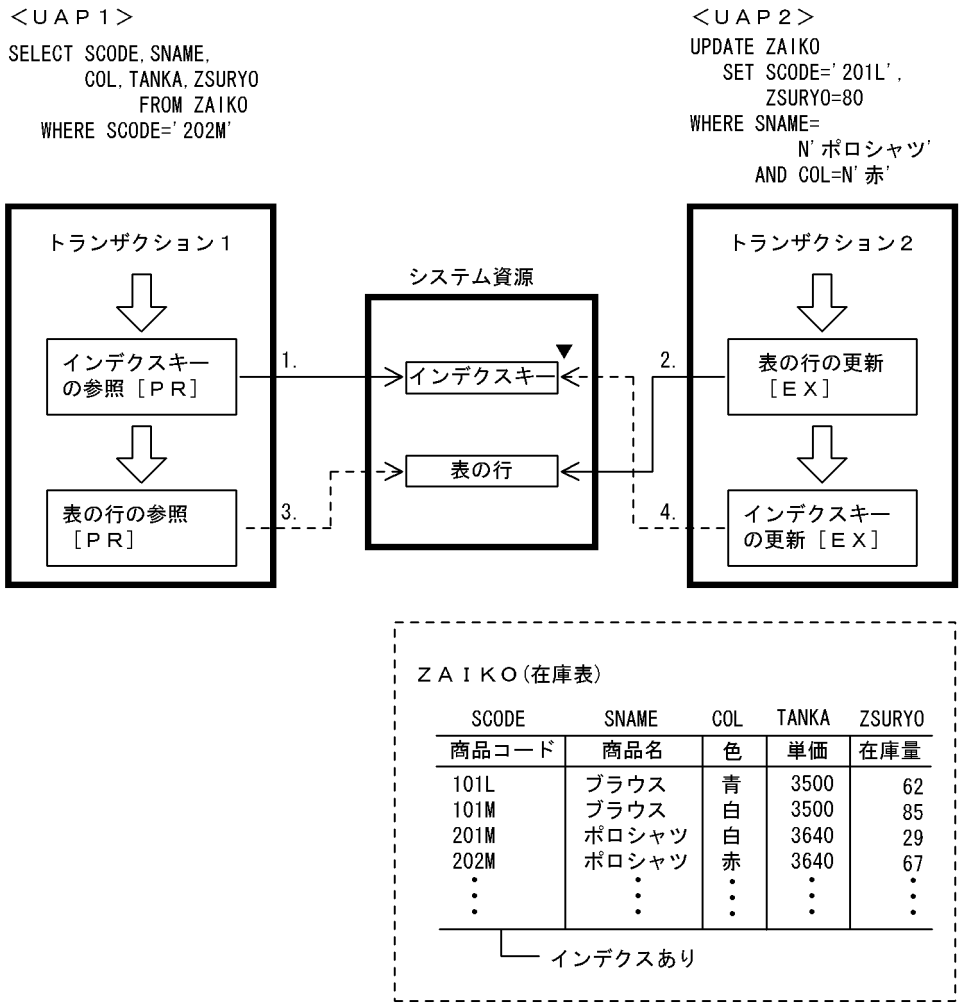
(1)　デッドロックの発生要因

二つのトランザクションが二つ以上の資源の確保をめぐって互いに相手を待つ状態となり，そこから先へ処理が進まなくなることをデッドロックといいます。

デッドロックは，一般に参照のトランザクションと更新（削除を含む）のトランザクションとの間で多く発生します。したがって，UAP のアクセス順序を変えることで，デッドロックの発生頻度を低減させることができます。

デッドロックの例として，二つのトランザクションが同一キーを持った行に対して同時に実行した場合に排他の掛かる順序とデッドロックの関係を次の図に示します。

図 3–12　デッドロックの例

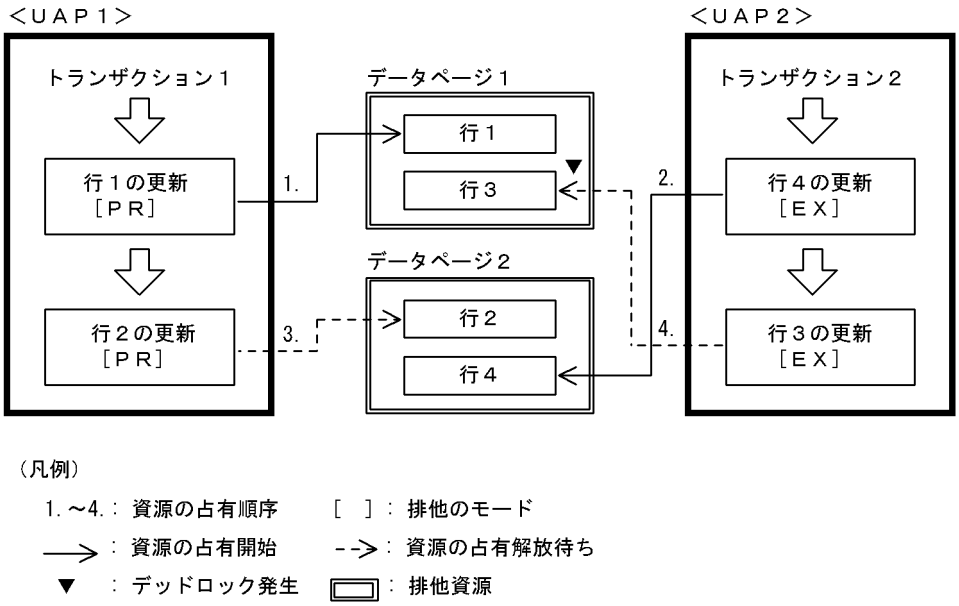


（凡例）　1.～4.：資源の占有順序
［　］：排他のモード
⟶：資源の占有開始
- - ->：資源の占有解放待ち
▼　：デッドロック発生

また，ページ排他の場合には，UAP のアクセス手順を統一しても，デッドロックが回避できないことがあります。

ページ排他でのデッドロックの例を次の図に示します。

図 3–13　ページ排他でのデッドロックの例



図「ページ排他でのデッドロックの例」で示した例の場合，クラスタキーを指定していないと，ページへの行の格納順序を一定にできないため，ページ単位に UAP のアクセス順序を統一できません。このような場合には，ALTER TABLE でページ排他を行排他に変更してデッドロックを回避してください。

(2)　サーバ間で発生するデッドロック

デッドロックは，一つのサーバ内で発生する以外にサーバとサーバの間でも発生します。HiRDB/パラレルサーバでは，サーバ間で発生するデッドロックのことをグローバルデッドロックといいます。

グローバルデッドロックは，一般に図「ページ排他でのデッドロックの例」で示したような一つのサーバ内で発生するデッドロックと同様に，参照のトランザクションと更新のトランザクションとの間で発生します。したがって，UAP のアクセス順序を変えることで，デッドロックの発生頻度を低減させることができます。

グローバルデッドロックの例として，二つのトランザクションが別々のサーバに格納された表に対して，検索と更新の順番を逆順で実行した場合に排他の掛かる順序とデッドロックの関係を次の図に示します。

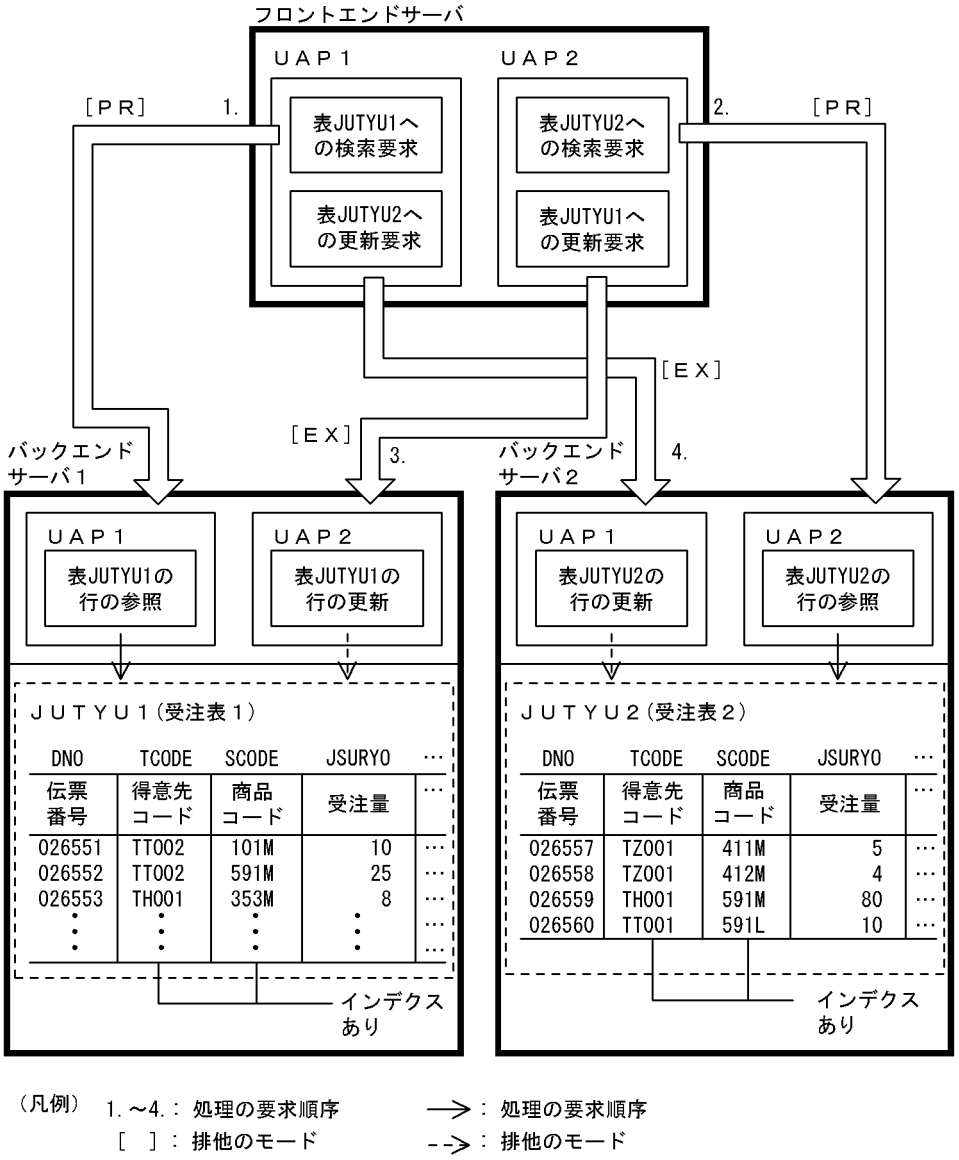
図 3-14　グローバルデッドロックの例

```
＜ＵＡＰ１＞
  SELECT TCODE,JSURYO FROM JUTYU1
     WHERE TCODE=TT002 AND JSURYO>10

  UPDATE JUTYU2 SET JSURYO=20
     WHERE SCODE='412M'
```

```
＜ＵＡＰ２＞
  SELECT TCODE,JSURYO FROM JUTYU2
     WHERE TCODE=TZ001 AND JSURYO<5

  UPDATE JUTYU1 SET JSURYO=50
     WHERE SCODE='591M'
```

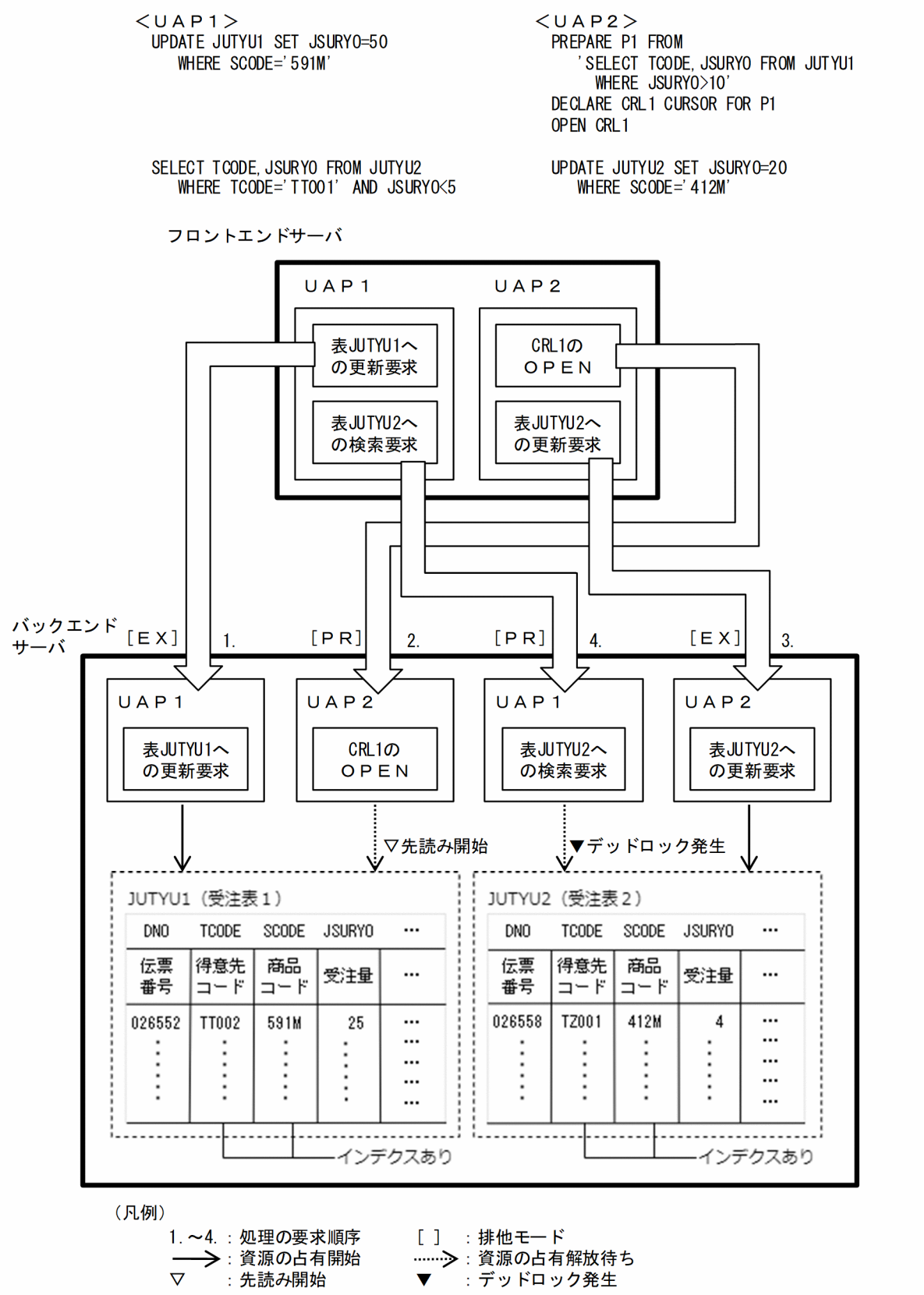


図「グローバルデッドロックの例」で示した例の場合，各バックエンドサーバでは，UAP1 と UAP2 との間の排他制御になりますが，フロントエンドサーバ側からは，お互いに排他待ちとなっているのでデッドロックになります。

(3)　データの先読みで発生するデッドロック

HiRDB/パラレルサーバでは，OPEN 文及び FETCH 文を発行したとき，処理性能向上のために FETCH 文で検索対象となる行のデータを先読みします。そのため，HiRDB/シングルサーバではデッドロックにならない UAP であっても，HiRDB/パラレルサーバではデッドロックが発生するおそれがあります。

HiRDB/パラレルサーバのデータの先読みで発生するデッドロックの例として，二つのトランザクションのうち一つのトランザクションでデータの先読みが発生した場合の，排他の掛かる順序とデッドロックの関係を次の図に示します。



［説明］

1. ［UAP1］JUTYU1 への更新要求で商品コード’591M’の行の EX 排他取得します。
2. ［UAP2］JUTYU1 へのカーソルオープンし UAP2 にオープン完了を応答します。その後，バックエンドサーバーが先読みを開始することで，1 で EX 排他を取得した行と競合し PR 排他取得待ちが発生します。
3. ［UAP2］UAP2 は 2 でオープン完了の応答を受け取ったことで，OPEN 文の次の JUTYU2 への更新要求を実行し，商品コード’412M’の行の EX 排他取得します。
4. ［UAP1］JUTYU2 への検索要求を実行しますが，3 で EX 排他を取得した行と競合するため PR 排他待ちが発生します。

   この時，UAP1，UAP2 ともに排他待ちとなっているため，デッドロックを検知します。

このような場合は，UAP1 又は UAP2 の SELECT 文のどちらかを WITHOUT LOCK NOWAIT を使用した検索に変更し，デッドロックを回避できないか検討してください。

(4)　デッドロックの検出

各ユニット内でのデッドロックは，それぞれのユニット内の排他制御機構が検出します。HiRDB では，各ユニット内で同一サーバが排他したリソースについては，そのサーバ内の複数トランザクション間のデッドロックの検出を行います。しかし，同一ユニット内であっても，別サーバにわたるリソース間でのデッドロックについては，排他制御のタイムアウトによって HiRDB が検出する以外では，検出できません。複数のユニットにわたるリソース間のデッドロックについては，ユニット内の別サーバ間のデッドロックと同じであり，タイムアウトによって検出します。

(a)　デッドロックの検出方法とタイミング

排他制御処理を分散させているかどうかによって，デッドロックが発生したときの検出方法とタイミングが異なります。排他制御処理の分散については，マニュアル「HiRDB システム運用ガイド」を参照してください。デッドロックの検出方法とタイミングを次の表に示します。

表 3–22　デッドロックの検出方法とタイミング

pd_lck_deadlock_check の値	pd_lck_pool_partition の値※	デッドロックの検出方法	デッドロックの検出タイミング
Y	2 以上	デッドロック監視プロセスによって，定期的に検出を実行します（インターバル監視方式）。	pd_lck_deadlock_check_interval オペランドに指定した間隔で検出します。この場合，デッドロック発生から検出までに時間差が発生します。
	1	サーバプロセスによって，排他待ちになった時点で自己検出します（即時検知方式）。	デッドロック発生後，即時に検出します。
N	−	デッドロックを検出しません。この場合，pd_lck_wait_timeout	−

pd_lck_deadlock_check の値	pd_lck_pool_partition の値※	デッドロックの検出方法	デッドロックの検出タイミング
		オペランドに指定した時間が経過するまで排他待ちを行った後，UAP に対して排他タイムアウトとしてエラーを発行します。	

（凡例） －：該当しません。

注※

フロントエンドサーバの場合は pd_fes_lck_pool_partition オペランドの値。

(b) デッドロック検出の無効化

pd_lck_deadlock_check オペランドに N を指定することで，排他制御でのデッドロック検出を無効にできます。

デッドロックの検出方法がインターバル監視方式の場合は，排他制御用プールパーティション数を増加するとデッドロックの検出タイミングごとに排他制御の性能が劣化するおそれがあります。そのため，デッドロックが発生しない業務システムでは，デッドロック検出をしないことで SQL の実行性能が改善することがあります。この場合は，デッドロックが発生しない業務システムを構築した上で，デッドロックの検出を無効にすることを推奨します。

反対に，デッドロックが発生する業務システムでは，デッドロック検出を無効にしないでください。デッドロック検出を無効にすると，デッドロックが発生したときに pd_lck_wait_timeout オペランドに指定した時間が経過してタイムアウトするまで SQL がエラー終了しません。また，HiRDB がデッドロック情報を出力しなくなるため，デッドロックが発生した要因が分からなくなるおそれがあります。

(5) デッドロックの対策

デッドロックの発生原因は，大きく分けると次の二つになります。

- UAP のアクセス順序（排他を掛ける順序）に依存する。
- 検索と更新との間の逆順処理をする。

デッドロックは，次の図で示した以外の種類もあります。

- 図「デッドロックの例」
- 図「ページ排他でのデッドロックの例」
- 図「グローバルデッドロックの例」

デッドロックの主な種類とその対策を次の表に示します。

表 3–23　デッドロックが発生する処理と対策

デッドロック資源	発生原因	対　　策
行と行	UAP のアクセス順序※1	• UAP のアクセス順序を統一する。 • LOCK TABLE によって表に排他を掛ける。 • デッドロック発生後に UAP を再実行する。
行とインデクスキー	検索と更新との間の逆順処理※2	• 検索行を更新しない。 • 列を同じ値で更新しない。 • インデクスの定義を必要最小限にする。 • LOCK TABLE によって表に排他を掛ける。 • NOWAIT 検索をする。 • デッドロック発生後に UAP を再実行する。 • インデクスキー値無排他を適用する。
インデクスキーと インデクスキー	UAP のアクセス順序	• UAP のアクセス順序を統一する。 • インデクスの定義を必要最小限にする。 • LOCK TABLE によって表に排他を掛ける。 • NOWAIT 検索をする。 • デッドロック発生後に UAP を再実行する。 • インデクスキー値無排他を適用する。
ページとページ	ページへの行格納順序の不定※3	• ALTER TABLE によってページ排他を行排他に変更する。

注※1　図「グローバルデッドロックの例」のような場合

注※2　図「デッドロックの例」のような場合

注※3　図「ページ排他でのデッドロックの例」のような場合

(6)　デッドロックプライオリティ値による排他制御

デッドロックが発生した場合，どちらのトランザクションをエラーとするかをデッドロックプライオリティ値で制御できます。この制御は，システム定義の pd_deadlock_priority_use オペランドで，デッドロック優先順位を制御する指定をし，クライアント環境定義の PDDLKPRIO オペランドでデッドロックプライオリティ値を指定した場合に，その指定値によって HiRDB がデッドロック優先順位を決定します。

指定した値が小さい方が処理の優先度が高く，値が大きくなるほどエラーになってロールバックする可能性が高くなります。また，デッドロックプライオリティ値が同じ場合，後発の方がエラーとなってロールバックさせられます。

PDDLKPRIO オペランドの指定を省略した場合，UAP，ユティリティ，及び運用コマンドの種別によって，自動的に HiRDB がデッドロックとなったトランザクションのどれかをエラーにしてロールバックさせます。PDDLKPRIO オペランドの指定値を省略した場合に仮定される値については，「クライアント環境定義の設定内容」を参照してください。

デッドロックによって暗黙的にロールバックされた UAP は，ROLLBACK 文，又は DISCONNECT 文でトランザクションを終了させないと，SQL 文を実行してもエラーになります。また，OLTP 環境で X/Open に

従ったアプリケーションプログラムをクライアントとした場合に，実行したアプリケーションプログラムがデッドロックになったときもトランザクションの終了が必要です。

なお，デッドロック発生時にデッドロック情報を出力したい場合，システム定義の pd_lck_deadlock_info オペランドに Y を指定する必要があります。pd_lck_deadlock_info オペランドについては，マニュアル「HiRDB システム定義」を参照してください。

(7)　デッドロックの回避策

デッドロックは排他の範囲を広くすることで発生頻度を低減できますが，同時実行性は低下します。したがって，排他の範囲を狭くすると同時実行性は向上しますが，不正参照，及び不正更新を引き起こしたり，デッドロックの発生率が増加します。同時実行性を保ちながら，デッドロックを回避するため，次に示す対策を考慮する必要があります。

- 頻繁に更新する列にはインデクスを付けないようにする。
- 検索条件列を更新しないようにする。
- カーソル定義の FOR UPDATE OF 句は，更新する列だけに指定する。
- 列（特にユニークインデクスの列）を同じ値で更新しないようにする。必ず更新する列だけを SET 句で指定する。
- カーソルを使って検索した行を更新，又は削除するときは，カーソル指定に FOR UPDATE 句を指定する。
- 検索後に更新を予定している列がある場合，WITH EXCLUSIVE LOCK を指定する。
- 複数列に条件を付ける場合，複数列インデクスの適用を検討する（単一列インデクスでの検索範囲を広くしないため）。
- WITHOUT LOCK NOWAIT を使用した検索を検討する。
- 二つ以上の表をアクセスする場合，アクセス順序を統一する。また，A，B の順でアクセスした場合，再度 A をアクセスしないようにする（A の値は保存しておきます）。
- LOCK TABLE を指定する。
- INSERT 文で行を挿入した直後に更新する場合，同一トランザクション内で処理をするようにする。
- 複数の UAP が，同一表に対して同時に AND の複数インデクス利用で更新する場合，システム定義の pd_work_table_option オペランドに 1 を指定する。AND の複数インデクス利用については，「クライアント環境定義の設定内容」の PDSQLOPTLVL を，pd_work_table_option については，マニュアル「HiRDB システム定義」を参照してください。
- インデクスキー値無排他を適用する。
- スナップショット方式を適用した場合，意図しないデッドロックが発生するおそれがあるため，デッドロック発生時にリトライできるような UAP を作成する。

以上のように，排他制御の範囲を考慮する以外にデッドロックを回避するためには，SQL 文の種類とインデクスの種別による排他制御の順序について考慮する必要があります。詳細については，「SQL 文の種類とインデクスの種別による排他制御の順序」を参照してください。

(8)　プラグインが使用する論理ファイルのデッドロック回避策

プラグインが論理ファイルを使用する場合，更新操作の場合は EX モード，検索操作の場合は PR モードで，論理ファイル単位に排他を掛けます。

論理ファイルの排他は，データの値に関係なく操作発生を契機に実行されます。したがって，論理ファイルを使用するプラグイン定義のある列をアクセスする場合に，更新トランザクションを実行すると，該当する列を操作するほかのすべてのトランザクションとの間に，論理ファイルの競合が発生します。このため，論理ファイルを使用するプラグイン定義のある列を更新するプログラムは，できるだけ単独で実行するようにしてください。

- デッドロック回避策 1

  LOCK TABLE を指定します。

- デッドロック回避策 2

  論理ファイルがデッドロック資源となった場合，論理ファイルがデータ型プラグインのものか，又はインデクス型プラグインのものかを確認して，次に示す表を参照してください。

  - 表「排他制御のモードの組み合わせの代表例（行排他の場合）(1/2)」
  - 表「排他制御のモードの組み合わせの代表例（行排他の場合）(2/2)」
  - 表「排他制御のモードの組み合わせの代表例（ページ排他の場合）(1/2)」
  - 表「排他制御のモードの組み合わせの代表例（ページ排他の場合）(2/2)」
  - 表「排他制御のモードの組み合わせの代表例（インデクスキー値無排他の場合）(1/2)」
  - 表「排他制御のモードの組み合わせの代表例（インデクスキー値無排他の場合）(2/2)」

  なお，デッドロックが発生した場合のデッドロック情報の出力内容については，マニュアル「HiRDB システム運用ガイド」を参照してください。

  排他情報：

  　種別 000e　−＞論理ファイル

  　排他情報の先頭 4 バイト　−＞ RD エリア番号

  RD エリア番号から RD エリア名を調査します。

  RD エリアが抽象データ型の LOB 属性格納用 RD エリアの場合：

  　「行」として扱います。

  RD エリアがプラグインインデクスの RD エリアの場合：

  　「インデクスキー」として扱います。

  注意

  　・プラグインインデクス検索の場合，NOWAIT 指定であっても論理ファイルは PR モードで排他されます。

  　・データ操作時，LOCK TABLE の排他を掛けていても，論理ファイルは EX 又は PR モードで排他されます。

・論理ファイルを使用するプラグイン定義のある列を複数定義した場合は，「デッドロック回避策2」では回避できません。「デッドロック回避策 1」で回避してください。

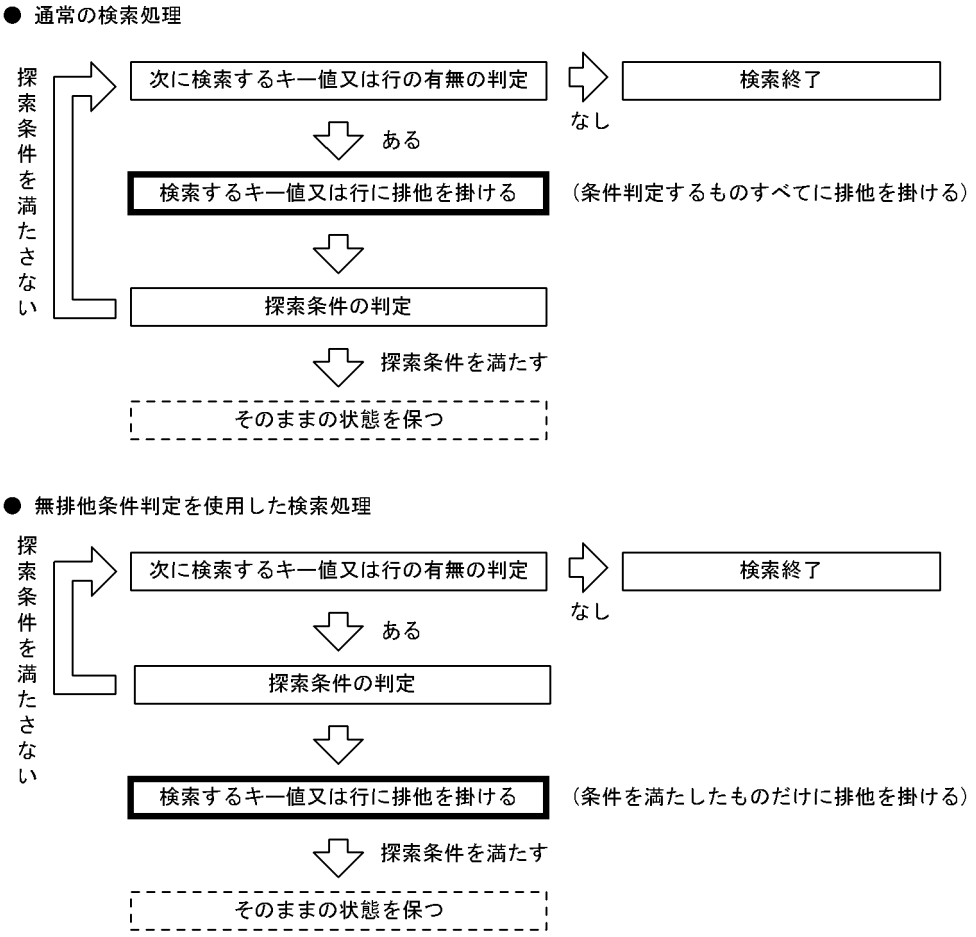
3.4.5　無排他条件判定

無排他条件判定は，検索処理時には排他を掛けないで，探索条件を満たした行，又はキー値に対してだけ排他を掛けます。無排他条件判定を使用すると通常の検索と比べて，探索条件を満たさない行，又はキー値に対して排他を掛けないため，検索時間が短縮できます。

また，更新処理と同時に検索処理を実行する場合，相手の更新処理が条件を満たさない行の更新，又は追加であれば，排他待ちが発生しません。このため，デッドロックや排他タイムアウトの発生が低減できます。

通常の検索処理と無排他条件判定の処理の流れを次の図に示します。

図 3–15　通常の検索処理と無排他条件判定の処理の流れ



無排他条件判定をするには，クライアント環境定義の PDLOCKSKIP オペランドで YES を指定します。

無排他条件判定は次に示す条件を満たす場合に効果が得られます。

- 条件判定をする件数に対して，条件を満たす件数が少ない検索

インデクスのキーによって，探索範囲をある程度絞り込んだ状態から，条件を切り出して検索する場合に，条件を満たすものだけに排他を掛けます。このため，探索範囲の件数に比べて，条件を満たす件数が少ないと，通常の検索処理に比べて（条件を満たす件数／探索範囲の件数）の割合で排他処理を削減できます。

- インデクスを使わない検索

  通常の検索処理で，インデクスを使わない検索は，一時的にすべての行に対して排他を掛けます。

  無排他条件判定を使用してインデクスを使わない検索をすると，探索条件を満たしたものだけに排他を掛けるため（条件を満たす件数／検索対象表の総行数）の割合で排他処理を削減できます。

- 条件を満たさない更新処理と同時に実行する検索

  ほかの更新トランザクションによって先に検索範囲が更新されていても，更新結果が条件を満たしていなければ，排他待ちにはなりません。

また，次に示す条件の場合に無排他条件判定を使用しても適用されません。

- WITHOUT LOCK NOWAIT のような排他を掛けない検索（システム定義 pd_isolation_level オペランドに 0 を指定，クライアント環境定義 PDISLLVL に 0 を指定，排他オプションを省略した検索）
- インデクスキー値無排他を適用したときのインデクスを利用する検索
- 次に示す探索条件で，かつ，インデクスを使わない探索条件
  - システム定義スカラ関数を指定している探索条件
  - ユーザ定義関数を指定している探索条件
  - 定義長 256 バイト以上の VARCHAR 型の列を指定している探索条件
  - 定義長 256 バイト以上の MVARCHAR 型の列を指定している探索条件
  - 定義長 128 文字以上の NVARCHAR 型の列を指定している探索条件
  - 定義長 256 バイト以上の BINARY 型の列を含む探索条件
  - BLOB 型の列を指定している探索条件
  - 繰返し列を指定している探索条件
  - コンポネント指定を使用している探索条件
  - 外への参照を含む副問合せを指定している探索条件
  - 限定述語又は IN 述語を指定している探索条件
  - ネストループ結合の内表として評価される探索条件を除く，2 表以上にわたる探索条件
  - 検索方式が AND の複数インデクス利用（AND PLURAL INDEXES SCAN）のインデクスで評価されない探索条件
  - 検索方式が OR の複数インデクス利用（OR PLURAL INDEXES SCAN）のインデクスで評価されない探索条件

なお，無排他条件判定は，排他を掛けないで条件判定をするため，コミットしていないデータを検索して条件判定するおそれがあります。例えば，更新トランザクションと同時に条件判定するとき，条件判定での検索結果と，更新トランザクションの処理結果との間に差異が発生することがあるので注意が必要です。

3.4.6　インデクスキー値無排他

インデクスキー値無排他とは，インデクスキー値で排他制御をしないことをいいます。この場合，表のデータだけで排他制御をします。

インデクスキー値無排他を適用した場合，インデクスを利用した検索処理ではインデクスキー値に対して排他は掛けません。また，表に対する更新処理（行挿入，行削除，及び列値更新）の場合にも，更新対象列に定義されているインデクスのインデクスキー値に対して排他は掛けません。

(1)　適用基準

インデクスキー値無排他は基本的に適用することをお勧めします。

ただし，ユニークインデクスの一意制制約保証処理の動作，残存エントリ，及び表データ更新時に出力されるシステムログ量を考慮した上で決めてください。ユニークインデクスの一意制制約保証処理の動作，残存エントリについては，「ユニークインデクスの残存エントリ」を参照してください。また，表データ更新時に出力されるシステムログ量については，マニュアル「HiRDB システム導入・設計ガイド」を参照してください。

(2)　インデクスキー値無排他の指定方法

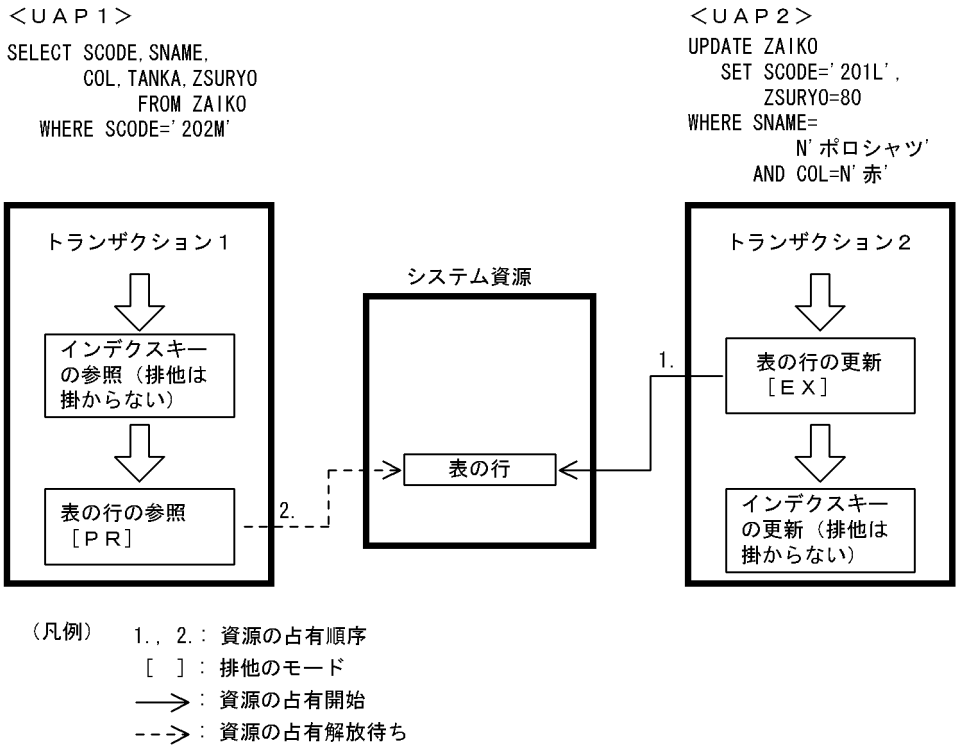
インデクスキー値無排他を適用する場合，システム定義の pd_indexlock_mode オペランドに NONE を指定します。pd_indexlock_mode オペランドについては，マニュアル「HiRDB システム定義」を参照してください。

なお，システム定義の pd_inner_replica_control オペランドの指定値が 1 より大きい場合，システム定義の pd_indexlock_mode オペランドの指定に関係なく，pd_indexlock_mode オペランドには NONE が仮定されます。

(3)　デッドロック回避の例

図「デッドロックの例」のようにデッドロックが発生した場合，インデクスキー値無排他を適用するとデッドロックを回避できます。インデクスキー値無排他によるデッドロック回避の例を次の図に示します。

図 3–16　インデクスキー値無排他によるデッドロック回避の例



(4)　注意事項

(a)　ユニークインデクスの一意性制約保証処理の動作

インデクスキー値無排他を適用している場合，一意性制約定義が指定されている表では，行追加更新時での一意性制約保証処理の動作がインデクスキー値方式（インデクスキー値無排他ではない方式）とは異なります。インデクスキー値無排他を適用する場合は，この動作の違いを考慮しておく必要があります。

一意性制約の保証処理とは，行データの挿入，又は列値更新のときに，インデクス（ユニークインデクス）を使用して追加しようとしているキーを持つデータが既に表にあるかをチェックするとともに，排他制御で追加キーの一意性を保証する処理のことをいいます。一意性制約の保障ができたら直ちにユニークインデクスのインデクスキーエントリを挿入し，その後に行データの挿入，又は列値更新が行われます。

一意性制約の保証処理では，**同一キーを持つインデクスキーエントリが見つかった場合，すぐにユニークエラーとなります。**そのインデクスキーを操作している相手トランザクションが未決着状態で，ロールバックする可能性があったとしても，排他制御でのチェックをしないですぐにユニークエラーとなります。ただし，相手トランザクションが，そのインデクスキーエントリに対する行データの挿入を終えていなかったときは，相手トランザクションが決着するまで排他待ちします。また，この排他待ちが関係しデッドロックが発生するおそれもあります。デッドロックになるのは，自トランザクションが排他を掛けている排他資源で，相手トランザクションも排他待ちした場合です。

すぐにユニークエラーにしないで，インデクスキーを操作している相手トランザクションの決着を待ってから挿入，又は更新処理を試みるようにしたい場合には，pd_lock_uncommited_delete_data オペランド

に WAIT を指定して，コミットしていない削除データに対して排他を行ってください。コミットしていない削除データの排他制御については，「コミットしていない削除データの排他制御」を参照してください。

(b)　ユニークインデクスの残存エントリ

インデクスキー値無排他を適用する場合，ユニークインデクスで排他待ち，及びデッドロックが発生することがあります。

インデクスキー値無排他でのユニークインデクスでは，一意性制約保証のために DELETE 文，又は UPDATE 文実行前のインデクスキーをインデクス上から削除しないで残すようにしています。この残っているインデクスキーのことを残存エントリといいます。

この残存エントリは，トランザクション決着後の適当なタイミングで削除されます。しかし，残存エントリと同一のキーに対する INSERT 文，又は UPDATE 文を実行した場合，タイミングによっては予想外に待たされたり，デッドロックが発生したりすることがあります。また，データディクショナリ表にはユニークインデクスが定義されています。定義系 SQL 文や pdmod を繰り返し実行した場合，タイミングによっては予想外に待たされたり，デッドロックが発生したりすることがあります。

残存エントリが削除されるタイミングについては，「コミットしていない削除データの排他制御」を参照してください。

残存エントリによる排他待ちかどうかを判断したい場合，システム共通定義 pd_unique_indexlock_info オペランドに Y を指定してください。残存エントリによる排他待ちのおそれがある場合だけ，排他待ち解除時に KFPH25005-I メッセージを出力します。タイムアウト情報やデッドロック情報と，KFPH25005-I メッセージの情報を比較し，次の内容がすべて合致する排他資源が，残存エントリによって排他待ちとなっていたおそれがあります。

- サーバ名
- プロセス ID
- 資源種別
- 資源情報

残存エントリによる排他待ちかどうかを判断する方法は，マニュアル「HiRDB メッセージ」の KFPH25005-I のメッセージ内容を参照してください。

残存エントリによる排他待ちの回避策と回避策の選び方を次に示します。

●回避策 1

一意性制約の列を更新しないように UAP を作成する

回避策 1 によって，UPDATE 文で一意性制約の列を更新する代わりに DELETE 文と INSERT 文で行の削除と追加をする場合でも，INSERT 処理のタイミングによっては残存エントリによる排他待ちが発生するおそれがあります。そのため，回避策 1 とともに，回避策 2 の適用も検討してください。

●回避策 2

システム共通定義の pd_dbreuse_remaining_entries オペランドに ALL 以外を指定する

残存エントリによる排他待ちが少なくなります。詳細は，「残存エントリによる排他待ちの回避（行識別子の再利用抑止）」を参照してください。

なお，pd_dbreuse_remaining_entries オペランドの指定値によって，排他待ちの軽減対象が異なります。詳細は，マニュアル「HiRDB システム定義」の「排他制御に関するオペランド」を参照してください。

●回避策 3

システム共通定義の pd_unique_check_mode オペランドに 1 を指定してユニークチェック用排他による一意性制約保証を使用する

詳細は，「残存エントリによる排他待ちの回避（ユニークチェック用排他使用）」を参照してください。

表 3–24　回避策の選び方

回避策	適用可能なケース	デメリット
回避策 1	一意製制約を指定した列を更新しないように AP を作成・改修できる場合	業務 AP の設計が限定される。
回避策 2	一意製制約を指定した列を更新する回数が少ない場合	更新する回数が多く，1 ページで使用スロット数が 255 を超える場合，残存エントリによる排他待ちを回避できない。
回避策 3	OS のメモリに余裕があり，更新により増加する排他資源を取得できる場合	HiRDB が使用する排他資源用プールサイズが増加する。

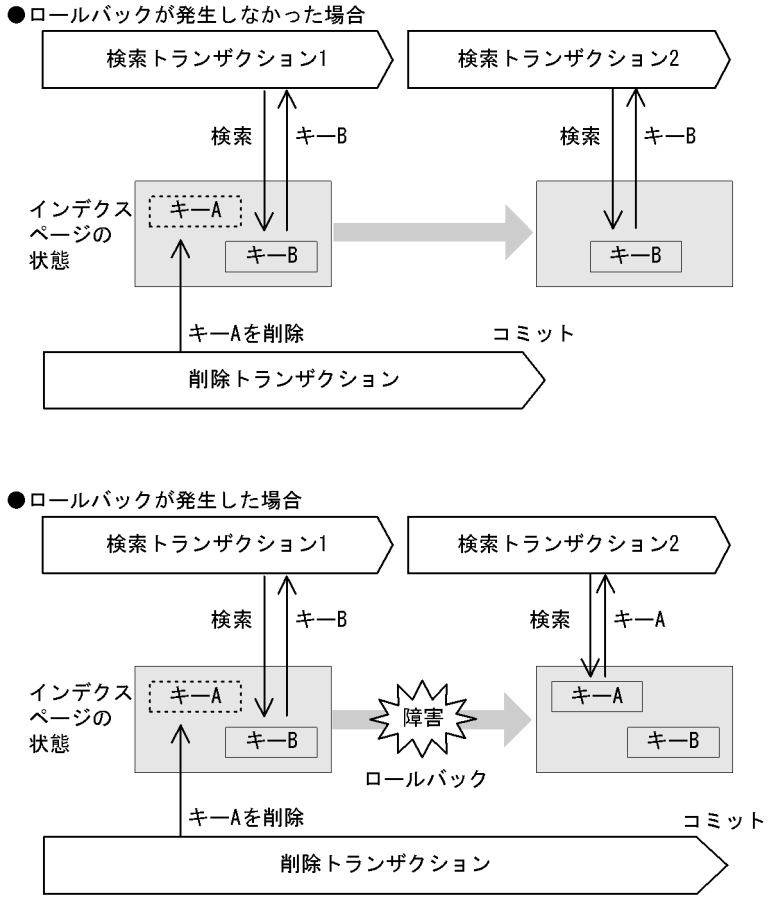
(c)　インデクスキー値無排他を適用しても回避できないデッドロック

UAP のアクセス順序によって，インデクスキーとインデクスキーとでデッドロックが発生することがあります。これを回避するためには，(b)と同様に一意性制約の列を更新しないように UAP を作成しなければなりません。

3.4.7　コミットしていない削除データの排他制御

通常，インデクスキーを削除すると，削除トランザクションがコミットする前に，そのインデクスキーはデータベースから削除されます。そのため，削除トランザクションがロールバックした場合，インデクスキーの内容は回復しますが，同時実行中のほかの検索トランザクションからは検索対象外となります。検索対象外となる削除インデクスキーの例を次の図に示します。

図 3–17　検索対象外となる削除インデクスキー



[説明]

削除トランザクションがコミットする前に，キー A はデータベースから削除されます。そのため，検索トランザクション 1 はキー A を読み飛ばし，キー B を検索に使用します。その後ロールバックが発生しなかった場合は，検索トランザクション 2 は検索トランザクション 1 と同様に，検索にキー B を使用します。

一方，ロールバックが発生してキー A の内容が回復した場合，検索トランザクション 2 は，検索にキー A を使用します。この場合，ロールバックの発生によって検索トランザクション 1 と検索トランザクション 2 は異なったインデクスキーを使用することになり，検索結果も異なってしまいます。

また，行削除の場合も同様です。コミットするまでデータベースから行データは削除されませんが，その間に同時実行中のほかの検索トランザクションからは検索対象外となります。

このように，コミットしていない削除データが同時実行中のほかの検索トランザクションから検索対象外となる状態は，削除トランザクションがコミットするまで排他を掛けることで回避できます。

(1)　適用基準

次のような業務の場合，コミットしていない削除データに排他を掛けることをお勧めします。

- 先行するトランザクションの処理結果によって，後続のトランザクションの処理が変わる業務
- ロールバック発生時に再実行を行わない業務

(2) 指定方法

pd_lock_uncommited_delete_data オペランドに WAIT を指定することで，コミットしていない削除データに排他を掛けます。pd_lock_uncommited_delete_data オペランドについては，マニュアル「HiRDB システム定義」を参照してください。

(3) コミットしていない削除データに排他を掛けた場合の効果

コミットしていない削除データに排他を掛けた場合の効果を次に示します。

- 検索中にコミットする前の削除データを検知した場合，検索トランザクションは削除トランザクションのコミット，又はロールバックの決着を待ってから検索処理を行います。これによって，削除トランザクションでロールバックが発生した場合でも，検索トランザクションからの検索読み飛ばしを防止します。
- インデクスキー値排他と同じ方式で一意性制約が保証されます。これによって，削除トランザクションのロールバック発生時のユニークエラーの検知を防止できます。

(4) コミットしていない削除データに排他を掛けた場合の検索トランザクションの動作

コミットしていない削除データに排他を掛けた場合と掛けなかった場合の検索トランザクションの動作について，SQL の実行条件ごとに，次の表に示します。

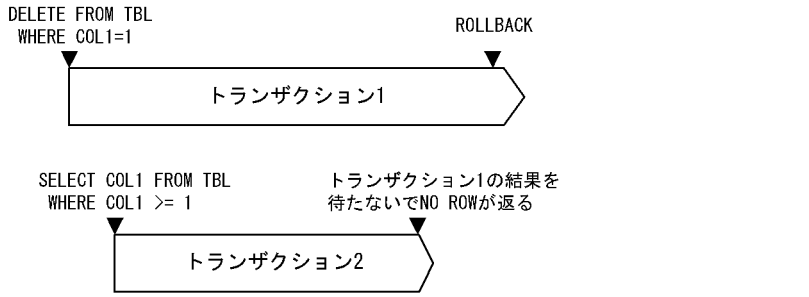
表 3–25 SQL の実行条件ごとの検索トランザクションの動作

項番	SQL の実行条件			UPDATE 文又は DELETE 文のトランザクションで ROLLBACK が発生した場合の検索トランザクションの動作	
	同時実行する SQL	UPDATE 文又は DELETE 文の内容	SELECT 文の内容	削除データに排他を掛けなかった場合	削除データに排他を掛けた場合
1	DELETE 文と SELECT 文の同時実行	行を削除する DELETE 文	DELETE 文で削除するキーが探索条件の範囲に含まれる検索，又は削除した行を参照する検索	DELETE 文を実行したトランザクションの決着を待たないで削除した行やキーを読み飛ばす	DELETE 文を実行したトランザクションの決着を待って，削除した行やキーを参照する
2	インデクスを定義している表に対し，UPDATE 文と SELECT 文を同時実行	インデクスの更新を伴う UPDATE 文	UPDATE 文で更新するインデクスのキーが探索条件の範囲に含まれる検索	UPDATE 文を実行したトランザクションの決着を待たないで更新前のキーを読み飛ばす	UPDATE 文を実行したトランザクションの決着を待って，更新前のキーを参照する
3	複数列インデクスを定義している表に対し UPDATE 文と SELECT 文を同時実行	複数列インデクスの一部の列を探索条件に指定し，ほかのインデクス構成列を更新する UPDATE 文	UPDATE 文で更新する複数列インデクスのキーが，探索条件の範囲に含まれる検索	UPDATE 文を実行したトランザクションの決着を待たないで更新前のキーを読み飛ばす	UPDATE 文を実行したトランザクションの決着を待って，更新前のキーを参照する

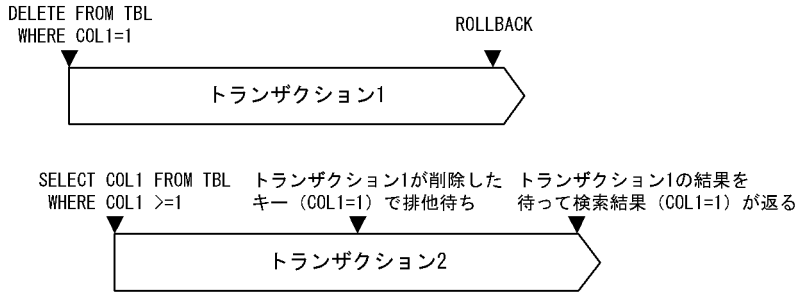
＜項番 1 の例＞

項番 1 の例として，列 COL1 にインデクスを定義した表 TBL の COL1=1 のデータに対し，DELETE 文と SELECT 文を同時実行した場合を次に示します。

●削除データに排他を掛けなかった場合



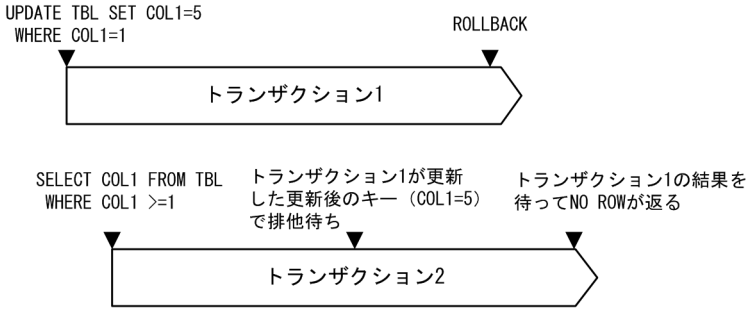
●削除データに排他を掛けた場合



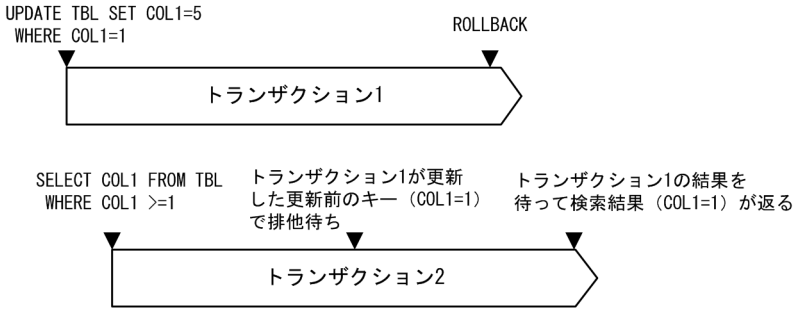
＜項番 2 の例＞

項番 2 の例として，列 COL1 にインデクスを定義した表 TBL の COL1=1 のデータに対し，UPDATE 文と SELECT 文を同時実行した場合を次に示します。

●削除データに排他を掛けなかった場合



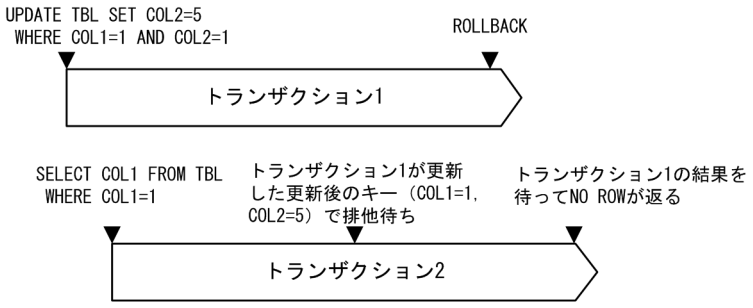
●削除データに排他を掛けた場合



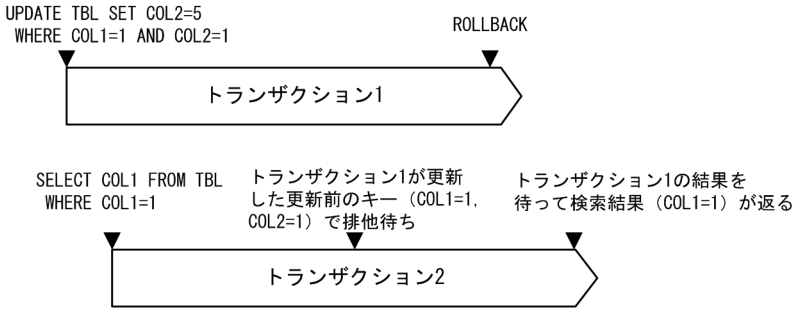
＜項番 3 の例＞

項番 3 の例として，列 COL1 と COL2 に複数列インデクスを定義している表 TBL の COL1=1，COL2=1 のデータに対し，UPDATE 文と SELECT 文を同時実行した場合を次に示します。

●削除データに排他を掛けなかった場合



●削除データに排他を掛けた場合



(5)　注意事項

(a)　インデクスの残存エントリ

コミットしていない削除データに排他を掛けた場合，削除，又は更新前のインデクスキーエントリをインデクス内に残すため，ナル値のユニークインデクスキー，及び非ユニークインデクスの残存エントリが発生します。この残存エントリに対して排他待ちが発生した場合，デッドロックが発生することがあります。また，残存エントリが大量にある場合は，検索性能が劣化するおそれがあります。

●回避策 1：残存エントリを減らす

基本的に，残存エントリはインデクスキーが更新，又は削除されるたびに増加します。残存エントリに対する排他待ちやデッドロックを回避するため，一定量の更新，又は削除が発生したら，データベース再編成ユティリティ（pdrorg），又は空きページ解放ユティリティ（pdreclaim）を実行し，残存エントリを削除してください。目安としては，「残存エントリが管理する行数の合計」／「インデクスが管理する総行数」が 30%を超えた時点を推奨します。残存エントリの総数は，データベース状態解析ユティリティ（pddbst）を実行して調べてください。

なお，残存エントリは次の場合に削除されます。

- 残存エントリが格納されているページに対してインデクスを追加しようとした場合に，ページ内の空き領域がなくなりインデクスページの分割が発生したとき

- 削除トランザクションの決着後に，そのインデクスに対して pdreclaim コマンドの -x オプションを実行した場合
- 削除トランザクションの決着後に，そのインデクス，又はインデクスに対応する表に対して pdrorg を実行した場合
- 削除したインデクスに対応する表に対して PURGE TABLE を実行した場合
- 削除したインデクス，又はインデクスに対応する表が格納されている RD エリアに対して再初期化を行った場合

また，残存エントリの発生を防ぐには，インデクスの更新頻度を少なくしておくことも有効です。

●回避策 2：残存エントリによる排他待ちを減らす

システム共通定義の pd_dbreuse_remaining_entries オペランドに NOTHING，ONLY_DIC，又は，NONE を指定すると，残存エントリによる排他待ちが少なくなります。詳細は，「残存エントリによる排他待ちの回避（行識別子の再利用抑止）」を参照してください。

(b)　表の残存エントリ

表の行データ格納領域についても，行削除後に残存エントリが発生します。コミットしていない削除データに排他を掛けた場合，これらの残存エントリに対してもチェックを行います。そのため，無効なデータの読み飛ばし処理や，排他制御のオーバヘッドが発生します。

●回避策 1：残存エントリを減らす

インデクスの残存エントリとは異なり，残存エントリに対する予想外の排他待ちは発生しませんが，一定量の更新，又は削除が発生したら，pdrorg，又は pdreclaim を実行し，残存エントリを削除してください。残存エントリの総数は，データベース状態解析ユティリティ（pddbst）を実行して調べてください。

また，残存エントリの発生を防ぐには，行削除の頻度を少なくしておくことも有効です。

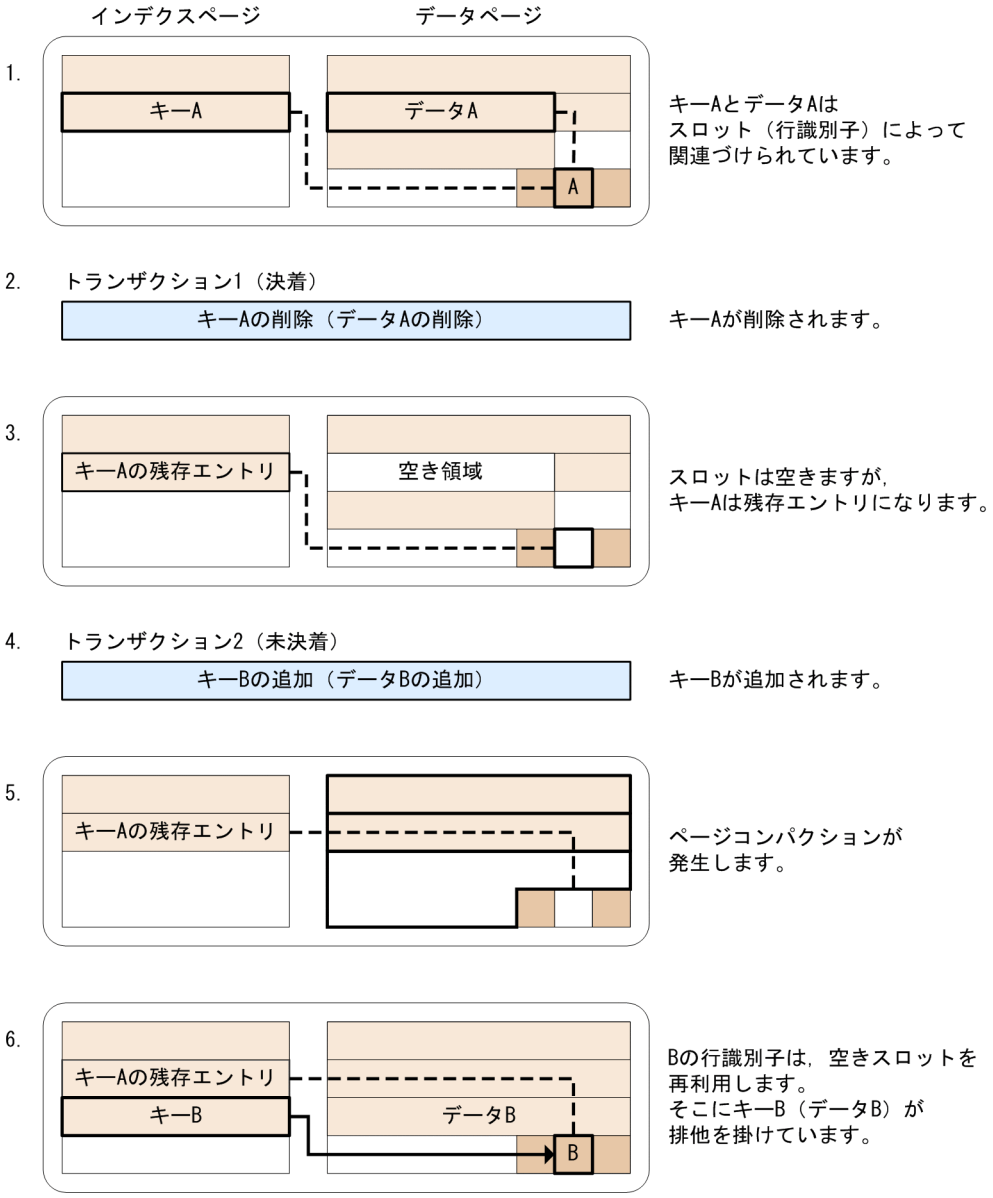
●回避策 2：残存エントリによる排他待ちを減らす

システム共通定義の pd_dbreuse_remaining_entries オペランドに NOTHING，ONLY_DIC，又は，NONE を指定すると，残存エントリによる排他待ちが少なくなります。詳細は，「残存エントリによる排他待ちの回避（行識別子の再利用抑止）」を参照してください。

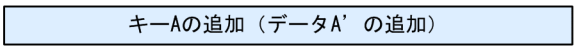
3.4.8　残存エントリによる排他待ちの回避（行識別子の再利用抑止）

インデクスキー値無排他を適用している場合，インデクスのキー値を削除した際に，インデクス格納ページに残存エントリが発生します。この場合，複数のトランザクションで INSERT 文，UPDATE 文，又は DELETE 文を実行したり，定義系 SQL 文や pdmod を繰り返し実行したりすると，異なるインデクスキー値の操作であっても排他待ちが発生することがあります。ユーザ表の排他待ち発生例を次の図に示します。

図 3–18　残存エントリによる排他待ち発生の例

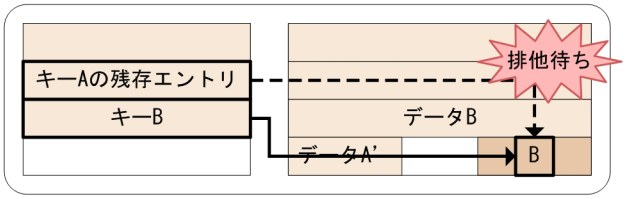


7. トランザクション3



キーAが追加されます。

8.



キーAの残存エントリが
スロットを参照しようとして
排他待ちが発生します。

（凡例）
：ページの未使用領域
：インデクス又はデータが格納されている領域
：スロット（データページの管理領域で，行識別子を示す）
：関連づけられている状態
：排他を取得しようとしている状態
：排他を取得している状態

[説明]

1. インデクスページには，キー A が格納されています。キー A は，データ A の行識別子を持っています。

   データページには，データ A が格納されています。また，データページの管理領域には，データ A に対応するスロット（行識別子）があります。

   キー A とデータ A は，このスロットが示す行識別子によって関連づけられています。

2. トランザクション 1 によってキー A 及びデータ A が削除されます。このとき，トランザクション 1 は決着します。

3. インデクスキー値無排他を適用しているため，インデクスページのキー A は残存エントリとなります。

   データページでは，データ A が削除され空き領域ができます。また，スロットの情報も削除され，空きスロットができます。

4. トランザクション 2 によってキー B 及びデータ B が追加されます。このとき，トランザクション 2 は未決着です。

5. 未使用領域にデータ B が入らないため，データページではページコンパクションが発生します。

6. インデクスページにキー B が格納されます。

   データページにはデータ B が格納されます。このとき，空きスロットが存在するため，データ B の行識別子はこの空きスロットを再利用します。

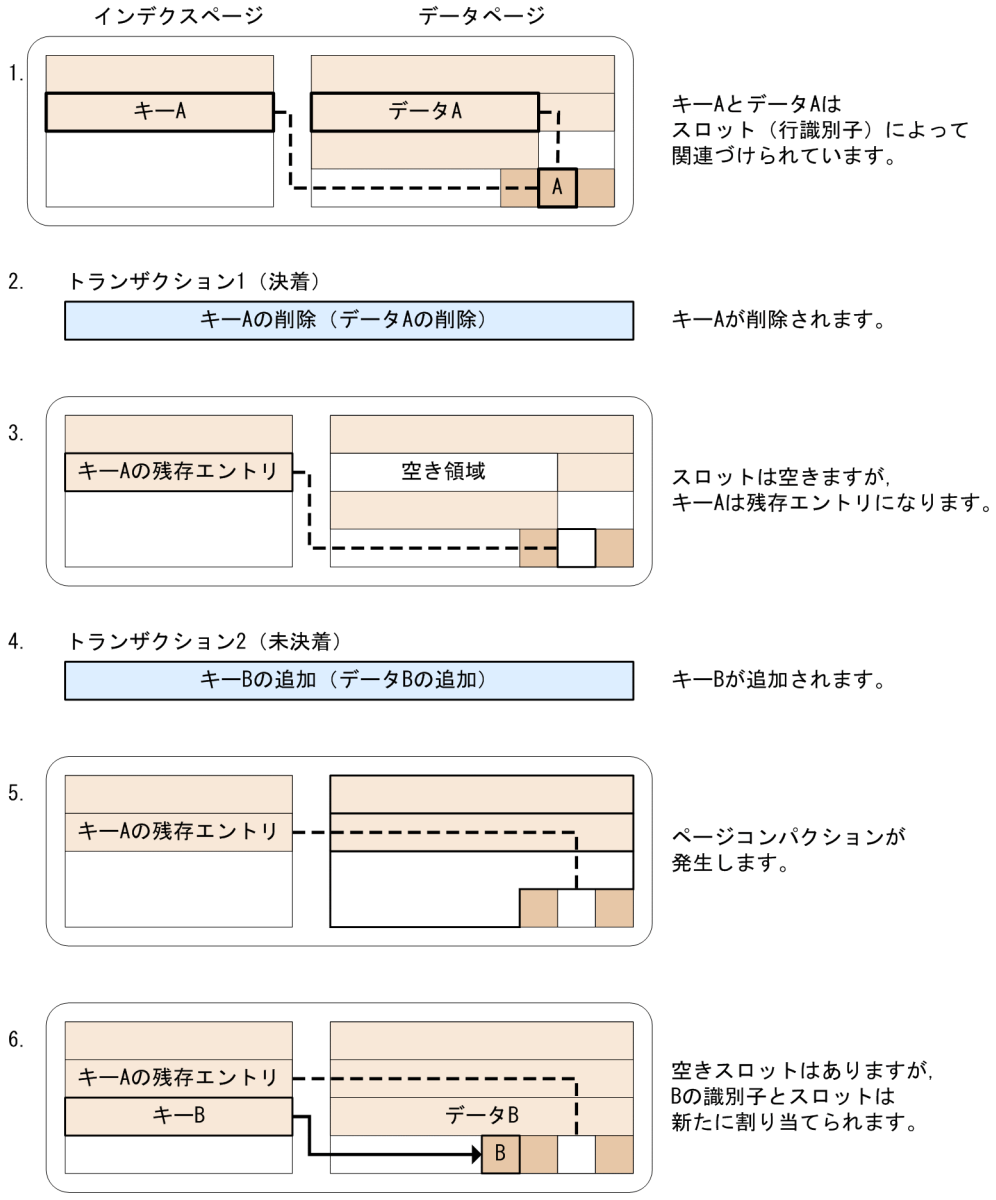
   なお，トランザクション 2 は未決着のため，キー B（データ B）はこのスロットに排他を掛けたままになっています。

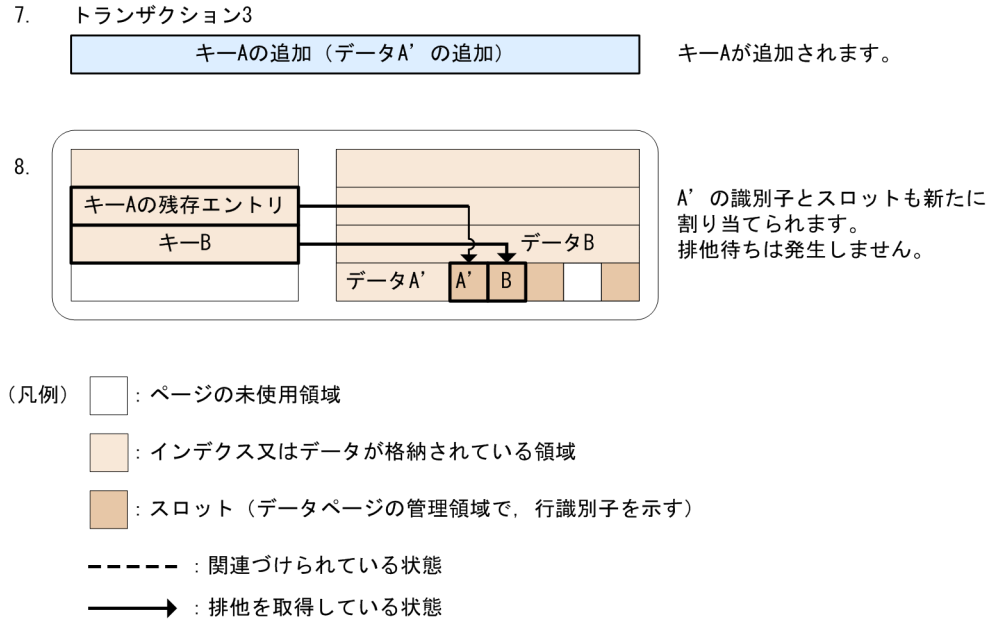
7. トランザクション 3 によって再びキー A 及びデータ A’が追加されます。

8. キー A には残存エントリがあるため，その残存エントリが前回使用していたスロットを再利用できるかどうか確認します。このとき，スロットは既にキー B（データ B）によって排他が掛けられているため，排他待ちが発生します。

この排他待ちは，行識別子の割り当て方法を変えることで回避できます。行識別子は，データを一意に識別するためにシステムが割り当てますが，行が削除されると，その行識別子は再利用されます。同様に，行識別子に対応していたスロットも再利用されます。そのため，残存エントリによる排他待ちが発生してしまいます。この行識別子の再利用を抑止することによって，スロットも再利用されなくなり，排他待ちを回避できます。行識別子の再利用抑止による排他待ちの回避例を次の図に示します。

図 3–19　行識別子の再利用抑止による排他待ちの回避例





[説明]

1. インデクスページには，キー A が格納されています。キー A は，データ A の行識別子を持っています。

   データページには，データ A が格納されています。また，データページの管理領域には，データ A に対応するスロット（行識別子）があります。

   キー A とデータ A は，このスロットが示す行識別子によって関連づけられています。

2. トランザクション 1 によってキー A 及びデータ A が削除されます。このとき，トランザクション 1 は決着します。

3. インデクスキー値無排他を適用しているため，インデクスページのキー A は残存エントリとなります。

   データページでは，データ A が削除され空き領域ができます。また，スロットの情報も削除され，空きスロットができます。

4. トランザクション 2 によってキー B 及びデータ B が追加されます。このとき，トランザクション 2 は未決着です。

5. 未使用領域にデータ B が入らないため，データページではページコンパクションが発生します。

6. インデクスページにキー B が格納されます。

   データページには，データ B が格納されます。データ B の行識別子には，新たなスロットが割り当てられます。このとき，トランザクション 2 は未決着のため，キー B（データ B）はこのスロットに排他を掛けたままになっています。

7. トランザクション 3 によって再びキー A 及びデータ A’が追加されます。

8. データ A’の行識別子にも新たなスロットが割り当てられるため，排他待ちは発生しません。

(1) 適用基準

行識別子の再利用を抑止すると，ページ内の管理領域（行識別子に対応するスロットの数）が増えるため，格納効率が低下する場合があります。

そのため，格納効率を下げてでも残存エントリの排他待ちを回避したい場合に適用してください。

格納効率の詳細は，「格納効率が低下するケース」を参照してください。

(2) 指定方法

行識別子の再利用を抑止するには，システム共通定義の **pd_dbreuse_remaining_entries** オペランドに **ALL** 以外を指定します。

なお，pd_dbreuse_remaining_entries オペランドの指定値によって，排他待ちの軽減対象が異なります。詳細は，マニュアル「HiRDB システム定義」の「排他制御に関するオペランド」を参照してください。

(3) 格納効率が低下するケース

行識別子を再利用しない場合，INSERT 文，UPDATE 文，及び DELETE 文を頻繁に実行したり，定義系 SQL 文や pdmod を繰り返し実行したりすると，ページ内の管理領域（行識別子に対応するスロットの数）が極端に増加します。管理領域は，1 ページ当たり最大で次の値まで増加します。

1 スロットの容量 2 バイト×1 ページ当たりの最大格納行数 255 行＝ 510 バイト

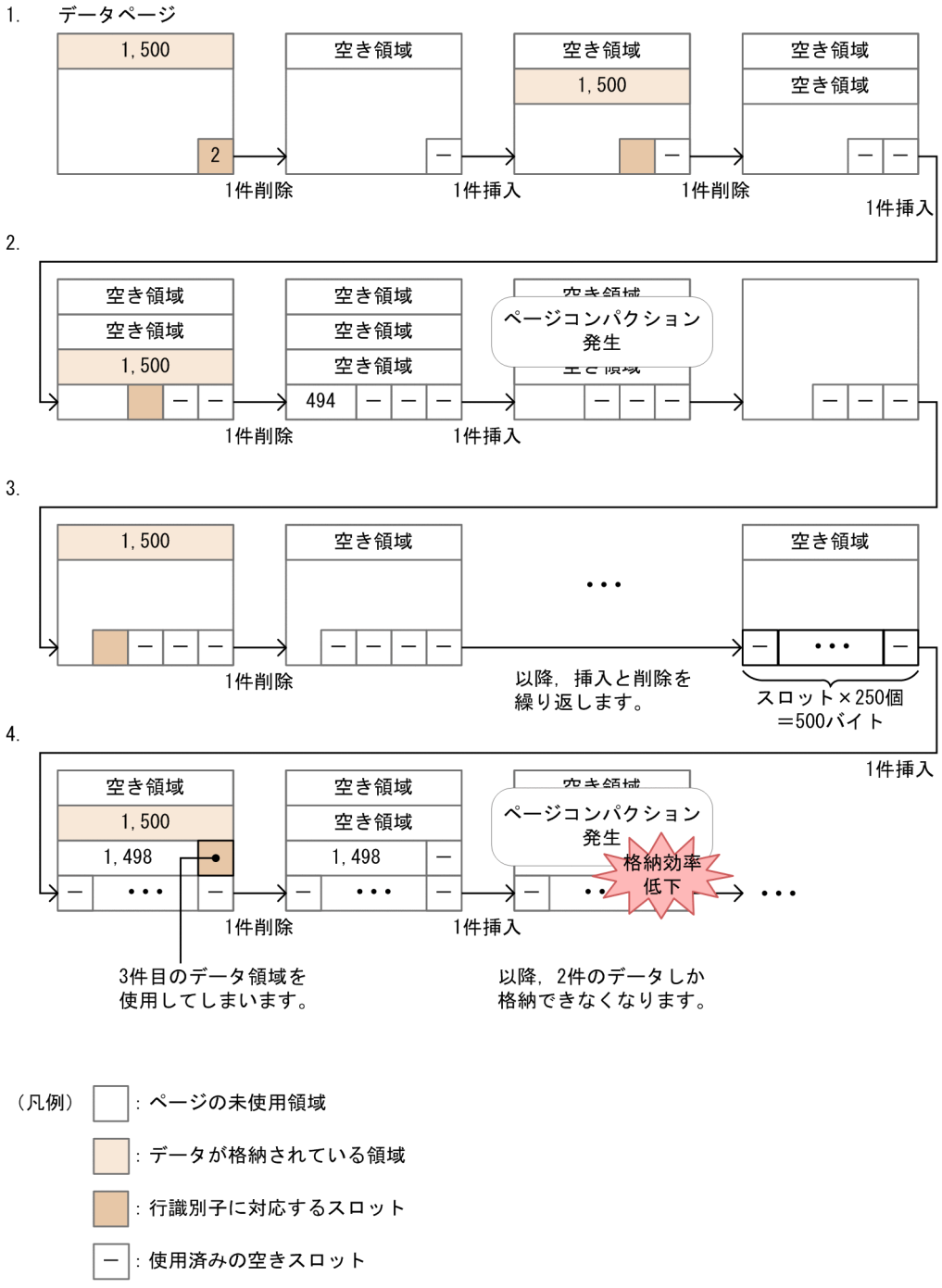
そのため，次のような場合は，1 ページに格納できるデータの件数が減少し，格納効率が低下することがあります。

- 1 件ごとのデータサイズが大きい，又はページサイズが小さいため，1 ページ当たりのデータ件数が少ない場合
- 未使用領域に余裕を持たせた見積もりをしていない場合

ユーザ表での行識別子の再利用抑止によって格納効率が低下する場合の例を次の図に示します。

この例では，データページのサイズが 5,000 バイトで，1 件のデータサイズが 1,500 バイトとします。行識別子の再利用を抑止している状態で，データの挿入と削除を繰り返します。

図 3–20　行識別子の再利用抑止による格納効率低下の例



[説明]

1. データの挿入と削除を繰り返します。行識別子の再利用を抑止しているため，その都度使用済みの空きスロットが増加します。
2. 未使用領域が 494 バイトとなり，1,500 バイトのデータが挿入できなくなったところで，ページコンパクションが発生します。このとき，空きスロットはそのまま残ります。
3. さらに挿入と削除を繰り返し，空きスロットが 250 個（500 バイト）になります。

4. ここでデータを挿入すると，新たなスロットが 3 件目のデータ領域を使用してしまいます。そのため，次にデータを挿入するときにはページコンパクションが発生します。以降，このページには 2 件のデータしか格納できなくなり，格納効率が低下します。

なお，スロット数が 1 ページ当たりの最大格納行数である 255 に達した場合は，ページ内に存在する残存エントリの中で行識別子の値が一番小さいものから順に再利用されます。

3.4.9　残存エントリによる排他待ちの回避（ユニークチェック用排他使用）

インデクスキー値無排他方式では，ユニークインデクスの一意性制約を保証するため行排他を使用して処理します。しかし，この方式では残存エントリによる排他待ちが発生し，新しいキー値を追加するとき残存エントリによる排他待ちが発生することがあります。ユニークチェック用排他による一意性制約保証は，ユニークチェック用の排他を使用して残存エントリによる排他待ちを回避します。ユニークチェック用の排他はユニークインデクスの一意性制約の確認にだけ使用しその他の処理では使用しないため，新たな排他待ちの要因にはなりません。

(1)　適用基準

ユニークチェック用排他による一意性制約保証を使用すると，ユニーク指定又はプライマリ指定のインデクスのキー値を削除するときに排他を取得します。そのため，排他制御に使用する共用メモリサイズが大きくなります。メモリに余裕があり，確実に残存エントリによる排他待ちを回避したい場合に指定します。増加する排他資源数については，マニュアル「HiRDB システム定義」の「付録 D　排他資源数の見積もり」を参照してください。

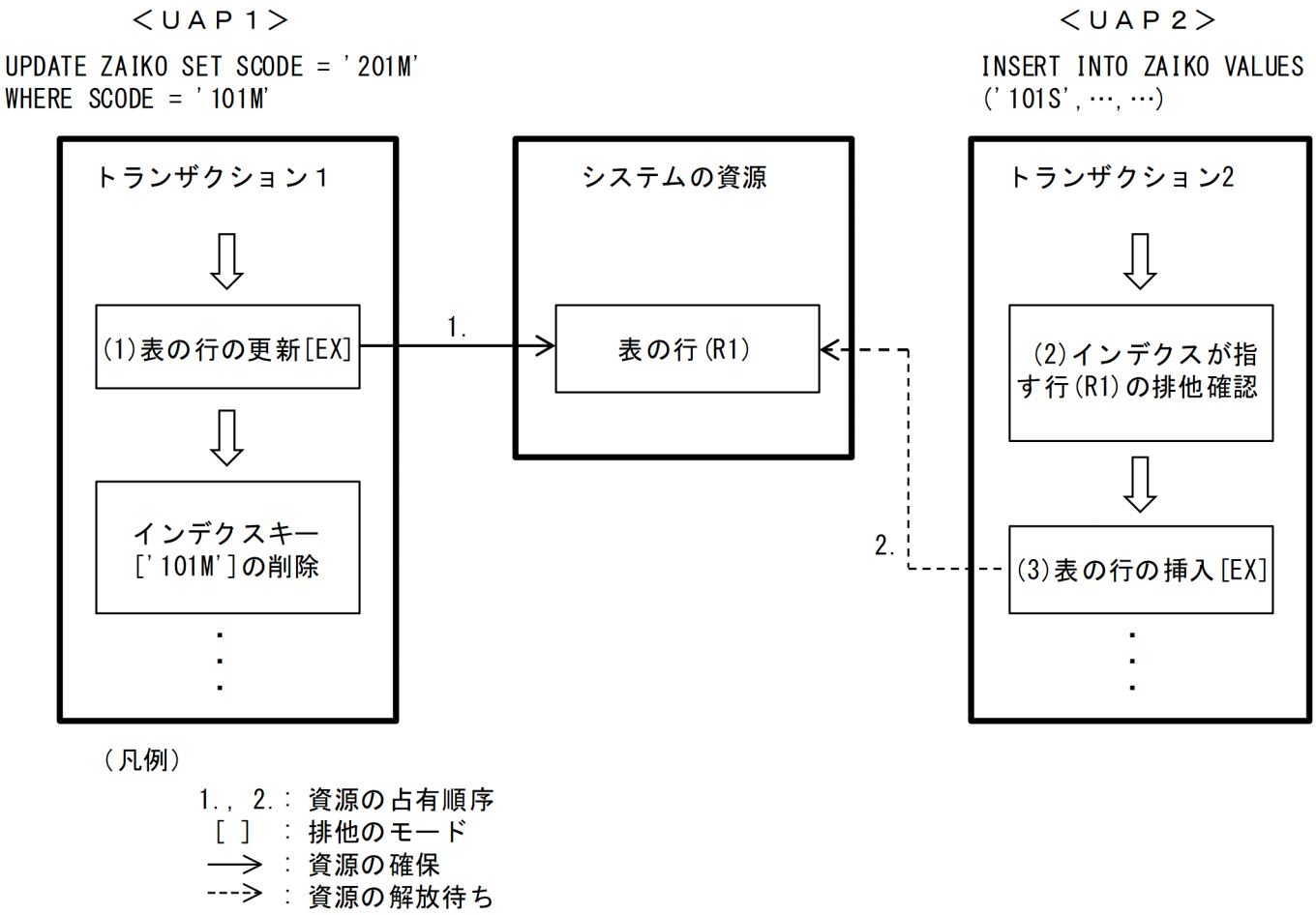
(2)　指定方法

ユニークチェック用排他による一意性制約保証を使用するには，システム共通定義の pd_unique_check_mode オペランドに 1 を指定します。詳細は，マニュアル「HiRDB システム定義」の「排他制御に関するオペランド」を参照してください。

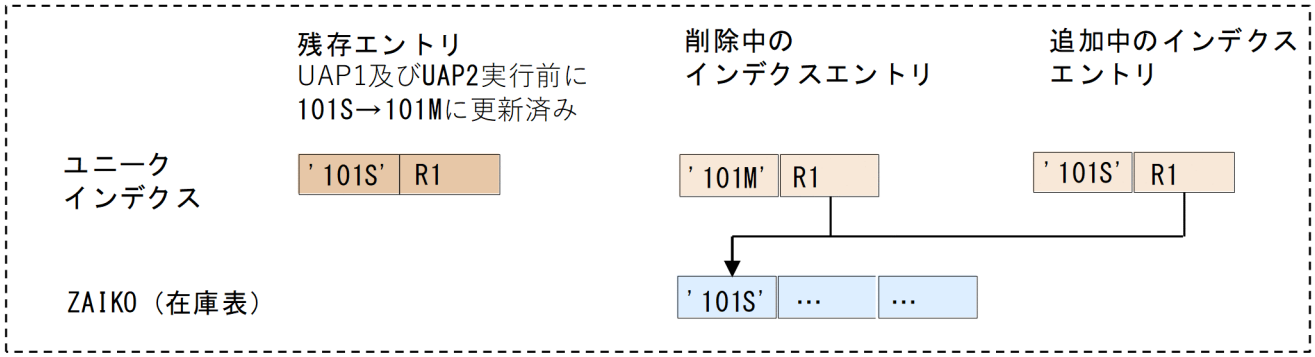
(3)　ユニークチェック用排他による一意性制約保証による残存エントリの排他待ちについて

ユニークチェック用排他による一意性制約保証を使用しない場合，行排他だけを使用して一意性制約を保証します。そのため，次の図に示すように過去に削除を行い完了したキー値を追加するときに残存エントリ（一意性制約を保障するために削除中に残している削除状態のエントリ）による排他待ちが発生することがあります。

図 3-21　残存エントリによる排他待ちが発生する例



■データベースの状態（ZAIKO(在庫表)のSCODE列にユニークインデクスを定義している）



[説明]

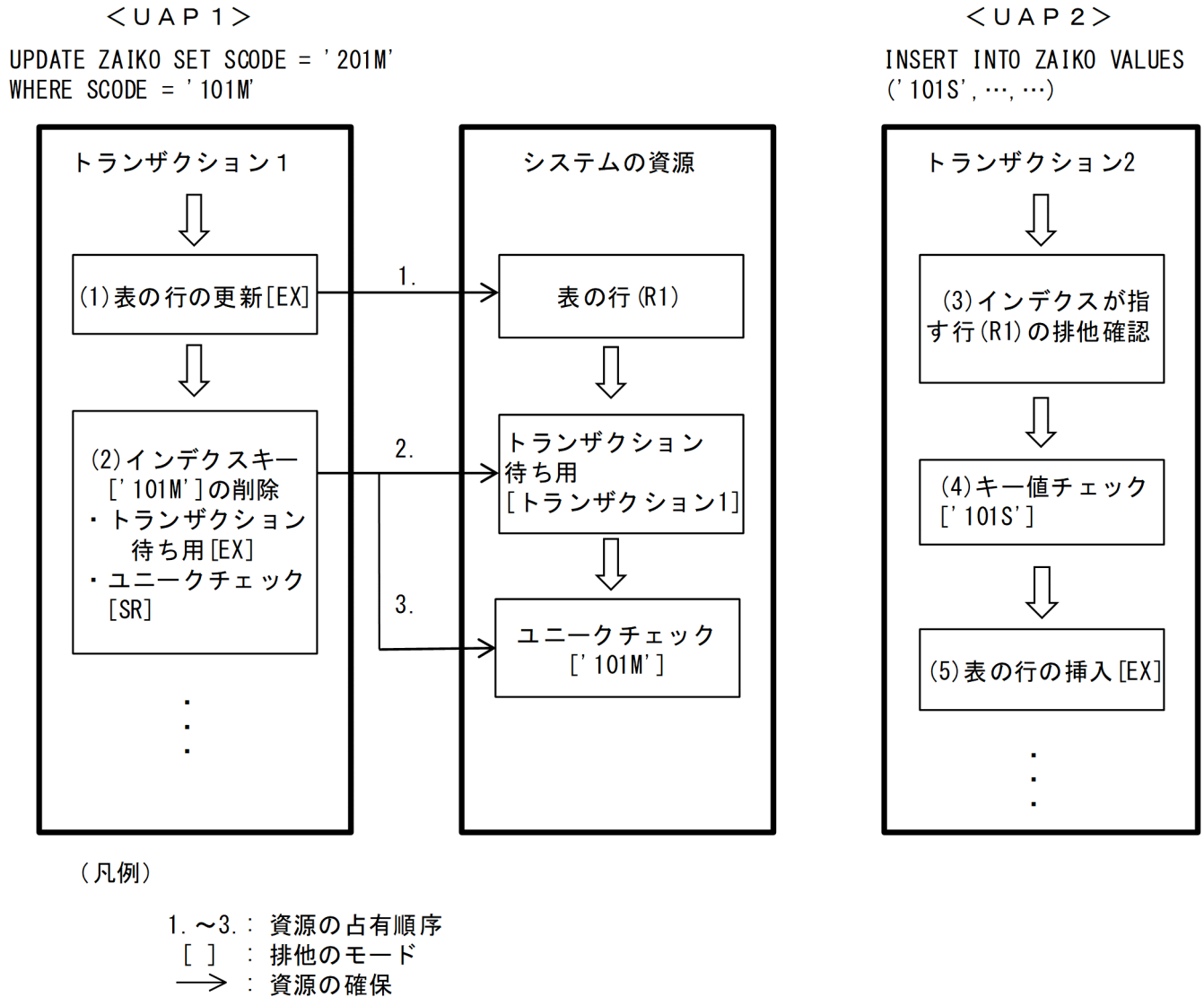
(1)行の更新のため，行(R1)に排他モードが EX の排他を取得する。

(2)インデクスキー追加前に削除中のインデクスエントリの行識別子(R1)で行に排他があるかを確認する。

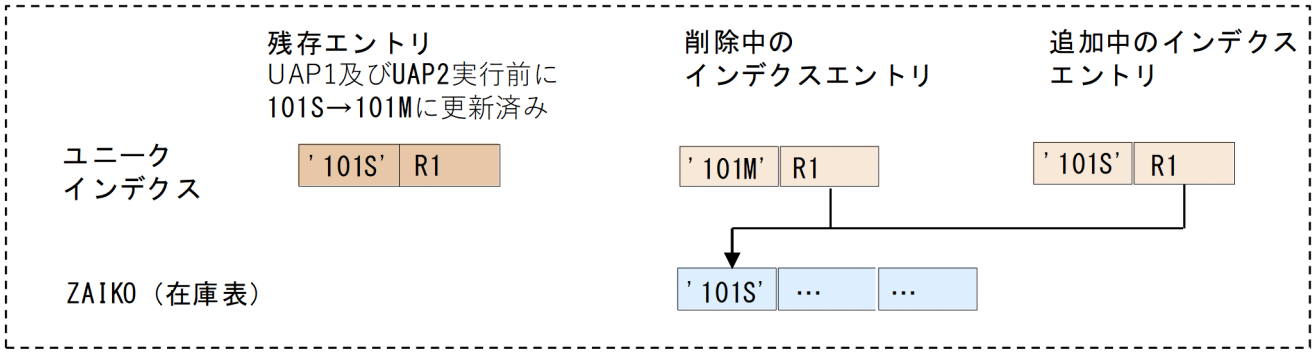
(3)行(R1)に排他があるため，トランザクション１の完了まで行(R1)の行排他で待つ。

ユニークチェック用排他による一意性制約保証を使用することで，次の図で示すように残存エントリによって待ちとなったり，デッドロックが発生したりすることがなくなります。

図 3-22　ユニークチェック用排他による一意性制約保証を使用することによって排他待ちを回避する例



■データベースの状態（ZAIKO(在庫表)のSCODE列にユニークインデクスを定義している）



[説明]

(1)行の更新のため，行(R1)に排他モードが EX の排他を取得する。

(2)インデクスキー削除時，トラザクション待ち用に排他モードが EX の排他を，キー値'101M'に対するユニークチェック用に排他モードが SR の排他を取得する。

(3)インデクスキー追加時，残存エントリの行識別子で行の排他があるかを確認する。

(4)(3)で行に排他があるため，行の排他を取得しているトランザクションがユニークチェック用の排他を取得しているかを確認する。

(5)挿入するキーと同じ値のユニークチェック用の排他を取得していないため，キー値の挿入を行ってトランザクション 2 の処理を続行する。

3.4.10　UAP でできる排他制御

排他制御は，HiRDB システムが自動的に制御していますが，処理形態によっては UAP で排他制御の単位を変更すると，オーバヘッドが削減できるので処理効率が向上します。UAP を作成するとき次に示す点を考慮してください。

(1)　検索の場合

1. 検索結果を一度参照するだけで，COMMIT するまでデータを占有する必要がないときは SELECT 文に WITHOUT LOCK を指定します。WITHOUT LOCK を指定すると，トランザクションの終了を待たないで排他が解除されるので，同時実行性が向上します。

   なお，WITHOUT LOCK NOWAIT を指定しても，データベース作成ユティリティ（option 文の nowait=no を指定した場合），又はデータベース再編成ユティリティ（-k unld の場合を除く）でデータ処理中の表は検索できません。

   また，WITHOUT LOCK NOWAIT を指定した場合，排他資源の表（NOWAIT 検索中）に対して，排他モード（EX）を取得する次のユティリティとの間にデッドロックが発生するおそれがあります。

   - データベース作成ユティリティ（pdload）
   - データベース再編成ユティリティ（pdrorg）

   データベース作成ユティリティの排他制御モード，及びデータベース再編成ユティリティの排他制御モードについては，マニュアル「HiRDB コマンドリファレンス」を参照してください。

2. 1 以外の場合は，検索する前に，対象の表に LOCK 文で SHARE モードの排他を掛けます。

   LOCK 文であらかじめ表に排他を掛けると，ページ単位，又は行単位の排他をしないため，オーバヘッドが大幅に削減できます。また，排他制御用のバッファ不足を防止することもできます。

3. 共用表を検索する場合，WITHOUT LOCK，又は WITH ROLLBACK を指定することをお勧めします。

(2)　更新の場合

1. 更新する前に，対象の表に LOCK 文で EXCLUSIVE モードの排他を掛けます。

   LOCK 文であらかじめ表に排他を掛けると，行単位の排他をしないため，オーバヘッドが大幅に削減できます。また，排他制御用のバッファ不足を防止することもできます。

2. 共用表に対して，インデクスのキー値の変更を伴う更新（追加，削除も含む）をする場合，及び大量の更新をする場合，必ず EXCLUSIVE 指定の LOCK 文で排他を掛けてください。なお，共用表に EXCLUSIVE 指定の LOCK 文で排他を掛けた場合，共用表に定義されているインデクスのインデクス格納用 RD エリア（共用 RD エリア）にも排他が掛かります。

(3) 削除の場合

1. 表の削除，インデクスの削除，又は全行削除をする場合，対象となる表のすべての使用中セグメントに対して EX モードの排他を掛けます。

   なお，表の使用中セグメント数が多い場合，すべてのセグメントに対して EX モードの排他を掛けるため，COMMIT するまでこれらすべてのセグメントを占有することになり，排他制御用バッファ内で多量の排他制御用資源管理テーブルを必要とするので注意する必要があります。特に，次に示す SQL 文を使用して表の削除，インデクスの削除，又は全行削除をする場合は，注意してください。

   - DROP SCHEMA
   - DROP TABLE
   - DROP INDEX
   - ALTER TABLE DROP PRIMARY KEY
   - PURGE TABLE

2. 複数の表を所有するスキーマを削除する場合，各々の表を削除してからスキーマを削除します。

   各表を削除した後にスキーマを削除すると，使用するメモリが軽減できます。

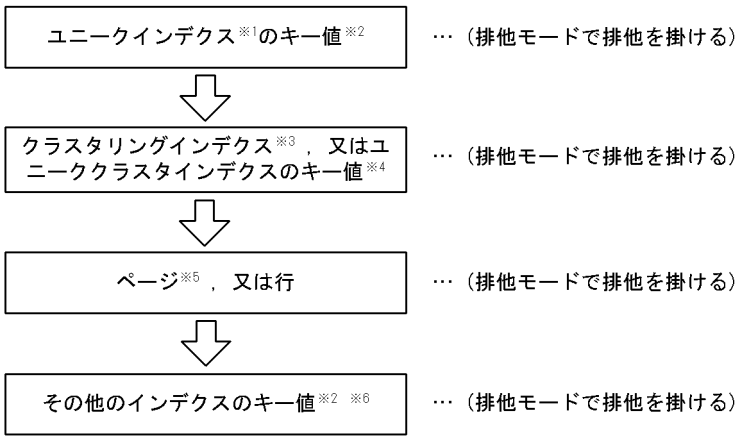
(4) 注意事項

1. LOCK 文で表に排他を掛けると，ほかのトランザクションが長時間待ち状態になるため，オンライン業務との同時実行は避けてください。ただし，非共用表の NOWAIT 検索については，待ち状態にはなりません。
2. 共用表に対して NOWAIT 検索をする場合，アクセスする RD エリアに対して，ほかのユーザが EXCLUSIVE 指定の LOCK 文を実行しているときはアクセスできません。

3.4.11　SQL 文の種類とインデクスの種別による排他制御の順序

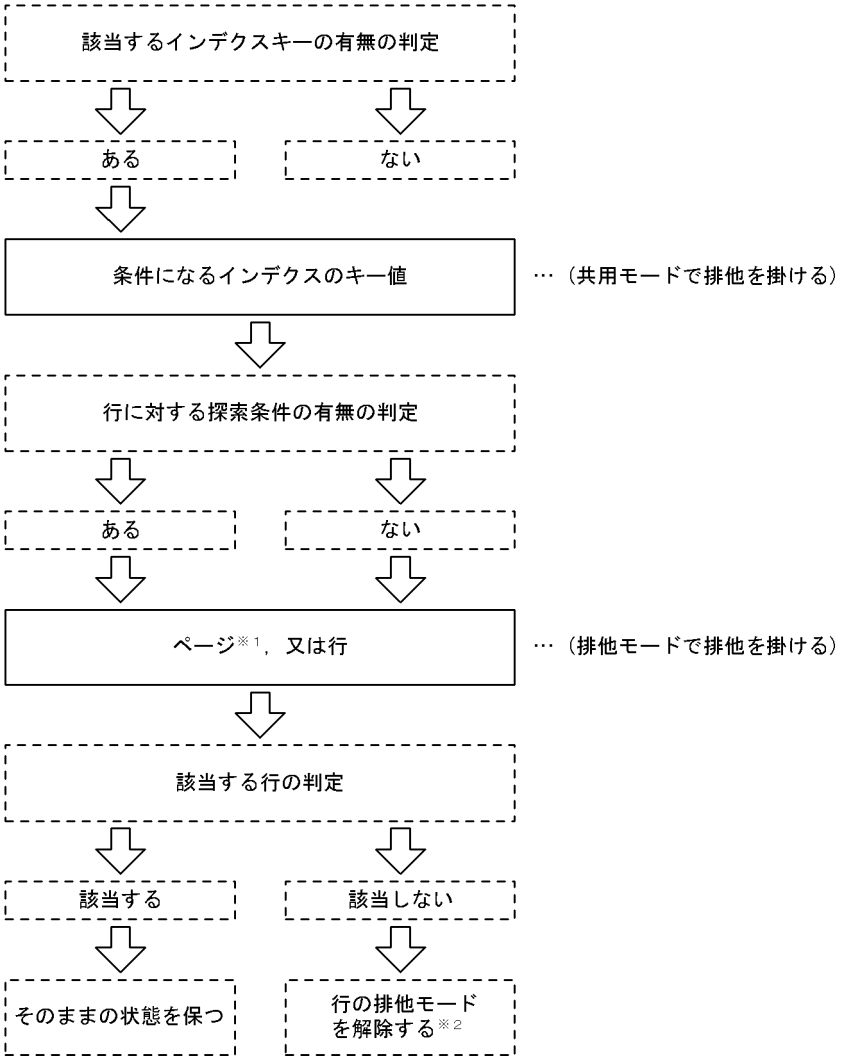
(1)　インデクスのキー値，及びデータページの行に対する操作系 SQL の排他制御の順序

(a)　INSERT 文の場合



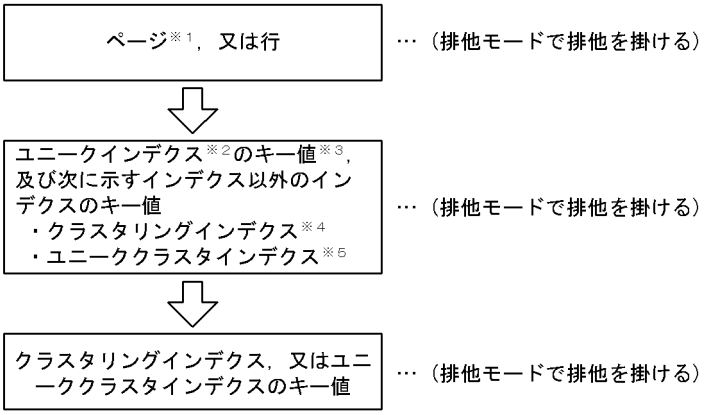
注※１ インデクスを定義する列に一意性制約(UNIQUE)を指定したときに作成されるインデクスのことです。
注※２ 定義数が複数の場合，定義した順序の逆順に処理します。
注※３ クラスタキーを指定した列に作成されるインデクスのことです。
注※４ クラスタキーとして定義した列に一意性制約(UNIQUE)を指定したときに作成されるインデクスのことです。
注※５ ページ排他表の場合に該当します。
注※６ ユニークインデクス，ユニーククラスタインデクス，及びクラスタリングインデクス以外のインデクスを示します。

(b)　カーソルを使用しない DELETE 文，又は UPDATE 文で条件に合うデータを探す場合



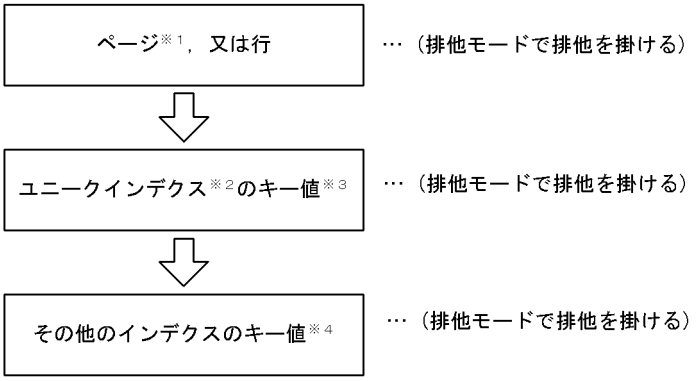
注※１　ページ排他表の場合に該当します。
注※２　既に共用モードの排他が掛かっている状態で排他モードの排他を掛けた場合，排他モードの処理が先に終了しても排他モードは解除されません。この場合，共用モードと排他モードの処理が終了したときに排他を解除します。

(c)　カーソルを使用した DELETE 文



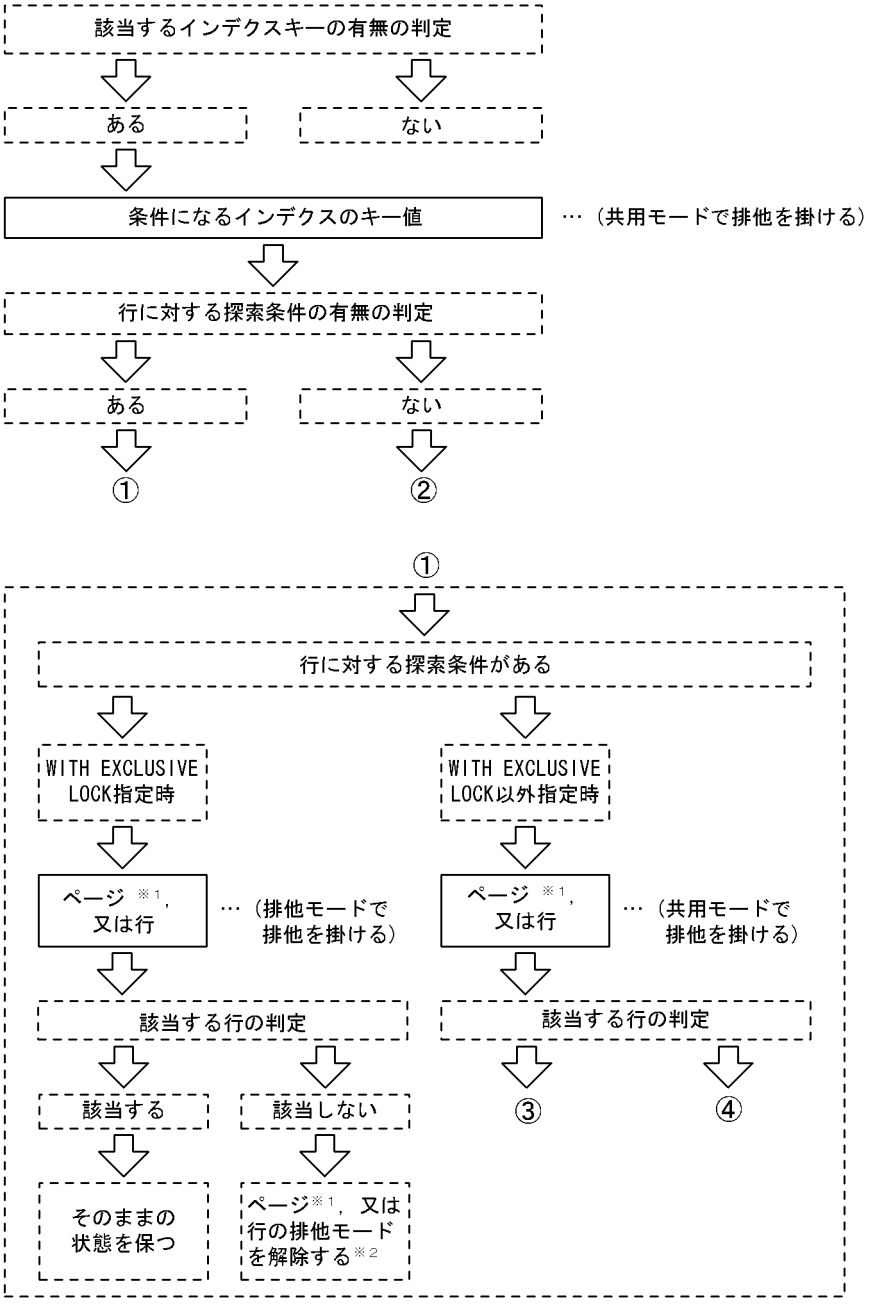
注※１　ページ排他表の場合に該当します。
注※２　インデクスを定義する列に一意性制約(UNIQUE)を指定したときに作成されるインデクスのことです。
注※３　定義数が複数の場合，定義した順序の逆順に処理します。
注※４　クラスタキーを指定した列に作成されるインデクスのことです。
注※５　クラスタキーとして定義した列に一意性制約(UNIQUE)を指定したときに作成されるインデクスのことです。

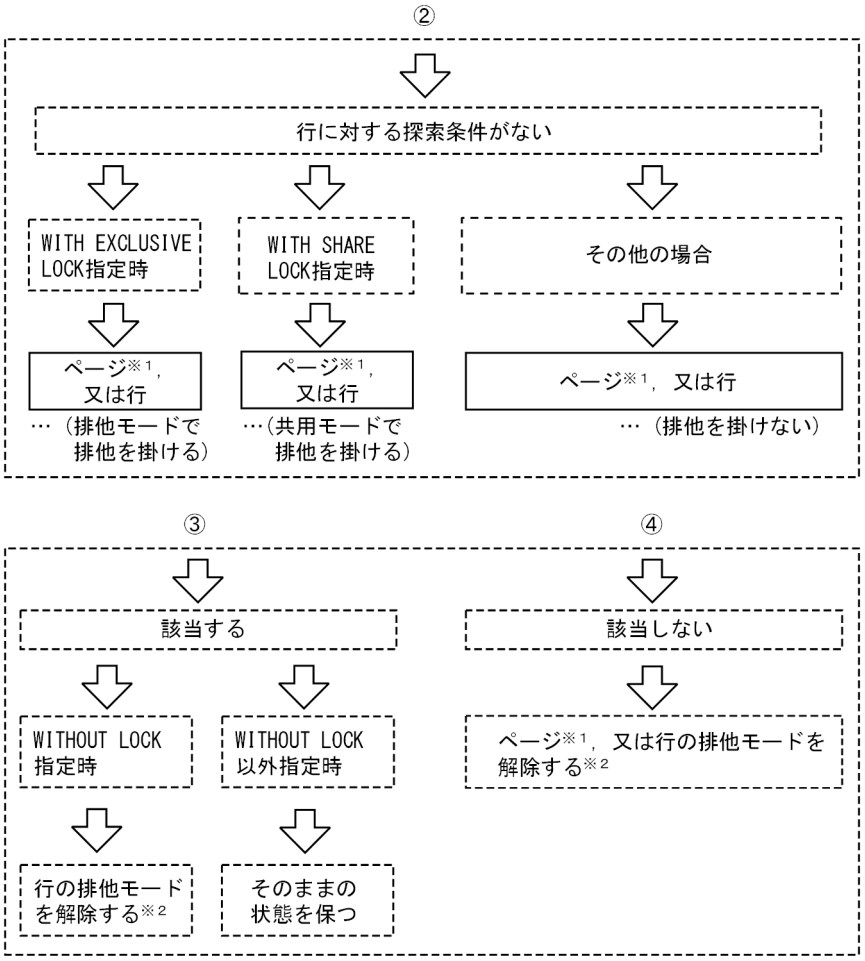
(d)　カーソルを使用した UPDATE 文の場合



注※１　ページ排他表の場合に該当します。
注※２　インデクスを定義する列に一意性制約(UNIQUE)を指定したときに作成されるインデクスのことです。
注※３　定義数が複数の場合，定義した順序の逆順に処理します。
注※４　ユニークインデクス，クラスタリングインデクス※5，及びユニーククラスタインデクス※6以外のインデクスを示します。
注※５　クラスタキーを指定した列に作成されるインデクスのことです。
注※６　クラスタキーとして定義した列に一意性制約(UNIQUE)を指定したときに作成されるインデクスのことです。

(e)　SELECT 文，又は FETCH 文の場合

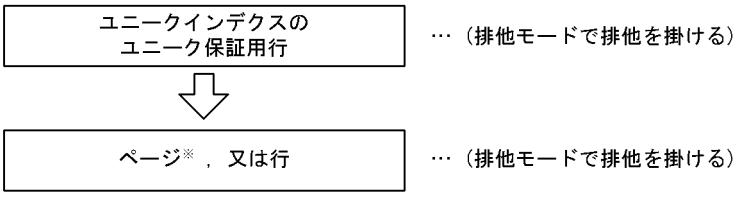




注※１ ページ排他表の場合に該当します。
注※２ 既に共用モードの排他が掛かっている状態で排他モードの排他を掛けた場合，排他モードの処理が先に終了しても排他モードは解除されません。この場合，共用モードと排他モードの処理が終了したときに排他を解除します。
また，図中の「該当する行の判定」で該当しない場合でも，HiRDBが選択したアクセスパスによっては排他を解除できない場合があります（表を結合するSQL文で，結合条件で取り出すデータを絞り込む場合など）。

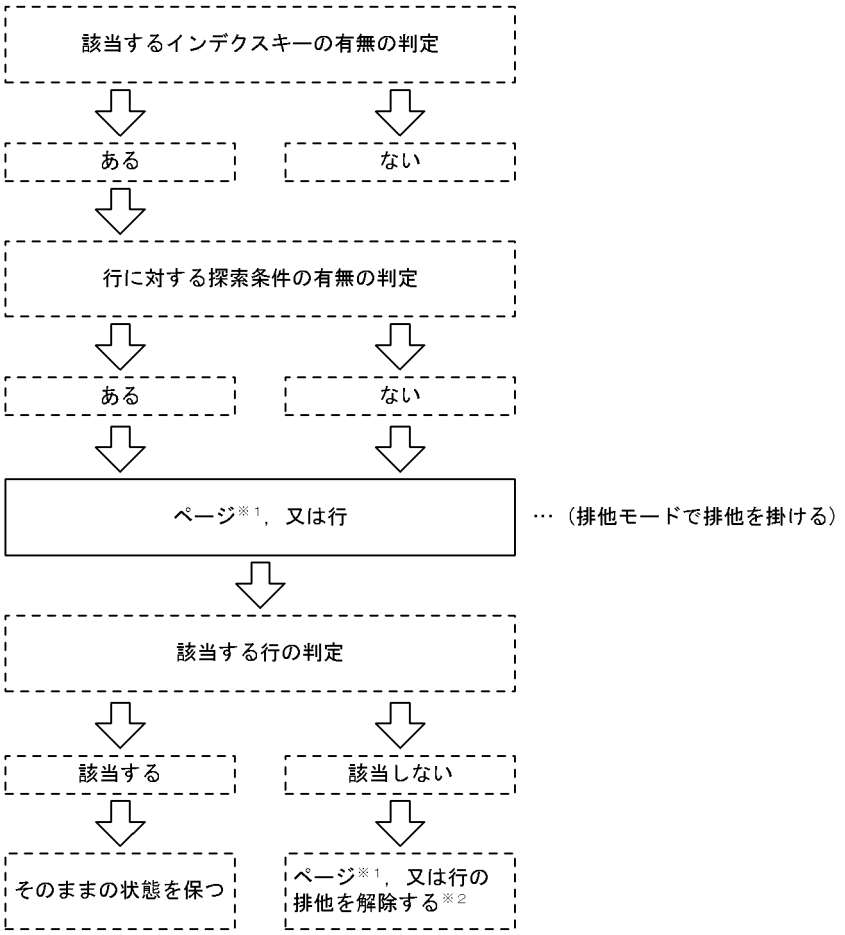
(2) インデクスキー値無排他を使用した場合の操作系 SQL の排他制御の順序

(a) INSERT 文の場合



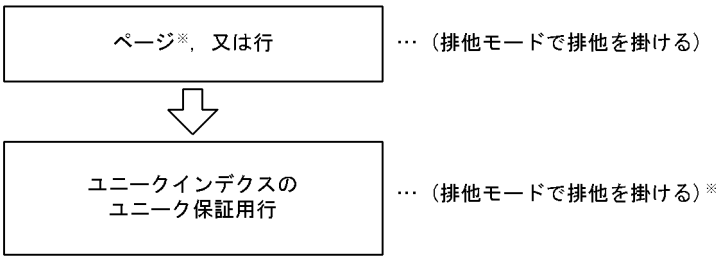
注※　ページ排他表の場合に該当します。

(b)　カーソルを使用しない DELETE 文，又は UPDATE 文で条件に合うデータを探す場合



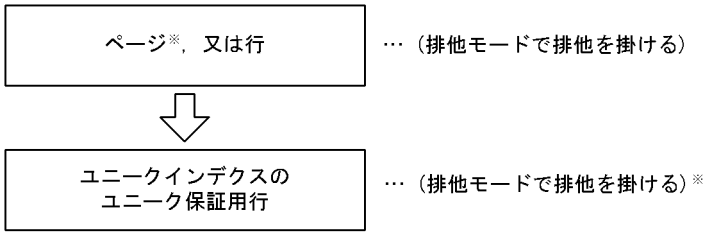
注※１　ページ排他表の場合に該当します。
注※２　既に共用モードの排他が掛かっている状態で排他モードの排他を掛けた場合，排他モードの処理が先に終了しても排他モードは解除されません。この場合，共用モードと排他モードの処理が終了したときに排他を解除します。

(c)　カーソルを使用した DELETE 文



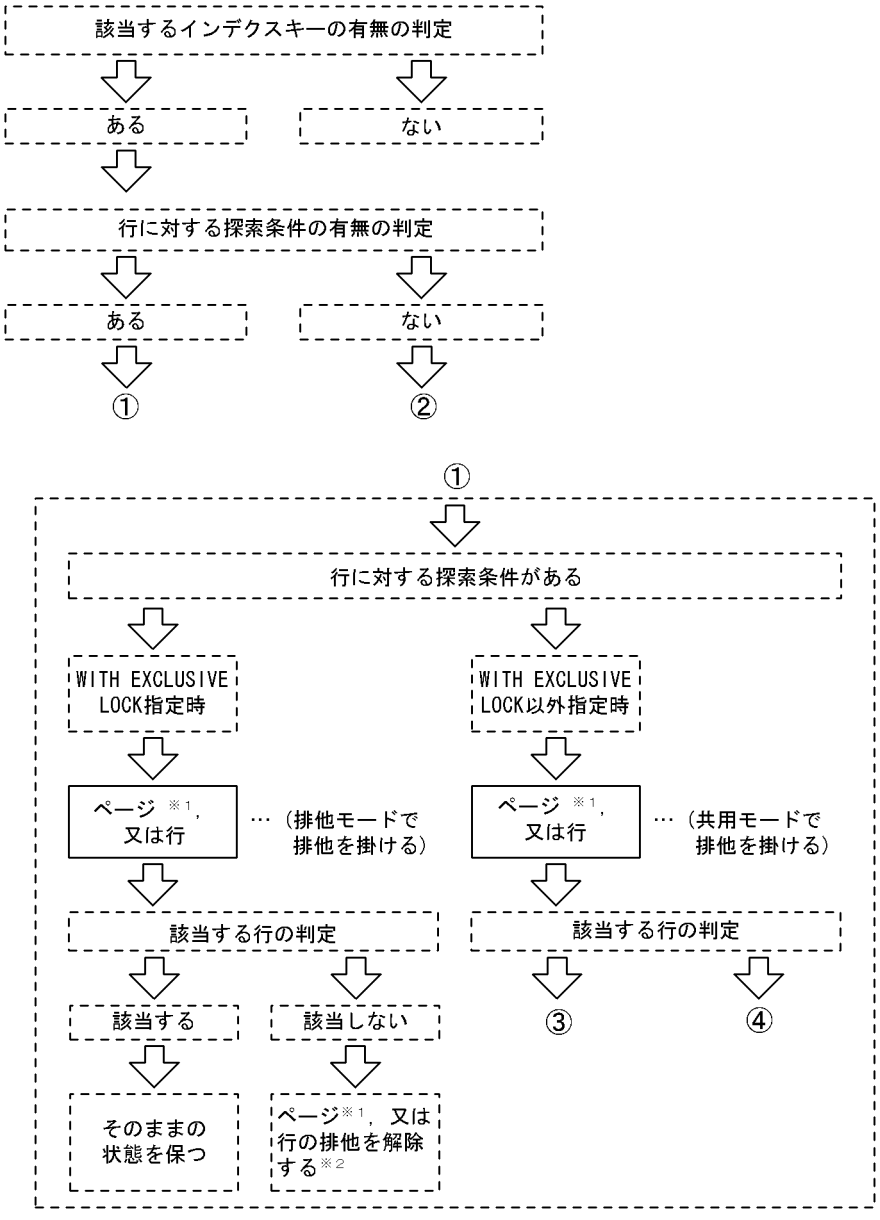
注※　ページ排他表の場合に該当します。

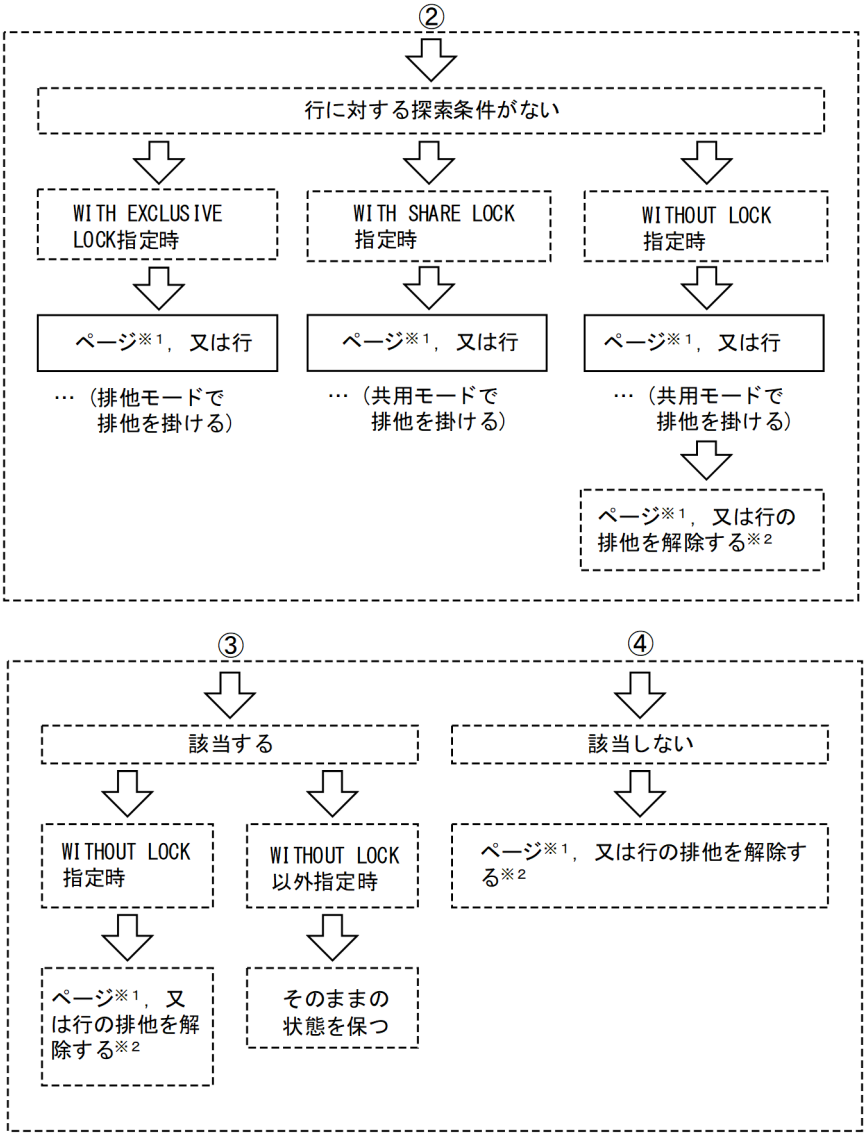
(d)　カーソルを使用した UPDATE 文の場合



注※　ページ排他表の場合に該当します。

(e)　SELECT 文，又は FETCH 文の場合

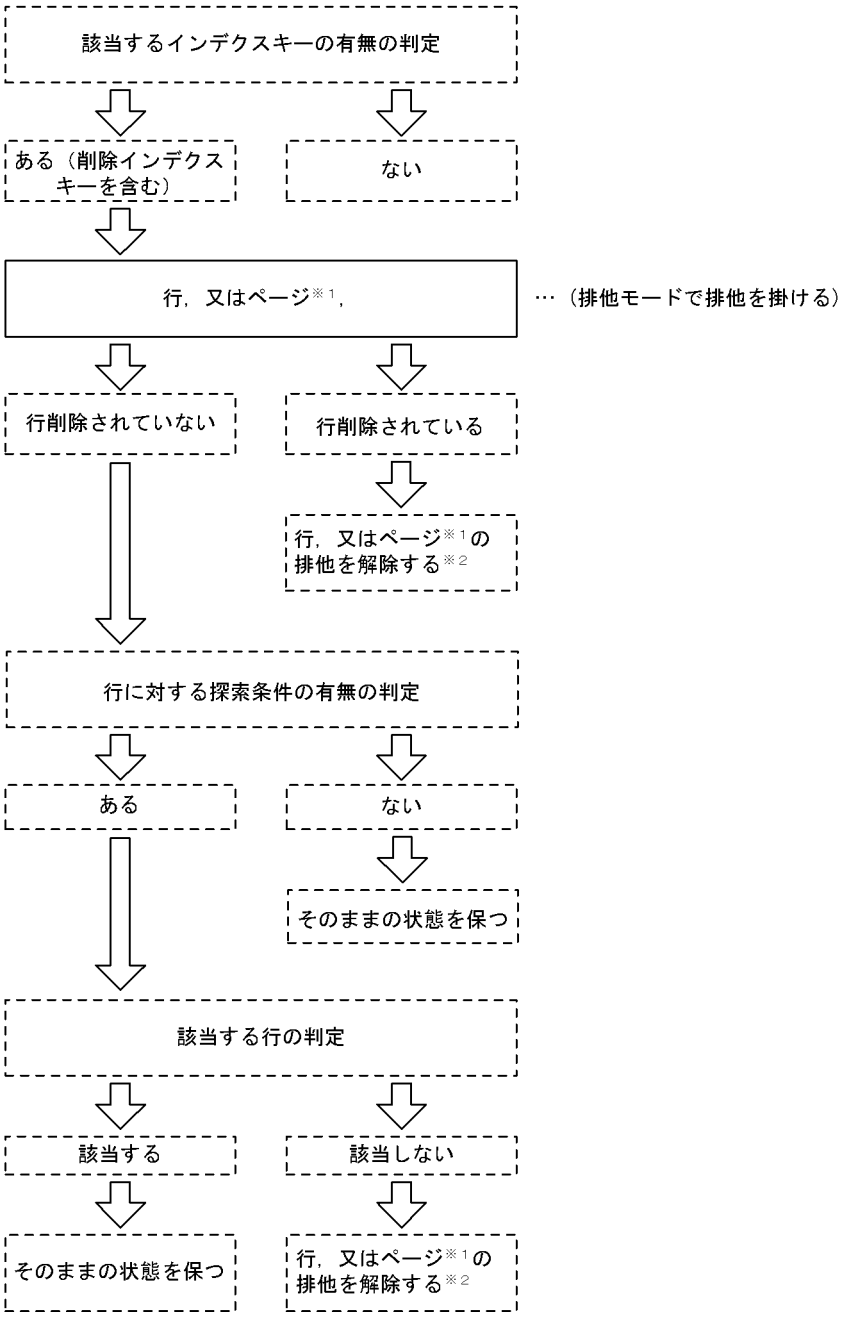




注※１ ページ排他表の場合に該当します。
注※２ 既に共用モードの排他が掛かっている状態で排他モードの排他を掛けた場合，排他モードの処理が先に終了しても排他モードは解除されません。この場合，共用モードと排他モードの処理が終了したときに排他を解除します。
また，図中の「該当する行の判定」で該当しない場合でも，HiRDBが選択したアクセスパスによっては排他を解除できない場合があります（表を結合するSQL文で，結合条件で取り出すデータを絞り込む場合など）。

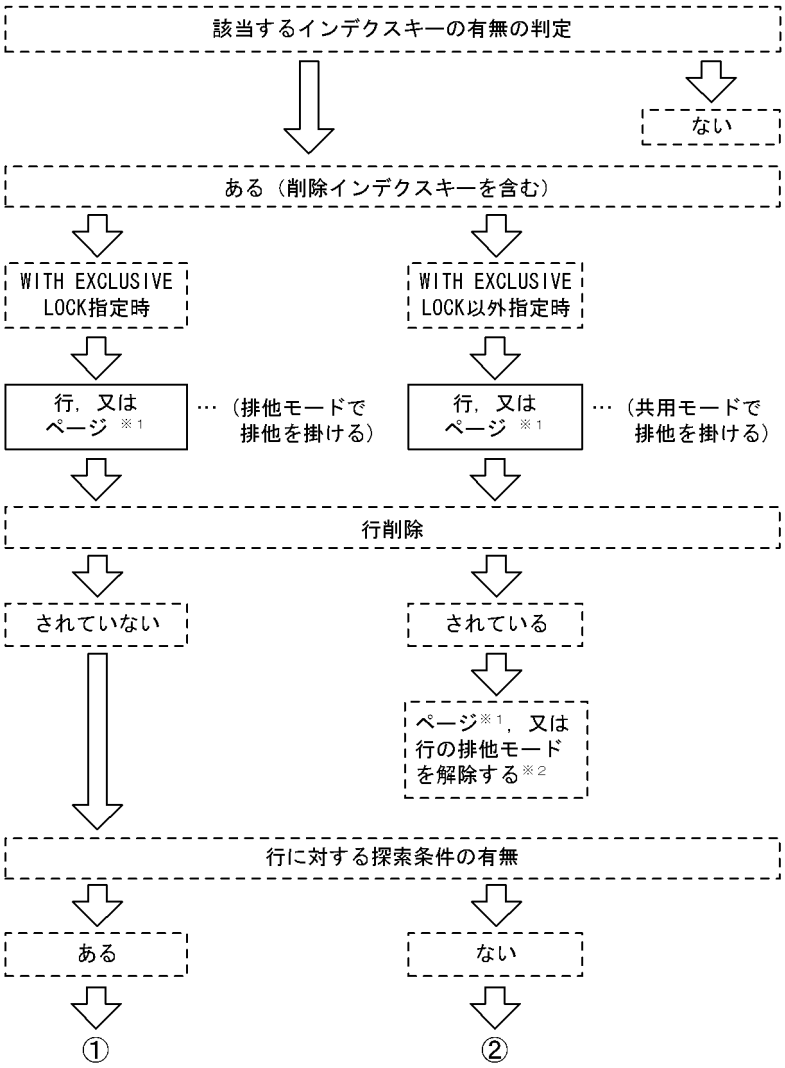
(3)　コミットしていない削除データの排他制御の順序

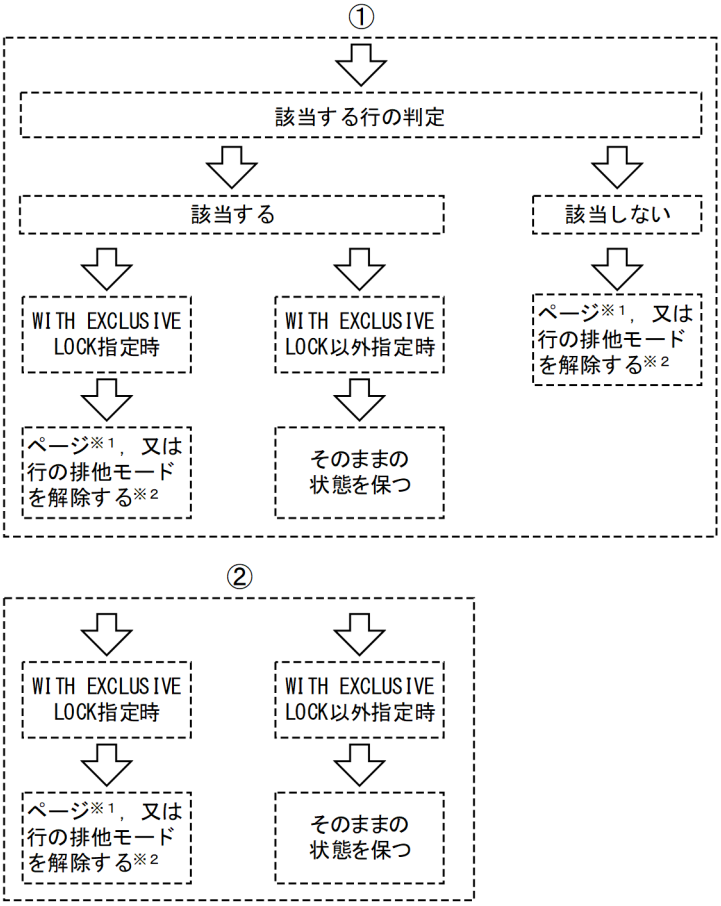
(a)　カーソルを使用しない DELETE 文，又は UPDATE 文で条件に合うデータを探す場合



注※１　ページ排他表の場合はページに排他を掛けます。
注※２　既に共用モードの排他が掛かっている状態で排他モードの排他を掛けた場合，排他モードの処理が先に終了しても排他モードは解除されません。

(b)　SELECT 文，又は FETCH 文の場合





注※１　ページ排他表の場合はページに排他を掛けます。
注※２　既に共用モードの排他が掛かっている状態で排他モードの排他を掛けた場合，排他モードの処理が先に終了しても排他モードは解除されません。また，図中の「該当する行の判定」で該当しない場合でも，HiRDBが選択したアクセスパスによっては排他を解除できない場合があります（表を結合するSQL文で，結合条件で取り出すデータを絞り込む場合など）。

3.4.12　行に掛かる排他制御の順序

HiRDB は SQL の最適化によって決定されたアクセスパスに従ってデータにアクセスします。通常，行排他の順序はこのアクセスパスによって決まります。ただし，SQL 実行時の条件によって行排他の順序が異なるケースがあります。次に示す SQL 実行時の条件では，行排他の順序が異なります。

- パラレルサーバで，かつ複数の BES を使用してデータにアクセスする場合
- スナップショット方式を適用する場合

ここでは，アクセスパスに従った行排他の順序，及び行排他の順序が異なるケースについて説明します。

(1)　アクセスパスに従った行排他の順序

HiRDB は SQL の最適化によって決定されたアクセスパスに従ってデータにアクセスします。行排他の順序はこのアクセスパスによって決まります。

基本的なデータアクセスの順序を次に示す例で説明します。

例

実行する SQL：

```
select * from t1,t2 where t1.c1=t2.c1 and t1.c1>3
```

[条件]

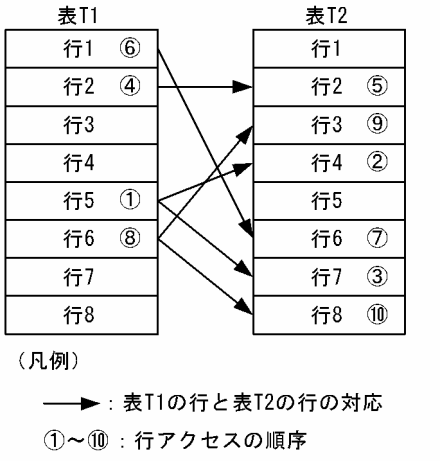
- HiRDB/シングルサーバで実行します。
- t1.c1 及び t2.c1 にインデクスを定義しています。
- t1.c1 のインデクスを用いて t1 にアクセスします。
- t1 の各行に対して t2.c1 のインデクスを用いてネストループジョインを行います。
- t1.c1>3 の条件で t1 の行 5，2，1，6 がヒットし，インデクスによってこの順にアクセスします。
- t1 の行に対してヒットする t2 行は次のとおりです。

  ・t1 の行 1 に対してヒットする t2 の行は 6 です。

  ・t1 の行 2 に対してヒットする t2 の行は 2 です。

  ・t1 の行 5 に対してヒットする t2 の行は 4，及び 7 であり，インデクスによってこの順にアクセスします。

  ・t1 の行 6 に対してヒットする t2 の行は 3，及び 8 であり，インデクスによってこの順にアクセスします。
- スナップショット方式は適用しません。



(2) 行排他の順序が異なるケース

SQL を実行するときの条件によって行排他の順序が異なるケースについて説明します。

(a) HiRDB/パラレルサーバの場合

処理性能向上のために，表を並列に検索，又は更新する場合，HiRDB はデータの先読みや並列での読み込みをします。そのため，結合する表の間でのデータのアクセス順序がシングルサーバの場合と異なること

があります。アクセス順序が異なるため，行排他の順序も異なります。その結果，デッドロックが発生するおそれがあります。

●分割表検索の場合

分割表検索の場合の，データアクセスの順序を次に示す例で説明します。

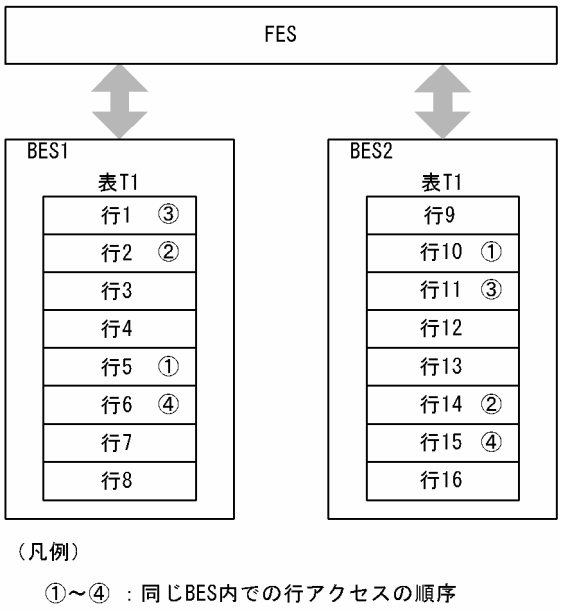
例

実行する SQL：

```
select * from t1 where t1.c1>3
```

[条件]

- HiRDB/パラレルサーバで実行します。
- 表 t1 は，行 1～8 を BES1 に，行 9～16 を BES2 に分割して格納しています。
- t1.c1>3 の条件で，表 T1 の行 1，2，5，6，10，11，14，15 がヒットします。このとき，BES1 中では行 5，2，1，6 の順序でインデクスによってアクセスします。BES2 中では行 10，14，11，15 の順序でインデクスによってアクセスします。
- スナップショット方式は適用しません。



[説明]

各 BES 内でのアクセス順序は決まりますが，データの先読みや並列での読み込みが発生するため，BES1 と BES2 のアクセス順序は決まりません。また，BES1 の行と BES2 の行とのアクセス順序も決まりません。

●結合検索の場合

結合検索の場合の，データアクセスの順序を次に示す例で説明します。

例

実行する SQL：

```
select * from t1,t2 where t1.c1=t2.c1 and t1.c1>3
```

[条件]

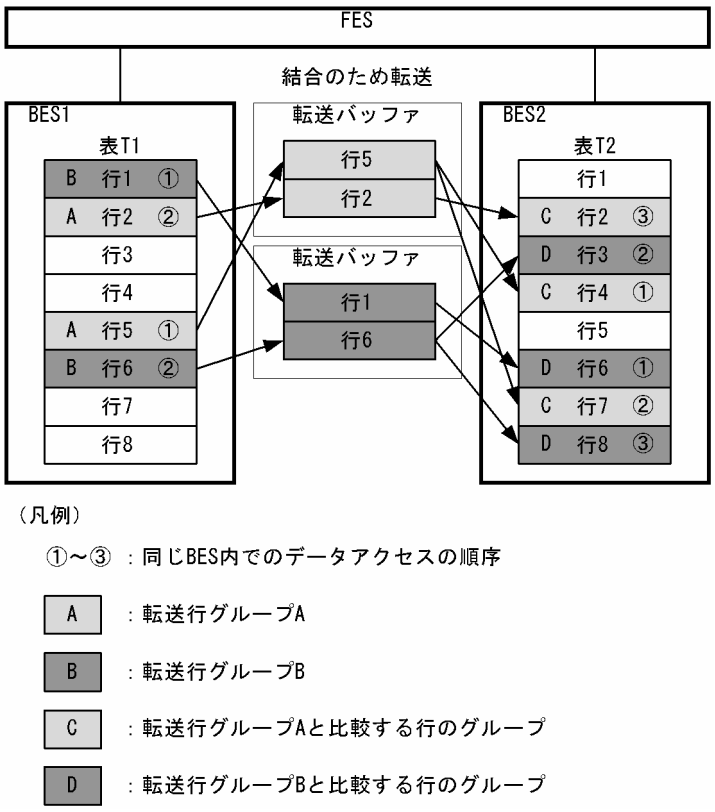
- HiRDB/パラレルサーバで実行します。
- t1.c1 及び t2.c1 にインデクスを定義しています。
- t1.c1 のインデクスを用いて t1 にアクセスします。
- t1 の各行に対して t2.c1 のインデクスを用いてネストループジョインを行います。
- t1.c1>3 の条件で t1 の行 5，2，1，6 がヒットし，インデクスによってこの順にアクセスします。
- t1 の行に対してヒットする t2 行は次のとおりです。

  ・t1 の行 1 に対してヒットする t2 の行は 6 です。

  ・t1 の行 2 に対してヒットする t2 の行は 2 です。

  ・t1 の行 5 に対してヒットする t2 の行は 4，及び 7 であり，インデクスによってこの順にアクセスします。

  ・t1 の行 6 に対してヒットする t2 の行は 3，及び 8 であり，インデクスによってこの順にアクセスします。
- スナップショット方式は適用しません。
- BES 間の転送行数は 2 行です。



[説明]

図中の A～D のアクセス順序は次のようになります。

- A～D 各グループ内のアクセス順序は丸付き数字の順になります。

- B，C のアクセスは A のアクセス後になります。
- データの先読み，又は並列読み込みが発生するため，B，C のアクセス順序は決まりません。
- データの先読み，又は並列読み込みが発生するため，B の行と C の行とのアクセス順序も決まりません。

(b)　スナップショット方式を適用する場合

スナップショット方式を適用する場合，SQL を実行するたびに行排他の順序が異なります。

次の三つのケースについて，行に掛かる排他制御の順序を説明します。

- スナップショット方式を適用する場合（同時実行するトランザクションなし）
- スナップショット方式を適用する場合（同時実行するトランザクションあり）
- スナップショット方式を適用しない場合

次の SQL を例とします。

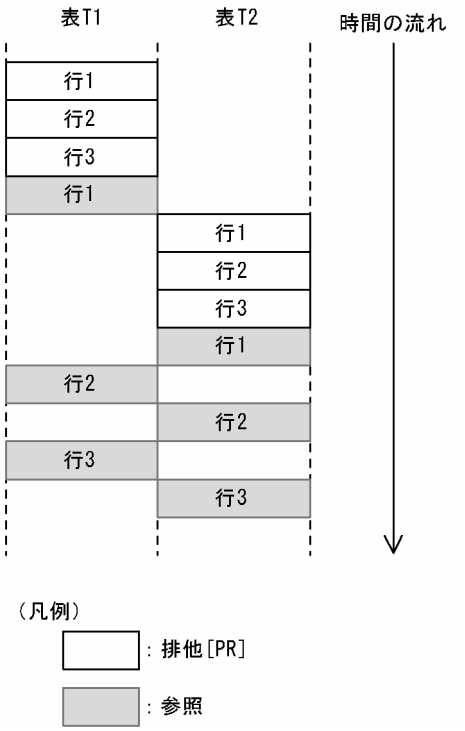
例

```
select * from t1,t2 where t1.c1=t2.c1 and t1.c1>3
```

●スナップショット方式を適用する場合（同時に実行するトランザクションなし）

スナップショット方式を適用する場合で，参照するページ内に更新中の行（排他オプション WITH EXCLUSIVE LOCK で検索した行を含む）がないとき，ページ内のすべての行にまとめて排他を掛けます。

SQL 実行時のデータアクセスの順序を次に示します。



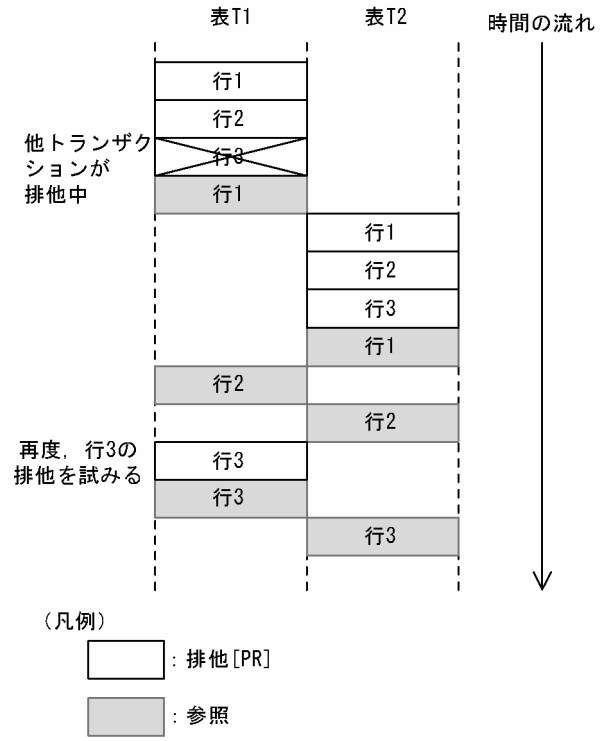
[説明]

- 表 T1，T2 の行 1 は，結合条件に指定した列の値が同じであることを示します。
- 表 T1，T2 の行はすべて同じページに格納されています。

●スナップショット方式を適用する場合（同時に実行するトランザクションあり）

スナップショット方式を適用する場合で，参照するページ内に更新中の行（排他オプション WITH EXCLUSIVE LOCK で検索した行を含む）があるとき，その直前の行までまとめて排他を掛けます。その後排他処理を打ち切り，それまでに排他を掛けた行を参照します。また，排他を掛けられなかった行については，実際に参照するときに排他を掛けます。

SQL 実行時のデータアクセスの順序を次に示します。



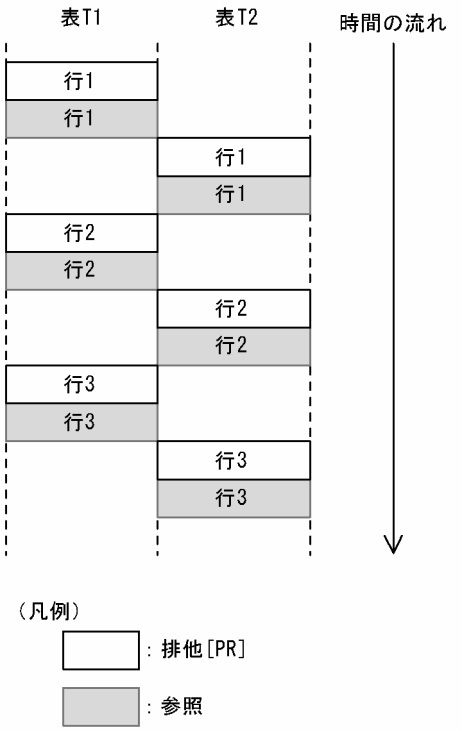
[説明]

- 表 T1，T2 の行 1 は，結合条件に指定した列の値が同じであることを示します。
- 表 T1，T2 の行はすべて同じページに格納されています。
- 表 T1 の行 3 に，ほかのトランザクションが EX モードで排他を掛けています。
- スナップショット方式を適用して結合検索を行う場合，表又はインデクスを更新する UAP や，排他オプション WITH EXCLUSIVE LOCK で検索する UAP がほかに存在すると，同じ SQL を実行しても行排他の順序が入れ替わります。
- スナップショット方式は参照するページの排他をまとめて行うため，インデクスページスプリットが発生した場合でも行排他の順序が入れ替わります。

●スナップショット方式を適用しない場合

スナップショット方式を適用しない場合，一行ごとに排他を掛けます。

SQL 実行時のデータアクセスの順序を次に示します。

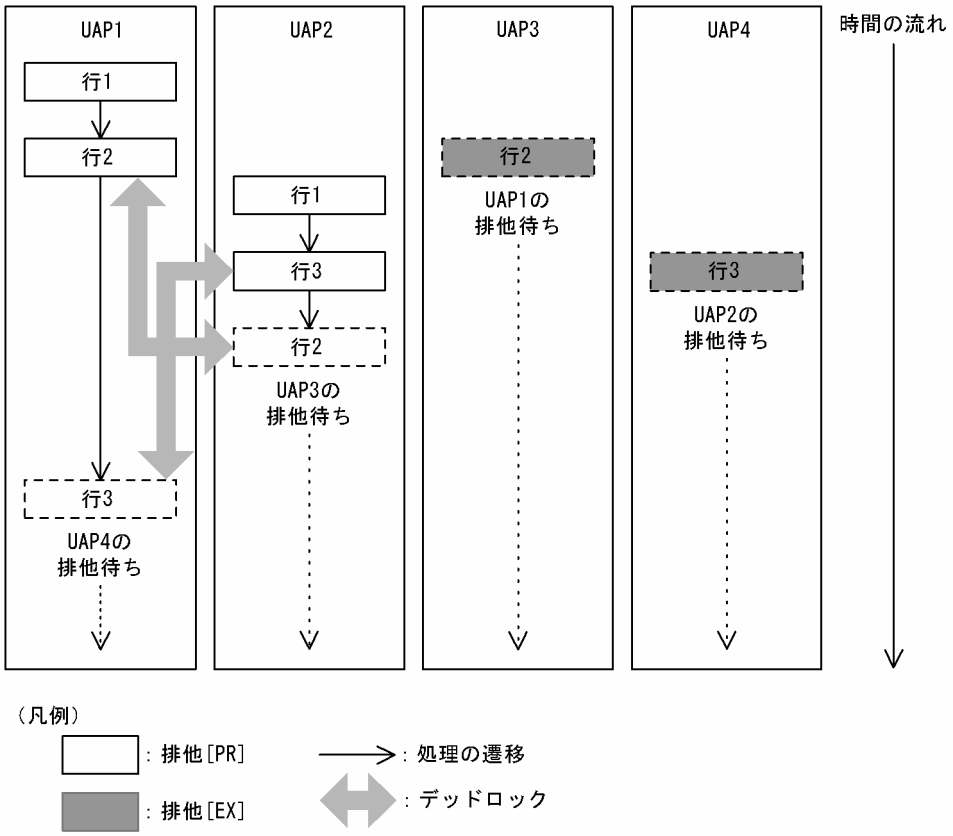


[説明]

表 T1，T2 の行 1 は，結合条件に指定した列の値が同じであることを示します。

(3)　デッドロックの対処方法

行排他の順序が異なるケースでは，EX モードで行排他を掛ける UAP が加わることによって，同じ SQL の同時実行であってもデッドロックが発生するおそれがあります。デッドロックの発生例を次に示します。



[説明]

- UAP1 と UAP2 は PR モードで行排他を掛ける UAP です。
- UAP1 と UAP2 は同じ SQL を実行する UAP ですが，(2)で説明した理由によって，排他の順序が次のように異なります。

  UAP1：行 1→行 2→行 3 の順序で排他を掛けます。

  UAP2：行 1→行 3→行 2 の順序で排他を掛けます。
- UAP3 と UAP4 は EX モードで行排他を掛ける UAP です。

デッドロックの対処方法を次に示します。

- 排他オプションに WITHOUT LOCK NOWAIT 又は WITHOUT LOCK WAIT を指定して検索を行う。
- IN EXCLUSIVE MODE の LOCK TABLE 文を実行してから検索を行う。
- トランザクションを再実行する。

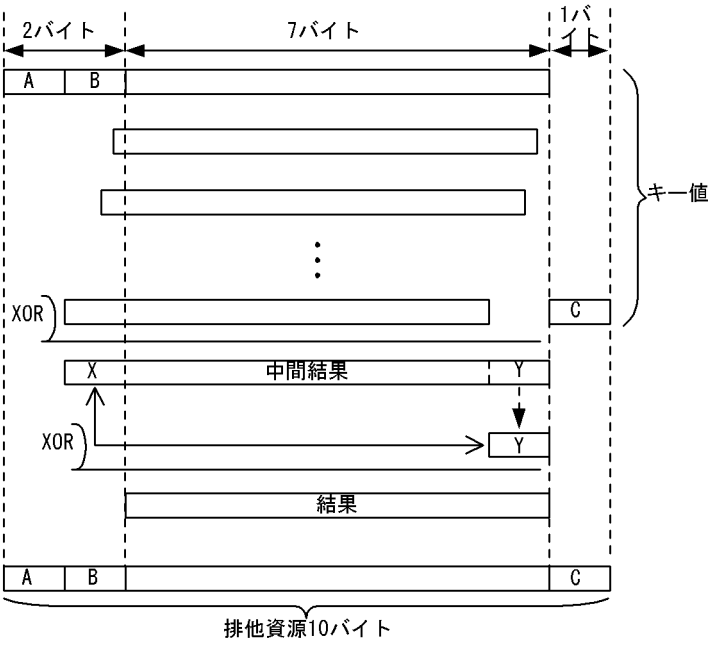
デッドロックが発生した場合については，「デッドロックと回避策」を参照してください。

3.4.13　インデクスキー値の排他資源の作成方法

インデクスのキー値が 10 バイトを超えた場合，システム定義の pd_key_resource_type オペランドの指定値によって，作成されるインデクスキー値の排他資源が変わります。システム定義の pd_key_resource_type オペランドについては，マニュアル「HiRDB システム定義」を参照してください。

pd_key_resource_type オペランドに TYPE1 を指定した場合の，キー値排他資源の作成方法を次の図に示します。

図 3–23　pd_key_resource_type=TYPE1 のキー値排他資源の作成方法



（凡例）A，B，Cは，それぞれ先頭1バイト，2バイト，下位1バイトを示します。

[説明]

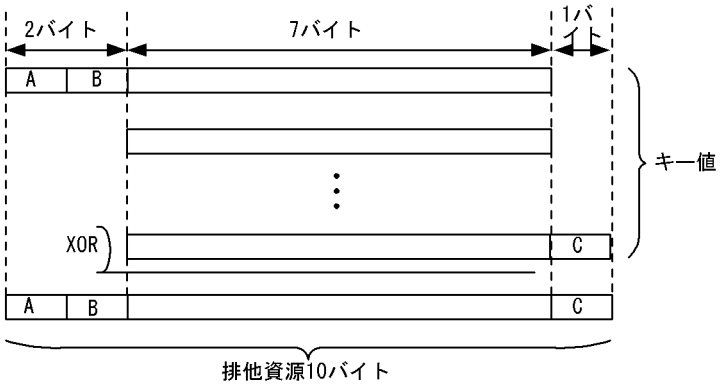
キー長が 10 バイトを超えると，キー値の先頭 2 バイトと下位 1 バイトを除いた長さを 7 バイト単位で切り出し，ビットシフトしながら排他的論理和をします。ビットシフトは，切り出した回数を 8 で割った余りの数で論理シフトし，8 バイトデータの排他的論理和をします。

排他的論理和の結果（中間結果）は，8 バイトの領域に格納し，先頭 1 バイト（X）と下位 1 バイト（Y）の排他的論理和をして，7 バイト分のデータ（結果）を作成します。

この 7 バイト分のデータ（結果）と，最初に除いた先頭 2 バイトと下位 1 バイトとを合わせた，10 バイトのデータがインデクスキー値の排他資源となります。

pd_key_resource_type オペランドに TYPE2 を指定した場合の，キー値排他資源の作成方法を次の図に示します。

図 3–24　pd_key_resource_type=TYPE2 のキー値排他資源の作成方法



（凡例）A，B，Cは，それぞれ先頭1バイト，2バイト，下位1バイトを示します。

[説明]

キー長が 10 バイトを超えると，キー値の先頭 2 バイトと下位 1 バイトを除いた長さを 7 バイト単位で排他的論理和をします。排他的論理和の結果の 7 バイトデータと先頭 2 バイト，下位 1 バイトを合わせた 10 バイトのデータがインデクスキー値の排他資源となります。

3.4.14　複数トランザクションで検索と更新をする場合

複数トランザクションで同一の表を検索及び更新する場合の一般的な処理方法と注意事項について説明します。

(1)　一般的な処理方法

SELECT 文で更新対象行を特定し，UPDATE 文で更新するトランザクションを実行する場合，SELECT 文を実行する前に同一トランザクション中で，LOCK 文を用いて排他モードで表に対して排他制御を行うか，又は SELECT 文の排他制御のモードを次のどちらかの方法で排他モード（EX：Exclusive）にするかしてください。

- クライアント環境定義に次のどちらかを指定し，SELECT 文に FOR UPDATE 指定をする
  - PDISLLVL に 2 を指定する
  - PDFORUPDATEEXLOCK に YES を指定する
- SELECT 文の排他オプションに WITH EXCLUSIVE LOCK を指定する

SELECT 文の排他制御のモードを排他モードにする場合の例を次に示します。

（例）

次の表「在庫テーブル」から「在庫状態」が「在庫確保中」の行を検索します。該当する行があった場合は，「在庫状態」を「在庫あり」に変更します。

表 3–26　例・在庫テーブル

在庫 ID（主キー）	在庫名	在庫状態
20345678	ネクタイ	在庫確保中
20345679	シャツ	在庫あり
20345680	靴下	在庫なし
20345681	ズボン	在庫確保中
20345682	スカート	在庫あり
20345683	パンツ	在庫確保中

```
DECLARE CUR1 CURSOR FOR SELECT 在庫状態 FROM 在庫テーブル
  WHERE 在庫状態='在庫確保中'
    WITH EXCLUSIVE LOCK FOR UPDATE OF 在庫状態
OPEN CUR1
WHILE(SQLCODE == 0){
    FETCH CUR1 INTO :在庫状態
    UPDATE 在庫テーブル SET 在庫状態='在庫あり'
      WHERE CURRENT OF CUR1
}
CLOSE CUR1
```

(2)　検索する行に対して更新する行が非常に少ない場合

SELECT 文でヒットする行に対して UPDATE 文で更新する行が少ない場合，SELECT 文の排他オプションに SHARE LOCK を指定することで，他トランザクションからも参照できるようになるため，同時実行性を向上できます。ただし，他トランザクションと更新する行が重なり，排他制御のモードが共有モードから排他モードに遷移することによってデッドロックが起こるおそれがあるため，注意してください。

(3)　排他の掛かる行を減らすために事前に無排他で検索する場合

SELECT 文を無排他検索にすることで，「検索する行に対して更新する行が非常に少ない場合」よりもさらに同時実行性を向上できます。ただし，無排他検索をしてから UPDATE 文を実行するまでの間に検索した行がほかのトランザクションによって更新される可能性があります。この場合，データの取り出しで読み込む値が，探索条件に合致しない値になることがあります。UPDATE 文を正しく実行するため，無排他にした SELECT 文の探索条件を，UPDATE 文又は排他制御のモードが排他モードの SELECT 文で再評価してください。

無排他にした SELECT 文の探索条件を，排他制御のモードが排他モードの SELECT 文で再評価する場合の例を次に示します。

（例）

無排他の SELECT 文に指定した探索条件「在庫状態='在庫確保中'」を WITH EXCLUSIVE LOCK の指定がある SELECT 文で再評価して，他トランザクションによる変更がないことを確認します。他トランザクションによる変更がなければ更新をします。

```
DECLARE CUR1 CURSOR FOR SELECT 在庫ID FROM 在庫テーブル
  WHERE 在庫状態='在庫確保中' WITHOUT LOCK
OPEN CUR1
WHILE(SQLCODE == 0){
    FETCH CUR1 INTO :在庫ID_WITHOUTLCK
    /* 行を絞り込むためのUAP側の処理 */
         .
         .
         .
    DECLARE CUR2 CURSOR FOR SELECT 在庫状態 FROM 在庫テーブル
      WHERE 在庫ID=:在庫ID_WITHOUTLCK AND 在庫状態='在庫確保中'
        WITH EXCLUSIVE LOCK FOR UPDATE OF 在庫状態
    OPEN CUR2
    FETCH CUR2 INTO :在庫状態_WITHLCK
    IF(SQLCODE == 0){
        UPDATE 在庫テーブル SET 在庫状態='在庫あり'
          WHERE CURRENT OF CUR2
    }
    CLOSE CUR2
}
CLOSE CUR1
```

3.5　カーソルの効果

UAP では，カーソルを利用して検索結果を取り出すことができます。

カーソルを使用する場合，DECLARE CURSOR でカーソルを宣言するか，又は ALLOCATE CURSOR でカーソルを割り当てます。

ここでは，カーソルを使用したときの効果，使用するときに考慮する内容について説明します。

3.5.1　カーソルを使用して表を操作するときの留意事項

(1)　カーソルの更新可能性，及びカーソルを使用した操作有無と，カーソルを使用しない操作との関連

カーソルを宣言，又は割り当てた後，OPEN 文でカーソルを開くと，データを取り出して，参照や更新などの操作ができます。また，カーソル宣言中，並びにカーソル宣言に指定した SQL 文識別子，及びカーソル割り当てに指定した拡張文名が識別する動的 SELECT 文中の，FOR READ ONLY 句，又は FOR UPDATE 句の指定の有無と，そのカーソルを使用した操作（更新，及び削除）の有無によって，カーソルを開いた後にカーソルを使用しない操作ができるかどうかが決まります。

カーソルの更新可能性とカーソルを使用しない操作との関連を次の表に示します。なお，SQL 最適化オプションの更新 SQL の作業表作成抑止を指定すると，カーソルを使用しない操作の制限が緩和されます。

表 3–27　カーソルの更新可能性とカーソルを使用しない操作との関連

条件				カーソルを使用しない操作							
カーソルの更新可能性の指定		カーソルを使用した操作		SQL 最適化オプションの更新 SQL の作業表作成抑止を適用しない場合				SQL 最適化オプションの更新 SQL の作業表作成抑止を適用して，インデクスキー値無排他機能を使用			
		更新	削除	検索	更新	削除	追加	検索	更新	削除	追加
静的 SQL	FOR READ ONLY 句指定あり	なし	なし	○	○	○	○	○	○	○	○
	FOR UPDATE OF 列名指定あり	なし	なし	○	△	×	×	○	○※2	○※2	○※2
		なし	あり	○	○	○	○	○	○	○	○
		あり	なし	○	△	×	×	○	○※2	○※2	○※2
		あり	あり	○	○	○	○	○	○	○	○
	FOR UPDATE 句指定あり	なし	なし	○	○	○	○	○	○	○	○
		なし	あり	○	○	○	○	○	○	○	○
		あり	なし	○	○	○	○	○	○	○	○

条件				カーソルを使用しない操作							
カーソルの更新可能性の指定		カーソルを使用した操作		SQL 最適化オプションの更新 SQL の作業表作成抑止を適用しない場合				SQL 最適化オプションの更新 SQL の作業表作成抑止を適用して，インデクスキー値無排他機能を使用			
		更新	削除	検索	更新	削除	追加	検索	更新	削除	追加
		あり	あり	○	○	○	○	○	○	○	○
	上記の指定なし※1	なし	なし	○	×	×	×	○	○※2	○※2	○※2
		なし	あり	○	○	○	○	○	○	○	○
		あり	なし	○	○	○	○	○	○	○	○
		あり	あり	○	○	○	○	○	○	○	○
動的SQL	FOR READ ONLY 句指定あり	なし	なし	○	○	○	○	○	○	○	○
	FOR UPDATE OF 列名指定あり	なし	なし	○	△	×	×	○	○※2	○※2	○※2
		なし	あり	○	○	○	○	○	○	○	○
		あり	なし	○	△	×	×	○	○※2	○※2	○※2
		あり	あり	○	○	○	○	○	○	○	○
	FOR UPDATE 句指定あり	なし	なし	○	○	○	○	○	○	○	○
		なし	あり	○	○	○	○	○	○	○	○
		あり	なし	○	○	○	○	○	○	○	○
		あり	あり	○	○	○	○	○	○	○	○
	上記の指定なし	なし	なし	○	×	×	×	○	○※2	○※2	○※2
		なし	あり	○	○	○	○	○	○	○	○
		あり	なし	○	○	○	○	○	○	○	○
		あり	あり	○	○	○	○	○	○	○	○

（凡例）

○：操作できます。

△：指定した列の更新ができます。

×：操作できません。

注※1

CURRENT OF カーソル名を指定した更新，又は削除が同一ポストソース内にある場合は，FOR UPDATE が仮定されます。

注※2

カーソルを使用した検索で使っているインデクスを更新した場合，カーソルでの検索結果は保証されません。例と対策方法を次に示します。

＜例＞

```
CREATE INDEX X1 ON T1(C1);
DECLARE CR1 CURSOR FOR SELECT C1 FROM T1 WHERE C1>0;

宣言したカーソルを使用して，次のFETCH文，UPDATE文を繰り返し実行します。

FETCH CR1 INTO :XX;
UPDATE T1 SET C1=10;

C1=10に更新した行が再度検索されます。
```

＜対策方法＞

次のどちらかの対策をしてください。

- UPDATE 文の更新値が，検索の探索条件を満たさないように探索条件を変更してください。

  （例）WHERE C1>0 AND C1 <>10
- 検索に使用するインデクスの構成列から，該当する列を削除してください。ただし，インデクス構成列の一部を削除した場合，その列が探索条件で十分に絞り込める列だったときは，性能が劣化するので注意してください。また，インデクス構成列の一部を削除した場合，インデクスキーの重複数が多くなり，排他待ち，及びデッドロックの多発を招くおそれがあるため，注意してください。したがって，この対策をする場合には，十分に検証した上で採用するようにしてください。

(2)　二つ以上のカーソルを同時に使用する場合

二つ以上のカーソルを使用して同じ表を同時に更新する場合，それぞれのカーソル宣言，又は動的 SELECT 文の FOR UPDATE 句の列名の並びに，更新する列をすべて指定します。例えば，カーソル 1 で列 1 を更新し，カーソル 2 で列 2 を更新する場合，カーソル 1，及びカーソル 2 を宣言するときに，FOR UPDATE 句に列 1，及び列 2 の両方を指定します。カーソル 1 で列 1 だけ，カーソル 2 で列 2 だけを指定した場合，更新時にエラーになります。

3.5.2　FOR UPDATE 句と FOR READ ONLY 句の使い分け

カーソルを使用して検索中の表に対して行の更新，削除，及び挿入する場合，DECLARE CURSOR，又は ALLOCATE CURSOR でカーソルを定義する必要があります。このとき，UAP の処理内容に応じて，FOR UPDATE 句（FOR UPDATE OF 句を含む），及び FOR READ ONLY 句を指定します。

カーソルを使用した行の更新又は削除をする場合，検索した行のほとんどを更新しないときは，排他オプションとして WITH SHARE LOCK を指定するとよいです。排他オプションを省略した場合は，WITH EXCLUSIVE LOCK が仮定されます。

FOR UPDATE 句（FOR UPDATE OF 句を含む），及び FOR READ ONLY 句は，指定によって処理効率が著しく低下する場合があるので注意が必要です。

FOR UPDATE 句（FOR UPDATE OF 句を含む），及び FOR READ ONLY 句を指定するときに考慮する内容を次の表に示します。

表 3–28　FOR UPDATE 句，及び FOR READ ONLY 句を指定するとき考慮する内容

指定句	用途	考慮する内容
FOR UPDATE 句	カーソルを使用して検索中の表に対して，そのカーソルを使用した行の更新，又は削除をして，更にカーソルを使用しない行の更新，削除，又は追加をする場合に指定する。	カーソルを使用した行の検索中に，対象のインデクスが更新された場合でも，動作を保証するため，1 回目の FETCH で内部的に作業表を作成する。この作業表の作成が検索時のオーバヘッドになる。
FOR UPDATE OF 句	カーソルを使用して検索中の表に対して，一部の列だけを更新する場合に指定する。	列名で指定された列に付けられたインデクスが検索に使用されると，1 回目の FETCH で内部的に作業表を作成する。この作業表の作成が検索時のオーバヘッドになる。
FOR READ ONLY 句	カーソルを使用して検索中に，ほかのカーソルを指定して更新（削除，挿入を含む）をする場合，又は直接探索条件を指定して更新（削除を含む）をする場合に指定する。	カーソルを使用した検索中にほかのカーソルで更新する場合，処理結果に影響しないようにするため，1 回目の FETCH で内部的に作業表を作成する。この作業表の作成が検索時のオーバヘッドになる。

FOR UPDATE 句，及び FOR READ ONLY 句を指定しない場合でも，更新（削除を含む）をするときは，1 回目の FETCH で内部的な作業表を作成することがあるので，オーバヘッドを考慮する必要があります。

なお，検索だけをするときは内部的な作業表を作成しないので，オーバヘッドを考慮する必要ありません。

3.5.3　カーソル宣言と排他の関係

FETCH を実行するとき，又は 1 行 SELECT 文を実行するときの排他制御モードはカーソル宣言時，動的 SELECT 文，又は 1 行 SELECT 文の前処理時の排他オプションが優先されます。排他オプションの指定がない場合は，データ保証レベル（データ保証レベル指定がない場合は 2 を仮定）に従います。データ保証レベルは，システム定義の pd_isolation_level オペランド，クライアント環境定義の PDISLLVL，又は手続き定義時若しくはトリガ定義時に指定する SQL コンパイルオプションの ISOLATION LEVEL で指定します。このとき，カーソルを使用した更新（又は削除）の有無，FOR UPDATE 時の WITH EXCLUSIVE LOCK 仮定によっても影響を受けます。

FOR UPDATE 時の WITH EXCLUSIVE LOCK 仮定の指定は，クライアント環境定義の PDFORUPDATEEXLOCK，手続き定義時又はトリガ定義時の SQL コンパイルオプションのデータ保証レベル（FOR UPDATE EXCLUSIVE を指定）で行います。

カーソル宣言時（DECLARE CURSOR）の排他オプション，カーソル宣言（DECLARE CURSOR）やカーソル割当て（ALLOCATE CURSOR）に指定した動的 SELECT 文の排他オプション，又は 1 行 SELECT 文に指

定した排他オプションによって，実行時の排他制御モードが異なります。カーソル宣言時，又は動的 SELECT 文前処理時の排他オプションと表操作時の排他オプションの関係を次の表に示します。

なお，カーソル宣言時の排他オプションについては，マニュアル「HiRDB SQL リファレンス」を参照してください。

表 3–29　カーソル宣言時，又は動的 SELECT 文前処理時の排他オプションと主問合せの表操作時の排他オプションの関係

主問合せの表への LOCK 文指定		SQL 文中の排他オプション		カーソルを使用した更新許可※	FOR UPDATE 時の WITH EXCLUSIVE LOCK 仮定	データ保証レベル	主問合せの表操作時の排他オプション
あり	EXCLUSIVE	あり	WITH EXCLUSIVE LOCK	－	－	－	WITH EXCLUSIVE LOCK
			WITH SHARE LOCK				WITH EXCLUSIVE LOCK
			WITHOUT LOCK WAIT				WITH EXCLUSIVE LOCK
			WITHOUT LOCK NOWAIT	あり			エラー（KFPA11156-E）
				なし			WITH EXCLUSIVE LOCK
		なし		－	－	－	WITH EXCLUSIVE LOCK
	SHARE	あり	WITH EXCLUSIVE LOCK	－	－	－	WITH EXCLUSIVE LOCK
			WITH SHARE LOCK				WITH SHARE LOCK
			WITHOUT LOCK WAIT				WITH SHARE LOCK
			WITHOUT LOCK NOWAIT	あり			エラー（KFPA11156-E）
				なし			WITH SHARE LOCK
		なし		あり	あり	－	WITH EXCLUSIVE LOCK
					なし	2	WITH EXCLUSIVE LOCK
						1	WITH SHARE LOCK
						0	WITH SHARE LOCK

<table>
<tr><th colspan="2">主問合せの表への LOCK 文指定</th><th colspan="2">SQL 文中の排他オプション</th><th>カーソルを使用した更新許可※</th><th>FOR UPDATE 時の WITH EXCLUSIVE LOCK 仮定</th><th>データ保証レベル</th><th>主問合せの表操作時の排他オプション</th></tr>
<tr><td colspan="2"></td><td colspan="2"></td><td>なし</td><td>－</td><td>－</td><td>WITH SHARE LOCK</td></tr>
<tr><td colspan="2" rowspan="8">なし</td><td rowspan="5">あり</td><td>WITH EXCLUSIVE LOCK</td><td rowspan="3">－</td><td rowspan="5">－</td><td rowspan="5">－</td><td>WITH EXCLUSIVE LOCK</td></tr>
<tr><td>WITH SHARE LOCK</td><td>WITH SHARE LOCK</td></tr>
<tr><td>WITHOUT LOCK WAIT</td><td>WITHOUT LOCK WAIT</td></tr>
<tr><td rowspan="2">WITHOUT LOCK NOWAIT</td><td>あり</td><td>エラー（KFPA11156-E）</td></tr>
<tr><td>なし</td><td>WITHOUT LOCK NOWAIT</td></tr>
<tr><td colspan="2" rowspan="7">なし</td><td>あり</td><td>あり</td><td>－</td><td>WITH EXCLUSIVE LOCK</td></tr>
<tr><td rowspan="3">なし</td><td rowspan="3">2</td><td>WITH EXCLUSIVE LOCK</td></tr>
<tr><td rowspan="2">1</td><td rowspan="2">WITHOUT LOCK WAIT</td></tr>
<tr><td rowspan="2">0</td></tr>
<tr><td rowspan="3">なし</td><td rowspan="3">－</td><td>2</td><td>WITH SHARE LOCK</td></tr>
<tr><td>1</td><td>WITHOUT LOCK WAIT</td></tr>
<tr><td>0</td><td>WITHOUT LOCK NOWAIT</td></tr>
</table>

（凡例）

－：指定内容に関係なく他の項目の指定が優先されます。

注

指定した排他オプションによって，実行時に次に示すような現象が発生する場合があります。

- WITH SHARE LOCK を指定した場合
  更新するときに表中の行を共用モードから排他モードにするため，デッドロックになることがあります。
- WITHOUT LOCK WAIT を指定した場合
  ほかのトランザクションによっては，不正更新（二重更新）や削除エラーになることがあります。

- WITHOUT LOCK NOWAIT を指定した場合
  WITHOUT LOCK NOWAIT を指定して検索した表に対して更新する SQL 文があるとエラーになります。

注※

次の場合，カーソルを使用した更新許可ありとなり，FOR UPDATE 句が仮定されます。

- FOR UPDATE 句を指定した場合
- FOR UPDATE 句を指定しないで，同一カーソル（カーソル宣言で指定したカーソル）を指定した UPDATE 文，又は DELETE 文がある場合

次の場合，カーソルを使用した更新許可なしとなります。

- FOR UPDATE 句を指定しないで，同一カーソル（カーソル宣言で指定したカーソル）を指定した UPDATE 文，又は DELETE 文もない場合

表 3–30　カーソル宣言時，又は動的 SELECT 文前処理時の排他オプションと副問合せの表操作時の排他オプションの関係

副問合せの表への LOCK 文指定		SQL 文中の排他オプション※1		カーソルを使用した更新許可※2	FOR UPDATE 時の WITH EXCLUSIVE LOCK 仮定	データ保証レベル	副問合せの表操作時の排他オプション
あり	EXCLUSIVE	あり	WITH EXCLUSIVE LOCK	－	－	－	WITH EXCLUSIVE LOCK
			WITH SHARE LOCK				WITH EXCLUSIVE LOCK
			WITHOUT LOCK WAIT				WITH EXCLUSIVE LOCK
			WITHOUT LOCK NOWAIT	あり			エラー（KFPA11156-E）
				なし			WITH EXCLUSIVE LOCK
		なし		－	－	－	WITH EXCLUSIVE LOCK
	SHARE	あり	WITH EXCLUSIVE LOCK	－	－	－	WITH SHARE LOCK
			WITH SHARE LOCK				WITH SHARE LOCK
			WITHOUT LOCK WAIT				WITH SHARE LOCK
			WITHOUT LOCK NOWAIT	あり			エラー（KFPA11156-E）
				なし			WITH SHARE LOCK

<table>
<tr><th colspan="2">副問合せの表への LOCK 文指定</th><th colspan="2">SQL 文中の排他オプション※1</th><th>カーソルを使用した更新許可※2</th><th>FOR UPDATE 時の WITH EXCLUSIVE LOCK 仮定</th><th>データ保証レベル</th><th>副問合せの表操作時の排他オプション</th></tr>
<tr><td colspan="2" rowspan="5"></td><td colspan="2" rowspan="5">なし</td><td rowspan="4">あり</td><td>あり</td><td>−</td><td>WITH SHARE LOCK</td></tr>
<tr><td rowspan="4">なし</td><td>2</td><td>WITH SHARE LOCK</td></tr>
<tr><td>1</td><td>WITH SHARE LOCK</td></tr>
<tr><td>0</td><td>WITH SHARE LOCK</td></tr>
<tr><td>なし</td><td>−</td><td>−</td><td>WITH SHARE LOCK</td></tr>
<tr><td rowspan="8">なし</td><td rowspan="5">あり</td><td>WITH EXCLUSIVE LOCK</td><td rowspan="3">−</td><td rowspan="5">−</td><td rowspan="5">−</td><td>WITH SHARE LOCK</td></tr>
<tr><td>WITH SHARE LOCK</td><td>WITH SHARE LOCK</td></tr>
<tr><td>WITHOUT LOCK WAIT</td><td>WITHOUT LOCK WAIT</td></tr>
<tr><td rowspan="2">WITHOUT LOCK NOWAIT</td><td>あり</td><td>エラー（KFPA11156-E）</td></tr>
<tr><td>なし</td><td>WITHOUT LOCK NOWAIT</td></tr>
<tr><td colspan="2" rowspan="7">なし</td><td>あり</td><td>あり</td><td>−</td><td>WITH SHARE LOCK</td></tr>
<tr><td rowspan="6">なし</td><td rowspan="3">なし</td><td rowspan="2">2</td><td>WITH SHARE LOCK</td></tr>
<tr><td rowspan="2">1</td><td rowspan="2">WITHOUT LOCK WAIT</td></tr>
<tr><td>0</td></tr>
<tr><td rowspan="3">なし</td><td rowspan="3">−</td><td>2</td><td>WITH SHARE LOCK</td></tr>
<tr><td>1</td><td>WITHOUT LOCK WAIT</td></tr>
<tr><td>0</td><td>WITHOUT LOCK NOWAIT</td></tr>
</table>

（凡例）

−：指定内容に関係なく他の項目の指定が優先されます。

注

指定した排他オプションによって，実行時に次に示すような現象が発生する場合があります。

- WITH SHARE LOCK を指定した場合

  更新するときに表中の行を共用モードから排他モードにするため，デッドロックになることがあります。

- WITHOUT LOCK WAIT を指定した場合
  ほかのトランザクションによっては，不正更新（二重更新）や削除エラーになることがあります。
- WITHOUT LOCK NOWAIT を指定した場合
  WITHOUT LOCK NOWAIT を指定して検索した表に対して更新する SQL 文があるとエラーになります。

注※1

主問合せに指定した SQL 文中の排他オプションを示します（副問合せに排他オプションは指定できません）。

注※2

次の場合，カーソルを使用した更新許可ありとなり，FOR UPDATE 句が仮定されます。

- 主問合せに FOR UPDATE 句を指定した場合（副問合せに FOR UPDATE 句は指定できません）
- 主問合せに FOR UPDATE 句を指定しないで，同一カーソル（カーソル宣言で指定したカーソル）を指定した UPDATE 文，又は DELETE 文がある場合

次の場合，カーソルを使用した更新許可なしとなります。

- 主問合せに FOR UPDATE 句を指定しないで，同一カーソル（カーソル宣言で指定したカーソル）を指定した UPDATE 文，又は DELETE 文もない場合

HiRDB では，FETCH 文で検索対象となる行を先読みする場合があります。先読みした行には排他を取得するため，FETCH 文を発行して取得している行数以上に排他を取得します。このため，検索対象となる行数ではなく，FETCH 文で取得する行数で排他資源数を見積もると，排他制御のためのバッファが不足する場合があります。カーソルを使用した操作をする場合は，検索対象となる行数で排他資源数を見積もってください。排他資源数を抑えるためには，探索条件で対象となる行を絞り込む必要があります。対象となる行を絞り込むことができない場合は，排他の単位を変更するなど排他制御の抑止を検討する必要があります。排他制御の抑止については，「UAP でできる排他制御」を参照してください。

3.5.4　ホールダブルカーソル

(1)　ホールダブルカーソルとは

ホールダブルカーソルとは，COMMIT 文を実行しても閉じないカーソルのことをいいます。

Type4 JDBC ドライバでは，コミット実行後も ResultSet オブジェクトを有効にする場合に，ホールダブルカーソル機能を使用します。

埋込み型 UAP の場合

ホールダブルカーソルを使用する場合，DECLARE CURSOR に UNTIL DISCONNECT，又は WITH HOLD を指定してカーソルを宣言します。この指定をすると，CLOSE 文，DISCONNECT 文，又は ROLLBACK 文（エラー発生などで暗黙的に実行される ROLLBACK や DISCONNECT 処理を含む）を実行するまでカーソルを開いたままにできます。

Type4 JDBC ドライバの場合

コミット実行後も ResultSet オブジェクトを有効にする場合，次のどれかを指定してください。これらの指定をした場合，ホールダブルカーソルを使用します。

- DriverManager.getConnection メソッドの引数で，ユーザプロパティ HIRDB_CURSOR に TRUE を指定する。
- DriverManager.getConnection メソッドの引数で，URL に HIRDB_CURSOR=TRUE を指定する。
- PrdbDataSource，PrdbConnectionPoolDataSource，PrdbXADataSource クラスの setHiRDBCursorMode メソッドの引数で，true を指定する。
- Connection.setHoldability メソッドの引数で，ResultSet.HOLD_CURSORS_OVER_COMMIT を指定する。
- SELECT 文に UNTIL DISCONNECT を指定する。
- 次のメソッドの引数で，ResultSet の保持機能として ResultSet.HOLD_CURSORS_OVER_COMMIT を指定する。

  Connection.createStatement

  Connection.prepareStatement

  Connection.prepareCall

また，次のどれかを実行するまで，ResultSet オブジェクトを有効なまま（ホールダブルカーソルを開いたまま）にできます。

- ResultSet.close
- Statement.close
- Connection.close（エラー発生時に，システムによって自動的に実行される切断処理も含む）
- Connection.rollback，XAResource.rollback（エラー発生時に，システムによって自動的に実行されるロールバックも含む）
- すべての検索行を取得

(2) ホールダブルカーソル使用時の効果

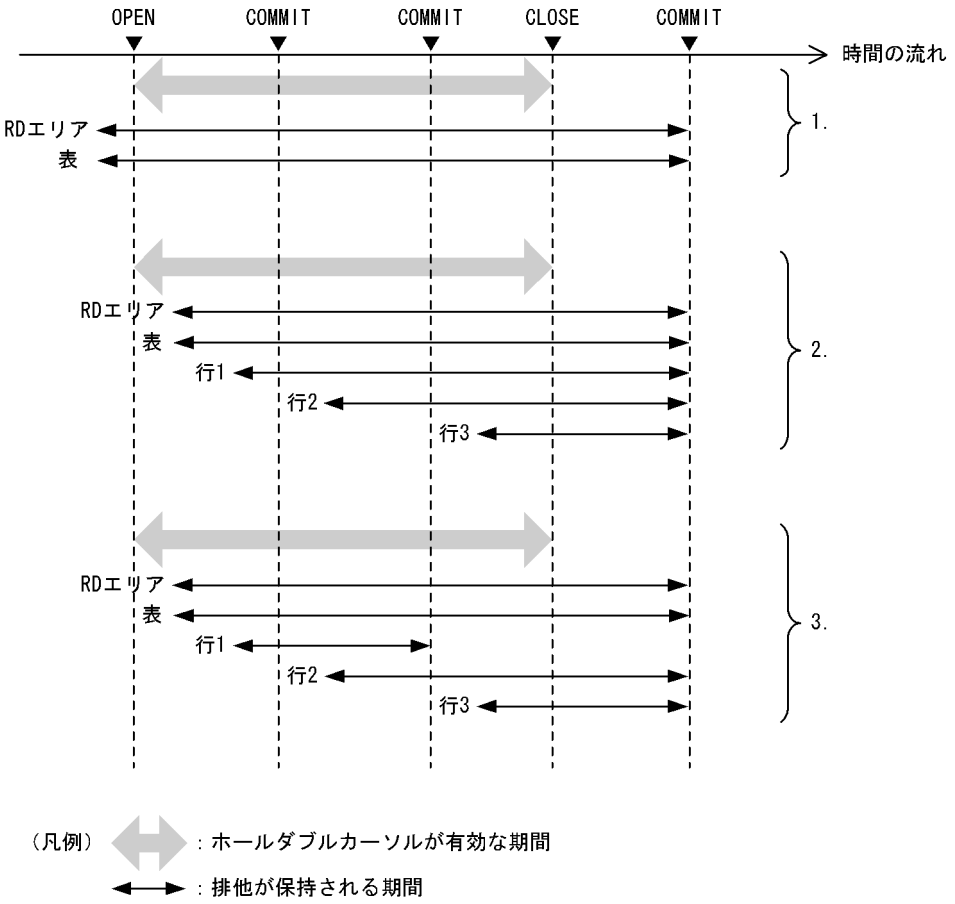
ホールダブルカーソルを使用すると，大量のデータを検索，又は更新する場合に，途中で COMMIT 文を実行できるため，排他資源の削減に有効となります。また，カーソルを開いたまま COMMIT 文を実行できるので，大量のデータを検索，又は更新する（トランザクションを長時間実行する）場合でも，シンクポイントを有効にし，再開始時の時間を短縮できます。

(3) ホールダブルカーソル使用時の処理

ホールダブルカーソルを使用した場合，作業表用ファイルの削除，及び作業表用ワークバッファの解放は，その作業表用ファイルを作成したホールダブルカーソルのクローズ後のコミットで行われます。

ホールダブルカーソルをオープンした場合，各バックエンドサーバプロセスは，トランザクションがなくても占有されます。したがって，ホールダブルカーソルを使用する場合は，最大サーバプロセス数の見積もり時に注意する必要があります。

UNITIL DISCONNECT 指定の LOCK 文の実行有無と，作業表を作成する検索，又はパラレルスキャンの有無によって，トランザクションを超えて引き継ぐ排他資源が異なります。引き継ぐ排他資源を次に示します。



[説明]

図中の番号の説明を次に示します。

番号	UNITIL DISCONNECT 指定の LOCK 文の実行	作業表を作成する検索，又はパラレルスキャン	引き継ぐ排他資源
1	あり	該当しません	LOCK 文の資源だけ引き継ぎます。
2	なし	あり	すべての資源を引き継ぎます。
3		なし	カーソル位置の資源だけ引き継ぎます。

OLTP 環境下の UAP でホールダブルカーソルを使用する場合，HiRDB のシステム定義の pd_oltp_holdcr オペランドに use を指定する必要があります。

また，OLTP 環境下の UAP でホールダブルカーソルを使用する場合は，次の条件を満たす必要があります。

- X/Open に準拠した API を使用して HiRDB にアクセスする UAP である。

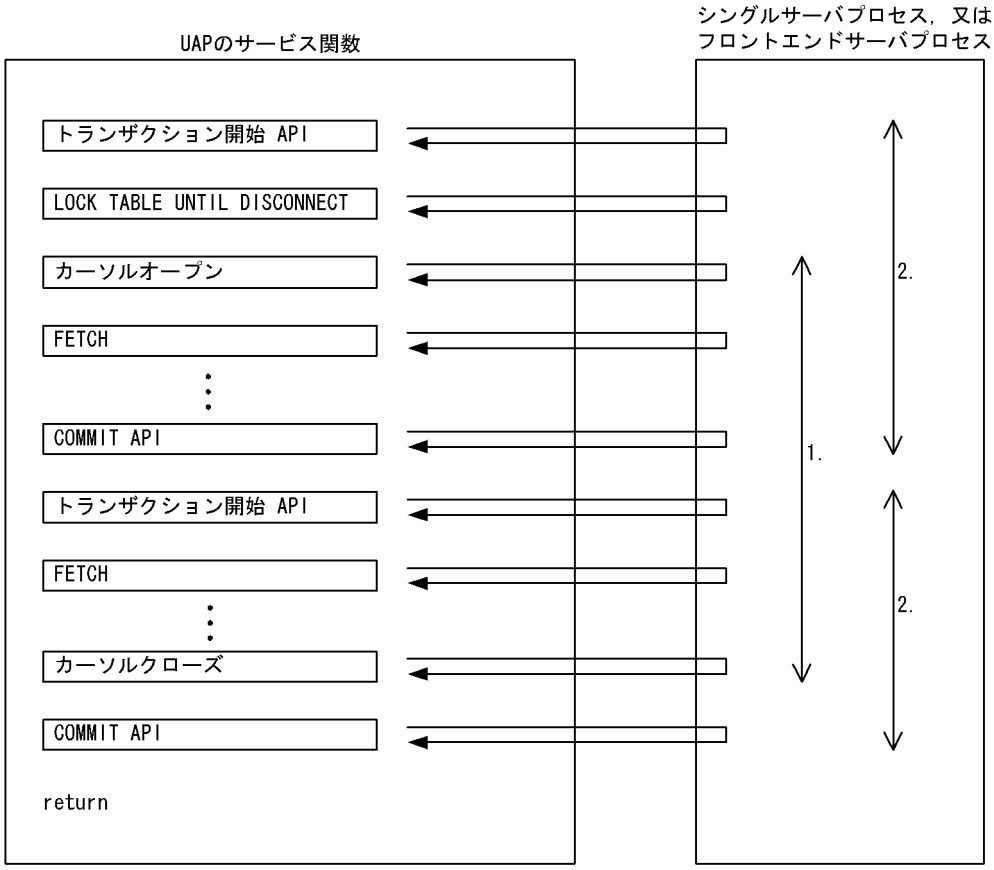
- ホールダブルカーソルを使用する UAP のサービス関数は，ホールダブルカーソルをオープンした後，該当するサービス関数が return する前に，カーソルの後処理をする。※

注※

カーソルの後処理をする方法を次に示します。

1. カーソルのクローズ
2. ROLLBACK 文実行
3. DISCONNECT 文実行
4. UAP の終了

UAP のサービス関数からの SQL の実行シーケンスを次に示します。図中のトランザクション開始 API，カーソル OPEN，カーソル CLOSE，及び COMMIT API の順序関係に注意してください。



[説明]

1. カーソルの保持範囲
2. トランザクションの範囲

(4)　ホールダブルカーソルの同時オープン数の上限

UNTIL DISCONNECT 指定の LOCK 文を実行しない表に対して，ホールダブルカーソルを使用する場合，そのホールダブルカーソルのトランザクション当たりの最大同時オープン数を，システム定義 pd_max_open_holdable_cursors に指定してください。

3.5.5　カーソルの使用例

カーソルを使用した例を次に示します。

(1)　カーソルを使用して行を検索しながら更新する例

在庫表（ZAIKO）から，カーソル（CR1）を使用してすべての行を検索しながら単価（TANKA）を 1 割引きにします。

```
          :
EXEC SQL BEGIN DECLARE SECTION;
  char xscode[5]; ..................................1
  char xsname[17];  ................................1
  char xcol[3]; ....................................1
  int xtanka; ......................................1
  int xzsuryo;  ....................................1
EXEC SQL END DECLARE SECTION;
          :
EXEC SQL DECLARE CR1 CURSOR FOR
    SELECT * FROM ZAIKO
      FOR UPDATE OF TANKA;  ........................2
EXEC SQL OPEN CR1;  ..................................3
EXEC SQL FETCH CR1
   INTO :xscode,:xsname,:xcol,:xtanka,:xzsuryo; .....4
EXEC SQL UPDATE ZAIKO
    SET TANKA=0.9*:xtanka
    WHERE CURRENT OF CR1; ............................5
EXEC SQL CLOSE CR1; ..................................6
          :
```

［説明］

1. 検索，更新，挿入で使用する埋込み変数を宣言します。
2. カーソル CR1 を宣言します。このとき，カーソル CR1 を使用して列 TANKA だけを更新するので，FOR UPDATE OF 列名を指定します。
3. カーソル CR1 をオープンします。
4. カーソル CR1 が指す行の列 TANKA の値を埋込み変数：xtanka に取り出します。
5. TANKA の値を 1 割引き（0.9＊:xtanka）します。
6. カーソル CR1 をクローズします。

(2)　カーソルを使用して行を検索しながら更新し，更に行を挿入する例

在庫表（ZAIKO）から，カーソル（CR1）を使用してすべての行を検索しながら更新し，更にカーソル（CR1）を使用しないで行を挿入します。

```
          :
EXEC SQL BEGIN DECLARE SECTION;
  char xscode[5];  .....................................1
  char xsname[17]; .....................................1
  char xcol[3];  .......................................1
  int xtanka;  .........................................1
  int xzsuryo; .........................................1
EXEC SQL END DECLARE SECTION;
          :
EXEC SQL DECLARE CR1 CURSOR FOR
    SELECT * FROM ZAIKO
      FOR UPDATE;  ....................................2
EXEC SQL OPEN CR1; ....................................3
EXEC SQL FETCH CR1
   INTO :xscode,:xsname,:xcol,:xtanka,:xzsuryo; .......4
EXEC SQL UPDATE ZAIKO
    SET SURYO=:xzsuryo+100
    WHERE CURRENT OF CR1;  ............................5
EXEC SQL INSERT INTO ZAIKO
    VALUES(:xscode,:xsname,:xcol,:xtanka,:xzsuryo); ...6
EXEC SQL CLOSE CR1;  ..................................7
          :
```

[説明]

1. 検索で使用する埋込み変数を宣言します。
2. カーソル CR1 を宣言します。このとき，カーソル CR1 を使用して更新し，更にカーソル CR1 を使用しないで行を挿入するため，FOR UPDATE 句を指定します。
3. カーソル CR1 をオープンします。
4. カーソル CR1 が指す行の値を埋込み変数に取り出します。
5. SURYO の値に 100 を加算します。
6. カーソル CR1 を使用しないで，ZAIKO 表に行を挿入します。
7. カーソル CR1 をクローズします。

(3)　ホールダブルカーソルを使用した例

在庫表（ZAIKO）から，カーソル（CR1）を使用してすべての行を検索しながら，単価（TANKA）を 50%にします。カーソル（CR1）はそのまま閉じないで，カーソル（CR1）を使用した別の操作をします。

```
          :
EXEC SQL BEGIN DECLARE SECTION:
  char xscode[5];  ..................................1
  char xsname[17]; ..................................1
  char xcol[3];  ....................................1
```

```
  int xtanka;  ....................................1
  int xzsuryo; ....................................1
END DECLARE SECTION;
          :
EXEC SQL LOCK TABLE ZAIKO
    IN EXCLUSIVE MODE UNTIL DISCONNECT;  ...........2
          :
EXEC SQL DECLARE CR1 CURSOR WITH HOLD FOR
    SELECT * FROM ZAIKO
      FOR UPDATE OF TANKA  ........................3
EXEC SQL OPEN CR1; ..............................4
EXEC SQL FETCH CR1
   INTO :xscode,:xsname,:xcol,:xtanka,:xzsuryo; ....5
EXEC SQL UPDATE ZAIKO SET TANKA=0.5*:xtanka
    WHERE CURRENT OF CR1;  ........................6
<1000行単位に次のCOMMIT文を実行する判定> ...........7
EXEC SQL COMMIT; ................................8
<更新する行がなくなった場合は次のCLOSE文を実行> ....9
EXEC SQL CLOSE CR1;  ............................10
          :
```

[説明]

1. 検索，及び更新で使用する埋込み変数（:xtanka）を宣言します。
2. ホールダブルカーソルを使用するため，UNTIL DISCONNECT 指定の LOCK 文で ZAIKO 表に排他を掛けます。また，カーソルを使用して更新をするので，排他モード（IN EXCLUSIVE MODE）を指定します。
3. カーソル CR1 を宣言します。このとき，宣言するカーソルはホールダブルカーソルなので WITH HOLD を指定します。また，更新する列は TANKA だけなので，FOR UPDATE OF 句に列 TANKA を指定します。
4. カーソル CR1 をオープンします。
5. カーソル CR1 が指す行の列 TANKA の値を埋込み変数：xtanka に取り出します。
6. TANKA の値を 50%（0.5＊:xtanka）にします。
7. 1000 行更新するごとに次の COMMIT 文を実行，又はそうでないときは更新処理を続行するような判定を記述します。
8. 更新処理をコミットします。
9. 更新する行がなくなった場合は次の CLOSE 文を実行，又は更新する行がまだある場合は更新処理を続行するような判定を記述します。
10. カーソル CR1 をクローズします。

3.6　SQL のエラーの判定と処置

UAP で SQL 文を実行する場合，SQL 文が正常に実行されたかを判定する必要があります。

ここでは，SQL 文が正常に実行されたかを判定する方法と，エラーを検知した場合の対処方法について説明します。

3.6.1　エラーの判定

(1)　リターンコードの参照

SQL 実行時に HiRDB システムでリターンコード（SQLCODE，及び SQLSTATE）が設定されます。ただし，DECLARE CURSOR のような宣言文の場合は，リターンコードが設定されません。リターンコードを参照する場合の変数名称は，次のとおりです。

- SQLCODE を参照する場合の変数名称：SQLCODE
- SQLSTATE を参照する場合の変数名称：SQLSTATE

SQLCODE 変数，及び SQLSTATE 変数の設定値を参照することで，SQL 文の実行状態が判定できます。

SQL 文の実行状態と変数に設定される値の関係を次の表に示します。

表 3–31　SQL 文の実行状態と変数に設定される値の関係

SQL 文の実行状態		SQLCODE 変数の値	SQLWARN0 の値	SQLWARN6 の値	SQLSTATE 変数の値
正常終了	警告なし	0	'△'	−	'00000'
	警告付き※4	0	'W'	−	'01nnn'※1 （nnn≠R00）
		>0（≠100,110）	−	−	'01R00'
	データなし※3	110	−	−	'R2000'
データなし		100	−	−	'02000'
エラー終了	暗黙的ロールバックなし	-1～-1999	−	'△'	'mmnnn'※2
	暗黙的ロールバックあり	-1～-1999	'W'	'W'	'40nnn'※1

（凡例）

mm：クラス

nnn：サブクラス

−：値は設定されません。

注※1

nnn には，SQLSTATE のサブクラスが設定されます。SQLSTATE については，マニュアル「HiRDB メッセージ」を参照してください。

注※2

mm には，SQLSTATE のクラスが設定されます。SQLSTATE については，マニュアル「HiRDB メッセージ」を参照してください。

注※3

リストを使用した検索で，リスト作成時にあった行が返らなかった場合の状態です。

注※4

警告情報の内容は，SQLWARN1～F に設定されるか，又は SQLCODE の値（100 以外の正数）で示されます。SQLWARN1～F に警告情報が設定される場合は，SQLWARN0 に'W'が設定されます。SQLWARN0 に'W'が設定されている場合，更に SQLWARN1～F の領域を確認します。

なお，SQLWARN0～F の内容については，「SQL 連絡領域」を参照してください。

警告情報の内容を SQLCODE の値（100 以外の正数）で示す場合は，SQLSTATE のサブクラス（nnn）が R00 になります。警告付き正常終了の場合の SQLSTATE，SQLCODE，及び SQLWARNO の値の関係を次の表に示します。

表 3–32　警告付き正常終了の場合の SQLSTATE，SQLCODE，及び SQLWARNO の値の関係

SQLSTATE の値	SQLCODE の値	SQLWARNO の場合
01nnn（nnn≠R00）	0	‘W’
01R00	100 以外の正数	空白，又は’W’

(a)　SQLCODE=100，又は SQLSTATE='02000'の場合

検索する行がなくなったことを判定します。

特に次に示す内容を判定するときに有効です。

- FETCH 文で取り出す行がなくなった
- 1 行 SELECT 文で行がなかった
- INSERT 文，DELETE 文，又は UPDATE 文で更新対象の行がなかった

(b)　SQLCODE<0，又は SQLSTATE='mmnnn'（mm が'00'，'01'，'02'でない，又はリストを使用した検索の場合は mm が'00'，'01'，'02'，'R2'でない）の場合

SQL エラーが発生したと判定します。

SQL エラーが発生した場合，暗黙的にロールバックされる場合とされない場合があります。
SQLWARN6='W'，又は SQLSTATE='40nnn'の場合，暗黙的にロールバックしたと判定します。

なお，エラーが発生した SQL を特定したい場合，SQL トレース情報を参照します。SQL トレース情報の内容については，「SQL トレース機能」を参照してください。

(c)　上記の(a)及び(b)以外の場合

正常終了したと判定します。

正常終了には，警告情報がある場合とない場合があります。SQLWARN0='W'，若しくは SQLCODE が 100 以上の正の値，又は SQLSTATE='01nnn'の場合，警告付き正常終了と判定します。

リストを使用した検索の場合，データなしの正常終了（リスト作成時にあった行が削除された）の可能性があります。SQLCODE が 110，又は SQLSTATE が'R2000'ならば，データなしの正常終了と判断して，検索行に対する処理をスキップする必要があります。

警告付き正常終了の内容については，表「警告付き正常終了の場合の SQLSTATE，SQLCODE，及び SQLWARNO の値の関係」を参照してください。

(2)　エラー検出時の対処

エラーを検知した場合，次に示す 1〜4 の順番で対処します。

1. リターンコードを出力，又は表示します。
2. リターンコードだけではエラーの内容が判別しにくい場合，各コードの付加情報を表示，又は出力します。また，必要に応じて，エラーになった SQL 文，又はエラーになった SQL 文を識別するための情報を表示します。

   リターンコードの付加情報と参照先を次の表に示します。

   表 3–33　リターンコードの付加情報と参照先

付加情報	参照先
SQLCODE に対応するメッセージ※	SQL 連絡領域中の SQLERRML フィールド，及び SQLERRMC フィールドの内容

   注※

   エラーが発生した後に再度 FETCH 文を実行した場合，HiRDB は前回発生したエラーのリターンコードを返しますが，メッセージの可変部の内容は保証しません。

3. トランザクションを取り消します（ROLLBACK，又は UAP を異常終了させます）。

   デッドロックによって暗黙的にロールバックされた UAP は次のようになります。

   通常の UAP の場合：

   暗黙的にロールバックされると，次に実行した SQL が新たなトランザクション開始となります（ROLLBACK，又は DISCONNECT もできます）。

   OLTP 下の UAP の場合：

   暗黙的にロールバックされると，OLTP 下の UAP からは DISCONNECT，又は ROLLBACK 以外は受け付けられません。

また，OLTP 環境で X/Open に従ったアプリケーションプログラムをクライアントとした場合に，実行したアプリケーションプログラムがデッドロックになったときもトランザクションの終了が必要です。

4. UAP の終了，又はトランザクションの開始（別のトランザクションの新規実行，又は同じトランザクションの再実行）をします。

   なお，同じトランザクションを再実行する場合，実行前にエラーの対策をしてください。エラーの原因が取り除かれない状態でトランザクションを再実行すると，無限ループになることもあります。また，再実行しても同一のエラーが発生するような場合，UAP の終了を考慮する必要があります。

3.6.2 エラーの自動判定

WHENEVER 文を使用すると，エラーが発生したかどうか自動的に判定できます。

WHENEVER 文では，次に示す内容について判定ができます。

- エラーが発生した
- 検索する行がなくなった
- 正常終了時の警告情報の有無

なお，WHENEVER 文の詳細については，マニュアル「HiRDB SQL リファレンス」を参照してください。

(1) エラーが発生した場合の判定（SQLCODE<0）

SQLERROR を指定した WHENEVER 文を使用して判定します。

エラーが発生したときに取らなければならない処置を指定することで，エラーが発生したときに指定した処置へ処理を移行します。また，エラーを参照する処理を指定すると，リターンコードと関連する情報を参照できます。

(2) 検索する行がなくなった（SQLCODE=100）

NOT FOUND を指定した WHENEVER 文を使用して判定します。

検索する行がなくなったときに取らなければならない処置を指定することで，検索する行がなくなったときに指定した処置へ処理を移行します。

(3) 正常終了時の警告情報の有無（SQLWARN0='W'，又は SQLCODE>0 かつ SQLCODE≠100）

SQLWARNING を指定した WHENEVER 文を使用して判定します。

正常終了で警告情報があったときに取らなければならない処置を指定することで，警告情報があるときだけ指定した処置へ処理を移行します。

リストを使用した検索の場合，データなしの正常終了（リスト作成時にあった行が削除された）の可能性があります。SQLCODE が 110，又は SQLSTATE が'R2000'ならば，データなしの正常終了と判断して，検索行に対する処理をスキップする必要があります。

4 性能向上，操作性向上に関する UAP の設計

この章では，性能向上及び操作性向上の UAP 設計時に考慮する点について説明します。

4.1　インデクス

インデクスを利用すると検索時の入出力回数を削減できます。

インデクスの設計については，マニュアル「HiRDB システム導入・設計ガイド」の「インデクスの設計」を参照してください。

ここでは，インデクス検索時の留意事項，及び HiRDB SQL Tuning Advisor を使ったインデクスの提案，未使用インデクスの調査方法について説明します。

4.1.1　インデクスの提案

HiRDB SQL Tuning Advisor のインデクス提案機能では，SQL に適切なインデクス定義を提案します。次のどちらかの方法で SQL を解析すると，その結果を基にインデクスを検証します。

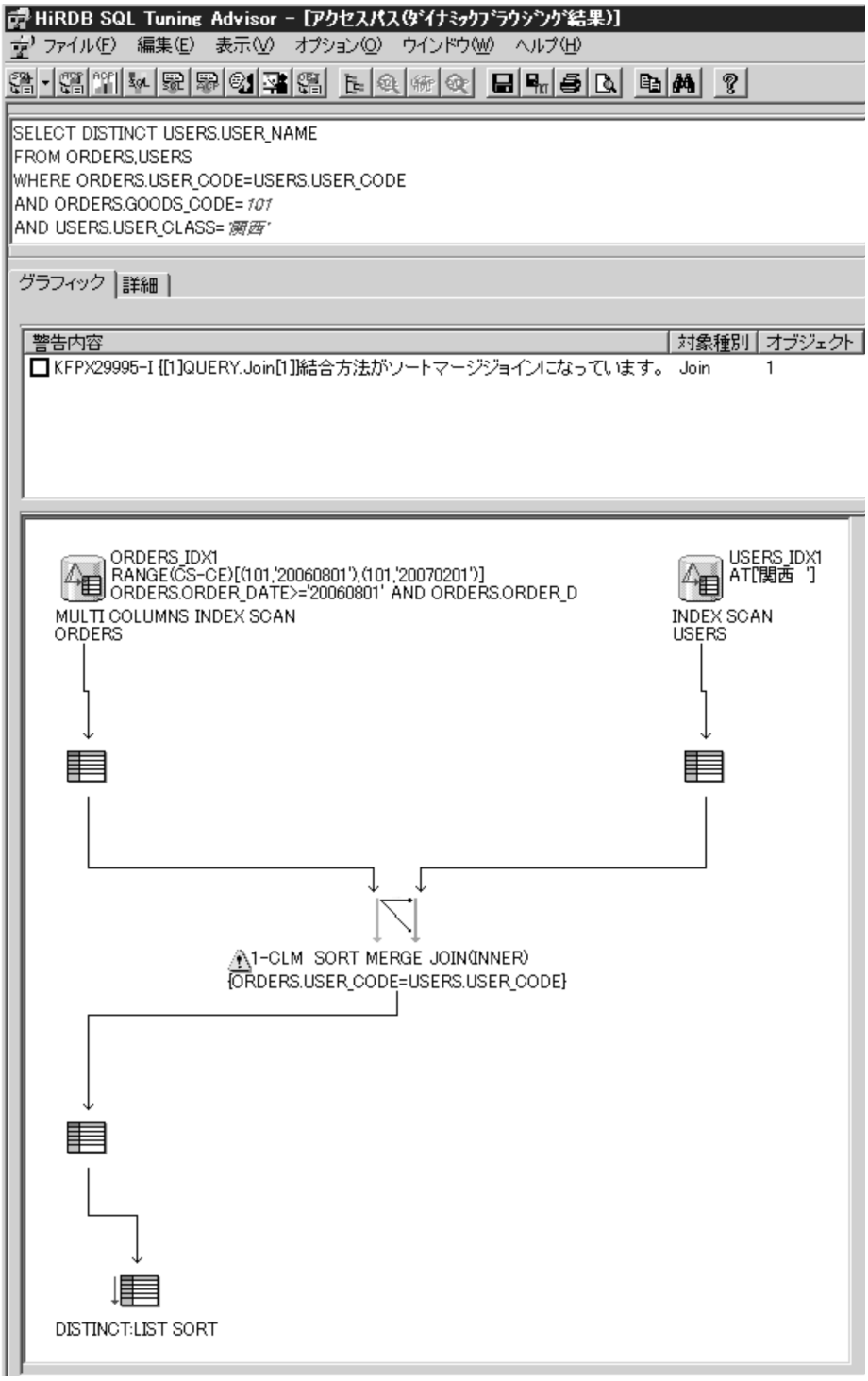
- ダイナミックブラウジング機能によるアクセスパス解析
  SQL を直接入力できるため，UAP 作成前のインデクス検証で使用します。
- アクセスパス情報ファイルからのアクセスパス解析
  中間結果情報も含めて検証できるため，環境構築後の性能検証で使用します。

ここでは，HiRDB SQL Tuning Advisor のダイナミックブラウジング機能を使って SQL を解析し，インデクスを提案させる手順を次に示します。

1. ［スタート］－［プログラム］－［HiRDB SQL Tuning Advisor］－［HiRDB SQL Tuning Advisor］を選択し，HiRDB SQL Tuning Advisor を起動します。
2. 接続の設定を行います。
   接続の設定方法については，「HiRDB SQL Tuning Advisor の環境設定」を参照してください。設定済みの場合，この手順は必要ありません。
3. ［ダイナミックブラウジング］ボタンをクリックします。
   ［ダイナミックブラウジング］画面が表示されます。



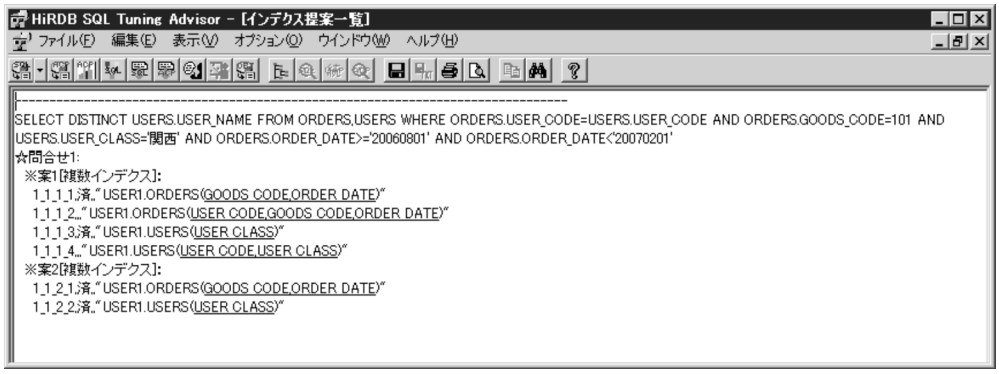
4. [SQL] テキストボックスに，検証したい SQL を入力し，[OK] ボタンをクリックします。
   [アクセスパス（ダイナミックブラウジング結果)] 画面が表示されます。
   その SQL に性能上の問題がある場合は，[警告内容] にガイダンスが表示されます。



5. ガイダンスの要因としてインデクスに問題がないかを確認する場合は，［インデクス提案］ボタンをクリックします。

   ［インデクス提案一覧］画面が表示されます。

   既に定義したインデクスには「済」と記載されていますので，「済」の記載がないインデクスについて定義してください。



4.1.2　未使用インデクスの調査

HiRDB SQL Tuning Advisor のアクセスパス集計機能では，未使用インデクスを調査できます。未使用インデクスを調査する手順を次に示します。

1. HiRDB SQL Tuning Advisor 用アクセスパス情報ファイルを取得します。

   取得方法については，「HiRDB SQL Tuning Advisor 用アクセスパス情報ファイル」を参照してください。検索処理の変更や，表及びインデクス定義を変更した場合は，変更前に取得したアクセスパス情報が，同一ファイルに含まれないようにしてください。

2. ［スタート］－［プログラム］－［HiRDB SQL Tuning Advisor］－［HiRDB SQL Tuning Advisor］を選択し，HiRDB SQL Tuning Advisor を起動します。

3. 接続の設定を行います。

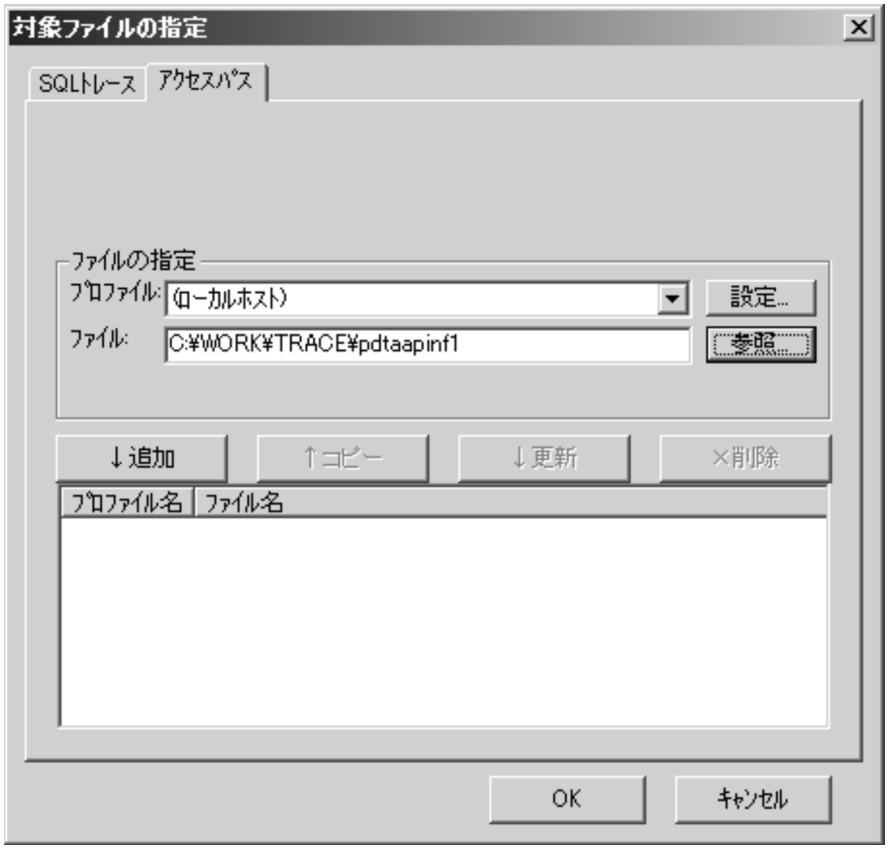
   接続の設定方法については，「HiRDB SQL Tuning Advisor の環境設定」を参照してください。設定済みの場合，この手順は必要ありません。

4. ［オプション］メニューから［対象ファイル指定］を選択します。

   ［対象ファイルの指定］画面が表示されます。

5. ［アクセスパス］タブで，アクセスパスファイル名を指定し，［追加］ボタンをクリックします。

   検索処理の変更や，表及びインデクス定義を変更した場合は，変更前に取得したアクセスパス情報ファイルが含まれないようにしてください。追加が完了したら，「OK」ボタンをクリックします。設定済みの場合，この手順は必要ありません。

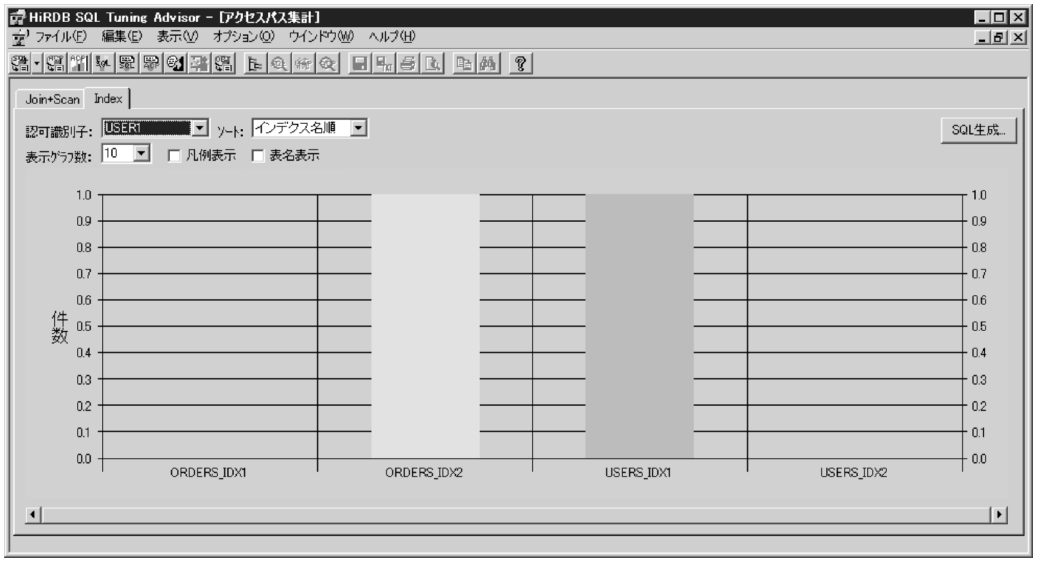


6. [アクセスパス集計] ボタンをクリックします。
   [アクセスパス集計] 画面が表示されます。「未使用のインデクスも表示する」をチェックします。



7. [OK] ボタンをクリックします。

［アクセスパス集計］画面が表示されます。［Index］タブを表示して，認可識別子で該当する内容を選択すると，インデクスの使用回数が表示されます。使用回数が 0 件のインデクスが未使用インデクスであることが分かります。



4.1.3　インデクス検索時の留意事項

この項では，インデクスの変更（データの変更に伴うインデクスのメンテナンス処理）の内部動作，及びインデクス検索時の UAP の設計指針について説明します。

インデクスは，探索条件に対する行の絞り込みを効率的に行い，データに対するアクセスや検索結果の返却を高速に行うためのアクセス手段です。

HiRDB は，複数のトランザクションによるインデクス検索と，インデクスの変更を同時に行うことによって，高レスポンス，高スループットのシステムを実現しています。しかし，その反面，インデクス変更の最中にインデクスを使用した検索を行うと，検索結果が変わることがあります。検索結果を変えないために，検索時に検索対象の表に対して排他を掛ける（LOCK TABLE 文を実行する）対処方法がありますが，システムの性能要件からこの対処方法が適用できないケースもあります。

このような場合に，複数トランザクションを同時実行し，順序関係に厳密な業務アプリケーションを実行するときは，この項で説明する UAP の設計指針を参考に業務アプリケーションを開発してください。

また，複数列インデクスを構成する列（例えば，ステータスの列など）に対する更新時のインデクス検索についても注意が必要です。

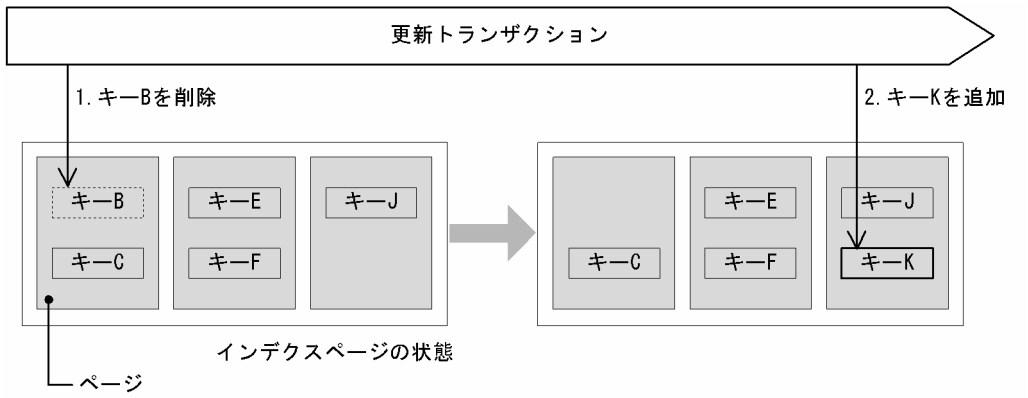
(1)　インデクス変更の内部動作

インデクス構成列の列値変更（UPDATE 文）に伴うインデクスの変更は，更新前の列値に対応するインデクスエントリの変更処理，及び更新後の列値に対応するインデクスエントリの変更処理の，二つの処理によって実現します。

ここでの更新前の列値に対応するインデクスエントリの変更とは，対応するインデクスキーがある行データが一つの場合，インデクスキーの削除を示します。更新後の列値に対応するインデクスエントリの変更とは，対応するインデクスキーがある行データが一つの場合，インデクスキーの追加を示します。また，これらの処理をインデクスキーの削除，インデクスキーの追加の順に実行します。これは，インデクスの変更時にインデクス容量の増加を抑えるためです。

インデクス変更の内部動作の例を次の図に示します。

図 4–1　インデクス変更の内部動作の例



[説明]

インデクスを定義している列を B から K に更新するトランザクションが実行します。

まず，1 で更新トランザクションが更新前の列値に対応するインデクスキー B を削除します。

その後，2 で更新後の列値に対応するインデクスキー K を追加します。

(2)　インデクス検索時の検索結果

インデクスの変更の最中にインデクス検索を行うと，検索結果が変わるケースがあります。具体的には，次の二つのケースがあります。

- 更新中の行が検索対象外となる。
- 更新対象の行が複数回検索結果に現れる。

(a)　更新中の行が検索対象外となる

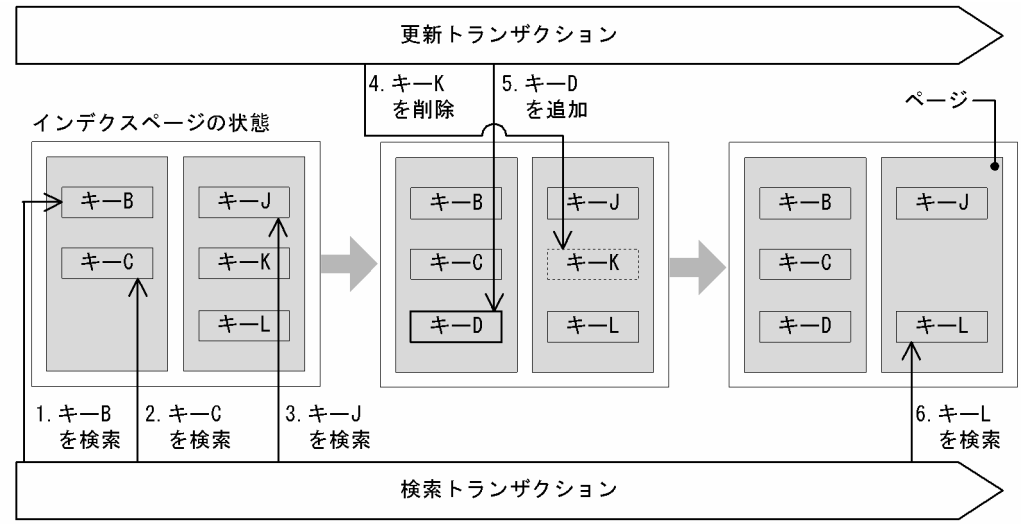
更新中の行が検索対象外となるケースについて説明します。

■ インデクス検索が終了していない範囲から，終了した範囲へのインデクス更新

UPDATE 文によって，インデクス検索が終了していない範囲から，終了した範囲へインデクスキーを更新すると，そのインデクスキーは検索対象外となります。

更新中の行が検索対象外となる例を次の図に示します。

図 4–2　更新中の行が検索対象外となる例（その 1）



[説明]

インデクスを定義している列を K から D に更新するトランザクションと，インデクスを検索するトランザクションが同時に実行します。

インデクス検索は，1〜3 でキー J に対する検索まで終了しています。

キー K を検索する時点で，更新トランザクションが 4 でキー K を削除，5 でキー D を追加するため，対応する行は検索対象外となります。

[補足]

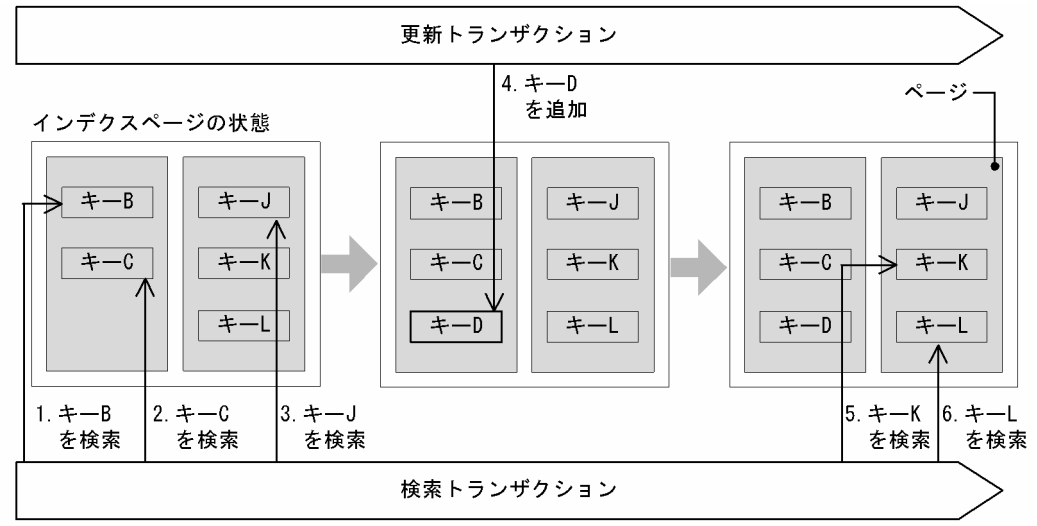
インデクス検索の探索条件がインデクスキー全体（複数列インデクスの場合，すべての構成列に対する探索条件）の場合，検索対象行が UPDATE 文によって探索条件から外れるため問題はありません。ただし，複数列インデクスで変更列以外の構成列に対する列を探索条件とした場合，業務によっては問題となるおそれがあります。この場合の対処方法については，「UAP の設計指針」を参照してください。

■ インデクス検索が終了した範囲へのインデクスキー追加

インデクス検索より先に INSERT 文を実行しても，インデクス検索が終了した範囲へインデクスキーを追加すると，そのインデクスキーは検索対象外となります。

更新中の行が検索対象外となる例を次の図に示します。

図 4–3　更新中の行が検索対象外となる例（その 2）



[説明]

インデクスを定義している列が D の行を追加するトランザクションと，インデクスを検索するトランザクションが同時に実行します。

HiRDB 内部の変更タイミングとして，更新トランザクションが 4 でキー D を追加する時点で，既にインデクス検索は 3 でキー J に対する検索まで終了しています。結果として，追加されたキー D に対応する行は検索対象外となります。

[補足]

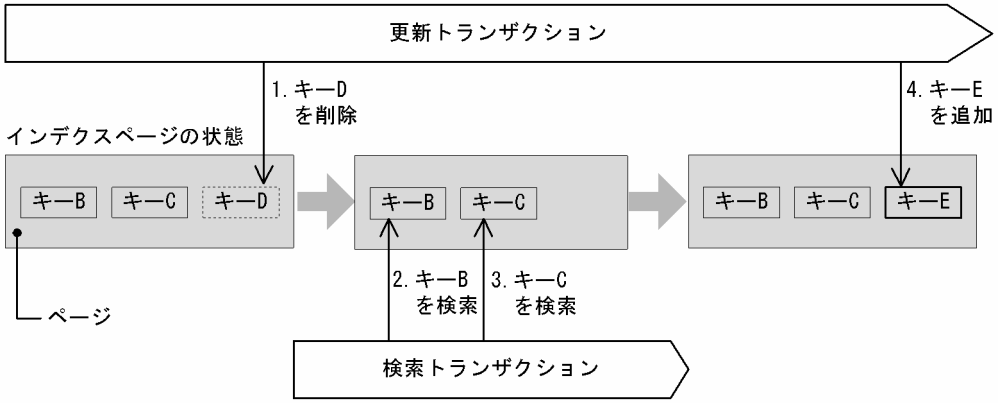
更新中の行が検索対象外となるために，データ操作に対して順序関係の厳密さを必要とする業務によっては問題となるおそれがあります。この場合の対処方法については，「UAP の設計指針」を参照してください。

■ UPDATE 文に伴うインデクスキーの変更

インデクスの変更の際，インデクスキーを削除した時点でインデクス検索が行われた場合，対応する行は検索対象外となります。

更新中の行が検索対象外となる例を次の図に示します。

図 4–4　更新中の行が検索対象外となる例（その 3）



[説明]

インデクスを定義している列の値を D から E に更新するトランザクションと，インデクス検索を行うトランザクションが同時に実行します。

更新トランザクションが 1 でキー D を削除した時点で，検索トランザクションは 2 でキー B，3 でキー C の検索を終了しています。このため，4 でのキー E の追加に対応する行は，検索対象外となります。

[補足]

- 更新前のインデクスキーと，更新後のインデクスキーが同じ場合，インデクスの変更は行いません。ただし，次のどれかの条件を満たす場合は，インデクスの変更（インデクスキーの削除及び追加）を行います。

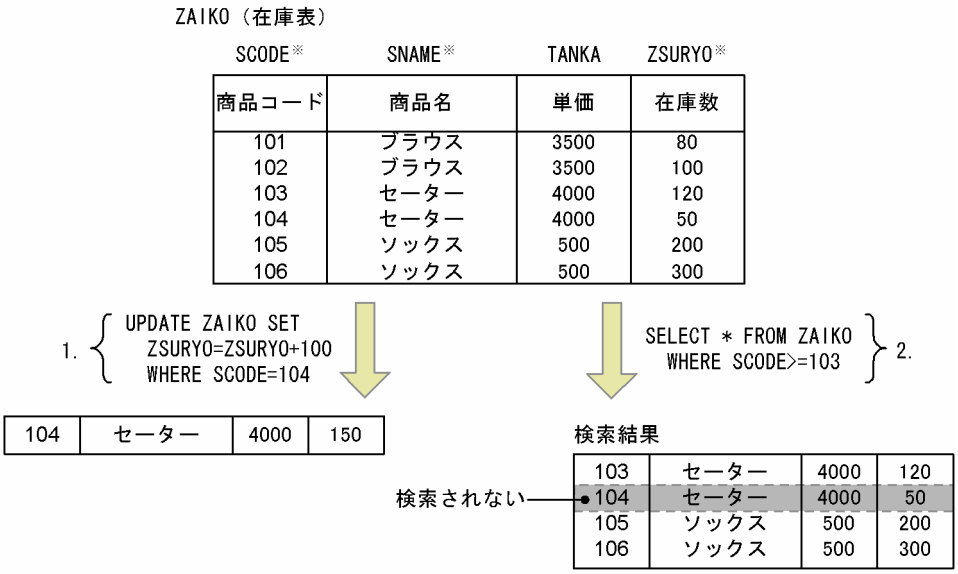
  ・定義長 256 バイト以上の可変長文字列型の列を構成列に持つインデクス

  ・繰返し列を構成列に持つインデクス

  ・部分構造インデクス

- インデクス検索の探索条件がインデクスキー全体の場合，検索対象行が UPDATE 文によって探索条件から外れるため問題はありません。ただし，複数列インデクスで変更列以外の構成列に対する列を探索条件とした場合，業務によっては問題となるおそれがあります。UPDATE 文に伴うインデクスキーの変更によって検索されなくなる例を次の図に示します。

  図 4–5　UPDATE 文に伴うインデクスキーの変更によって検索されなくなる例



注※　複数列インデクスを構成する列です。

[説明]

商品コード 104 の商品が 100 個入荷されたので，在庫数に 100 を加算します。

SCODE，SNAME，及び ZSURYO で構成される複数列インデクスを利用して検索します。この場合，1 で更新中の行は検索対象にならないため，検索結果に含まれません。

この場合の対処方法については「UAP の設計指針」を参照してください。

(b)　更新対象の行が複数回検索結果に現れる

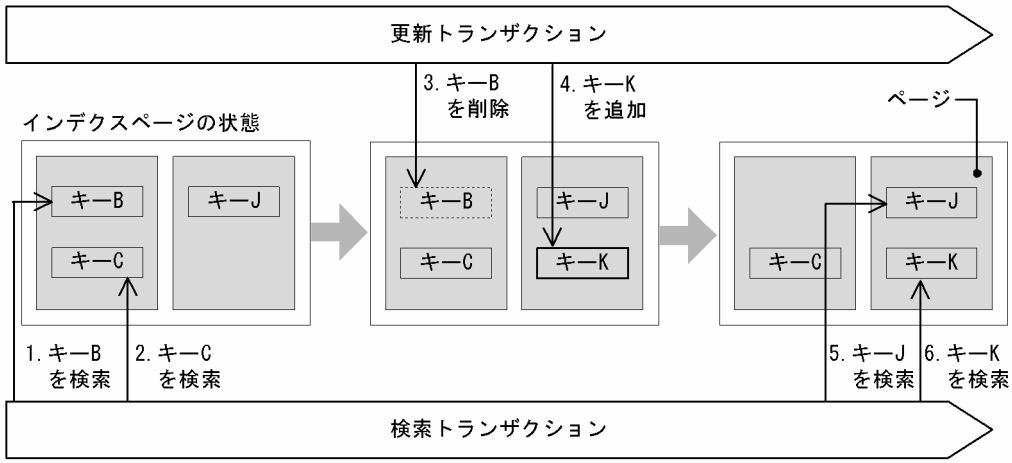
更新対象の行が複数回検索結果に現れるケースについて説明します。

■ インデクス検索が終了した範囲から，終了していない範囲へのインデクス更新

排他オプションに WITHOUT LOCK WAIT を指定した場合，トランザクション完了までの検索結果は保証されないため，タイミングによって更新対象の行が検索結果として複数回現れることがあります。排他オプションに WITHOUT LOCK NOWAIT を指定した場合も同様です。具体的には，UPDATE 文によってインデクス検索が終了した範囲から，終了していない範囲へインデクスキーを更新すると，そのインデクスキーは再度検索されます。

更新対象の行が複数回検索結果に現れる例を次の図に示します。

図 4–6　更新対象の行が複数回検索結果に現れる例



[説明]

インデクスを定義している列を B から K に更新するトランザクションと，インデクスを検索するトランザクションが同時に実行します。

1 でキー B を検索する時点で，排他オプションの指定によって排他が解除されているため，更新トランザクションは対応行を更新できます。

検索トランザクションが 5 でキー J を検索する前に，更新トランザクションが対応行を更新（3 でのキー B の削除，及び 4 でのキー K の追加）したため，6 でのキー K の検索によって，対応する更新行が検索結果に 2 回現れます。

[補足]

統計情報など，厳密さを必要としないデータの検索結果として扱う場合は問題ありませんが，検索結果がその後の業務処理に影響を及ぼすような厳密さを必要とする場合は，問題となるおそれがあります。この場合の対処方法については，「UAP の設計指針」を参照してください。

(3)　インデクス検索の動作

INSERT 文，UPDATE 文，及び DELETE 文実行中にインデクスを検索した場合，検索結果が変わるケースが発生する可能性を，次の表に示します。

表 4–1　検索結果が変わるケースが発生する可能性

後続処理（インデクスを使用した検索）	先行処理		
	INSERT 文	UPDATE 文	DELETE 文
SELECT 文	×	×※	○
UPDATE 文	×	×	○
DELETE 文	×	×	○

（凡例）

○：検索結果は変わりません。

×：検索結果が変わる（検索対象外となる）可能性があります。

注※

排他オプション WITHOUT LOCK WAIT 又は WITHOUT LOCK NOWAIT の場合，更新対象の行が複数回検索結果に現れる可能性があります。

INSERT 文ではインデクスを使用した検索は行わないため，後続処理として INSERT 文で発生する可能性はありません。

なお，更新対象の行が複数回検索結果に現れるかどうかは，更新前後のインデクスキー値の変化に依存します。更新トランザクションがインデクスを更新したときの，インデクスキー値の変化の方向と，検索トランザクションのインデクス検索方向が一致する場合，次に示す表のように，複数回検索結果に現れる可能性があります。

インデクス検索方向	UPDATE 文によるインデクスキー値の変化		
	インデクスキー値が大きくなる	インデクスキー値が小さくなる	インデクスキー値が同じ（同値更新）
昇順 （キー値の大きい方向）	×	○	○
降順 （キー値の小さい方向）	○	×	○

（凡例）

○：検索結果は変わりません。

×：検索結果が変わる（複数回検索結果に現れる）可能性があります。

(4)　UAP の設計指針

複数トランザクションを同時に実行する場合で，順序関係に厳密さを必要とする業務アプリケーションを開発するときは，UAP が表に対する排他を取得することで，インデクスの変更とインデクス検索をシリアライズする必要があります。ただし，性能面で影響がある場合には，次の対処を検討してください。

1. インデクス構成列に更新項目を含めないようにしてください。
2. 検索頻度，条件などでインデクス構成列に更新項目を含めなければならない場合は，更新項目以外のインデクス構成列を探索条件としたインデクス検索で，業務上の問題が発生しないかどうか確認してください。問題が発生する場合には，その探索条件でこのインデクスを検索に使用しないようにしてください。
3. 2.が採用できない場合は，UAP で検索結果に対する再チェック処理を組み込んでください。

4.2　表に対する操作

4.2.1　FIX 属性の表

FIX 属性の表は，行が固定長なので，1 行を一つの列とみなした操作など，表の列数が多いほど有効です。

行単位で操作をする場合，列単位で操作する場合と比較して次に示す特長があります。

- 処理時間が短い。
- 処理する列数が増えても処理時間に影響しない。
- 1 行分のデータを 1 個のデータとして受け渡しができるため，UAP の作成や保守がしやすい。

FIX 属性の表に対する次の操作では，行単位で操作をした方が処理効率が向上します。

- すべての列，又はほとんどの列を検索する場合
- すべての列，又はほとんどの列を更新する場合
- データを挿入する場合

なお，一つの行全体が操作の対象なので，表に列を追加したときは，データを受け渡す埋込み変数も宣言し直してください。

FIX 属性の表は，可変長の列やナル値の列がないことが条件になります。しかし，列数が多いなど，1 行を一つの列とみなした操作をした方が効率が良い場合，次に示す方法で可変長の列やナル値の列をなくすことを検討してください。

- 定義長の小さい可変長データ，及び実長の取り得る範囲が狭いデータを固定長にする。
- ナル値をほかの値（例えば，数データの場合は 0，文字データの場合は空白）で代用する。

4.2.2　採番業務で使用する表

採番業務では次の 2 種類の方法で採番できます。

- WITHOUT ROLLBACK オプション指定の表での採番
- 自動採番機能

ここでは，WITHOUT ROLLBACK オプション指定の表での採番について説明します。自動採番機能については，「自動採番機能」を参照してください。

(1) どのようなときに使用するか

実際の業務では，伝票番号や書類の番号の管理など，様々な採番業務があります。採番業務では，あるユーザが伝票番号を取得しようとしたとき，それと同時に他ユーザも伝票番号を取得しようとするケースが考えられます。

また，あるユーザが伝票番号を取得しようとしたときには，ほかのユーザが以前に取得した伝票番号と重複しないように，番号をカウントアップしておく必要があります。

このようなケースでは，あるユーザが伝票番号の取得処理中に，他ユーザが待ち状態になる可能性があります。このようなことを考慮し，HiRDB では，排他待ちの影響を少なくして，採番業務をするための機能を提供しています。

(2) 表の設計

採番業務を効率良く実施するため，排他待ちの影響が少なくなるように表を設計する必要があります。採番を管理する表への排他の影響を少なくするために，トランザクションのコミットを待たないで表への更新（追加，削除を含む）処理が完了した時点で，その行への排他を解除し，それ以降はロールバックされなくなるという機能を提供しています。この機能を実現するためには，表の設計者が，表定義時に CREATE TABLE の WITHOUT ROLLBACK オプションを指定する必要があります。

(3) 業務への適用条件

WITHOUT ROLLBACK オプションを指定して行う採番は，欠番を許容できない業務には適していません。番号が連続でなくてもよい場合に適用するようにしてください。欠番は次に示す場合に発生します。

表定義時に WITHOUT ROLLBACK オプションを指定した表の行を更新したトランザクションが ROLLBACK 文や SQL 実行中のエラーによりロールバックした場合，取得した番号を使用した業務の表に対してはロールバックされます。この場合，整合性が保たれますが，取得した番号はロールバックされません。このため，ロールバックしたトランザクションで取得した値は HiRDB システム内で使われずに次の番号が取得され，欠番となります。

また，表定義時に WITHOUT ROLLBACK オプションを指定した表の行を更新したトランザクションが完了する前に HiRDB システムが異常終了した場合，HiRDB システムの再開始後に採番した値が戻るときがあります。

採番した値を HiRDB システム内に閉じて使用する場合，採番した値が戻っても，その値を使用したトランザクションもロールバックするため，HiRDB システム内で一意性が保てます。HiRDB システム外で使用する場合は採番した値が戻ると一意性が保てなくなります。HiRDB システム外で使用する場合は，更新する SQL 文に WRITE IMMEDIATE オペランドを指定してください。WRITE IMMEDIATE オペランドを指定すると，HiRDB システムが異常終了しても値が戻りません。ただし，WRITE IMMEDIATE オペランドを指定すると 1 行の更新ごとにシステムログの書き出しが行われ，その書き出し時間が SQL の実行時間に加算されます。

(4)　採番を管理する表の例

採番を管理する表の例を次の図に示します。

図 4–7　採番を管理する表の例

採番管理表

種類	採番
伝票番号	23
書類番号	17

業務表（伝票番号を使用する表）

伝票番号	品名
1	ＡＡＡ
2	ＢＢＢ
⋮	⋮
23	WWW
⋮	⋮

注

表の定義例（WITHOUT ROLLBACK オプションの指定）については，マニュアル「HiRDB システム導入・設計ガイド」を参照してください。

(5)　採番業務のアプリケーションプログラムの例

採番業務のアプリケーションプログラムの例を次に示します。なお，採番管理表と業務表を操作するアプリケーションプログラムは同一トランザクションを想定します。

（例）

伝票番号と書類番号を管理する採番管理表があるものとします。採番管理表から，最新の伝票番号を取得し，それを業務で使用する SQL の例を次に示します。

```
INSERT INTO 採番管理表 VALUE('伝票番号',1)  ....1

              :

DECLARE CUR1 CURSOR FOR  .....................2
  SELECT 採番 FROM 採番管理表
    WHERE 種類='伝票番号' FOR UPDATE OF 採番
OPEN CUR1  ...................................3
FETCH CUR1 INTO :x_採番  .....................4
UPDATE 採番管理表 SET 採番=:x_採番+1  ........5
    WHERE CURRENT OF CUR1
CLOSE CUR1  ..................................6

              :

取得した番号を利用した業務表へのアクセス処理  ..7

              :
```

[説明]

1. 採番管理表に，伝票番号の初期値として 1 を挿入します。
2. 採番管理表から最新の伝票番号を検索する，カーソル CUR1 を宣言します。
3. カーソル CUR1 をオープンします。
4. 伝票番号を x_採番に取り出します。
5. 次に検索するユーザのために，採番をカウントアップします（最新の採番にするため）。この処理が終了した時点で，コミットを待たないで行への排他を解除します。
6. カーソル CUR1 をクローズします。
7. x_採番に取り出した伝票番号を基に，ユーザ任意の業務を実施します。

なお，採番ごとに 3～7 を繰り返します。

(6) 複数種類の採番を管理するときの考慮点

(a) 排他について

WITHOUT ROLLBACK オプションを指定した表に複数の行を格納する場合，その表にインデクスを定義していないときは，すべての行が検索対象となるため，すべての行に対して一時的に排他が掛かります。このような場合，例えば，伝票番号の採番処理と書類番号の採番処理との間で，排他待ちになることがあります。これを回避するためには，クライアント環境定義の PDLOCKSKIP に YES を指定して，無排他条件判定をする必要があります。無排他条件判定をした場合には，検索処理時には排他を掛けないで，探索条件を満たした行に対してだけ排他を掛けます。

(b) ロールバックについて

複数種類の採番を扱う場合，1 回の SQL で複数行を更新するような処理はしないでください。排他の解除，及びロールバックがされなくなるタイミングが各行単位に，それぞれの行の更新処理が完了した時点となります。そのため，複数行を更新する UAP が異常終了すると，一部の行の更新がロールバックされない場合があります。

(7) ストアドプロシジャを使用した採番の例

採番業務では，ある決まったパターンで処理することが多いため，その処理をストアドプロシジャとして登録しておくと便利です。

表の定義例，ストアドプロシジャの例を例 1～例 3 に示します。

（例 1）

WITHOUT ROLLBACK 指定の表と，ストアドプロシジャを使用して，順序番号を割り当てます。

初期値 1，増分値 1 で INTEGER の最大値までの値を採番します。

INTEGER の最大値を超えた場合は，オーバフローのエラーが返されます。ただし，既定値設定機能（PDDFLNVAL）使用時はオーバフローのエラーにはならないで，ナル値となるため，非ナル値制約違反のエラーとなります。また，初期値を持つ行が挿入されていない場合，表中に行がない状態になるた

め，UPDATE 文実行時にカーソルが行に位置づけられていないというエラーになります。複数行が挿入されている場合には，2 行目以降は無視されます。

```
CREATE FIX TABLE
    owner_id.sequence_tbl(sequence_no INTEGER NOT NULL)
    WITHOUT ROLLBACK;  .....................................1
CREATE PROCEDURE owner_id.nextval(OUT next_no INTEGER)
  BEGIN
    DECLARE update_no INTEGER;  ............................2
    DECLARE cr1 CURSOR FOR
      SELECT sequence_no FROM owner_id.sequence_tbl
        FOR UPDATE;
    OPEN cr1;
    FETCH cr1 INTO update_no;  .............................3
    SET next_no=update_no;  ................................4
    UPDATE owner_id.sequence_tbl SET sequence_no=update_no+1
      WHERE CURRENT OF cr1;  ...............................5
    CLOSE cr1;  ............................................3
  END  .....................................................2
COMMIT WORK;  ..............................................6
INSERT INTO owner_id.sequence_tbl(sequence_no) VALUES(1);  ...7
COMMIT WORK;  ..............................................8

〈順序番号の割り当て〉  ....................................9
CALL owner_id.nextval(OUT:xnext_no);
                :
割り当てた順序番号xnext_noを使用した処理
                :
CALL owner_id.nextval(OUT:xnext_no);
                :
```

[説明]

1. INTEGER の値を採番するための表 owner_id.sequence_tbl を定義します。
2. 順序番号を割り当てて，パラメタ next_no で出力する手続き owner_id.nextval を定義します。
3. 表 owner_id.sequence_tbl の列 sequence_no を検索します。
4. 検索した値をパラメタ next_no に設定します。
5. 表 owner_id.sequence_tbl の列 sequence_no に増分値 1 を加えた値に更新します。
6. 表と手続きの定義を有効にするため，トランザクションをコミットします。
7. INSERT 文で，初期値 1 を持つ行をあらかじめ挿入しておきます。
8. 挿入した行を有効にするため，トランザクションをコミットします。
9. 手続き owner_id.nextval を CALL 文で呼び出し，順序番号を割り当て，パラメタ next_no で値を取得します。CALL 文を実行するごとに次の順序番号が割り当てられます。

（例 2）

WITHOUT ROLLBACK 指定の表と，ストアドプロシジャを使用して，2 種類以上の順序番号を割り当てます。

順序番号を識別するキーごとに，初期値 1，増分値 1 で INTEGER の最大値までの値を採番します。

INTEGER の最大値を超えた場合は，オーバフローのエラーが返されます。ただし，既定値設定機能（PDDFLNVAL）使用時はオーバフローのエラーにはならないで，ナル値となるため，非ナル値制約違反のエラーとなります。また，順序番号を識別するために指定したキー値に対して，初期値を持つ行が挿入されていない場合，表中に行がない状態になるため，UPDATE 文実行時にカーソルが行に位置づけられていないというエラーになります。順序番号を識別するために指定したキー値に対して，複数行が挿入された場合には 2 行目以降は無視されます。

注 1

WITHOUT ROLLBACK 指定の表には，インデクスを定義できません。排他の競合を防ぐためにクライアント環境定義の PDLOCKSKIP=YES を指定する必要があります。

注 2

WITHOUT ROLLBACK 指定の表には，インデクスを定義できないので，順序番号の種類が非常に多い場合は表及び手続きを分けてください。

```
CREATE FIX TABLE
    owner_id.sequence_tbl(sequence_key CHAR(30) NOT NULL,
                          sequence_no INTEGER  NOT NULL)
    WITHOUT ROLLBACK;  .....................................1
CREATE PROCEDURE owner_id.nextval(IN input_key CHAR(30),
                                  OUT next_no INTEGER)
  BEGIN
    DECLARE update_no INTEGER;  ............................2
    DECLARE cr1 CURSOR FOR
      SELECT sequence_no FROM owner_id.sequence_tbl
        WHERE sequence_key=input_key FOR UPDATE OF sequence_no;
    OPEN cr1;
    FETCH cr1 INTO update_no;  .............................3
    SET next_no=update_no;  ................................4
    UPDATE owner_id.sequence_tbl SET sequence_no=update_no+1
      WHERE CURRENT OF cr1;  ...............................5
    CLOSE cr1;  ............................................3
  END  .....................................................2
COMMIT WORK;  ..............................................6
INSERT INTO owner_id.sequence_tbl(sequence_key,sequence_no)
    VALUES('key_value_1',1);  ..............................7
COMMIT WORK;  ..............................................8
INSERT INTO owner_id.sequence_tbl(sequence_key,sequence_no)
    VALUES('key_value_2',1);  ..............................7
COMMIT WORK;  ..............................................8
       :
（順序番号の種類数分，初期値の行を挿入します）

<'key_value_1'の順序番号の割り当て>  ......................9
xinput_key <-- 'key_value_1'
CALL owner_id.nextval(IN:xinput_key,OUT:xnext_no);
                  :
'key_value_1'に対して割り当てた順序番号xnext_noを使用した処理
                  :
xinput_key <-- 'key_value_1'
CALL owner_id.nextval(IN:xinput_key,OUT:xnext_no);
                  :

<'key_value_2'の順序番号の割り当て>  ......................9
```

```
xinput_key <-- 'key_value_2'
CALL owner_id.nextval(IN :xinput_key,OUT:xnext_no);
                    :
'key_value_2'に対して割り当てた順序番号xnext_noを使用した処理
                    :
xinput_key <-- 'key_value_2'
CALL owner_id.nextval(IN:xinput_key,OUT:xnext_no);
                    :
```

[説明]

1. 順序番号を識別するキーごとに，INTEGER の値を採番するための表 owner_id.sequence_tbl を定義します。
2. 順序番号を識別するキーをパラメタ input_key で入力し，そのキーに対して順序番号を割り当てて，パラメタ next_no で出力する手続き owner_id.nextval を定義します。
3. 表 owner_id.sequence_tbl の列 sequence_key に対して順序番号を識別するキーを指定して，列 sequence_no を検索します。
4. 検索した値をパラメタ next_no に設定します。
5. 表 owner_id.sequence_tbl の列 sequence_no を増分値 1 を加えた値に更新します。
6. 表と手続きの定義を有効にするため，トランザクションをコミットします。
7. 順序番号を識別するキーごとに，INSERT 文で初期値 1 を持つ行をあらかじめ挿入しておきます。
8. 挿入した行を有効にするため，トランザクションをコミットします。
9. 手続き owner_id.nextval を CALL 文で呼び出し，順序番号を割り当て，パラメタ next_no で値を取得します。CALL 文を実行するごとに次の順序番号が割り当てられます。

(例 3)

WITHOUT ROLLBACK 指定の表と，ストアドプロシジャを使用して，最小値と最大値の間の値を循環させて順序番号を割り当てます。

初期値を持つ行が挿入されていない場合，表中に行がない状態になるため，UPDATE 文実行時にカーソルが行に位置づけられていないというエラーになります。また，複数行が挿入されている場合には，2 行目以降は無視されます。

```
CREATE FIX TABLE
     owner_id.sequence_tbl(sequence_no INTEGER NOT NULL)
     WITHOUT ROLLBACK;  .....................................1
CREATE PROCEDURE owner_id.nextval(OUT next_no INTEGER)
  BEGIN
    DECLARE update_no INTEGER;  ..............................2
    DECLARE cr1 CURSOR FOR
      SELECT sequence_no FROM owner_id.sequence_tbl FOR UPDATE;
    OPEN cr1;
    FETCH cr1 INTO update_no;  ...............................3
    SET next_no=update_no;  ..................................4
    IF update_no=2147483647 THEN
      SET update_no=-2147483648;
    ELSE
      SET update_no=update_no+1;
    END IF;  .................................................5
```

```
    UPDATE owner_id.sequence_tbl SET sequence_no=update_no
      WHERE CURRENT OF cr1;  .....................................6
    CLOSE cr1;  ................................................3
  END  .........................................................2
COMMIT WORK;  ..................................................7
INSERT INTO owner_id.sequence_tbl(sequence_no)VALUES(1);  .......8
COMMIT WORK;  ..................................................9

<順序番号の割り当て>  ..........................................10
CALL owner_id.nextval(OUT:xnext_no);
                :
割り当てた順序番号xnext_noを使用した処理
                :
CALL owner_id.nextval(OUT:xnext_no);
                :
```

[説明]

1. INTEGER の値を採番するための表 owner_id.sequence_tbl を定義します。
2. 表 owner_id.sequence_tbl の列 sequence_no を，増分値 1，最小値−2,147,483,648，最大値 2,147,483,647 で，最大値の次の値が最小値となるように値を循環させて順序番号を割り当てる手続き owner_id.nextval を定義します。
3. 表 owner_id.sequence_tbl の列 sequence_key を検索します。
4. 検索した値をパラメタ next_no に設定します。
5. 検索した値が最大値 2,147,483,647 ならば，最小値−2,147,483,648 を順序番号の次の値とし，そうでなければ増分値 1 を加えた値を順序番号の次の値とします。
6. 表 owner_id.sequence_tbl の列 sequence_no を順序番号の次の値に更新します。
7. 表と手続きの定義を有効にするため，トランザクションをコミットします。
8. INSERT 文で，初期値 1 を持つ行をあらかじめ挿入しておきます。
9. 挿入した行を有効にするため，トランザクションをコミットします。
10. 手続き owner_id.nextval を CALL 文で呼び出し，順序番号を割り当て，パラメタ next_no で値を取得します。CALL 文を実行するごとに次の順序番号が割り当てられます。

4.2.3　文字集合を使用した表

文字集合を定義すると，表の列ごとに異なる文字集合の文字列データを格納できます。

(1)　文字集合を指定した文字列データの受け渡し

文字集合を指定した文字列データを受け渡す場合の例について説明します。

（例）

表 T1 の列 C2（文字集合 EBCDIK）を検索します。表 T1，及び検索 SQL を次に示します。

- 表 T1 は，次のように定義されています（下線部分が文字集合指定です）。

```
CREATE TABLE T1
  (C1 INT, C2 CHAR(30) CHARACTER SET EBCDIK)
```

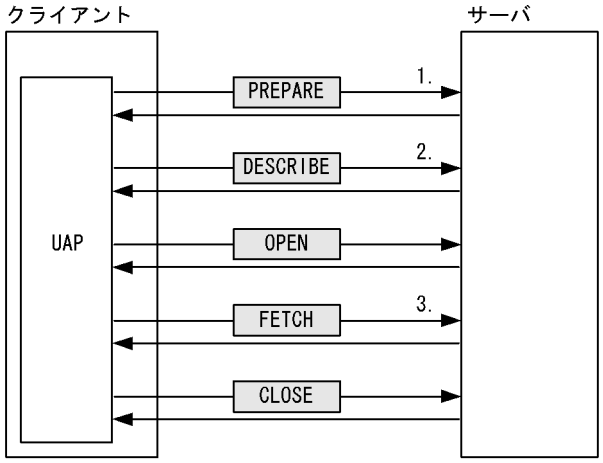
- 検索 SQL は次のようになります。

```
＜埋込み変数の宣言＞

char DATA[31];
     :
SELECT C2 FROM T1
  WHERE C2 = :DATA
```

この場合のデータの受け渡しの流れを次の図に示します。

図 4–8　文字集合を指定した文字列データの受け渡し



[説明]

1. 列 C2 の文字集合の情報がディクショナリ表から取得され，SQL オブジェクトに格納されます。
2. 文字集合名記述領域※に C2 の文字集合名が設定されます。
3. 入力変数の文字集合と文字集合名記述領域に設定された文字集合が異なる場合，変換して代入されます。

   出力変数の文字集合と文字集合名記述領域に設定された文字集合が異なる場合，変換して格納されます。

注※

UAP 実行時に動的に決定した変数(DATA)の文字集合名の情報を記述した領域のことです。文字集合名記述領域の内容を介して，次の処理が行われます。

- クライアント側で指定した変数の文字集合名情報を HiRDB サーバに通知する
- HiRDB サーバで前処理した SQL 文の検索項目や，？パラメタの文字集合名情報をクライアント側で受け取る

文字集合名記述領域については，「文字集合名記述領域」を参照してください。

(2)　文字コード変換

クライアントとサーバの文字集合が異なる場合，サーバ側で文字コードを変換します。クライアントとサーバの両方が既定文字集合の場合，文字コード変換はしません。ただし，クライアント環境定義の PDCLTCNVMODE の指定がある場合は，それに従って変換します。

文字集合の定義の有無と，クライアントとサーバ間での文字コード変換について次の表に示します。

表 4–2　クライアントとサーバ間での文字コード変換

クライアント側の文字コード		サーバ側の文字コード							
		SJIS		UJIS	UTF-8，IVS 対応 UTF-8		LANG-C	CHINESE	CHINESE-GB18030
		なし	EK	なし	なし	U16	なし	なし	なし
SJIS	なし	−	SVR	CLT	CLT	C-S	△	×	×
	EK	SVR	−	×	×	×	×	×	×
UJIS	なし	×	×	−	×	×	△	×	×
UTF-8	なし	×	×	×	−	SVR	△	×	×
	U16C	×	×	×	SVR	−※	×	×	×
UCS-2	なし	×	×	CLT	CLT	C-S	×	×	×
LANG-C	なし	−	SVR	−	−	SVR	−	−	−
CHINESE	なし	×	×	×	×	×	△	−	×
CHINESE-GB18030	なし	×	×	×	×	×	×	×	−

（凡例）

なし：文字集合を定義していません。

EK：文字集合に EBCDIK を定義しています。

U16：文字集合に UTF16 を定義しています。

U16C：文字集合に UTF16，UTF-16LE，UTF-16BE のどれかを定義しています。

SVR：サーバ側で文字コード変換をします。

CLT：クライアント側で文字コード変換をします（クライアント環境定義 PDCLTCNVMODE で変換を指定）。

C-S：クライアント側でサーバ側の文字コード変換をして（クライアント環境定義 PDCLTCNVMODE で変換を指定），サーバ側で文字集合の文字コード変換をします。

−：文字コード変換が不要です。

△：文字コード変換が不要です。ただし，クライアント環境定義 PDCLTCNVMODE に NOUSE を指定している場合は文字コード変換をします（全角文字は変換しません）。

×：文字コード変換ができません。

注※

クライアント側の文字コードとサーバ側の文字コードのエンディアンが異なる場合，エンディアン変換をします。

(a) ODBC 対応アプリケーションプログラム，又は ADO.NET 対応アプリケーションプログラムからのアクセス

ODBC 対応アプリケーションプログラム，又は ADO.NET 対応アプリケーションプログラムからのアクセスの場合も，表「クライアントとサーバ間での文字コード変換」の変換規則に従います。

ただし，次に示すクライアントのバージョンでは文字集合名記述領域をそれぞれ仮定します。

クライアントのバージョンが 08-05 より前の場合

文字集合名記述領域なしと仮定して，表「クライアントとサーバ間での文字コード変換」の変換規則に従います。

クライアントのバージョンが 08-05 以降の場合

クライアントの文字コードが UCS-2 で，サーバ側の文字コードが UTF-8 の場合，文字集合名記述領域は UTF-16LE を仮定して，表「クライアントとサーバ間での文字コード変換」の変換規則に従います。

(b) OLE DB 対応アプリケーションプログラム，又は Type2 JDBC ドライバを使用する JDBC 対応アプリケーションプログラムからのアクセス

文字集合名記述領域なしと仮定して，表「クライアントとサーバ間での文字コード変換」の変換規則に従います。

(c) Type4 JDBC ドライバを使用する JDBC 対応アプリケーションプログラムからのアクセス

クライアントのバージョンが 08-05 より前，又はサーバ側の文字コードが UTF-8 以外の場合

JDBC ドライバが，Java 仮想マシンによって提供されるエンコーダを利用してサーバ側の文字コードに変換します。サーバ側に文字集合の指定がある場合は，サーバ側で文字集合の文字コードに変換します。

クライアントのバージョンが 08-05 以降，かつサーバ側の文字コードが UTF-8 の場合

サーバ側に文字集合 UTF16 の指定がある場合は，UTF-16BE でサーバとデータの受け渡しをします。サーバ側に文字集合 UTF16 の指定がない場合は，Java 仮想マシンによって提供されるエンコーダを利用して UTF-8 に変換します。

4.3　ストアドプロシジャ，ストアドファンクション

ストアドプロシジャ，ストアドファンクションの定義方法について説明します。

ストアドプロシジャ，ストアドファンクションを定義する場合，事前に必要な RD エリアを作成しておいてください。ストアドプロシジャ及びストアドファンクションの運用については，マニュアル「HiRDB システム運用ガイド」を参照してください。

ストアドプロシジャ，ストアドファンクションは，処理手続きを SQL，Java，又は C 言語で記述できます。SQL で記述したものを，**SQL ストアドプロシジャ**，**SQL ストアドファンクション**，Java で記述したものを，**Java ストアドプロシジャ**，**Java ストアドファンクション**，C 言語で記述したものを，**C ストアドプロシジャ**，**C ストアドファンクション**といいます。

また，ストアドプロシジャとストアドファンクションを総称して，**ストアドルーチン**といいます。さらに，すべての利用者を示す PUBLIC を所有者として定義するストアドルーチンを**パブリックルーチン**といいます。

Java ストアドプロシジャ，Java ストアドファンクションについては，「Java ストアドプロシジャ，Java ストアドファンクション」を参照してください。C ストアドプロシジャ，C ストアドファンクションについては，「C ストアドプロシジャ，C ストアドファンクション」を参照してください。

- **注意事項**

  SQL ストアドプロシジャ，SQL ストアドファンクション実行中にエラーが発生した場合は，エラーが発生した時点で SQL ストアドプロシジャ，SQL ストアドファンクションの処理を終了します（SQL ストアドプロシジャ，SQL ストアドファンクションの制御から抜けます）。したがって，SQL ストアドプロシジャ，SQL ストアドファンクション中にはエラー処理を記述できません。

4.3.1　ストアドプロシジャの定義

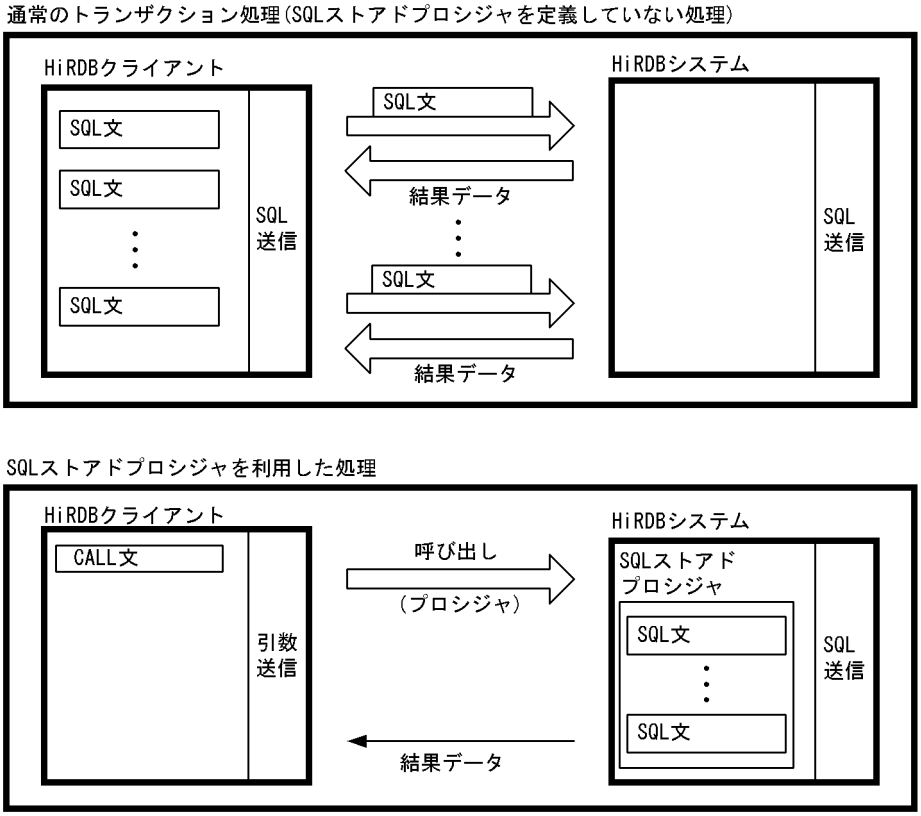
ストアドプロシジャは，SQL で記述した一連のデータベース処理手続きを，手続きとしてデータベースに登録しておく機能です。

(1)　SQL ストアドプロシジャの効果

データベースを操作する場合，FETCH 文で検索し，データの有無によって UPDATE 文，又は INSERT 文の発行を繰り返すと，クライアントとサーバ間のオーバヘッドが増大します。このようなデータベースアクセス処理を手続き（プロシジャ）として定義しておくと，CALL 文で手続きを呼び出すだけで，一連の処理を実行できるため，クライアントサーバ間のデータの受渡しなどオーバヘッドの抑制が図れます。また，手続き中の SQL 文はコンパイルされた形式（SQL オブジェクト）でサーバ側に登録されているため，処理を共用化できる以外に，SQL 解析のオーバヘッドも削減できます。

SQL ストアドプロシジャの効果を次の図に示します。

図 4–9　SQL ストアドプロシジャの効果



(2)　SQL ストアドプロシジャの定義と実行

SQL ストアドプロシジャは，CREATE PROCEDURE 又は CREATE TYPE 実行時に手続きがデータベースに登録され，DROP PROCEDURE 実行時に削除されます。

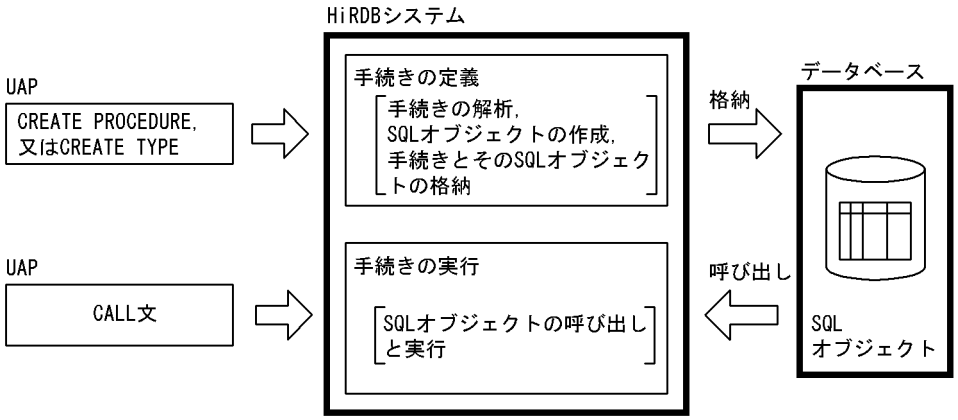
登録した手続きは，CALL 文で呼び出して実行します。

SQL オブジェクトが無効になっている手続きがある場合は，ALTER PROCEDURE 又は ALTER ROUTINE で手続きを再作成することもできます。

また，既に SQL ストアドプロシジャがある場合には，pddefrev コマンドを実行すれば，既にある SQL ストアドプロシジャの定義系 SQL を作成できます。処理が似ている SQL ストアドプロシジャを新たに作成するときに使用すると便利です。pddefrev コマンドについては，マニュアル「HiRDB コマンドリファレンス」を参照してください。

SQL ストアドプロシジャの定義と実行を次の図に示します。

図 4–10　SQL ストアドプロシジャの定義と実行



パブリックプロシジャ

他ユーザが定義したストアドプロシジャを使用する場合は，UAP 中からストアドプロシジャを呼び出すときに，所有者の認可識別子とルーチン識別子を指定する必要があります。しかし，CREATE PUBLIC PROCEDURE を実行してパブリックプロシジャとして定義すると，他ユーザが定義したストアドプロシジャを使用する場合でも，UAP 中からストアドプロシジャを呼び出すときに，所有者の認可識別子を指定する必要がなくなります（ルーチン識別子だけ指定します）。

(3)　SQL ストアドプロシジャの例

SQL ストアドプロシジャの例として，複数の SQL 文と，それらを制御する文（ルーチン制御 SQL）を組み合わせて一つの手続き（プロシジャ）として定義する例とそれを呼び出して実行する例を次の図に示します。

図 4-11　SQL ストアドプロシジャの例

SQLストアドプロシジャの定義

```
CREATE PROCEDURE proc_1 (IN fromdate date, IN todate date) ----------1.
  BEGIN ---------------------------------------------------------------2.
    DECLARE x_goods_no INTEGER; ------------------------------------3.
    DECLARE x_quantity_1, x_total_quantity DECIMAL(17,2); ----------3.
    DECLARE cr1 CURSOR FOR
      SELECT goods_no, quantity_1, entrydate FROM tran_t1
        WHERE entrydate BETWEEN fromdate AND todate
        ORDER BY entrydate;

    OPEN cr1;
    while_loop:
    WHILE SQLCODE = 0 DO ------------------------------------ 4.
      FETCH cr1 INTO x_goods_no, x_quantity_1;
        IF SQLCODE=100 THEN
        LEAVE while_loop;  ---------------------------------------5.
      END IF;
      SELECT total_quantity INTO x_total_quantity
        FROM master_t1 WHERE goods_no = x_goods_no;
      IF SQLCODE=100 THEN ------------------------------------ 6.
        INSERT INTO master_t1 VALUES(x_goods_no, x_quantity_1);
      ELSE
        SET x_total_quantity = x_total_quantity + x_quantity_1; -------7.
        UPDATE master_t1 SET total_quantity = x_total_quantity
          WHERE goods_no = x_goods_no;
      END IF  ------------------------------------------------ 8.
    END while_loop; ----------------------------------------------9.
    CLOSE cr1;
  END; ---------------------------------------------------------------10.
```

SQLストアドプロシジャの呼び出し

```
          :
strcpy(e_fromdate, "1996-06-01");
strcpy(e_todate, "1996-06-30");
EXEC SQL CALL proc_1(IN :e_fromdate, IN :e_todate); ---------------- 11.
          :
```

[説明]

1. 手続きの名前と SQL パラメタの定義
2. 複合文の開始
3. SQL 変数の宣言
4. 文の繰り返しの指定
5. 文の途中終了の指定
6. 条件分岐の指定
7. 値の代入の指定
8. 条件分岐の終了
9. 文の繰り返しの終了
10. 複合文の終了
11. 手続きの呼び出し

注 1

各 SQL 文については，マニュアル「HiRDB SQL リファレンス」を参照してください。

注 2

この例では，entrydate でソートするためにカーソル宣言の SELECT 句に選択項目として entrydate を指定しています。ただし，この値は参照しないため，FETCH 文では対応する埋込み変数を省略して，entrydate の値は取り出しません。

(4) SQL ストアドプロシジャのデバッグ

SQL ストアドプロシジャのデバッグをする場合，参照する SQL 変数，SQL パラメタなどを，ルーチン制御 SQL の WRITE LINE 文を使用してクライアント側のファイルに出力して行います。WRITE LINE 文については，マニュアル「HiRDB SQL リファレンス」を参照してください。

SQL ストアドプロシジャ中に WRITE LINE 文を記述する例を次に示します。

```
CREATE PROCEDURE proc_1 (IN fromdate date, IN todate date)
  BEGIN
    :
    :
    WRITE LINE  'fromdate=' ||char(fromdate);   .........................1
    WRITE LINE  'todate=' ||char(todate);   ...........................2
    :
    :
```

[説明]

1. SQL パラメタ「fromdate」の値を文字列に変換して，ファイルに出力する指定です。
2. SQL パラメタ「todate」の値を文字列に変換して，ファイルに出力する指定です。

WRITE LINE 文を記述した SQL ストアドプロシジャから，WRITE LINE 文の値式の値をクライアント側のファイルに出力する場合，クライアント環境定義 PDWRTLNFILSZ を設定し，UAP から SQL ストアドプロシジャを呼び出します。例を次に示します。

csh（C シェル）の場合の PDWRTLNFILSZ の設定例（HiRDB クライアントが UNIX 版の場合）

```
setenv PDWRTLNFILSZ 4096
```

PDWRTLNFILSZ の設定例（HiRDB クライアントが Windows 版の場合）

```
PDWRTLNFILSZ=4096
```

SQL ストアドプロシジャの呼び出し

```
strcpy(e_fromdate, "2003-06-01");
strcpy(e_todate, "2003-06-30");
EXEC SQL CALL proc_1(IN :e_fromdate, IN :e_todate);
```

出力ファイルの内容

```
fromdate=2003-06-01
todate=2003-06-30
```

注　出力ファイルは，クライアント環境定義 PDWRTLNPATH で設定します。

デバッグ終了後，WRITE LINE 文を記述した SQL ストアドプロシジャから，WRITE LINE 文の値式の値をファイルに出力する必要がなくなった場合，クライアント環境定義 PDWRTLNFILSZ を省略して UAP を実行してください。PDWRTLNFILSZ の指定を省略すると，SQL ストアドプロシジャ中の WRITE LINE 文は実行されません。

(5)　ストアドプロシジャ中のトランザクションの決着

(a)　トランザクションを決着する SQL

次の SQL 文をストアドプロシジャ中で実行すると，トランザクションを決着できます。ただし，C ストアドプロシジャ中では，SQL 文を実行できません。

- COMMIT 文
- ROLLBACK 文

次の SQL を実行した場合，自動的に COMMIT が行われます。

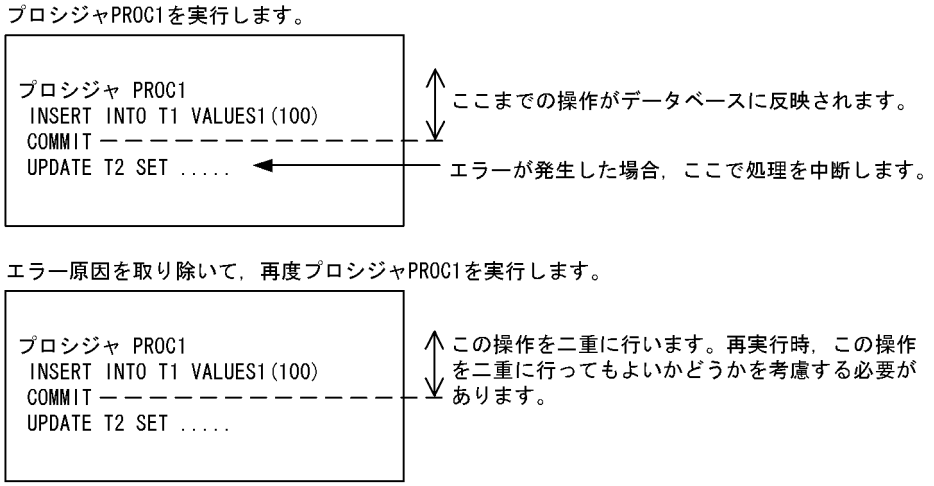
- PURGE TABLE 文
- 定義系 SQL（Java ストアドプロシジャだけ）

次の条件に該当する場合，自動的に ROLLBACK が行われます。

- ROLLBACK が必要なエラー

(b)　ストアドプロシジャを再実行する場合の注意事項

ストアドプロシジャ中でトランザクションを決着した後にエラーが発生した場合，そのプロシジャの実行は途中で終了します。エラーになったストアドプロシジャを再実行する場合，プロシジャの処理は先頭から実行されるため，エラー発生前のトランザクション解決までに行った操作が，二重に実行されてよいかどうかを考慮する必要があります。例を次に示します。



(6) 結果集合返却機能（SQL ストアドプロシジャ限定）

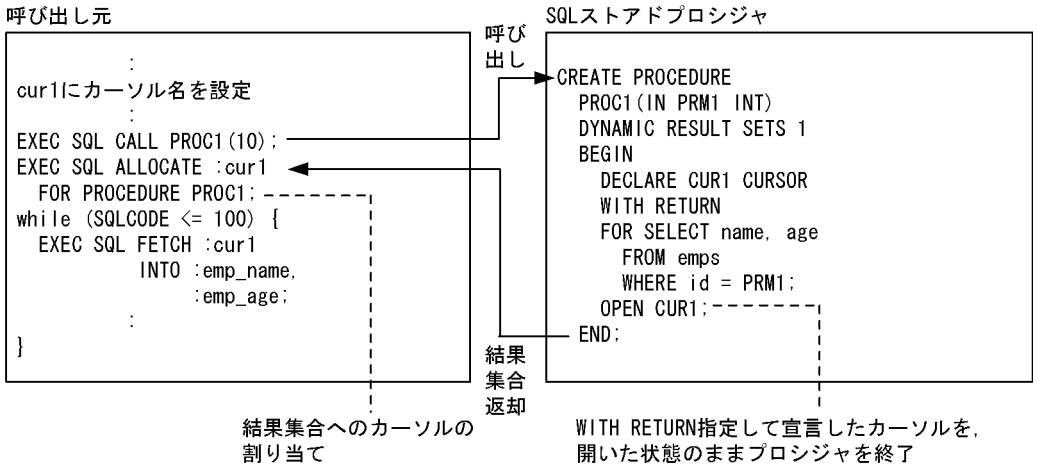
SQL ストアドプロシジャ定義時に，CREATE PROCEDURE の DYNAMIC RESULT SETS 句に 1 以上の値を指定した場合，結果集合返却機能を使用できます。なお，SQL ストアドファンクションについては，結果集合返却機能は使用できません。

(a) 結果集合返却機能とは

SQL ストアドプロシジャ内での，SELECT 文の実行によって得られるカーソルを，SQL ストアドプロシジャの呼び出し元で参照する機能を，**結果集合返却機能**といいます。

結果集合返却機能の概要を次の図に示します。

図 4–12 結果集合返却機能の概要（SQL ストアドプロシジャの場合）



(b) 結果集合返却機能を使用できる呼び出し元の言語

結果集合返却機能を使用できる呼び出し元の言語を次に示します。

- Java
- C

- C++
- COBOL※
- OOCOBOL

注※

RDB ファイル入出力機能を使用していない場合，使用できます。

(c) 結果集合返却機能の使用例

SQL ストアドプロシジャ内で，表 emps_1，及び表 emps_2 に対して，id<10 の条件を満たす列 id，name，及び age を取得します。呼び出し元で 2 個の結果集合を受け取り，これらを操作します。

- **SQL ストアドプロシジャの定義**

```
CREATE PROCEDURE proc2(IN param1 INTEGER) ..............  1
  DYNAMIC RESULT SETS 2 ................................  2
  BEGIN
    DECLARE CUR1 CURSOR WITH RETURN ....................  3
       FOR SELECT id,name,age FROM emps_1
            WHERE id < param1 ORDER BY id;
    DECLARE CUR2 CURSOR WITH RETURN ....................  4
       FOR SELECT id,name,age FROM emps_2
            WHERE id < param1 ORDER BY id;
    OPEN CUR1; ..........................................  5
    OPRN CUR2; ..........................................  6
  END; ..................................................  7
```

[説明]

1. プロシジャ名，パラメタの定義
2. 返却する検索結果情報数の指定
3. カーソル（CUR1）の宣言
4. カーソル（CUR2）の宣言
5. カーソル（CUR1）のオープン
6. カーソル（CUR2）のオープン
7. 呼び出し終了，及び結果集合返却

- **呼び出し元（C 言語の埋込み型 UAP の場合）**

```
#include <stdio.h>
#include <string.h>

main()
{
  EXEC SQL BEGIN DECLARE SECTION;
    struct {
      short len;
      char str[31];
    } cur1;
```

```
    int emp_id;
    char emp_name[13];
    int emp_age;
  EXEC SQL END DECLARE SECTION;
  -------- (HiRDBへのCONNECT処理 (省略) ) --------

  cur1.len = sprintf(cur1.str, "cursor1"); ..... 1

  EXEC SQL CALL PROC(10); ..... 2

  If (SQLCODE == 120) { ..... 3

    EXEC SQL ALLOCATE GLOBAL :cur1
               FOR PROCEDURE PROC2; ..... 4
    printf("*** emps_1 ***¥n"); ..... 5
    while (1) { ..... 5
      EXEC SQL FETCH GLOBAL :cur1 ..... 5
                 INTO :emp_id,:emp_name,:emp_age; ..... 5
      if (SQLCODE<0 || SQLCODE==100) break; ..... 5
      printf("ID=%d  NAME=%s  AGE=%d¥n", ..... 5
            emp_id, emp_name, emp_age); ..... 5
    } ..... 5
    CLOSE GLOBAL :cur1; ..... 6
    if (SQLCODE==121) { ..... 7
      printf("*** emps_2 ***¥n"); ..... 8
      while (1) { ..... 8
    EXEC SQL FETCH GLOBAL :cur1 ..... 8
                   INTO :emp_id,:emp_name,:emp_age; ..... 8
        if (SQLCODE<0 || SQLCODE==100) break; ..... 8
        printf("ID=%d  NAME=%s  AGE=%d¥n", ..... 8
              emp_id, emp_name, emp_age); ..... 8
      } ..... 8
      CLOSE GLOBAL :cur1; ..... 9
    }
  }
}
```

[説明]

1. カーソル名の設定
2. CALL 文の実行
3. 返却結果集合があるかどうかの確認
4. カーソルの割り当て（1 個目の結果集合とカーソルを関連づけます）
5. 1 個目の結果集合からの情報を出力
6. カーソルを閉じる（2 個目の結果集合とカーソルを関連づけます）
7. 次の結果集合があるかどうかの確認
8. 2 個目の結果集合からの情報を出力
9. カーソルを閉じる

(d)　結果集合返却機能を使用する場合の注意事項

- **CREATE PROCEDURE での SQL ストアドプロシジャ定義時**

  1. 結果集合として返却するカーソルのカーソル宣言には，「WITH RETURN」を指定してください。
  2. 結果集合として返却するカーソルは，「WITH RETURN」指定で宣言したカーソルのうち，プロシジャ終了時に開いた状態のカーソルだけです。
  3. 返却する結果集合が 2 個以上の場合，カーソルを開いた順序で返却されます。

- **呼び出し元作成時**

  1. 結果集合を返却するプロシジャを実行した場合，SQLSTATE に 0100C，SQLCODE に 120 が設定されます。
  2. 埋込み型 UAP，及び SQL ストアドプロシジャで結果集合を受け取る場合には，ALLOCATE CURSOR 文で結果集合の組にカーソルを割り当て，その先頭の結果集合とカーソルを関連づけます。返却される結果集合が 2 個以上の場合，2 個目以降の結果集合は，前の結果集合を参照しているカーソルに対して CLOSE 文を実行すると，カーソルと関連づけられます。CLOSE 文を実行した場合に，次の結果集合があり，その結果集合とカーソルを関連づけたときには，SQLSTATE に 0100D，SQLCODE に 121 が設定されます。次の結果集合がない場合には，SQLSTATE に 02001，SQLCODE に 100 が設定されます。

4.3.2　ストアドファンクションの定義

ストアドファンクションは，SQL で記述した一連のデータベース操作を，ユーザ定義関数としてデータベースに登録しておく機能です。

(1)　SQL ストアドファンクションの定義と実行

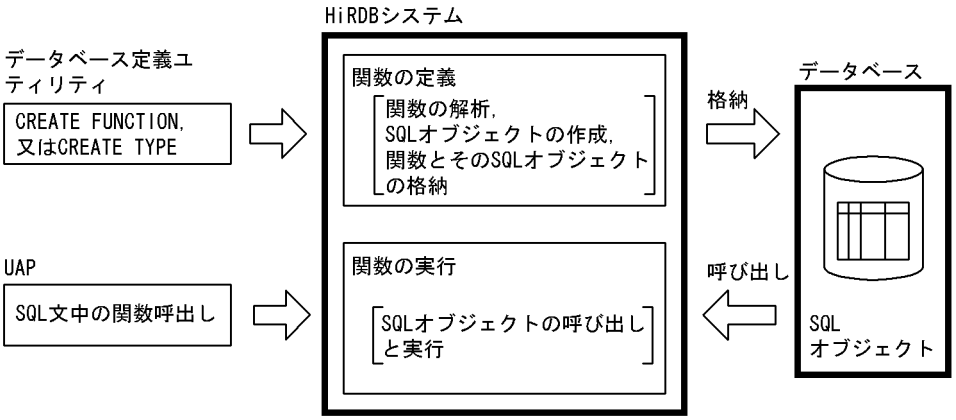
SQL ストアドファンクションは，CREATE FUNCTION 又は CREATE TYPE 実行時にユーザ定義関数がデータベースに登録され，DROP FUNCTION 実行時に削除されます。

登録したユーザ定義関数は，SQL 文中に関数呼出しを指定すれば，その関数を呼び出して実行します。

SQL オブジェクトが無効になっている関数がある場合には，ALTER ROUTINE で関数を再作成することもできます。

SQL ストアドファンクションの定義と実行を次の図に示します。

図 4–13　SQL ストアドファンクションの定義と実行



パブリックファンクション

他ユーザが定義したストアドファンクションを使用する場合は，UAP 中からストアドファンクションを呼び出すときに，所有者の認可識別子とルーチン識別子を指定する必要があります。しかし，CREATE PUBLIC FUNCTION を実行してパブリックファンクションとして定義すると，他ユーザが定義したストアドファンクションを使用する場合でも，UAP 中からストアドファンクションを呼び出すときに，所有者の認可識別子を指定する必要がなくなります（ルーチン識別子だけ指定します）。

(2)　SQL ストアドファンクションの例

SQL ストアドファンクションの例として，ルーチン制御 SQL を組み合わせてユーザ定義関数（ファンクション）として定義する例とそれを呼び出して実行する例を次の図に示します。

図 4–14　SQL ストアドファンクションの例

SQLストアドファンクションの定義

```
CREATE FUNCTION age(birthday date)------------------------ 1.
  RETURNS int ---------------------------------------- 2.
  BEGIN ---------------------------------------------- 3.
    DECLARE today date;-------------------------------- 4.
    SET today = CURRENT DATE;-------------------------- 5.
    RETURN YEAR(today - birthday);--------------------- 6.
  END;----------------------------------------------- 7.
```

SQLストアドファンクションの呼び出し

```
SELECT PID,AGE(BIRTHDAY) FROM EMP;------------------------ 8.
```

[説明]

1. ユーザ定義関数の名前と SQL パラメタの定義
2. 関数の戻り値の指定
3. 複合文の開始

4. SQL 変数の宣言

5. 値の代入の指定

6. 関数の戻り値を返却する指定

7. 複合文の終了

8. 関数呼出しを使用した検索

注

各 SQL 文については，マニュアル「HiRDB SQL リファレンス」を参照してください。

関数の定義例：

- 指定した日付の月末の日付を求める関数

```
CREATE FUNCTION LASTDAY(INDATE DATE) RETURNS DATE
  BEGIN
    DECLARE MM1 INTEGER;
    SET MM1=MONTH(INDATE)-1;
    RETURN (INDATE-MM1 MONTHS+(31-DAY(INDATE)) DAYS+MM1 MONTHS);
  END
```

- 指定した日付の曜日を 0（日）～6（土）の整数で求める関数

```
CREATE FUNCTION DNOFWEEK(INDATE DATE) RETURNS INTEGER
  BEGIN
    RETURN MOD(DAYS(INDATE),7);
  END
```

- 指定した日付の曜日を日本語で求める関数

```
CREATE FUNCTION 曜日(INDATE DATE) RETURNS NCHAR
  BEGIN
    RETURN (CASE MOD(DAYS(INDATE),7) WHEN 0 THEN N'日'
      WHEN 1 THEN N'月'
      WHEN 2 THEN N'火'
      WHEN 3 THEN N'水'
      WHEN 4 THEN N'木'
      WHEN 5 THEN N'金'
      ELSE N'土'  END);
  END
```

- 指定した日付の曜日を英語で求める関数

```
CREATE FUNCTION DAYOFWEEK(INDATE DATE) RETURNS CHAR(3)
  BEGIN
    RETURN (CASE MOD(DAYS(INDATE),7) WHEN 0 THEN 'SUN'
      WHEN 1 THEN 'MON'
      WHEN 2 THEN 'TUE'
      WHEN 3 THEN 'WED'
      WHEN 4 THEN 'THU'
      WHEN 5 THEN 'FRI'
      ELSE 'SAT'  END);
  END
```

- 指定した日付の直後の，指定した曜日の日付を求める関数

  （引数の曜日が日本語（N'日'～N'土'）の場合）

```
CREATE FUNCTION NEXT_DAY(INDATE DATE,曜日 NCHAR)
  RETURNS DATE
  BEGIN
    DECLARE SDOW,TDOW INTEGER;
    SET TDOW=(CASE 曜日 WHEN N'日' THEN 0
      WHEN N'月' THEN 1
      WHEN N'火' THEN 2
      WHEN N'水' THEN 3
      WHEN N'木' THEN 4
      WHEN N'金' THEN 5
      ELSE 6 END);
    SET SDOW=MOD(DAYS(INDATE),7);
    RETURN (INDATE + (CASE WHEN TDOW>SDOW THEN TDOW-SDOW
      ELSE 7+TDOW-SDOW END) DAYS);
  END
```

  （引数の曜日が英語（'SUN'～'SAT'）の場合）

```
CREATE FUNCTION NEXT_DAY(INDATE DATE,DAYOFWEEK CHAR(3))
  RETURNS DATE
  BEGIN
    DECLARE SDOW,TDOW INTEGER;
    SET TDOW=(CASE DAYOFWEEK WHEN 'SUN' THEN 0
      WHEN 'MON' THEN 1
      WHEN 'TUE' THEN 2
      WHEN 'WED' THEN 3
      WHEN 'THU' THEN 4
      WHEN 'FRI' THEN 5
      ELSE 6 END);
    SET SDOW=MOD(DAYS(INDATE),7);
    RETURN (INDATE + (CASE WHEN TDOW>SDOW THEN TDOW-SDOW
      ELSE 7+TDOW-SDOW END) DAYS);
  END
```

  （引数の曜日が整数（0～6）の場合）

```
CREATE FUNCTION NEXT_DAY(INDATE DATE,DNOFWEEK INTEGER)
  RETURNS DATE
  BEGIN
    DECLARE SDOW,TDOW INTEGER;
    SET TDOW=DNOFWEEK;
    SET SDOW=MOD(DAYS(INDATE),7);
    RETURN (INDATE + (CASE WHEN TDOW>SDOW THEN TDOW-SDOW
      ELSE 7+TDOW-SDOW END) DAYS);
  END
```

- 20 日締めで指定した日付の年月（'yyyy-mm'）を求める関数

```
CREATE FUNCTION YYYYMM20(INDATE DATE) RETURNS CHAR(7)
  BEGIN
    RETURN SUBSTR(CHAR(INDATE+1 MONTH -20 DAYS),1,7);
  END
```

- 3 月 20 日締めで指定した日付の年度（'yyyy'）を求める関数

```
CREATE FUNCTION YYYY0320(INDATE DATE) RETURNS CHAR(4)
  BEGIN
    RETURN SUBSTR(CHAR(INDATE-2 MONTHS -20 DAYS),1,4);
  END
```

- 3 月 20 日締めで指定した日付の年度と四半期（'yyyy-nQ'）を求める関数

```
CREATE FUNCTION YYYYNQ0320(INDATE DATE) RETURNS CHAR(7)
  BEGIN
    DECLARE WORKDATE DATE;
    SET WORKDATE=(INDATE -2 MONTHS -20 DAYS);
    RETURN (SUBSTR(CHAR(WORKDATE), 1, 5) ||
            SUBSTR(DIGITS((MONTH(WORKDATE)+2)/3), 10, 1) || 'Q');
  END
```

- 3 月 20 日締めで指定した日付の年度と期（'yyyy-nH'）を求める関数

```
CREATE FUNCTION YYYYNH0320(INDATE DATE) RETURNS CHAR(7)
  BEGIN
    DECLARE WORKDATE DATE;
    SET WORKDATE=(INDATE -2 MONTHS -20 DAYS);
    RETURN (SUBSTR(CHAR(WORKDATE),1,5) ||
            SUBSTR(DIGITS((MONTH(WORKDATE)+5)/6), 10, 1) || 'H');
  END
```

- 日付間（第一引数−第二引数）の月数を整数で求める関数（日数は切り捨て）

```
CREATE FUNCTION MONTHBETWEEN0(INDATE1 DATE,INDATE2 DATE)
  RETURNS INTEGER
  BEGIN
    DECLARE YMINTERDATE INTERVAL YEAR TO DAY;
    SET YMINTERDATE=INDATE1-INDATE2;
    RETURN (YEAR(YMINTERDATE)*12+MONTH(YMINTERDATE));
  END
```

- 日付間（第一引数−第二引数）の月数を小数点以下まで求める関数（ただし，1 日の月数は小さい方の日付の日を 1 か月の起点とし，大きい方の日付を含む 1 か月の日数分の 1 とします）

```
CREATE FUNCTION MONTHBETWEEN(INDATE1 DATE,INDATE2 DATE)
  RETURNS DECIMAL(29,19)
  BEGIN
    DECLARE INTERDATE INTERVAL YEAR TO DAY;
    DECLARE DMONTHS DEC(29,19);
    DECLARE YYI,MMI INTEGER;
    DECLARE WDATE DATE;
    DECLARE SIGNFLAG DEC(1);
    IF INDATE1>INDATE2 THEN
      SET INTERDATE=INDATE1-INDATE2;
      SET WDATE=INDATE2;
      SET SIGNFLAG=1;
    ELSEIF INDATE1<INDATE2 THEN
      SET INTERDATE=INDATE2-INDATE1;
      SET WDATE=INDATE1;
      SET SIGNFLAG=-1;
    ELSE RETURN 0;
    END IF;
    SET YYI=YEAR(INTERDATE);
```

```
    SET MMI=MONTH(INTERDATE);
    SET WDATE=WDATE+YYI YEARS+MMI MONTHS;
    SET DMONTHS=YYI*12+MMI
        +DEC(DAY(INTERDATE),2)/(DAYS(WDATE+1 MONTH)-DAYS(WDATE));
    IF SIGNFLAG=1 THEN RETURN DMONTHS;
    ELSE RETURN -DMONTHS;
    END IF;
  END
```

- 日付間（第一引数−第二引数）の年数を小数点以下まで求める関数（ただし，1 日の年数は小さい方の日付の月日を 1 か年の起点とし，大きい方の日付を含む 1 か年の日数分の 1 とします）

```
CREATE FUNCTION YEARBETWEEN(INDATE1 DATE,INDATE2 DATE)
  RETURNS DECIMAL(29,19)
  BEGIN
    DECLARE INTERDATE INTERVAL YEAR TO DAY;
    DECLARE DYEARS DEC(29,19);
    DECLARE YYI,MMI INTEGER;
    DECLARE WDATE1,WDATE2 DATE;
    DECLARE SIGNFLAG DEC(1);
    IF INDATE1>INDATE2 THEN
      SET INTERDATE=INDATE1-INDATE2;
      SET WDATE1=INDATE1;
      SET WDATE2=INDATE2;
      SET SIGNFLAG=1;
    ELSEIF INDATE1<INDATE2 THEN
      SET INTERDATE=INDATE2-INDATE1;
      SET WDATE1=INDATE2;
      SET WDATE2=INDATE1;
      SET SIGNFLAG=-1;
    ELSE RETURN  0;
    END IF;
    SET YYI=YEAR(INTERDATE);
    SET WDATE2=WDATE2+YYI YEARS;
    SET DYEARS=YYI
          +DEC(DAYS(WDATE1)-DAYS(WDATE2),3)
          /(DAYS(WDATE2+1 YEAR)-DAYS(WDATE2));
    IF SIGNFLAG=1 THEN RETURN DYEARS;
    ELSE RETURN -DYEARS;
    END IF;
  END
```

(3)　呼び出す関数の決定規則と結果のデータ型

- 認可識別子，ルーチン識別子，引数の数が一致し，引数のデータ型に抽象データ型を含まないで，かつ引数の順序に対応してパラメタのデータ型が完全一致する場合は，この関数を呼び出します。また，この場合の関数の結果のデータ型は，呼び出す関数の RETURNS 句のデータ型になります。
- 認可識別子，ルーチン識別子，引数の数のどれかが一致しない関数の場合は，この関数は呼び出しの対象とはなりません。
- 認可識別子，ルーチン識別子，引数の数は一致しているが，引数のデータ型に抽象データ型を含む場合，又は引数の順序に対応したパラメタのデータ型が完全に一致しない場合は，呼び出す関数は次のように決定します。

- 抽象データ型を含まない場合

  左側の引数から順番に各引数の既定義型を基準として，基準と優先度が同じか又はより優先度が低いデータ型の中で最も優先度の高い既定義型をパラメタに持つ関数を呼び出します。既定義型の優先度を次の表に示します。また，この場合，呼び出す関数が SQL 解析時に一意に決まるので，関数の結果のデータ型は呼び出す関数の RETURNS 句のデータ型となります。

表 4–3　既定義型の優先度

各引数のデータ型	優先度
数データ	SMALLINT→INTEGER→DECIMAL→SMALLFLT→FLOAT
文字データ	CHAR→VARCHAR
各国文字データ	NCHAR→NVARCHAR
混在文字データ	MCHAR→MVARCHAR

（凡例）　A→B：A が B より優先度が高いことを示します。

- 抽象データ型を含む場合

  抽象データ型を含む場合，次の順番で呼び出す関数を決定します。

1. 基本となる関数の決定

   基本となる関数の決定方法は，左側の引数から順番に各引数のデータ型を基準として，基準と優先度が同じか又はより優先度が低いデータ型の中で最も優先度の高いデータ型をパラメタに持つ関数を，基本となる関数とします。データ型が既定義型の場合は，表「既定義型の優先度」の優先度に準じます。データ型が抽象データ型の場合は，次の表に示す優先度に準じます。

表 4–4　抽象データ型の優先度

各引数のデータ型	優先度
抽象データ型	同じデータ型→スーパタイプ※

（凡例）A→B：A が B より優先度が高いことを示します。

注※　抽象データ型定義中の UNDER 句で直接指定するスーパタイプの方が，そのほかのスーパタイプよりも優先度が高くなります。

2. 候補となる関数の決定

   引数が抽象データ型の場合，引数のデータとして取り得る実際の値のデータ型は，引数の定義の抽象データ型と同じデータ型又はサブタイプとなります。そのため，基本となる関数のほかに，引数の抽象データ型と同じデータ型又はサブタイプの抽象データ型を対応するパラメタに持つすべての関数が候補となる関数となります。

   候補となる関数が，基本となる関数一つだけの場合，その関数が呼び出す関数となります。関数の結果のデータ型は，呼び出す関数の RETURNS 句のデータ型になります。

3. RETURNS 句のデータ型を用いた候補となる関数の絞り込み

   基本となる関数の RETURNS 句のデータ型と，基本となる関数以外の候補となる関数の RETURNS 句のデータ型の互換性のチェックをします。RETURNS 句のデータ型が互換性のない関数の場合，候補となる関数ではなくなります。また，互換性のチェックの後，残った候補となる関数の RETURNS

句のデータ型を基に，関数の結果のデータ型を決定します。結果のデータ型及びデータ長は，集合演算（UNION 〔ALL〕，又は EXCEPT 〔ALL〕）の結果のデータ型及びデータ長と同じになります。詳細（問合せ式）については，マニュアル「HiRDB SQL リファレンス」を参照してください。

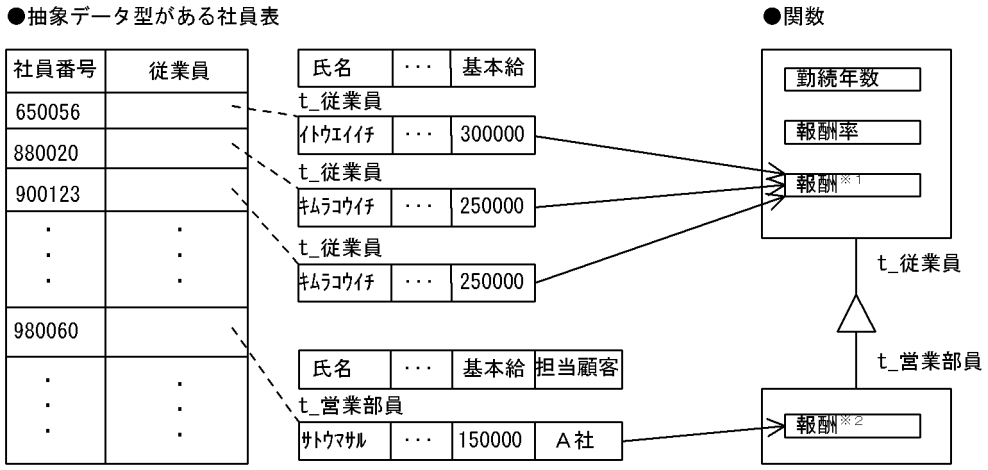
ただし，抽象データ型の場合は，基本となる関数の RETURNS 句の抽象データ型になります。

4. SQL 文実行時の関数の決定

2 及び 3 で関数が一意に決まらない場合，SQL 文実行時に，抽象データ型の引数の実際のデータ型によって，候補となる関数の中から呼び出す関数を一つに決定します。左側の引数より順番に，各引数の実際の値がナル値以外の場合はその値のデータ型を基準として，ナル値の場合はその引数のデータ型を基準とし，その基準のデータ型と同じか又はより優先度が低いデータ型の中で最も優先度の高いデータ型をパラメタとして持つ関数を候補となる関数の中から一つ決定し，呼び出す関数とします。

HiRDB では，関数を多重定義できるため，呼び出す関数の候補が複数ある場合があります。関数の呼び出しの記述と，関数の定義がどのように一致するかによって，呼び出す関数が決定されます。抽象データ型がある表と呼び出す関数の対応を次の図に示します。

図 4–15　抽象データ型がある表と呼び出す関数の対応



[説明]

例えば，社員表に対して抽象データ型関数「報酬」を使用して検索する，次のような SQL 文があるとします。

```
SELECT 社員番号 FROM 社員表 WHERE 報酬(従業員)>=200000
```

この場合，パラメタの値のデータ型が t_従業員か t_営業部員かによって，それぞれのデータ型に対応した関数が決定され，呼び出されます。

なお，この社員表の定義内容については，マニュアル「HiRDB システム導入・設計ガイド」を参照してください。

注※1　報酬＝基本給×報酬率()

注※2　報酬＝顧客総数()×1000 ＋基本給×報酬率()

（例） 抽象データ型を含む場合の呼び出し関数の決定

A，B，C を抽象データ型とし，C をBのスーパタイプ，B を A のスーパタイプとします（抽象データ型の優先度：A→B→C）。

（例 1）

＜前提条件＞

表定義

CREATE TABLE T1(C1 C)

関数定義

f(A)，f(B)，f(C)

SQL 文

SELECT f(C1) FROM T1

＜結果＞

基本となる関数

f(C)

関数呼出しが f(C1)の場合の候補となる関数

f(A),f(B),f(C)

呼び出し関数

SQL 文実行時に呼び出す関数を次に示します。

T1.C1 の実際の値	呼び出し関数
A 型	f(A)
B 型	f(B)
C 型	f(C)
NULL 値	f(C)

（例 2）

＜前提条件＞

表定義

CREATE TABLE T1(C1 C,C2 B)

関数定義

f(A,A)，f(A,B)，f(A,C)，f(B,A)，f(B,C)，f(C,A)，f(C,B)，f(C,C)

SQL 文

SELECT f(C1,C2) FROM T1

＜結果＞

基本となる関数

f(C,B)

関数呼出しが f(C1,C2)の場合の候補となる関数

f(A,A),f(A,B),f(A,C),f(B,A),f(B,C),f(C,A),f(C,B)

呼び出し関数

SQL 文実行時に呼び出す関数を次に示します。

T1.C1 の実際の値	T1.C2 の実際の値	呼び出し関数
A 型	A 型	f(A,A)
	B 型	f(A,B)
	NULL 値	f(A,B)
B 型	A 型	f(B,A)
	B 型	f(B,C)
	NULL 値	f(B,C)
C 型	A 型	f(C,A)
	B 型	f(C,B)
	NULL 値	f(C,B)
NULL 値	A 型	f(C,A)
	B 型	f(C,B)
	NULL 値	f(C,B)

4.3.3　ストアドファンクションを定義，又は削除するときの注意事項

ストアドファンクションを定義，又は削除するときの注意事項について説明します。

(1)　ストアドファンクション定義時

- **ストアドファンクションを作成すると，既存のストアドファンクションが無効になることがあります**
  次の条件を満たす，既存のストアドファンクションが無効になります。
  - 作成するストアドファンクションと同じ名称（認可識別子及びルーチン識別子が同じ）で，かつパラメタ数が同じストアドファンクションを呼び出している

  この場合，ALTER ROUTINE で無効になったストアドファンクションを再作成してください。
- **ストアドファンクションを作成すると，既存のストアドプロシジャが無効になることがあります**

ストアドファンクションを作成すると，既存のストアドプロシジャが無効になることがあります。次に示す条件を満たすストアドプロシジャが無効になります。

- 作成するストアドファンクションと名称（認可識別子及びルーチン識別子），及びパラメタ数が同じストアドファンクションを呼び出しているストアドプロシジャ

この場合，無効になったストアドプロシジャを ALTER PROCEDURE 又は ALTER ROUTINE で再作成してください。

- **ストアドファンクションを作成すると，既存のトリガが無効になることがあります**

  ストアドファンクションを作成すると，既存のトリガが無効になることがあります。次に示す条件を満たすトリガが無効になります。

  - 作成するストアドファンクションと名称（認可識別子及びルーチン識別子），及びパラメタ数が同じストアドファンクションを呼び出しているトリガ

  この場合，無効になったトリガを ALTER TRIGGER 又は ALTER ROUTINE で再作成してください。

- **作成したストアドファンクションが無効になることがあります**

  次の手順でストアドファンクションを作成すると，そのストアドファンクションが無効になることがあります。

  1. プラグインをインストール
  2. 1 のプラグイン関数を呼び出すストアドファンクションを作成
  3. 1 のインストールしたプラグインとは別のプラグインをインストール

  [説明]

  1 と 3 で，インストールするプラグインが提供する関数の，関数名とパラメタ数が同じ場合，3 の操作をすると 2 で作成したストアドファンクションが無効になります。

  この場合，ALTER ROUTINE で無効になったストアドファンクションを再作成してください。

(2) ストアドファンクション削除時

- **ストアドファンクションを削除すると，削除したストアドファンクション以外のストアドファンクションが無効になることがあります**

  次の条件を満たす，そのほかのストアドファンクションが無効になります。

  - 削除するストアドファンクションと同じ名称（認可識別子及びルーチン識別子が同じ）で，かつパラメタ数が同じストアドファンクションを呼び出している

  この場合，ALTER ROUTINE で無効になったストアドファンクションを再作成してください。

- **ストアドファンクションを削除すると，同じ名称のストアドプロシジャが無効になることがあります**

  次の条件を満たすストアドプロシジャが無効になります。

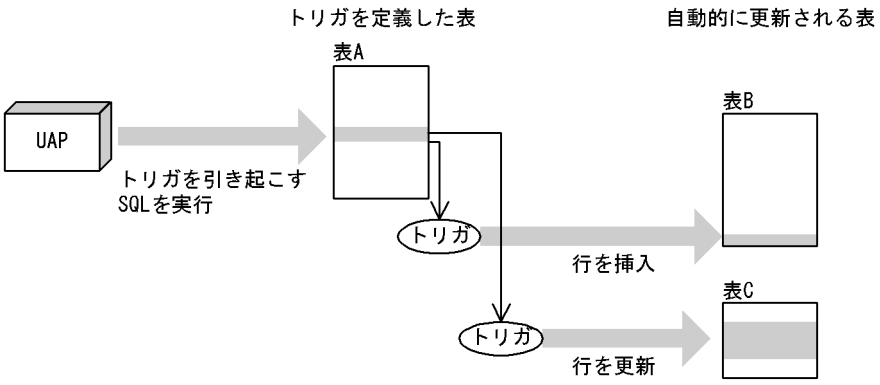
  - 削除するストアドファンクションと同じ名称（認可識別子及びルーチン識別子が同じ）で，かつパラメタ数が同じストアドファンクションを呼び出している

  この場合，ALTER PROCEDURE，又は ALTER ROUTINE で無効になったストアドプロシジャを再作成してください。

4.4　トリガ

トリガを定義すると，ある表への操作（更新，挿入，及び削除）を契機に自動的に SQL 文を実行させることができます。トリガは，定義する表，トリガを動作させる契機となる SQL（**トリガを引き起こす SQL**），自動的に実行させる SQL 文（**トリガ SQL 文**），その動作が実行される条件（**トリガ動作の探索条件**）などを指定して定義します。トリガを定義した表にトリガ動作の探索条件を満たす SQL 文が実行されると，トリガ SQL 文が自動的に実行されます。トリガの概要を次の図に示します。

図 4–16　トリガの概要



[説明]

UAP からトリガを引き起こす SQL が実行されると，トリガを定義した表 A でトリガが呼び出され，トリガ動作の探索条件を満たしている場合，自動的にトリガ SQL 文（この場合，表 B への行の挿入や表 C への行の更新）が実行されます。

トリガを使用すると，次のような操作を UAP で記述する必要がなくなります。

- ある表の更新に伴ってほかの表を必ず更新する
- ある表の更新に伴って，その更新行中のある列を必ず更新する（列と列を関連づける）

例えば，商品管理表の価格が変更されると商品管理履歴表に変更内容を蓄積するという場合，トリガを使用しないと，商品管理表を更新する UAP は常に商品管理履歴表も更新する必要があります。トリガを使用すると，商品管理履歴表への操作を自動化できるため，商品管理表を更新する UAP は商品管理履歴表の更新を考慮する必要がありません。このように，トリガを適切に使用すると UAP を作成するときの負荷を軽減できます。

なお，トリガを定義すると，その表を使用する関数，手続き，及びトリガの SQL オブジェクトは無効になるため，再作成する必要があります。また，トリガが使用している資源（表，インデクスなど）が定義，定義変更，又は削除された場合，トリガの SQL オブジェクトは無効になるため，再作成する必要があります。

トリガの詳細については，マニュアル「HiRDB システム導入・設計ガイド」を参照してください。

4.5 SQL の最適化

HiRDB には，SQL 文の検索効率を良くするための最適化機能があります。

最適化処理には，方式の異なる SQL 最適化モードがあります。SQL 拡張最適化オプションの指定値，及び SQL の構文から，HiRDB が SQL ごとに SQL 最適化モードを決定します。

SQL 最適化モードの種類を次に示します。

- コストベース最適化モード 1（バージョン 06-00 より前の HiRDB で使用している最適化処理方式）
- コストベース最適化モード 2（バージョン 06-00 以降の HiRDB で使用できる最適化処理方式）

さらに，データベースの状態を考慮して，最も効率的なアクセスパスを決定するための最適化の方法を指定できます。最適化の方法には次の三つがあります。

- SQL 最適化指定
- SQL 最適化オプション
- SQL 拡張最適化オプション

SQL 最適化指定：

SQL 最適化指定は，SQL 文中に指定できます。指定した SQL に対して，最適化が適用されます。

SQL 最適化指定については，マニュアル「HiRDB SQL リファレンス」を参照してください。

SQL 最適化オプション，SQL 拡張最適化オプション：

SQL 最適化オプション，SQL 拡張最適化オプションには，それぞれ複数の機能が割り当てられていて，その中から必要とする機能を選択できます。SQL 最適化オプションで指定する機能は，コストベース最適化モード 1 及びコストベース最適化モード 2 の両方で有効となります。SQL 拡張最適化オプションで指定する機能は，コストベース最適化モード 2 の場合だけ有効となります。

SQL 最適化オプション，SQL 拡張最適化オプションについては，「クライアント環境定義の設定内容」を参照してください。

注意事項：

SQL 最適化モードを選択する場合の指標を次に示します。

＜バージョン 06-00 以降の HiRDB を初めて導入する場合＞

- コストベース最適化モード 2 を使用することをお勧めします。
- コストベース最適化モード 2 を使用する場合は，最適化の精度を更に向上させるため，必要に応じて最適化情報収集ユティリティを実行してください。最適化情報収集ユティリティの実行要否については，マニュアル「HiRDB コマンドリファレンス」を参照してください。
- SQL 最適化オプション，SQL 拡張最適化オプションには，指定した方がよい推奨値があります。この推奨値は必ず指定するようにして，更にそれ以外に使用できる機能がないか検討してください。

＜バージョン 06-00 より前の HiRDB をバージョンアップする場合＞

バージョンアップ前と同じ状態で使用するために，コストベース最適化モード 1 を使用することをお勧めします。ただし，SQL 文によっては，常にコストベース最適化モード 2 を使用する場合があるため，既に構築されている環境で新しい運用を始めるときには，SQL 拡張最適化オプションの指定値についても検討してください。また，SQL 最適化オプションの指定値は変更しないでください。

4.5.1　SQL 最適化モード

(1)　SQL 最適化モードの特徴

SQL 最適化モードの特徴を次の表に示します。

表 4–5　SQL 最適化モードの特徴

SQL 最適化モード	説明	長所	短所	選択方法
コストベース最適化モード 1	バージョン 06-00 より前の HiRDB のコストベース最適化処理方式です。バージョン 06-00 以降の HiRDB でも使用できます。	バージョン 06-00 より前の HiRDB からバージョンアップしても，バージョンアップ前と同じアクセスパスで検索できます。 なお，高速に検索する目的として，アクセスパスを変更することもあります。	候補とするアクセスパスの種類が少ないため（ハッシュジョインなどの機能を候補として選択しません），必ずしも最適なアクセスパスが選択されるとは限りません。	SQL 拡張最適化オプションに"NONE"又は 0 を指定してください。 なお，SQL 文によっては，常にコストベース最適化モード 2 を使用する場合があります。詳細については，「強制的にコストベース最適化モード 2 を適用する SQL」を参照してください。
コストベース最適化モード 2	バージョン 06-00 以降の HiRDB で，高速に検索できるようにしたコストベース最適化処理方式です。	結合検索，副問合せ処理に対して，ハッシングを組み合わせたアクセスパスを候補として選択するため，高速に検索できます。	複雑な最適化処理をするため，最適化処理に時間が掛かります。	SQL 拡張最適化オプションにコストベース最適化モード 2 の適用を指定するか，又は SQL 拡張最適化オプションを省略してください。

(2)　強制的にコストベース最適化モード 2 を適用する SQL

コストベース最適化モード 1 を使用していても，コストベース最適化モード 2 が強制的に適用される場合があります。強制的にコストベース最適化モード 2 が適用される SQL を次に示します。

- UPDATE 文の SET 句での副問合せ
- OUTER JOIN，INNER JOIN，又は CROSS JOIN
- 集合演算結果の COUNT(*)
- DISTINCT 集合関数の値式
- ビュー表，WITH 句の問合せ名の外結合（OUTER JOIN）への指定

- BLOB データ，BINARY データの部分的な更新・検索
- SQL 最適化指定
- 定義長が 255 バイトを超える値式のソート
- 先頭から n 行の検索結果を取得する機能
- BINARY 型を使用した検索
- 内部導出表が 2 段以上入れ子になるビュー表，WITH 句の検索
- マトリクス分割
- 結合表に対する副問合せ
- 繰返し列での集合関数 MIN，MAX 適用
- 行値構成子
- CASE 式中の副問合せ
- 値式 2 が BLOB 型の POSITION 関数
- 参照制約
- 検査制約
- 定義長 256 バイト以上のデータの制限解除
- 更新，削除，又は追加をする表の副問合せへの指定
- FROM 句での繰返し列の平坦化機能
- LIMIT 句
- 内部導出表が 2 段以上入れ子になる検索
- 問合せ式本体の指定箇所拡大
- ウィンドウ関数
- SIMILAR 述語
- XML 型を使用した検索
- 文字集合
- RD エリア名を指定した検索，更新，又は削除
- 圧縮列へのアクセスを含む操作系 SQL
- 一時表に対して操作をする SQL
- 列追加定義（ALTER TABLE 文）で ON ROW EXISTS 指定の列を追加した表の検索

それぞれの SQL に該当する適用条件及び例を次に示します。

(a)　UPDATE 文の SET 句での副問合せ

- UPDATE 文の SET 句に，スカラ副問合せ又は行副問合せを指定した場合

例：

```
UPDATE T1 SET(C1,C2)=(SELECT MAX(C1),MAX(C2) FROM T2) WHERE C3=1
```

注　下線部分が該当箇所です。

(b)　OUTER JOIN，INNER JOIN，又は CROSS JOIN

- FROM 句に，[INNER] JOIN を指定した場合

  例：

```
SELECT T1.C1,T2.C2 FROM T1 INNER JOIN T2 ON T1.C1=T2.C1
```

注　下線部分が該当箇所です。

- FROM 句に，LEFT [OUTER] JOIN を含む表参照と任意の表参照をコンマで区切り複数指定した場合

  例：

```
SELECT T1.C1,T2.C2 FROM T1 LEFT OUTER JOIN T2 ON T1.C1=T2.C1,
  T3 WHERE T1.C1=T3.C1
```

注　下線部分が該当箇所です。

- FROM 句に，表参照 1 LEFT [OUTER] JOIN 表参照 2 を指定し，更に表参照 2 に LEFT [OUTER] JOIN をネストして指定した場合

  例：

```
SELECT T1.C1,T2.C2,T3.C2 FROM T1 LEFT OUTER JOIN
  (T2 LEFT OUTER JOIN T3 ON T2.C1=T3.C1)
  ON T1.C1=T3.C1
```

注　下線部分が該当箇所です。

- FROM 句に RIGHT [OUTER] JOIN を指定した場合

  例：

```
SELECT T1.C1,T2.C2 FROM T1 RIGHT OUTER JOIN T2 ON T1.C1=T2.C1
```

注　下線部分が該当箇所です。

- FROM 句に CROSS JOIN を指定した場合

  例：

```
SELECT T1.C1,T2.C2 FROM T1 CROSS JOIN T2
```

注　下線部分が該当箇所です。

(c)　集合演算結果の COUNT(*)

- FROM 句に指定した問合せ式本体が集合演算を含む場合

  例：

```
SELECT COUNT(*) FROM (SELECT C1 FROM T1 UNION SELECT C1 FROM T2)
```

注　下線部分が該当箇所です。

(d)　DISTINCT 集合関数の値式

- DISTINCT 集合関数（COUNT，SUM，又は AVG）の引数に，列指定を除く値式を指定した場合

  例：

```
SELECT AVG(DISTINCT C1+C2) FROM T1
```

注　下線部分が該当箇所です。

(e)　ビュー表，WITH 句の問合せ名の外結合（OUTER JOIN）への指定

- FROM 句にビュー表又は WITH 句の問合せ名に対する LEFT [OUTER] JOIN を指定し，そのビュー表又は WITH 句の問合せ名から内部導出表が作成される場合

  例：

```
WITH W1(C1,C2) AS (SELECT C1,COUNT(*) FROM T1 GROUP BY C1)
  SELECT W1.C1,W1.C2,T2.C2 FROM W1 LEFT JOIN T2 ON W1.C1=T2.C1
```

注　下線部分が該当箇所です。

(f)　BLOB データ，BINARY データの部分的な更新・検索

- スカラ関数 SUBSTR の値式 1 に，BLOB 型を指定した場合

  例：

```
SELECT SUBSTR(C1,1,500) FROM T1
```

注　下線部分が該当箇所です。C1がBLOB型の列です。

- UPDATE 文の更新対象が BLOB 型の列で，更新値に連結演算を指定した場合

  例：

```
UPDATE T1 SET C1=C1||?
```

注　下線部分が該当箇所です。C1がBLOB型の列です。

- UPDATE 文の更新対象が BLOB 型の列又は BLOB 属性で，更新値に列指定又はコンポネント指定を指定した場合

  例：

```
UPDATE T1 SET C1=C2
```

注　下線部分が該当箇所です。C1，C2がBLOB型の列です。

(g) SQL 最適化指定

- 使用インデクスの SQL 最適化指定を指定した場合

  例：

```
SELECT T1.C1 FROM T1 WITH INDEX(idx1) WHERE T1.C2<=500
```

  注　下線部分が該当箇所です。

- 結合方式の SQL 最適化指定を指定した場合

  例：

```
SELECT T1.C1,T2.C2 FROM T1 INNER JOIN BY NEST T2 ON T1.C1=T2.C1
```

  注　下線部分が該当箇所です。

- 副問合せ実行方式の SQL 最適化指定を指定した場合

  例：

```
SELECT T1.C1 FROM T1 WHERE T1.C1=ANY
  (HASH SELECT T2.C1 FROM T2 WHERE T2.C2='302S')
```

  注　下線部分が該当箇所です。

(h) 定義長が 255 バイトを超える値式のソート

- ORDER BY 句のソートのキーになる項目に，定義長が 256 バイト以上の CHAR，VARCHAR，MCHAR，及び MVARCHAR，並びに 128 文字以上の NCHAR 及び NVARCHAR を指定した場合

  例１：

```
SELECT C1,C2 FROM T1 ORDER BY C2
```

  注　下線部分が該当箇所です。C2はVARCHAR(300)の列です。

  例２：

```
SELECT C1,C3||C4 FROM T1 ORDER BY 2
```

  注　下線部分が該当箇所です。C3||C4はNCHAR(150)の値式です。

(i) 先頭から n 行の検索結果を取得する機能

- ORDER BY 句の直後に LIMIT 句を指定する場合

  例：

```
SELECT SCODE,ZSURYO FROM ZAIKO WHERE  ZSURYO>20 ORDER BY 2,1 LIMIT 10
```

  注　下線部分が該当箇所です。

(j) BINARY 型を使用した検索

- BINARY 型の列を検索する場合

例：

```
SELECT C1 FROM T1

注　下線部分が該当箇所です。C1はBINARY型の列です。
```

(k)　内部導出表が 2 段以上入れ子になるビュー表，WITH 句の検索

- FROM 句にビュー表又は WITH 句の問合せ名を指定した問合せ指定があり，更にこのビュー表定義中又は WITH 句中の導出問合せ式の FROM 句に，内部導出表となるビュー表又は WITH 句を指定している場合

  例：

```
WITH Q1(QC1,QC2) AS (SELECT C1,C2 FROM V1 GROUP BY C1,C2)

SELECT AVG(QC1),QC2 FROM Q1 GROUP BY QC2

注　下線部分が該当箇所です。V1は内部導出表となるビュー表です。
```

(l)　マトリクス分割

- マトリクス分割表に対して，検索，更新，削除，及びリストの操作をする場合

  例：

```
SELECT * FROM T1

注　下線部分が該当箇所です。T1はマトリクス分割表です。
```

(m)　結合表に対する副問合せ

- 結合表を含む問合せ指定を指定し，FROM 句の ON 探索条件，WHERE 句，又は HAVING 句に副問合せを指定する場合

  例：

```
SELECT * FROM T1 LEFT JOIN T2 ON T1.C1=T2.C1
  WHERE T1.C1=ANY(SELECT C1 FROM T3)

注　下線部分が該当箇所です。
```

(n)　繰返し列での集合関数 MIN，MAX 適用

- 集合関数の MIN 又は MAX に，FLAT 指定の繰返し列を指定する場合

  例：

```
SELECT MIN(FLAT(C1)) FROM T1

注　下線部分が該当箇所です。C1は繰返し列です。
```

(o)　行値構成子

- 行値構成子を指定する場合

例：

```
SELECT * FROM T1 WHERE (C1,C2,C3)>(1,2,3)
```

注　下線部分が該当箇所です。

(p)　CASE 式中の副問合せ

- CASE 式中に副問合せを指定する場合

  例：

```
SELECT CASE(SELECT C1 FROM T1) WHEN 1 THEN C2 ELSE C1 END FROM T1
```

注　下線部分が該当箇所です。

(q)　値式 2 が BLOB 型のスカラ関数 POSITION

- スカラ関数 POSITION の値式 2 に BLOB 型を指定する場合

  例：

```
SELECT POSITION(? AS BLOB(1K) IN C1) FROM T1
```

注　下線部分が該当箇所です。C1はBLOB型の列です。

(r)　参照制約

- 被参照表又は参照表に対して，挿入，更新，及び削除をする場合

  例：

```
UPDATE T1 SET C1=?
```

注　下線部分が該当箇所です。T1は被参照表又は参照表です。

(s)　検査制約

- 検査制約を定義した列に対して，挿入，及び更新をする場合

  例：

```
INSERT INTO T1(C1,C2) VALUES(?,?)
```

注　下線部分が該当箇所です。C1は検査制約を定義した列です。

(t)　定義長 256 バイト以上のデータの制限解除

- GROUP BY 句に次の値式を指定する場合
  - 定義長が 256 バイト以上の CHAR，VARCHAR，MCHAR，又は MVARCHAR 型
  - 128 文字以上の NCHAR，又は NVARCHAR 型
  - 256 バイト以上の BINARY 型

例：

```
SELECT C1,COUNT(*) FROM T1 GROUP BY C1
```

注　下線部分が該当箇所です。T1.C1が256バイト以上の文字列です。

- 集合関数の引数に次の値式を指定する場合
  - 定義長が 256 バイト以上の CHAR，VARCHAR，MCHAR，又は MVARCHAR 型
  - 128 文字以上の NCHAR，又は NVARCHAR 型
  - 256 バイト以上の BINARY 型

  例：

```
SELECT MIN(C1) FROM T1
```

注　下線部分が該当箇所です。T1.C1が256バイト以上の文字列です。

- ビュー表，WITH 句，又は FROM 句に問合せ式本体を指定し，内部導出表を作成する場合で，内部導出表の選択式に次の値式を指定するとき
  - 256 バイト以上の CHAR，VARCHAR，MCHAR，又は MVARCHAR 型
  - 128 文字以上の NCHAR，又は NVARCHAR 型
  - 256 バイト以上の BINARY 型

  例：

```
WITH W1(C1,C2) AS (SELECT DISTINCT C1,C2 FROM T1)
    SELECT C2,COUNT(*) FROM W1 GROUP BY C2
```

注　下線部分が該当箇所です。T1.C1が256バイト以上の文字列です。

(u)　更新，削除，又は追加をする表の副問合せへの指定

- 更新，削除，又は追加をする表を副問合せに指定する場合

```
例1：

UPDATE T1 SET C1=NULL WHERE C1=(SELECT MIN(C1) FROM T1)

例2：

DELETE FROM T1 WHERE C1=(SELECT MIN(C1) FROM T1)

例3：

INSERT INTO T1(C1,C2) VALUES((SELECT MIN(C1) FROM T1),NULL)
```

注　下線部分が該当箇所です。

- INSERT 文の問合せ式本体に，追加する表と同一表を指定する場合

  例：

```
INSERT INTO T1(C1,C2) SELECT C1,C2+1 FROM T1
```

注　下線部分が該当箇所です。

(v)　FROM 句での繰返し列の平坦化機能

- FROM 句に FLAT を指定する場合

  例：

```
SELECT C1,C2 FROM T1(FLAT(C1,C2)) WHERE C1<10 AND C2 >20
```

注　下線部分が該当箇所です。C1，C2は繰返し列です。

(w)　LIMIT 句

- LIMIT 句を指定する場合

  例：

```
SELECT SCODE, ZSURYO FROM ZAIKO WHERE ZSURYO > 20ORDER BY 2, 1 LIMIT 20, 10
```

注　下線部分が該当箇所です。

(x)　内部導出表が 2 段以上入れ子になる検索

- 内部導出表を作成する問合せ指定の FROM 句に，更に内部導出表となる問合せ指定を指定している場合

  例：

```
SELECT AVG(QC1),QC2 FROM(SELECT C1,C2 FROM V1 GROUP BY C1,C2) AS Q1(QC1,QC2)
```

注　下線部分が該当箇所です。V1は内部導出表となるビュー表です。

(y)　問合せ式本体の指定箇所拡大

- ビュー表，WITH 句，又は FROM 句に集合演算を指定し，この問合せが内部導出表を作成する場合

  例：

```
WITH V1(C1,C2) AS (SELECT C1,C2 FROM T1 UNION SELECT C1,C2 FROM T2)
    SELECT C1 FROM V1 WHERE C2>0
```

注　下線部分が該当箇所です。

- INSERT 文に集合演算を指定する場合

  例：

```
INSERT INTO T3 (C1,C2)
    SELECT C1,C2 FROM T1 UNION ALL SELECT C1,C2 FROM T2
```

注　下線部分が該当箇所です。

- 副問合せに集合演算を指定する場合

例：

```
SELECT C1, C2 FROM T3
    WHERE EXISTS(SELECT C1 FROM T1 EXCEPT SELECT C1 FROM T2)

注　下線部分が該当箇所です。
```

(z)　ウィンドウ関数

- 選択式の中にウィンドウ関数を含む場合

  例：

```
SELECT C1,C2,COUNT(*) OVER() FROM T1

注　下線部分が該当箇所です。
```

(aa)　SIMILAR 述語

- SIMILAR 述語を指定する場合

  例：

```
SELECT C1 FROM T1 WHERE C2 SIMILAR TO '%(b|g)%'

注　下線部分が該当箇所です。
```

(ab)　XML 型を使用した検索

- XML 型を使用した検索を行った場合

  例：

```
SELECT C1 FROM T1
    WHERE XMLEXISTS('/書籍情報[価格=1000]'
                    PASSING BY VALUE C2)

注　下線部分が該当箇所です。T1.C2がXML型の列です。
```

(ac)　文字集合

- SQL 中に文字集合を指定した列を含む場合

  例：

```
SELECT C1, C2 FROM T1 WHERE C1='HiRDB'

注　下線部分が該当箇所です。T1.C1が文字集合を指定した列です。
```

(ad)　RD エリア名を指定した検索，更新，又は削除

- SQL 中にアクセス先の RD エリア名を指定した場合

  例：

```
SELECT C1 FROM T1 IN ( 'RU01,RU02' ) WHERE C1='HiRDB'
```

注　下線部分が該当箇所です。RU01，RU02がアクセス先のRDエリアです。

(ae)　圧縮列へのアクセスを含む操作系 SQL

- SQL 中に圧縮列を含む場合

  例：

```
SELECT C1,C2 FROM T1 WHERE C2<?
```

注　下線部分が該当箇所です。T1.C1が圧縮列です。

(af)　一時表に対して操作をする SQL

- 一時表に対して操作する場合

  例：

```
SELECT C1 FROM TTMP1
```

注　下線部分が該当箇所です。TTMP1が一時表です。

(ag)　列追加定義（ALTER TABLE 文）で ON ROW EXISTS 指定の列を追加した表を含む操作系 SQL

- 操作系 SQL 中に列追加定義（ALTER TABLE 文）で ON ROW EXISTS 指定の列を追加した表を含む場合

  例：

```
CREATE TABLE T1(C1 INT)
ALTER TABLE T1 ADD C2 INT DEFAULT 2 ON ROW EXISTS
SELECT C1 FROM T1
```

注　下線部分が該当箇所です。T1が列追加定義(ALTER TABLE文)でON ROW EXISTS指定の列を追加した表です。

(3)　SQL 最適化オプション，SQL 拡張最適化オプションの有効範囲

SQL 最適化オプション，SQL 拡張最適化オプションが有効となる SQL 最適化モードを次の表に示します。

表 4–6　SQL 最適化オプション，SQL 拡張最適化オプションが有効となる SQL 最適化モード

SQL 最適化モード	SQL 最適化オプション	SQL 拡張最適化オプション
コストベース最適化モード 1	○	×
コストベース最適化モード 2	○	○

（凡例）

○：有効となります。

×：無効となります。

(4) 最適化処理で選択された SQL 最適化モードを確認する方法

アクセスパス表示ユティリティを使用すると，SQL 文ごとに最適化処理で選択された SQL 最適化モードを確認できます。アクセスパス表示ユティリティについては，マニュアル「HiRDB コマンドリファレンス」を参照してください。

(5) 注意事項

1. SQL 最適化モードを変更すると，アクセスパスが変更になるため，SQL 文の検索性能が低下することがあります。本番運用などで十分に性能評価ができない環境では，SQL 最適化モードを変更しないことをお勧めします。
2. HiRDB を新規導入する場合，コストベース最適化モード 2 を使用することをお勧めします。また，ほかの SQL 拡張最適化オプションを使用する場合は，コストベース最適化モード 2 に追加する形で使用してください。コストベース最適化モード 2 を使用すると，最適化処理で選択できるアクセスパスの種類が多いため，より高速に検索できるアクセスパスを選択できます。

   なお，HiRDB システム定義の pd_additional_optimize_level オペランドの省略値はコストベース最適化モード 2 が含まれるため，通常はコストベース最適化モード 2 が適用されます。
3. バージョン 06-00 より前の HiRDB からバージョンアップする場合，バージョンアップ前と同じ状態で使用するためにコストベース最適化モード 1 をそのまま使用することをお勧めします。ただし，SQL 文によっては，常にコストベース最適化モード 2 を使用することがあります。
4. 通常は絞り込み条件を考慮して最適化をしますが，SQL 拡張最適化オプションにハッシュジョイン，副問合せのハッシュ実行を適用した場合，絞り込み条件がなかったり，絞り込み条件で行数があまり絞り込めなかったりすると，行数の多い表を内表にしたハッシュジョインを適用したり，行数の多い表の転送が発生したりします。このような場合は，表の行数の情報を最適化に反映させるために，必要に応じて次のどちらかの方法で最適化情報収集ユティリティを実行してください。最適化情報収集ユティリティの実行要否については，マニュアル「HiRDB コマンドリファレンス」を参照し，性能について十分に検証するようにしてください。
   - 表にデータを格納した状態で，最適化情報収集レベルを lvl1 にして（-c オプションに lvl1 を指定して）実行します。lvl1 では，表の行数の情報だけを取得するため，比較的短時間で最適化情報収集ユティリティを実行できます。また，-t オプションに ALL を指定すると，スキーマ内のすべての表に対して行数を取得できます。
   - 表にデータを格納できない場合や，テスト環境の場合は，本番環境での表の行数（NROWS）を最適化パラメタファイルに記述して，表ごとに-s オプションを指定して実行します。表の行数を 1,000 行にする場合の最適化パラメタファイルの記述例を次に示します。

     # 表最適化情報

     NROWS 1000 # 表の全行数
5. コストベース最適化モード 1 を使用する場合，通常は最適化情報収集ユティリティを実行する必要はありませんが，実行するときは最適化情報収集レベルは lvl1 にしないでください。

4.5.2　最適化方法の種類

SQL 最適化指定，SQL 最適化オプション，及び SQL 拡張最適化オプションの最適化方法の種類について説明します。

(1)　SQL 最適化指定

SQL 最適化指定には，次の最適化方法があります。

- 使用インデクスの SQL 最適化指定
- 結合方式の SQL 最適化指定
- 副問合せ実行方式の SQL 最適化指定

(2)　SQL 最適化オプション

SQL 最適化オプションには，次の最適化方法があります。

1. ネストループジョイン強制
2. 複数の SQL オブジェクト作成
3. フロータブルサーバ対象拡大（データ取り出しバックエンドサーバ）
4. ネストループジョイン優先
5. フロータブルサーバ候補数の拡大
6. OR の複数インデクス利用優先
7. 自バックエンドサーバでのグループ化，ORDER BY，DISTINCT 集合関数処理
8. AND の複数インデクス利用の抑止
9. グループ分け高速化処理
10. フロータブルサーバ対象限定（データ取り出しバックエンドサーバ）
11. データ収集用サーバの分離機能
12. インデクス利用の抑止
13. 複数インデクス利用強制
14. 更新 SQL の作業表作成抑止
15. 探索高速化条件の導出
16. スカラ演算を含むキー条件の適用
17. プラグイン提供関数からの一括取得機能
18. 導出表の条件繰り込み機能

(3)　SQL 拡張最適化オプション

SQL 拡張最適化オプションには，次の最適化方法があります。

1. コストベース最適化モード 2 の適用
2. ハッシュジョイン，副問合せのハッシュ実行
3. 値式に対する結合条件適用機能
4. スカラ演算を含む条件に対するサーチ条件適用
5. パラメタを含む XMLEXISTS 述語への部分構造インデクスの有効化
6. FROM 句の導出表のマージ適用
7. 外結合内結合変換機能

4.5.3　SQL の最適化の指定方法

(1)　指定できる箇所

(a)　SQL 最適化指定

SQL 最適化指定は，次の SQL に指定できます。

- 副問合せ
- 表式
- DELETE 文
- UPDATE 文

(b)　SQL 最適化オプション，SQL 拡張最適化オプション

SQL 最適化オプション，及び SQL 拡張最適化オプションは，次の箇所で指定できます。なお，通常はシステム共通定義で指定してください（すべての SQL に対して有効となります）。

- システム共通定義の pd_optimize_level，pd_additional_optimize_level オペランド
- フロントエンドサーバ定義の pd_optimize_level，pd_additional_optimize_level オペランド
- クライアント環境定義の PDSQLOPTLVL，PDADDITIONALOPTLVL
- SQL コンパイルオプション（CREATE PROCEDURE と CREATE TYPE の各 SQL に指定した手続き本体，及び ALTER PROCEDURE，ALTER ROUTINE，CREATE TRIGGER，並びに ALTER TRIGGER）

(2) 優先順位

SQL 最適化オプション，及び SQL 拡張最適化オプションを複数の箇所に指定した場合の優先順位について説明します。なお，SQL 文中に SQL 最適化指定を指定している場合は，SQL 最適化オプション及び SQL 拡張最適化オプションより SQL 最適化指定が優先されます。

(a) ストアドルーチン中以外，及びトリガ中以外の操作系 SQL

優先順位は次のようになります。

1. クライアント環境定義の PDSQLOPTLVL，PDADDITIONALOPTLVL
2. フロントエンドサーバ定義の pd_optimize_level，pd_additional_optimize_level オペランド
3. システム共通定義の pd_optimize_level，pd_additional_optimize_level オペランド

(b) ストアドルーチン中，及びトリガ中の操作系 SQL

優先順位は次のようになります。

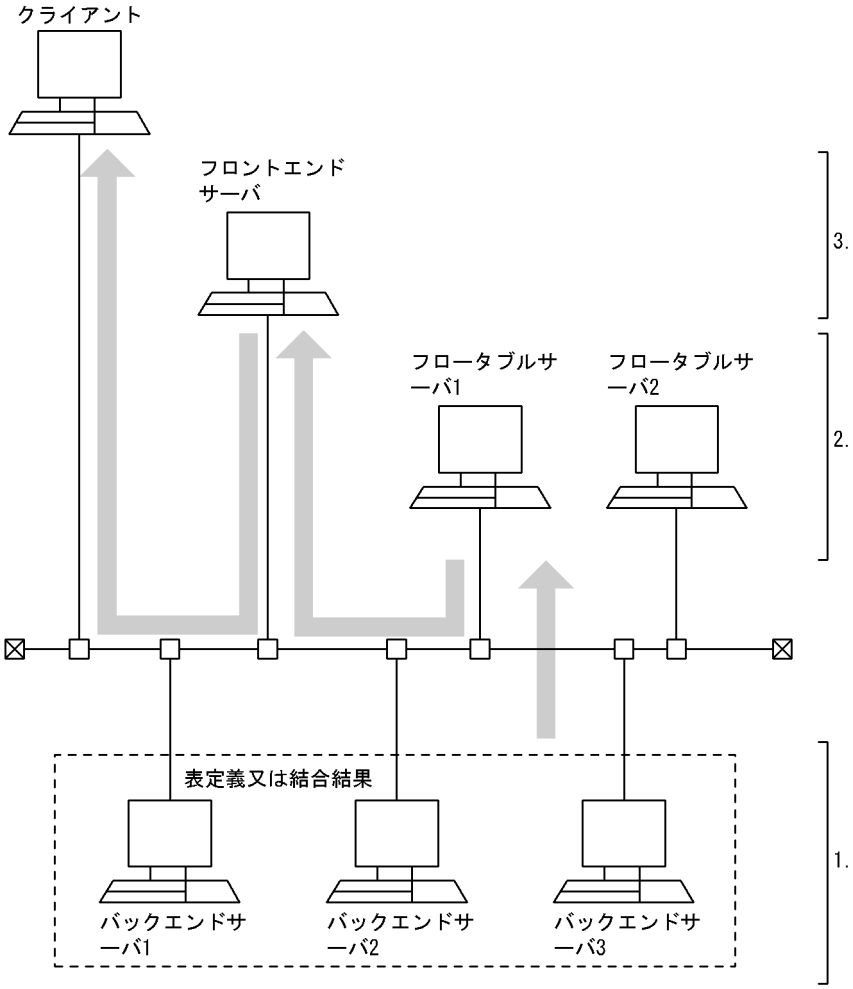
1. SQL コンパイルオプション（ALTER PROCEDURE，ALTER ROUTINE，ALTER TRIGGER，CREATE PROCEDURE，及び CREATE TRIGGER，並びに CREATE TYPE の手続き本体）
2. フロントエンドサーバ定義の pd_optimize_level，pd_additional_optimize_level オペランド
3. システム共通定義の pd_optimize_level，pd_additional_optimize_level オペランド

4.5.4 フロータブルサーバの割り当て方法（HiRDB/パラレルサーバ限定）

(1) HiRDB での問合せ処理方式

HiRDB/パラレルサーバでの SQL 文の問合せ処理は，主に 3 ステップに分けて実行しています。HiRDB/パラレルサーバでの SQL 文の問合せ処理方式を次の図に示します。

図 4–17　HiRDB/パラレルサーバでの SQL 文の問合せ処理方式



[説明]

1. バックエンドサーバでデータを取り出します。2 表以上の問合せの場合，結合方式によってはバックエンドサーバ間でデータの通信が実行され，ステップ 1.が更に数段階になるときがあります。
2. フロータブルサーバでグループ化，ソート，重複排除，集合演算処理などをします。処理方式によっては，フロータブルサーバを使用しない場合や，フロータブルサーバ間でデータの通信が実行され，ステップ 2.が更に数段階になるときがあります。
3. フロントエンドサーバで問合せ結果を集めて，クライアントに転送します。

HiRDB では，ステップ 2 で使用するフロータブルサーバは，SQL ごとにその SQL でアクセスしないバックエンドサーバを自動的に割り当てます。ただし，SQL 最適化オプションを指定した場合には，フロータブルサーバの割り当て方法を変更できます。

フロータブルサーバを使用する SQL と，フロータブルサーバの割り当てに関する最適化を次に示します。なお，これらの最適化の特徴については，「フロータブルサーバの割り当てに関する最適化の特徴」を参照してください。

- フロータブルサーバ対象拡大（データ取り出しバックエンドサーバ）

- フロータブルサーバ対象限定（データ取り出しバックエンドサーバ）
- データ収集用サーバの分離機能

また，次の最適化を適用すると，フロータブルサーバの割り当て台数を最大値まで増やせます。なお，この最適化の特徴については，「フロータブルサーバの割り当て候補数に関する最適化の特徴」を参照してください。

- フロータブルサーバ候補数の拡大

(2)　フロータブルサーバを使用する SQL

次のどれかの条件に当てはまる場合，SQL 文の問合せ処理でフロータブルサーバを使用します。

- UNION [ALL]句又は EXCEPT [ALL]句を指定して検索する場合
- SELECT 文で次に示す指定をする場合
  - 複数の表を結合して検索するとき
  - インデクスを定義していない列に対して ORDER BY 句を指定するとき
  - 横分割表に対して ORDER BY 句を指定するとき
  - 選択式中に集合関数を含む値式を指定しているとき
  - 選択式中にウィンドウ関数 COUNT(*) OVER()を含む値式を指定しているとき
  - GROUP BY 句を指定するとき
  - DISTINCT 句を指定するとき
  - FOR UPDATE 句を指定するか又はこのカーソルを使用した更新があり，インデクスを定義した列に検索条件を指定するとき
  - FOR READ ONLY 句を指定するとき
  - 限定述語の副問合せを指定するとき
- INSERT 文の挿入元の問合せ本体に，次に示す指定をする場合
  - 外への参照がある副問合せに更新表を指定するとき
  - 挿入元の問合せ式本体の主問合せに更新表を指定するとき
- UPDATE 文に次に示す指定をする場合
  - 探索条件中又は更新値中に，外への参照がある副問合せを指定し，その副問合せ中に更新表を指定するとき
  - 探索条件中に限定述語の副問合せを指定するとき
- DELETE 文に次に示す指定をする場合
  - 探索条件中の外への参照がある副問合せに，更新表を指定するとき
  - 探索条件中に限定述語の副問合せを指定するとき

(3)　フロータブルサーバの割り当てに関する最適化の特徴

フロータブルサーバの割り当てに関する最適化の特徴を次の表に示します。

表 4–7　フロータブルサーバの割り当てに関する最適化の特徴

最適化方法	長　所	短　所
省略した場合	データ取り出しバックエンドサーバに，ソートなどの負荷が掛かる処理を割り当てないため，同時に同じバックエンドサーバからデータを取り出す SQL 文を実行した場合，高速に検索できます。	データ取り出し以外のバックエンドサーバをフロータブルサーバとして割り当てるため，通信負荷が高くなります。
フロータブルサーバ対象拡大（データ取り出しバックエンドサーバ）	「フロータブル候補数の拡大」と組み合わせると，フロータブルサーバをすべてのバックエンドサーバに割り当てるため，フロータブルサーバでのソートなどの並列処理の効果が高くなります。	複数の SQL 文を同時に実行する場合，同一フロータブルサーバに複数の処理が割り当てられるため，同時実行性が損なわれます。また，通信負荷も高くなります。
フロータブルサーバ対象限定（データ取り出しバックエンドサーバ）	検索に使用する表が定義されているバックエンドサーバにだけ，フロータブルサーバを割り当てるため，表定義によって使用するバックエンドサーバを使い分けることができます。	分割数が少ない表に大量のデータが格納されている場合は，使用できるフロータブルサーバが少なくなるため，すべてのバックエンドサーバを有効に使用できなくなります。
データ収集用サーバの分離機能	データ収集用サーバにデータを送る場合，同じサーバと別サーバから同時にデータが転送されると，同じサーバからの転送を優先します。これによって，別サーバの処理は後になってしまいます。データ収集用サーバの分離機能を適用すると，すべてのサーバを別サーバとして扱うことができるので，すべてのサーバから均等にデータを受け取れるようになります。	1SQL 文中に集合演算，副問合せを伴った検索など，問合せが複数個ある場合は，常に同じフロータブルサーバが使用されるため，同時実行性が損なわれます。

(4)　フロータブルサーバの割り当て候補数に関する最適化の特徴

フロータブルサーバの割り当て候補数に関する最適化の特徴を次の表に示します。

表 4–8　フロータブルサーバの割り当て候補数に関する最適化の特徴

最適化方法	長　所	短　所
省略した場合	データ件数が多い検索では，フロータブルサーバの割り当て台数を多くします。データ件数が少ない検索では，フロータブルサーバの割り当て台数を少なくします。	探索条件に=や BETWEEN などの絞り込める述語を指定している場合，HiRDB はデータ件数が少ないと判断します。実際に=や BETWEEN などであまり絞り込めない場合でも，フロータブルサーバの割り当て台数を少なくするため，サーバの処理負荷が高くなります。

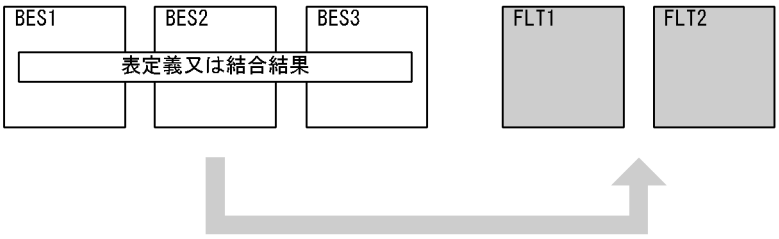
最適化方法	長　所	短　所
フロータブルサーバ候補数の拡大	データ件数が多い検索では，すべてのフロータブルサーバを使用して効率的に検索できます。	データ件数が少ない検索でもすべてのフロータブルサーバを使用するため，SQL 文の同時実行性が損なわれます。また，サーバ間の通信経路が複雑になるため，表の分割数が多い場合には，通信負荷も高くなります。

(5)　各最適化でのフロータブルサーバの割り当て方法

(a)　最適化方法を省略した場合

最適化方法を省略した場合のフロータブルサーバの割り当てを次の図に示します。

図 4–18　最適化方法を省略した場合のフロータブルサーバの割り当て



（凡例）
BES：バックエンドサーバ
FLT：フロータブルサーバ
：候補となるフロータブルサーバ

[説明]

「フロータブルサーバ候補数の拡大」を指定しない場合は，FLT1 及び FLT2 から，HiRDB がフロータブルサーバとして必要と判断した台数を割り当てます。ただし，FLT1 及び FLT2 の両方を割り当てるとは限りません。

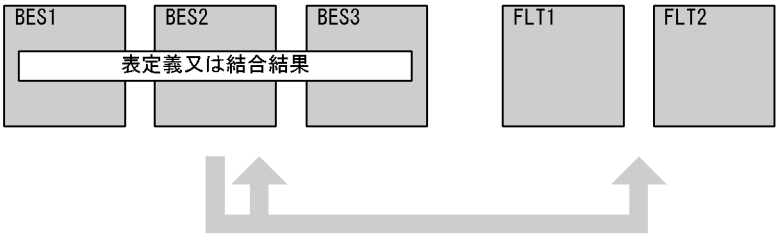
「フロータブルサーバ候補数の拡大」を指定した場合は，FLT1 及び FLT2 の両方を割り当てます。

表を定義していないバックエンドサーバがある場合で，複数の SQL から同じデータを取り出し，データ取り出しバックエンドサーバをデータ取り出し処理にだけ使用するときに適用してください。

(b)　フロータブルサーバ対象拡大（データ取り出しバックエンドサーバ）を適用する場合

フロータブルサーバ対象拡大（データ取り出しバックエンドサーバ）を適用する場合のフロータブルサーバの割り当てを次の図に示します。

図 4–19　フロータブルサーバ対象拡大（データ取り出しバックエンドサーバ）を適用する場合のフロータブルサーバの割り当て



（凡例）
BES：バックエンドサーバ
FLT：フロータブルサーバ
：候補となるフロータブルサーバ

[説明]

「フロータブルサーバ候補数の拡大」を指定しない場合は，BES1，BES2，BES3，FLT1，及び FLT2 から，HiRDB がフロータブルサーバとして必要と判断した台数を割り当てます。ただし，すべてのサーバを割り当てるとは限りません。

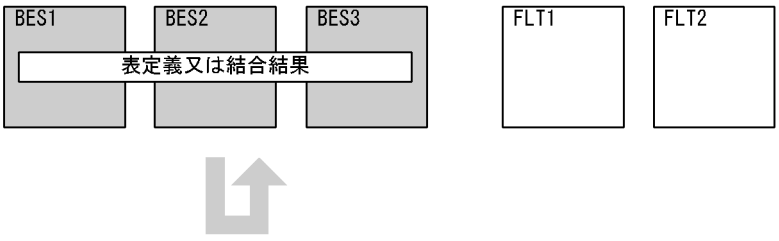
「フロータブルサーバ候補数の拡大」を指定した場合は，BES1，BES2，BES3，FLT1，及び FLT2 を割り当てます。

SQL 文を単独で実行する場合で，すべてのバックエンドサーバを効率的に使用するときに適用してください。

(c)　フロータブルサーバ対象限定（データ取り出しバックエンドサーバ）を適用する場合

フロータブルサーバ対象限定（データ取り出しバックエンドサーバ）を適用する場合のフロータブルサーバの割り当てを次の図に示します。

図 4–20　フロータブルサーバ対象限定（データ取り出しバックエンドサーバ）を適用する場合のフロータブルサーバの割り当て



（凡例）
BES：バックエンドサーバ
FLT：フロータブルサーバ
：候補となるフロータブルサーバ

[説明]

「フロータブルサーバ候補数の拡大」を指定しない場合は，BES1，BES2，及び BES3 から，HiRDB がフロータブルサーバとして必要と判断した台数を割り当てます。ただし，BES1，BES2，及び BES3 すべてを割り当てるとは限りません。

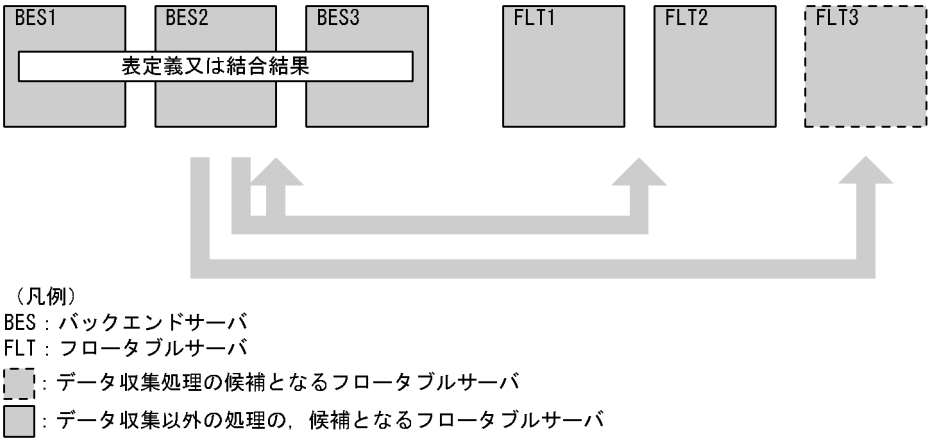
「フロータブルサーバ候補数の拡大」を指定した場合は，BES1，BES2，及び BES3 を割り当てます。

複数の SQL 文を実行する場合で，それぞれの検索が別のバックエンドサーバに定義されている表をアクセスし，表ごとに使用するバックエンドサーバを分けて運用するときに適用してください。

(d)　データ収集用サーバの分離機能を適用する場合

データ収集用サーバの分離機能を適用する場合のフロータブルサーバの割り当てを次の図に示します。

図 4–21　データ収集用サーバの分離機能を適用する場合のフロータブルサーバの割り当て



[説明]

複数の BES から 1 か所の BES にデータを集める（データ収集処理）必要のある SQL に対しては，FLT1，FLT2，及び FLT3 から，データ収集用に FLT3 を割り当てます。SQL 文中でデータ収集処理を複数回実行する場合は，必ずこのデータ収集用サーバ（FLT3）を割り当てます。

データ収集以外の処理をする場合，「フロータブルサーバ候補数の拡大」を指定しないときは，BES1，BES2，BES3，FLT1，及び FLT2 から，HiRDB がフロータブルサーバとして必要と判断した台数を割り当てます。ただし，BES1，BES2，BES3，FLT1，及び FLT2 すべてを割り当てるとは限りません。

「フロータブルサーバ候補数の拡大」を指定した場合は，BES1，BES2，BES3，FLT1，及び FLT2 を割り当てます。

SQL 文中にデータ収集処理がない場合は，「フロータブルサーバ対象拡大（バックエンドサーバ）」を適用したときと同じ動作となります。

4.5.5　グループ分け処理方式（HiRDB/パラレルサーバ限定）

グループ分け処理方式に影響がある最適化を次に示します。

- グループ分け高速化処理
- 自バックエンドサーバでのグループ化，ORDER BY，DISTINCT 集合関数処理

HiRDB がグループ分けのためのソート又はハッシング処理を不要と判断した場合は，より高速に処理できる方式が選択されます。グループ分け処理方式については，マニュアル「HiRDB コマンドリファレンス」を参照してください。

グループ分け処理方式に関する最適化の特徴を次の表に示します。

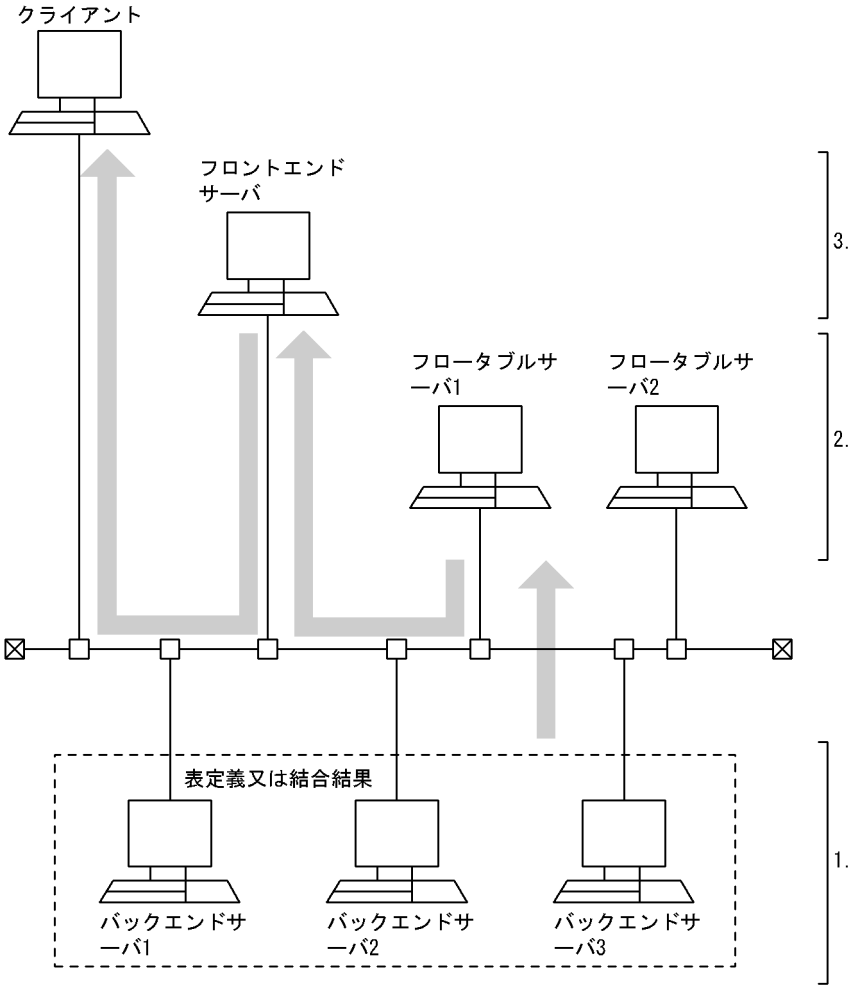
表 4–9　グループ分け処理方式に関する最適化の特徴

最適化方法 （グループ分け処理方式 の種類）	長　所	短　所
最適化を省略した場合 (FLOATABLE SORT)	バックエンドサーバ間のデータ件数に偏りがあり，グループ化してもデータ件数があまり減らない場合は高速に検索できます。	通信量が多くなるため，グループ化数が少なく，データ件数が多い場合には性能が悪くなります。
グループ分け高速化処理 (HASH)	グループ数が少ない場合は高速に検索できます。	ハッシングでグループ化をするため，グループ数が多い場合には性能が悪くなります。
自バックエンドサーバでのグループ化，ORDER BY，DISTINCT 集合関数処理 (LIST SORT)	グループ化することで，データ件数が大幅に減る場合は高速に検索できます。また，分割キーでグループ化する場合も高速に検索できます。	各バックエンドサーバでソート処理をするため，バックエンドサーバ間のデータ件数に偏りがある場合は，データ件数の多いサーバの処理時間が長くなるため，性能が悪くなります。

(1)　最適化を省略した場合のグループ分け処理方式

最適化を省略した場合のグループ分け処理方式を次の図に示します。

図 4–22　最適化を省略した場合のグループ分け処理方式



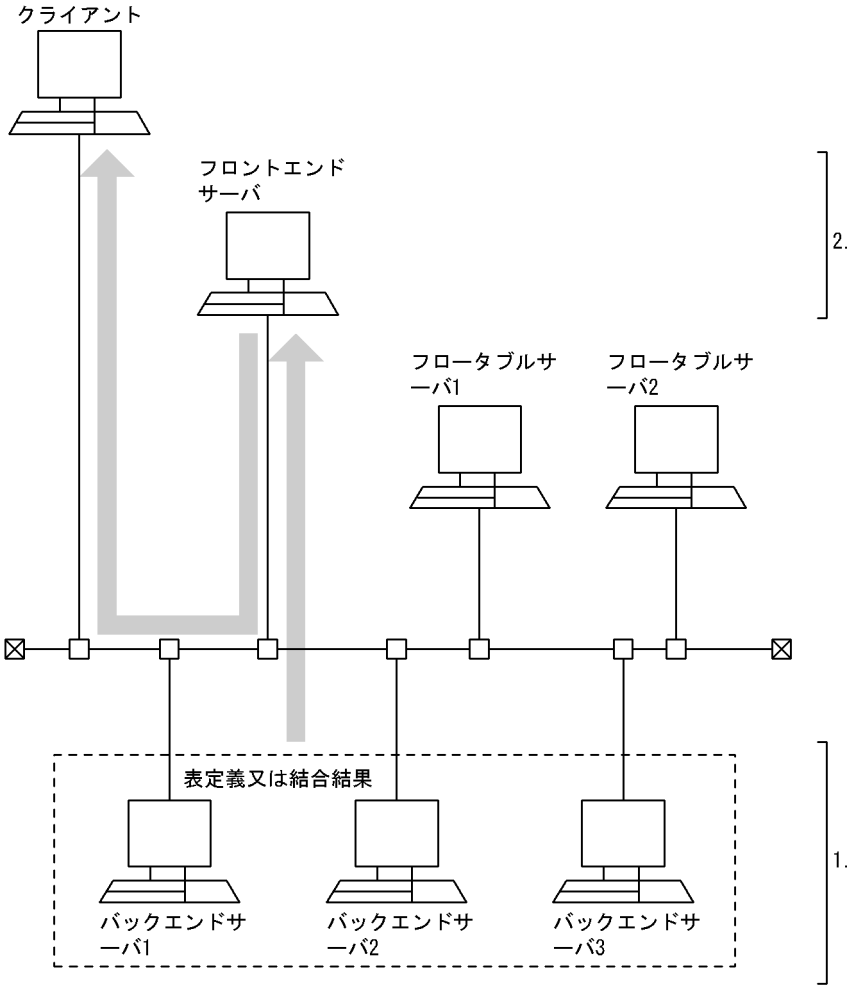
[説明]

1. データ取り出し処理だけをします。
2. グループ化列でのソート，グループ化だけをします。
3. グループ化処理結果を集めて，クライアントに転送します。

(2)　グループ分け高速化処理を適用した場合のグループ分け処理方式

グループ分け高速化処理を適用した場合のグループ分け処理方式を次の図に示します。

図 4–23　グループ分け高速化処理を適用した場合のグループ分け処理方式



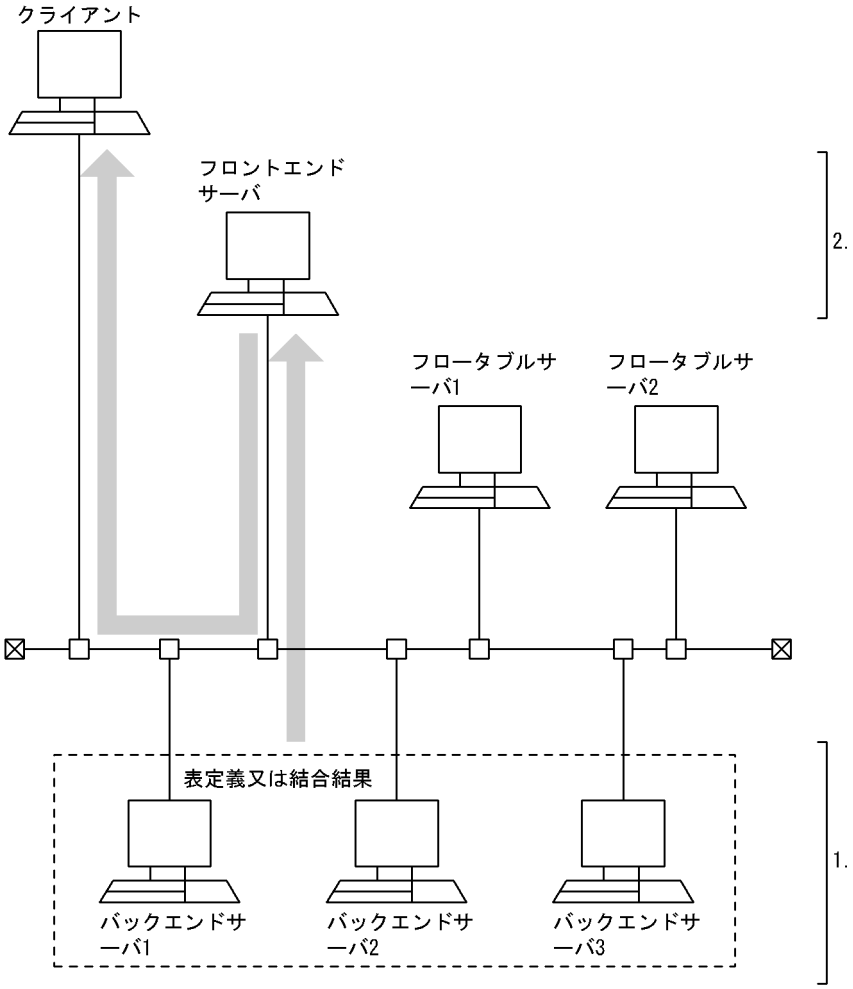
[説明]

1. データ取り出し後，グループ化列でハッシングしてグループ化をします（フロータブルサーバは使用しません）。
2. 各バックエンドサーバでのグループ化結果を集めて，更に全体をグループ化して，結果をクライアントに転送します。

(3)　自バックエンドサーバでのグループ化，ORDER BY，DISTINCT 集合関数処理を適用した場合のグループ分け処理方式

自バックエンドサーバでのグループ化，ORDER BY，DISTINCT 集合関数処理を適用した場合のグループ分け処理方式を次の図に示します。ただし，この図は 1 表を検索するときの処理方式です。

図 4–24　自バックエンドサーバでのグループ化，ORDER BY，DISTINCT 集合関数処理を適用した場合のグループ分け処理方式



［説明］

1. データ取り出し後，グループ化列でソート，グループ化をします（フロータブルサーバは使用しません）。
2. 各バックエンドサーバでのグループ化結果を集めて，更に全体をグループ化して，結果をクライアントに転送します。

4.5.6　結合方式

(1)　結合方式の種類

結合方式の種類及び特徴（直積を除く）を次の表に示します。この表にある結合方式を適用できなかった場合は，直積が適用されます。

表 4–10　結合方式の種類及び特徴

結合方式	処理方式	初回のデータ取り出し	長所	短所
マージジョイン	結合列でソートして，結合列の値が小さいものから順に突き合わせ処理をします。	遅い	ヒット件数の多い表同士でも，少ないメモリで結合できるので，ほかの方式に比べて性能劣化が少ないです。 また，結合列のデータがソート済みで，マージジョインのためのソート処理をキャンセルできる場合は，高速に検索できます。	結合する列のデータがソートされていない場合，ソート処理の負荷が高くなり，性能が悪くなります。
ネストループジョイン	外表の結合列の値を使用して，内表の結合列に定義されているインデクスをサーチして，突き合わせを入れ子にしたものを繰り返して処理します。	速い	内表を結合列に指定したインデクスで絞り込める場合，高速に検索できます。	外表から行を 1 件取り出すごとに，インデクスを使用して内表を検索するため，外表のヒット件数が多い場合は性能が悪くなります。
ハッシュジョイン	内表の結合列からハッシュ表を作成し，外表の結合列をハッシングして内表から作成したハッシュ表と突き合わせ処理をします。	内表のヒット件数が少ない場合は速い（ネストループジョインよりは遅く，マージジョインよりは速い）	内表のヒット件数が少なく，外表のヒット件数が多い場合，高速に検索できます。	内表のヒット件数が多い場合は使用するメモリが多くなります。また，メモリを使用できなくなった分については，いったんファイルに退避するため，性能が悪くなります。
SELECT-APSL	？パラメタを含んだ条件がある場合，あらかじめ結合方法の候補を複数作成しておいて，？パラメタの値を入力した時点で最適な検索方法を決定します。	実際に選択された検索方法によって異なる	？パラメタの値を入力する時点で，最適な検索方法を選択できます。	最適化情報収集ユティリティ（pdgetcst）を実行しておく必要があります。※ また，複数の結合方法の候補を作成するため，SQL オブジェクトのサイズが大きくなります。

注※

最適化情報収集ユティリティを実行していても，最適なアクセスパスを選択できない場合があります。最適化情報収集ユティリティの実行要否については，マニュアル「HiRDB コマンドリファレンス」を参照し，性能について十分に検証するようにしてください。

(2)　処理方式

(a)　マージジョイン

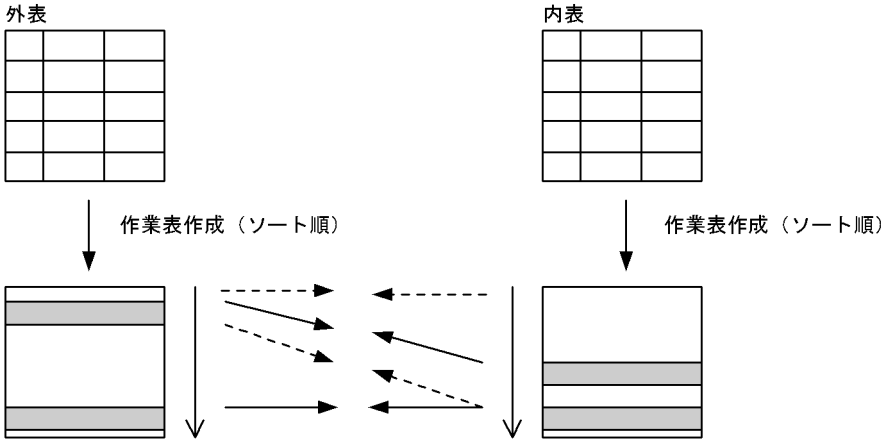
マージジョインは，外表をあまり絞り込めないときに有効です。

SORT MERGE JOIN：

外表と内表から行を取り出して，それぞれ作業表を作成しソートします。その後，結合条件を満たした行同士を結合します。

SORT MERGE JOIN の処理方式を次の図に示します。

図 4–25　SORT MERGE JOIN の処理方式

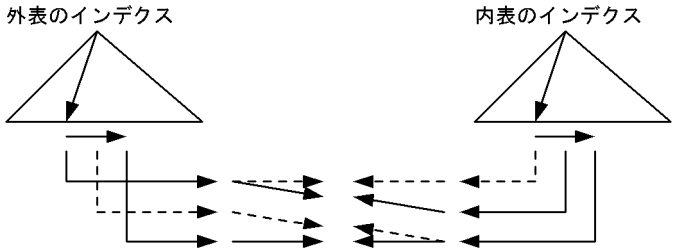


KEY SCAN MERGE JOIN

外表と内表から，KEY SCAN で行を取り出します。その後，結合条件を満たした行同士を結合します。

KEY SCAN MERGE JOIN の処理方式を次の図に示します。

図 4–26　KEY SCAN MERGE JOIN の処理方式

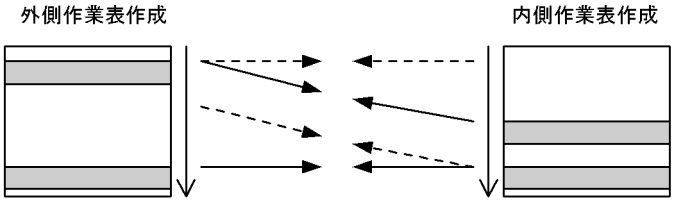


LIST SCAN MERGE JOIN

外表と内表から作業表を作成し，ソートをしないで結合列の昇順に行を取り出します。その後，結合条件を満たした行同士を結合します。

LIST SCAN MERGE JOIN の処理方式を次の図に示します。

図 4–27　LIST SCAN MERGE JOIN の処理方式



L-KEY R-LIST MERGE JOIN

外表は KEY SCAN で行を取り出し，内表は作業表を作成し，ソートをしないで行を取り出します。その後，結合条件を満たした行同士を結合します。

L-KEY R-SORT MERGE JOIN

外表は KEY SCAN で行を取り出し，内表は作業表を作成し，ソートをして行を取り出します。その後，結合条件を満たした行同士を結合します。

L-LIST R-KEY MERGE JOIN

外表は作業表を作成し，ソートをしないで行を取り出し，内表は KEY SCAN で行を取り出します。その後，結合条件を満たした行同士を結合します。

L-LIST R-SORT MERGE JOIN

外表は作業表を作成し，ソートをしないで行を取り出します。また，内表は作業表を作成し，ソートをして行を取り出します。その後，結合条件を満たした行同士を結合します。

L-SORT R-KEY MERGE JOIN

外表は作業表を作成し，ソートをして行を取り出し，内表は KEY SCAN で行を取り出します。その後，結合条件を満たした行同士を結合します。

L-SORT R-LIST MERGE JOIN

外表は作業表を作成し，ソートをして行を取り出し，内表は作業表を作成し，ソートをしないで行を取り出します。その後，結合条件を満たした行同士を結合します。

(b)　ネストループジョイン

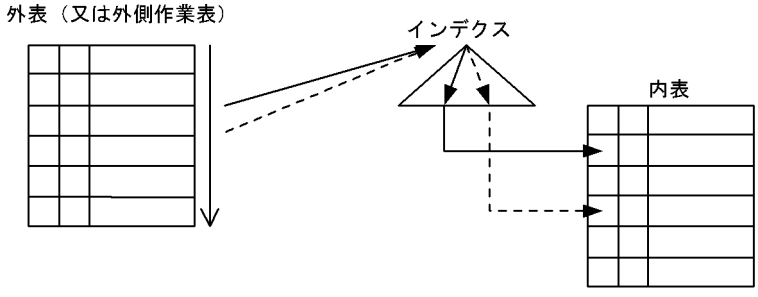
ネストループジョインは，内表にインデクスが定義されていて，外表をかなり絞り込めるときに有効です。

NESTED LOOPS JOIN

外表から 1 行ずつ行を取り出し，内表のそれぞれの行に突き合わせて，結合条件を満たす行を取り出す入れ子型のループ処理をする結合方法です。

NESTED LOOPS JOIN の処理方式を次の図に示します。

図 4–28　NESTED LOOPS JOIN の処理方式



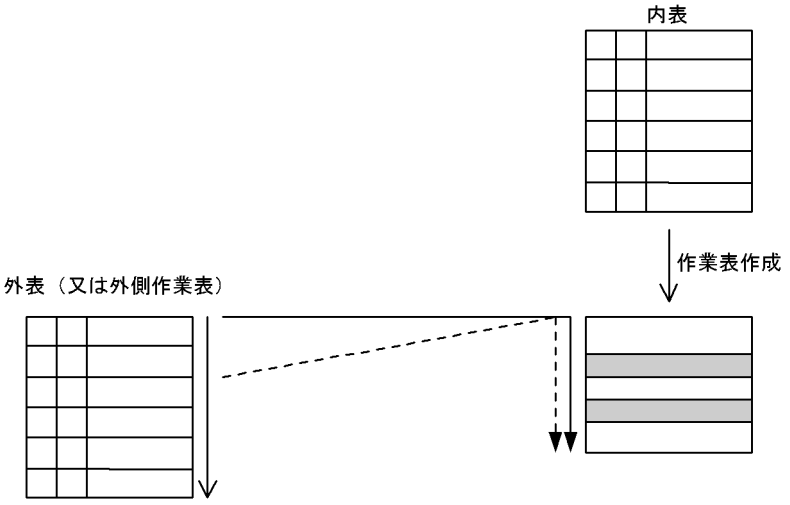
注　外表は，インデクスを使用した検索をする場合もあります。

R-LIST NESTED LOOPS JOIN

内表から行を取り出して作業表を作成します。その後，外表から 1 行ずつ行を取り出して，それぞれの行に対して内表から作成した作業表を突き合わせ，結合条件を満たす行を取り出す入れ子型のループ処理をする結合方法です。

R-LIST NESTED LOOPS JOIN の処理方式を次の図に示します。

図 4–29　R-LIST NESTED LOOPS JOIN の処理方式



注　外表は，インデクスを使用した検索をする場合もあります。

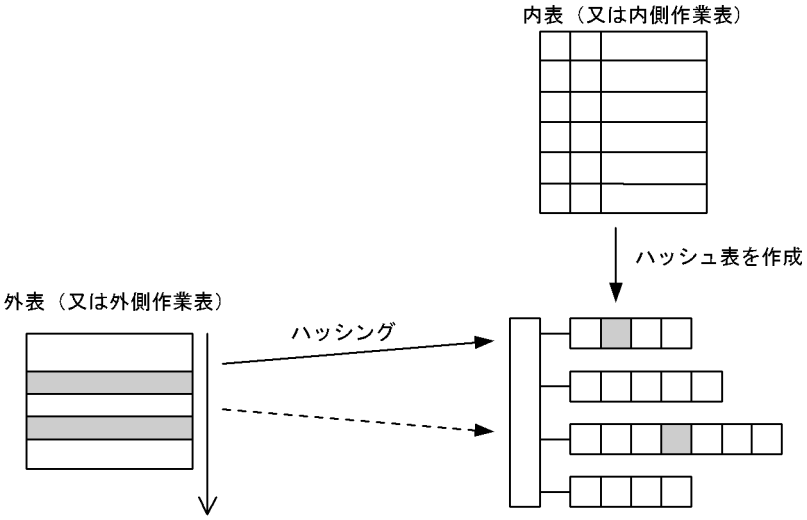
(c)　ハッシュジョイン

HASH JOIN

あらかじめ内表の結合列の値でハッシングしてハッシュ表を作成しておいて，外表を 1 行取り出すごとに外表の結合列の値でハッシングして，内表から作成しておいたハッシュ表と突き合わせて結合します。

HASH JOIN の処理方式を次の図に示します。

図 4–30　HASH JOIN の処理方式



ハッシュジョインには，4 種類の処理方式があります。ハッシュジョインの処理方式と特徴を次の表に示します。

表 4–11　ハッシュジョインの処理方式と特徴

処理方式	内容	長所	短所	選択方法
一括ハッシュジョイン	内表から作成したハッシュ表を，すべて作業表用バッファ領域に展開し，ハッシュジョインをする方式です。 メモリ使用量はやや多くなり，処理性能もやや良くなります。	ハッシュ表を，すべて作業表用バッファ領域に展開してハッシュジョインをするため，高速に処理できます。	内表のハッシュ表サイズが大きい場合，使用する作業表用バッファ領域が大きくなるので SQL 文の同時実行性が損なわれます。	ハッシュ表サイズを変更します。※
バケット分割ハッシュジョイン	内表，外表を複数のバケットに分割し，内表の一部のバケットからハッシュ表を作成して作業表用バッファ領域に展開します。さらに，残りのバケットを作業表用ファイルに退避しておいて，作業表用バッファ領域に展開された内表のバケットと，同じ値の領域の外表バケットを読み出して，ハッシュジョインします。そして，作業表用ファイルから，内表を少しずつ作業表用バッファ領域に展開してハッシュジョインをする方式です。 メモリ使用量は少なくなり，処理性能はやや悪くなります。	作業表用バッファ領域が少ない環境でも，ハッシュジョインができます。	内表，外表の行をいったん作業表用ファイルに退避するため，作業表用バッファ領域だけでハッシュジョインする場合に比べると，性能が悪くなります。	
連続ハッシュジョイン	3 表以上を検索する場合に，最も外側の表以外の表から，ハッシュ表を作成して作業表用バッファ領域に展開しておいて，連続してハッシュジョインをする方式です。 メモリ使用量は多くなり，処理性能は良くなります。	ハッシュ表をすべて作業表用バッファ領域に展開してハッシュジョインをするので，高速に処理できます。 また，最も外側の表だけが大きい場合，高速に処理できます。	連続実行する表の数が多くなると，使用する作業表用バッファ領域が大きくなります。	選択できません。 表の行数を基に，HiRDB が自動的に最適な方式を選択します。
断続ハッシュジョイン	3 表以上を検索する場合，表又は作業表を結合するごとに，結合結果を作業表用ファイルに退避してハッシュジョインをする方式です。 メモリ使用量は少なくなり，処理性能は悪くなります。	作業表用バッファ領域が少ない環境でも，3 表以上のハッシュジョインができます。	表又は作業表を結合するごとに，結合結果をいったんファイルに退避するため，入出力が多くなり性能が悪くなります。	

注※

ハッシュ表サイズの変更については，「ハッシュジョイン，副問合せのハッシュ実行を適用する場合の準備」を参照してください。

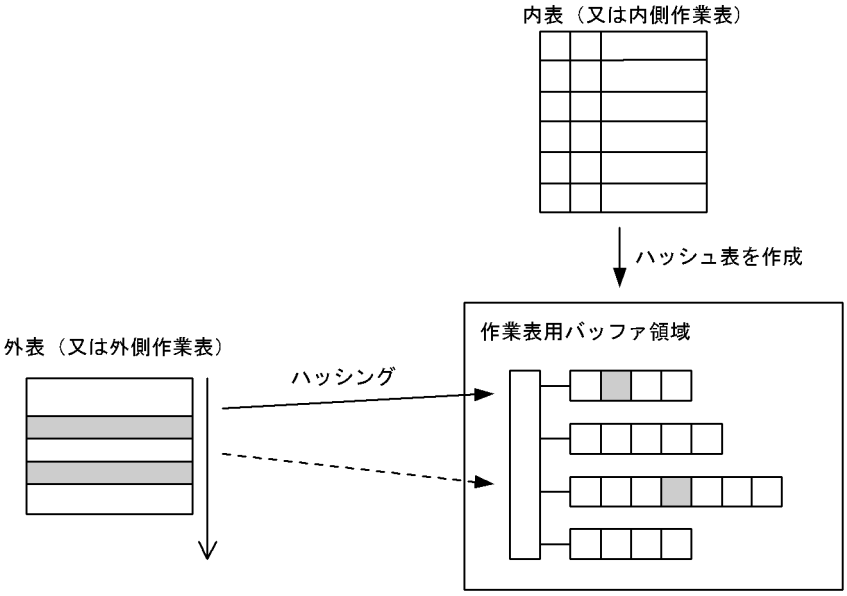
各処理方式の概要を次に示します。

●一括ハッシュジョイン

内表から作成したハッシュ表を，すべて作業表バッファ領域に展開してハッシュジョインします。

一括ハッシュジョインの処理方式を次の図に示します。

図 4–31　一括ハッシュジョインの処理方式



●バケット分割ハッシュジョイン

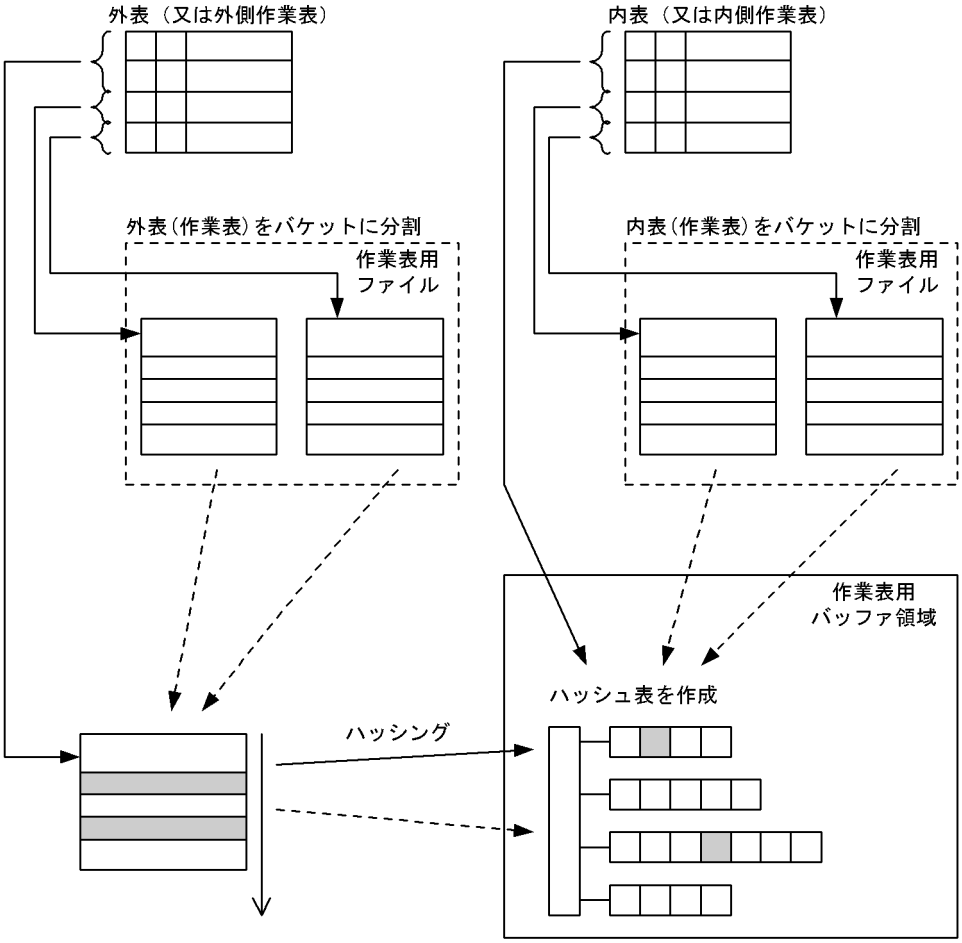
内表，外表をバケットに分割し，内表の一部を作業表用バッファ領域に展開し，残りを作業表用ファイルに退避します。

バケットとは，表の結合列の値でハッシングして，複数の小さな表に分割することをいいます。

結合処理は，作業表用バッファ領域に展開された内表の一部で行います。まず，内表からハッシュ表を作成し，外表から 1 行ずつ取り出して内表から作成したハッシュ表と突き合わせて結合をします。作業表バッファ領域にある表同士の結合が終わった時点で，作業表用ファイルから外表，内表のバケットを作業表用バッファ領域に展開し，同様に結合処理をします。そして，表全体を作業表用バッファ領域に展開して結合した時点で終了となります。

バケット分割ハッシュジョインの処理方式を次の図に示します。

図 4–32　バケット分割ハッシュジョインの処理方式



●連続ハッシュジョイン

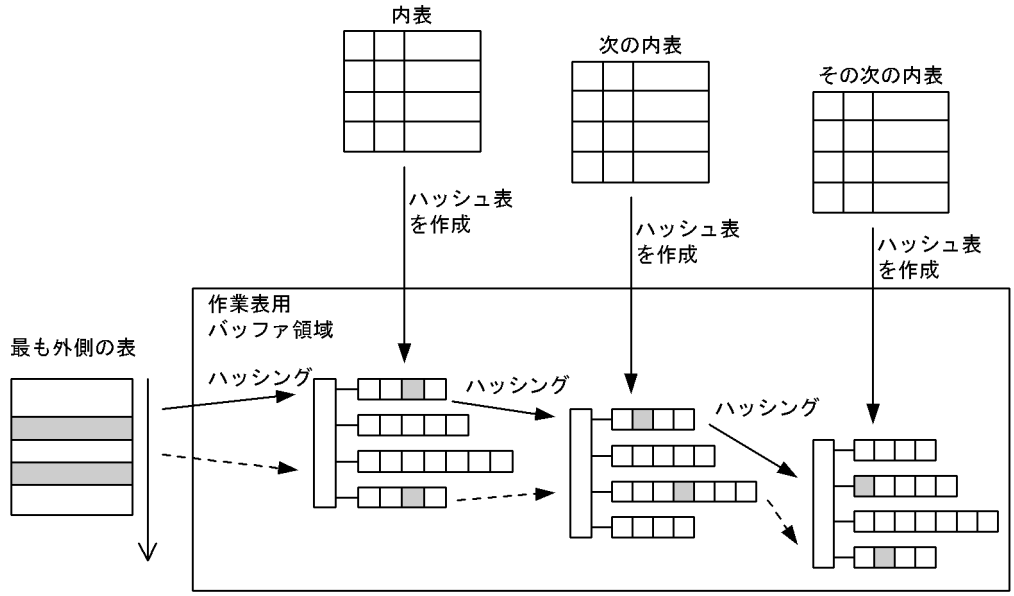
3 表以上の検索に適用します。

まず，最も外側の表以外の表からハッシュ表を作成し，作業表用バッファ領域に展開します。次に，外表から 1 行取り出してハッシングして，内表から作成したハッシュ表と突き合わせて結合します。結合条件を満たす場合には，結合結果でハッシングしてハッシュ表と突き合わせて結合します。

最後の行まで結合し終わるか，又は条件が偽になった時点で，最も外側の表まで戻り，次の行を取り出して同様に結合処理を繰り返します。結合途中で内表の結合キー値が重複している箇所がある場合には，そこまで戻って結合処理を繰り返します。重複キー値の処理がすべて終了したら，最も外側の表まで戻り，次の行を取り出して同様に結合処理を繰り返します。

連続ハッシュジョインの処理方式を次の図に示します。

図 4–33　連続ハッシュジョインの処理方式



●断続ハッシュジョイン

3 表以上の検索に適用します。

まず，最初の結合の内表からハッシュ表を作成し，作業表用バッファ領域に展開します。次に，外表を 1 行ずつ取り出して外表の結合列の値でハッシングし，内表から作成したハッシュ表と突き合わせて結合します。外表からすべての行を取り出して結合し終わったら，次の結合処理に移ります。

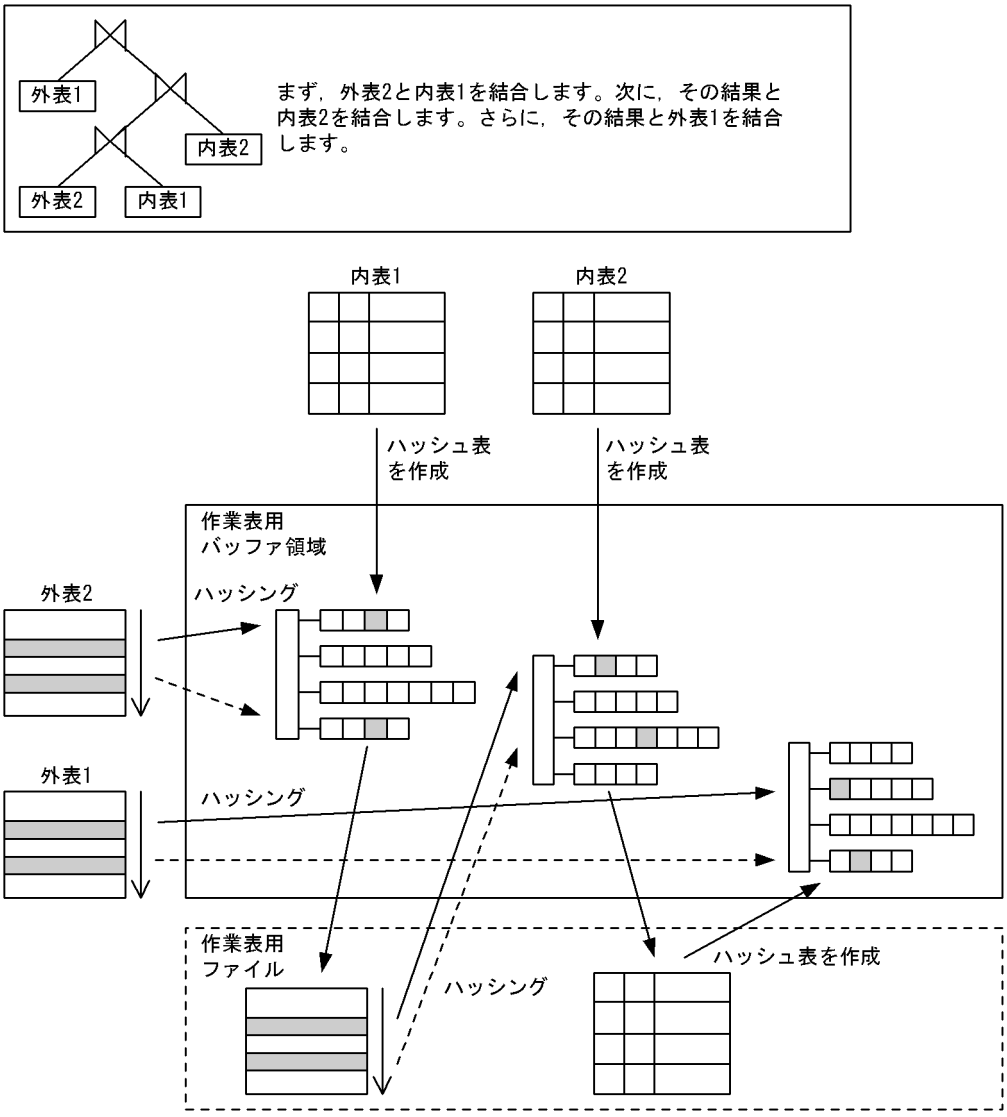
結合結果が外表になる場合と，内表になる場合とで処理が変わります。

結合処理が外表になる場合は，次の結合の内表からハッシュ表を作成し，結合結果から 1 行ずつ取り出して，内表から作成したハッシュ表と突き合わせて結合します。

処理結果が内表になる場合は，結合結果からハッシュ表を作成し，外表から 1 行ずつ取り出して，結合結果から作成したハッシュ表と突き合わせて結合します。

断続ハッシュジョインの処理方式を次の図に示します。なお，この処理方式では，外表 1→（(外表 2→内表 1）→内表 2）と結合した場合を例にしています。

図 4–34　断続ハッシュジョインの処理方式



(d)　SELECT-APSL

SELECT-APSL は，SQL 実行時に結合方法を動的に決定する方式です。

SELECT-APSL（HiRDB/パラレルサーバ限定）

条件に？パラメタを含んでいる場合，？パラメタの値によって最適な結合方法が変わることがあります。また，SQL 最適化処理時に？パラメタの値が決まらないため，最適な結合方法が決定できません。そのため，SQL 実行時にヒット率を計算して結合方法を決定します。

SELECT-APSL について，アクセスパス表示ユティリティ（pdvwopt）の表示例を使用して説明します。

```
条件        T1(外表).C1=？パラメタ
基準値      0.047
  [1] ネストループジョイン
  [2] マージジョイン
```

[説明]

- 述語 T1(外表).C1=？パラメタのヒット率が，基準値（0.047）より小さい場合

ヒット率が小さく，外表をかなり絞り込めるので，実行時にネストループジョインが選択されます。

- 述語 T1(外表).C1=？パラメタのヒット率が，基準値（0.047）以上の場合
  ヒット率が大きく，外表をあまり絞り込めないので，実行時にマージジョインが選択されます。

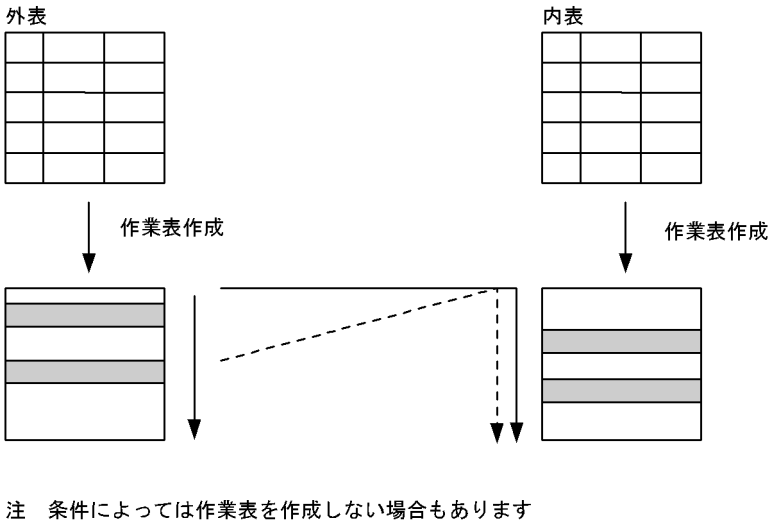
(e)　直積

CROSS JOIN

外表のすべての行と，内表のすべての行をそれぞれ組み合わせて結合します。2 表にわたった条件があれば，結合した後に判定します。

CROSS JOIN の処理方式を次の図に示します。

図 4–35　CROSS JOIN の処理方式



注　条件によっては作業表を作成しない場合もあります

4.5.7　検索方式

(1)　検索方式の種類

検索方式の種類及び特徴（リストスキャン（LIST SCAN），及び ROWID FETCH を除く）を次の表に示します。

リストスキャンは，ビュー表の検索，WITH 句問合せ式などで，いったん作業表を作成して検索する場合に適用されます。ROWID FETCH は，カーソルを使用した場合などに適用されます。

表 4–12　検索方式の種類及び特徴

検索方式	処理方式	長所	短所
テーブルスキャン (TABLE SCAN)	表が格納されているページ（データページ）を順次サーチ	全件検索をする場合は高速に検索できます。	条件によって検索結果を絞り込める場合でも，すべてのデータペー

検索方式	処理方式	長所	短所
	して，すべての行を参照する方式です。 初回のデータ取り出しは，やや遅いです。	また，インデクスでの絞り込みができない場合も，高速に検索できます。	ジを参照するため，性能は悪くなります。
インデクススキャン (INDEX SCAN， MULTI COLUMNS INDEX SCAN， PLUGIN INDEX SCAN)	インデクスをバイナリサーチして，目的のデータの行識別子を取得するごとに，その行識別子が指すデータページ中の行を参照する方式です。 初回のデータ取り出しは，速いです。	インデクスで絞り込める場合は高速に検索できます。 クラスタキーインデクスの場合は，あまり絞り込めない場合でも高速に検索できます。 また，ソート処理を省略できる場合があります。※1	インデクスであまり絞り込めない場合は，データページに対するランダムな入出力が増え，性能が悪くなります。
キースキャン (KEY SCAN， MULTI COLUMNS KEY SCAN， PLUGIN KEY SCAN)	インデクスをバイナリサーチして，インデクス中のデータ（インデクス構成列の値又は行識別子）だけを参照する方式です。 インデクスの構成列又は行識別子だけを参照する場合に適用されます。 初回のデータ取り出しは，速いです。	インデクスであまり絞り込めない場合でも，データページの入出力がなく，インデクスページを参照するだけなので，高速に検索できます。 また，ソート処理を省略できる場合があります。※1	特にありません。
SELECT-APSL	？パラメタを含んだ条件がある場合，あらかじめ検索方法の候補を複数作成しておいて，？パラメタの値を入力する時点で最適な検索方法を決める方式です。 初回のデータ取り出しの速さは，実際に選択された検索方法によって異なります。	？パラメタの値を入力する時点で，インデクスでの絞り込み率を考慮して最適な検索方法を選択できます。	最適化情報収集ユティリティ（pdgetcst）を実行しておく必要があります。※2 また，複数の検索方法の候補を作成するため，SQL オブジェクトのサイズが大きくなります。
AND の複数インデクス利用 (AND PLURAL INDEXES SCAN)	複数のインデクスを使用して複数の作業表を作成し，作業表間の積集合，和集合，及び差集合を組み合わせて結果を求める方式です。 初回のデータ取り出しは，遅いです。	積集合，和集合，及び差集合を組み合わせて結果を求めるため，複雑な条件を指定した場合でも，インデクスを使用しての評価ができます。	作業表を複数個作成し，それぞれの作業表に対してソートをするため，インデクスであまり絞り込めない場合は，ソートする件数が多くなるため，性能が悪くなります。
OR の複数インデクス利用 (OR PLURAL INDEXES SCAN)	複数のインデクスを使用して検索した結果を，一つの作業表に格納して，最後に重複排除をして結果を求める方式です。 初回のデータ取り出しは，遅いです。	OR 演算子で結ばれたそれぞれの探索条件に対して，インデクスで絞り込める場合は，高速に検索できます。	複数のインデクスを使用して検索した結果を，一つの作業表に格納してソート及び重複排除をするため，重複排除する前の件数が多い場合には，性能が悪くなります。

注※1

ORDER BY 処理や GROUP BY 処理で，インデクスでソートできる場合，ソートをしないでデータを取り出します。ソート処理を省略しているかどうかは，アクセスパスで確認できます。

注※2

最適化情報収集ユティリティを実行した場合でも，最適なアクセスパスを選択できないときがあります。最適化情報収集ユティリティの実行要否については，マニュアル「HiRDB コマンドリファレンス」を参照し，性能について十分な検証をするようにしてください。

(2) 処理方式

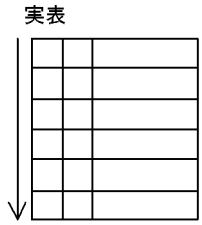
(a) インデクスを使用しない検索

TABLE SCAN

インデクスを使用しないで，表のデータページを検索します。

TABLE SCAN の処理方式を次の図に示します。

図 4–36 TABLE SCAN の処理方式



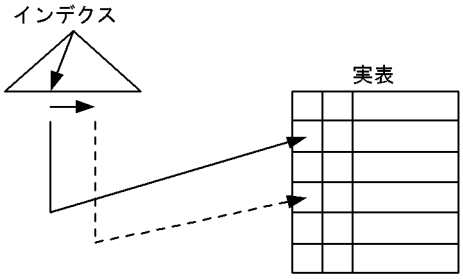
(b) 一つのインデクスを使用した検索

INDEX SCAN

単一列インデクスのインデクスページを検索して絞り込んだ後，表のデータページを検索します。

INDEX SCAN の処理方式を次の図に示します。

図 4–37 INDEX SCAN の処理方式

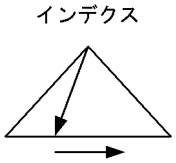


KEY SCAN

単一列インデクスのインデクスページだけを検索します。データページは検索しません。

KEY SCAN の処理方式を次の図に示します。

図 4–38　KEY SCAN の処理方式

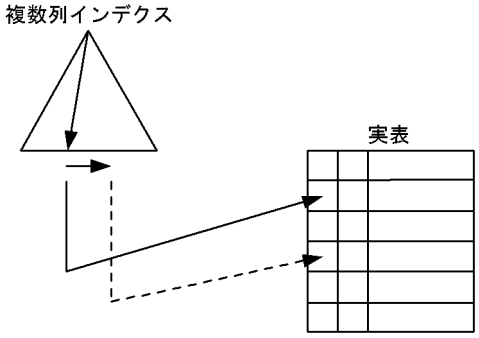


MULTI COLUMNS INDEX SCAN

複数列インデクスのインデクスページを検索して絞り込んだ後，表のデータページを検索します。

MULTI COLUMNS INDEX SCAN の処理方式を次の図に示します。

図 4–39　MULTI COLUMNS INDEX SCAN の処理方式

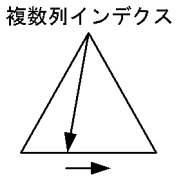


MULTI COLUMNS KEY SCAN

複数列インデクスのインデクスページだけを検索します。データページは検索しません。

MULTI COLUMNS KEY SCAN の処理方式を次の図に示します。

図 4–40　MULTI COLUMNS KEY SCAN の処理方式

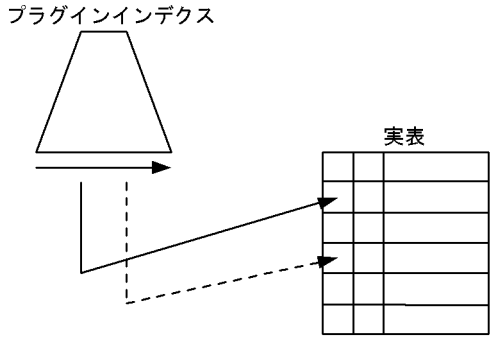


PLUGIN INDEX SCAN

プラグインインデクスを使用して絞り込んだ後，表のデータページを検索します。

PLUGIN INDEX SCAN の処理方式を次の図に示します。

図 4–41　PLUGIN INDEX SCAN の処理方式



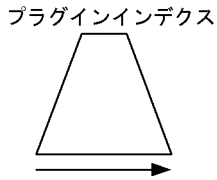
注　プラグインインデクスの構造は，プラグインによって異なります。

PLUGIN KEY SCAN

プラグインインデクスを使用してインデクスページだけを検索します。データページは検索しません。

PLUGIN KEY SCAN の処理方式を次の図に示します。

図 4–42　PLUGIN KEY SCAN の処理方式



注　プラグインインデクスの構造は，プラグインによって異なります。

(3)　SELECT-APSL

SELECT-APSL（HiRDB/パラレルサーバの場合）

条件に？パラメタを含んでいる場合，？パラメタの値によって最適な検索方法が変わることがあります。また，前処理時に？パラメタの値が決まらないため，最適な検索方法が決定できません。そのため，SQL 実行時にヒット率を計算して検索方法を決定します。

(4)　複数のインデクスを使用した検索

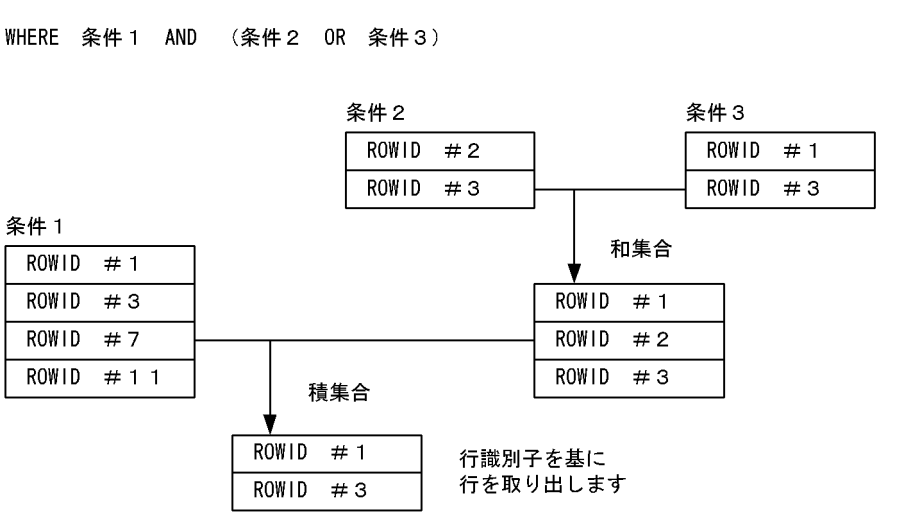
AND PLURAL INDEXES SCAN

AND 演算子又は OR 演算子で結ばれた探索条件に対して，それぞれのインデクスを使用して検索し，行識別子（ROWID）をそれぞれの作業表に格納します。これらの作業表を，AND 演算子の場合は積集合，OR 演算子の場合は和集合，ANDNOT 演算子（ASSIGN LIST 文でだけ使用できます）の場合は，差集合をとって一つの作業表にまとめます。その後，この作業表の行識別子を基に行を取り出します。

各条件から行識別子の作業表を作成する場合に，条件列にインデクスがなくても，TABLE SCAN によって作業表を作成するときがあります。

AND PLURAL INDEXES SCAN の処理方式を次の図に示します。

図 4–43　AND PLURAL INDEXES SCAN の処理方式



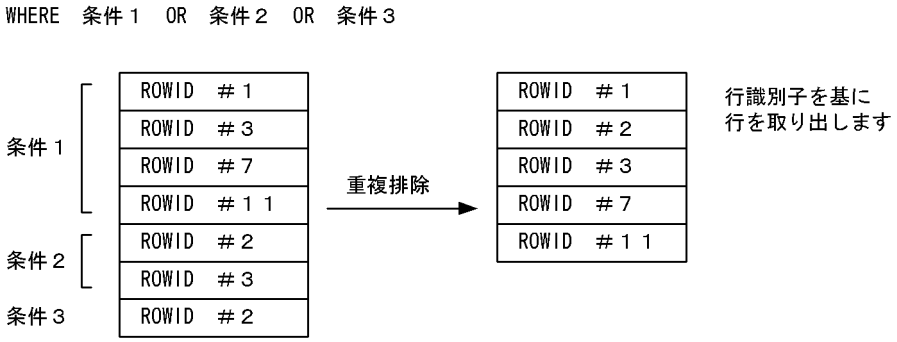
OR PLURAL INDEXES SCAN

OR 演算子で結ばれた探索条件に対して，それぞれのインデクスを使用して検索し，行識別子（ROWID）を一つの作業表に格納します。この作業表の重複する行を重複排除した後，行識別子を基に行を取り出します。

各条件から行識別子の作業表を作成する場合に，条件列にインデクスがなくても，TABLE SCAN によって作業表を作成するときがあります。

OR PLURAL INDEXES SCAN の処理方式を次の図に示します。

図 4–44　OR PLURAL INDEXES SCAN の処理方式



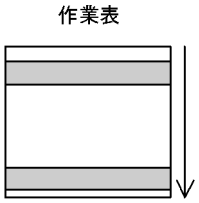
(5)　内部的に作成した作業表の検索

LIST SCAN

内部的に作成した作業表を検索します。

LIST SCAN の処理方式を次の図に示します。

図 4–45　LIST SCAN の処理方式



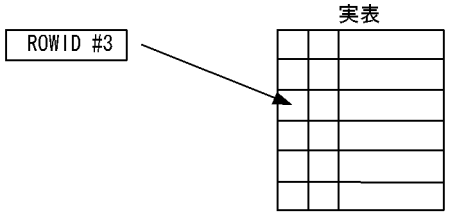
(6)　行識別子を使用した検索

ROWID FETCH

行識別子（ROWID）をキーにして表を検索します。行を取り出す必要がない場合は検索しません。

ROWID FETCH の処理方式を次の図に示します。

図 4–46　ROWID FETCH の処理方式



4.5.8　外への参照のない副問合せの実行方式

(1)　実行方式の種類

外への参照のない副問合せの実行方式及び特徴を次の表に示します。

表 4–13　外への参照のない副問合せの実行方式及び特徴

実行方式	処理方式	長所	短所
作業表 ATS 実行	あらかじめ結果を求めて作業表を作成しておいて，外側の問合せに対してインデクスを使用して検索する場合に，副問合せ結果から作成した作業表を使用してサーチ範囲を絞り込む方式です。	外側の問合せに対してインデクスを使用できます。したがって，副問合せのヒット件数が少く，外側の問合せの件数が多い場合，インデクスを使用することでサーチ範囲を絞り込めるときに高速に検索できます。	副問合せの結果の行数分，外側の問合せに対してインデクスを使用して検索するため，副問合せのヒット件数が多い場合には性能が悪くなります。
作業表実行	あらかじめ副問合せの結果を求めて作業表を作成しておいて，外側の問合せを 1 行検索するごとに副問合せの結果から作成した作業表と突き合わせて，副問合せを含む述語を評価する方式です。	作業表を必要とするすべての副問合せの条件に対して適用できます。	外側の問合せの件数が多い場合には性能が悪くなります。

実行方式	処理方式	長所	短所
行値実行	あらかじめ副問合せを求めておいて（作業表は作成しません），外側の問合せを検索する場合に副問合せの結果の値を使用することで，副問合せを含む条件を評価する方式です。	外側の問合せに対してインデクスを使用できます。したがって，外側の問合せの件数が多い場合，インデクスを使用することでサーチ範囲を絞り込めるときは高速に検索できます。	外側の問合せの件数が多く，インデクスを使用して副問合せを含む述語を絞り込めない場合は，性能が悪くなります。
ハッシュ実行	あらかじめ副問合せ結果からハッシュ表を作成しておいて，外側の問合せを 1 行取り出すごとに外側の問合せの値をハッシングして，ハッシュ表と突き合わせをする方式です。	副問合せのヒット件数が少なく，外側の問合せの件数が多い場合，高速に検索できます。	副問合せのヒット件数が多い場合は，使用する作業表用バッファサイズが大きくなります。使用する作業表バッファサイズを指定できますが，作業表用バッファが一杯になった場合は，いったん作業表用ファイルへ退避するため，性能が悪くなります。

外への参照のない副問合せの最適な実行方式を次の表に示します。

表 4–14　外への参照のない副問合せの最適な実行方式

<table>
<tr><th>副問合せ</th><th colspan="2">最適な実行方式</th></tr>
<tr><td rowspan="9">=ANY，=SOME の限定述語の右側，及び IN 述語の右側に指定した表副問合せ</td><td colspan="2">外側の問合せ，副問合せのデータ件数によって異なります。</td></tr>
<tr><td>外側の問合せ：多い
副問合せ　　：少ない</td><td rowspan="3">作業表 ATS 実行，又はハッシュ実行が有効です。</td></tr>
<tr><td>外側の問合せ：中間
副問合せ　　：少ない</td></tr>
<tr><td>外側の問合せ：少ない
副問合せ　　：少ない</td></tr>
<tr><td>外側の問合せ：多い
副問合せ　　：中間</td><td rowspan="2">ハッシュ実行が有効です。</td></tr>
<tr><td>外側の問合せ：中間
副問合せ　　：中間</td></tr>
<tr><td>外側の問合せ：少ない
副問合せ　　：中間</td><td>ハッシュ実行，又は作業表実行が有効です。</td></tr>
<tr><td>外側の問合せ：多い
副問合せ　　：多い</td><td rowspan="2">ハッシュ実行が有効です（データ件数が多いので性能向上は期待できません）。</td></tr>
<tr><td>外側の問合せ：中間
副問合せ　　：多い</td></tr>
<tr><td>外側の問合せ：少ない
副問合せ　　：多い</td><td>ハッシュ実行，又は作業表実行が有効です。なお，外への参照のある EXISTS 述語に変換すると，高速に検索できることがあります。</td></tr>
</table>

副問合せ	最適な実行方式
限定述語（=ANY 及び=SOME を除く）の右側，及び IN 述語の右側に指定した表副問合せ	常に作業表実行が適用されます。
EXISTS 述語の副問合せ	常に行値実行が適用されます。
上記以外の副問合せ（スカラ副問合せ，及び行副問合せ）	

(2)　処理方式

(a)　作業表 ATS 実行

WORK TABLE ATS SUBQ

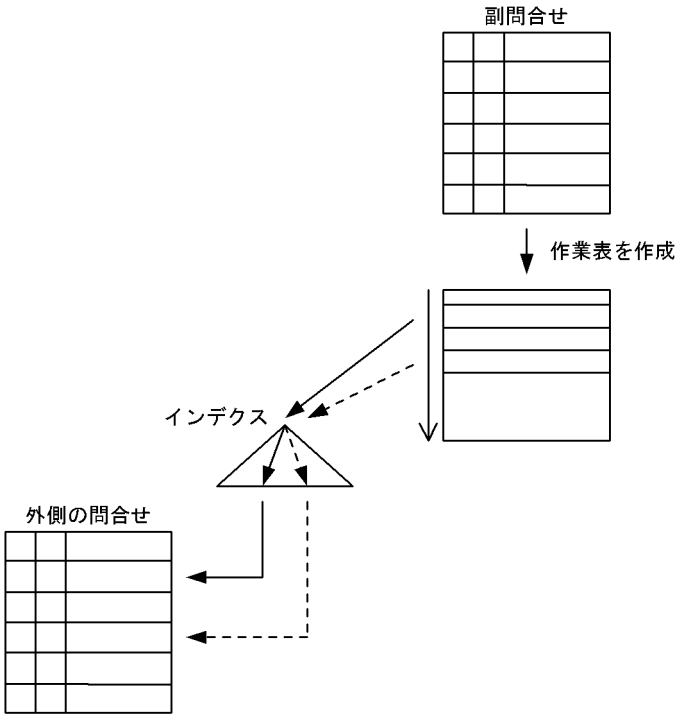
=ANY，=SOME の限定述語の右側，及び IN 述語の右側に指定した表副問合せに適用します。

まず，副問合せの選択式の値を求めて，作業表を作成します。次に，インデクスを使用して外側の問合せを検索します。このとき，副問合せの結果を使用してインデクスのサーチ範囲を絞り込んで検索します。問合せのサーチ条件種別は ATS 又は RANGES となります。

なお，副問合せに対して，HiRDB が内部的に重複排除（DISTINCT）をすることがあります。

WORK TABLE ATS SUBQ の処理方式を次の図に示します。

図 4–47　WORK TABLE ATS SUBQ の処理方式



限定述語と比較述語の例を次に示します。

例：

```
SELECT C1 FROM T1 WHERE C2=ANY(SELECT C2 FROM T2)
```

注　T1(C2)にインデクスが定義されているものとします。

まず，副問合せの表 T2 を検索して，T2.C2 の値から作業表を作成します。次に，作業表から T2.C2 の値を 1 行ずつ取り出して，外側の問合せの T1.C2 に定義したインデクスのサーチ範囲を絞り込んで検索します。

(b)　作業表実行

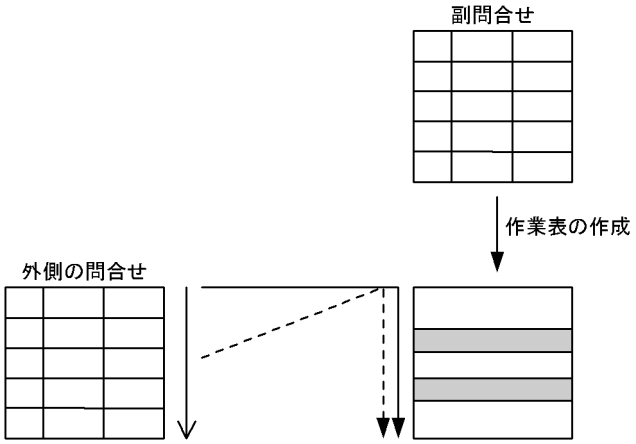
WORK TABLE SUBQ

限定述語の右側，及び IN 述語の右側の表副問合せに適用します。

まず，副問合せの選択式の値を求めて，作業表を作成します。次に，外側の問合せを検索します。このとき，外側の問合せを 1 行検索するごとに副問合せの結果と突き合わせて探索条件を評価します。

WORK TABLE SUBQ の処理方式を次の図に示します。

図 4–48　WORK TABLE SUBQ の処理方式



例：

```
SELECT T1.C1 FROM T1 WHERE T1.C2=ANY(SELECT C2 FROM T2)
```

まず，副問合せの表 T2 を検索して，T2.C2 の値から作業表を作成します。次に，外側の問合せを実行して 1 行ずつ取り出し，T1.C2 の値を副問合せから作成した作業表と突き合わせて探索条件を評価します。

(c)　行値実行

ROW VALUE SUBQ

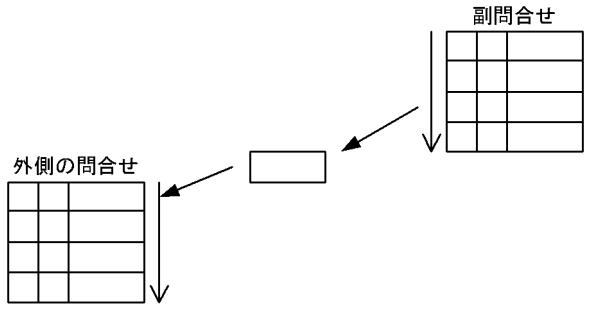
行副問合せ，スカラ副問合せ，及び EXISTS 述語に対して適用します。

まず，副問合せの選択式の値を求めます。次に，副問合せ結果の値を使用して，外側の問合せの副問合せを含む条件を評価します。

比較述語では，外側の問合せを検索する場合に HiRDB がインデクスを使用した方がよいと判断したときには，インデクスを使用して検索します。

ROW VALUE SUBQ の処理方式を次の図に示します。

図 4–49　ROW VALUE SUBQ の処理方式



例を次に示します。

例：

```
SELECT T1.C1 FROM T1 WHERE T1.C2<(SELECT MAX(C2) FROM T2)
```

まず，副問合せの表 T2 を検索して，MAX(T2.C2)の値を取り出します（作業表は作成しません）。次に，外側の問合せ中の副問合せを含む条件を，MAX(T2.C2)の値を使用して評価します。

(d)　ハッシュ実行

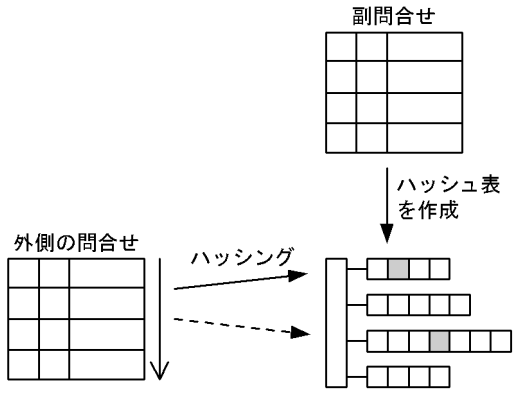
HASH SUBQ

限定述語の右側，及び IN 述語の右側の表副問合せに適用します。

まず，副問合せの選択式の値を求めます。このとき，選択式の値からハッシュ表を作成します。次に，外側の問合せを実行し，限定述語及び IN 述語の左側に指定した列の値でハッシングして，副問合せから作成したハッシュ表と突き合わせて検索します。

HASH SUBQ の処理方式を次の図に示します。

図 4–50　HASH SUBQ の処理方式



例を次に示します。

例：

```
SELECT T1.C1 FROM T1 WHERE T1.C2=ANY(SELECT C2 FROM T2)
```

まず，副問合せの表 T2 を検索して，T2.C2 の値からハッシュを作成します。次に，外側の問合せを実行し，T1.C2 の値でハッシングして，副問合せから作成したハッシュ表と突き合わせて検索します。

4.5.9　外への参照のある副問合せの実行方式

(1)　実行方式の種類

外への参照のある副問合せの実行方式及び特徴を次の表に示します。

表 4–15　外への参照のある副問合せの実行方式及び特徴

実行方式	処理方式	長所	短所
ネストループ作業表実行	外側の問合せを 1 行検索するごとに，副問合せを実行して作業表を作成して，副問合せを含む条件を評価する方式です。	副問合せの探索条件のうち，外への参照列を含む条件に対して，インデクスを使用できます。したがって，副問合せの探索条件が，インデクスを使用することでサーチ範囲を絞り込める場合に高速に検索できます。 外側の問合せの検索で，外への参照列が同値の行を連続して検索した場合には，副問合せの検索を省略できます。	外側の問合せのヒット件数が多い場合には，性能が悪くなります。
ネストループ行値実行	外側の問合せを 1 行検索するごとに，副問合せを実行して（作業表は作成しない），副問合せを含む条件を評価する方式です。	副問合せの探索条件のうち，外への参照列を含む条件に対してインデクスを使用できます。したがって，副問合せの探索条件が，インデクスを使用することでサーチ範囲を絞り込める場合には高速に検索できます。 外側の問合せの検索で，外への参照列が同値の行を連続して検索した場合には，副問合せの検索を省略できます。	外側の問合せのヒット件数が多い場合には，性能が悪くなります。
ハッシュ実行	あらかじめ副問合せ結果からハッシュ表を作成しておいて，外側の問合せを 1 行取り出すごとに外側の問合せの値をハッシングし，ハッシュ表と突き合わせをする方式です。	外への参照列を含む条件を除いた副問合せのヒット件数が少なく，外側の問合せの件数が多い場合，高速に検索できます。	外への参照列を含む条件にインデクスを使用できません。 外への参照列を含む条件を除いた副問合せのヒット件数が多い場合，使用する作業表用バッファサイズが大きくなります。使用する最大作業表用バッファサイズを指定できますが，作業表バッファ領域が一杯になった場合は，いったんファイルに退避するため，性能が悪くなります。 副問合せがジョインしている場合，外への参照列を含む条件の評価はジョイン後になります。

(2) 処理方式

(a) ネストループ作業表実行

NESTED LOOPS WORK TABLE SUBQ

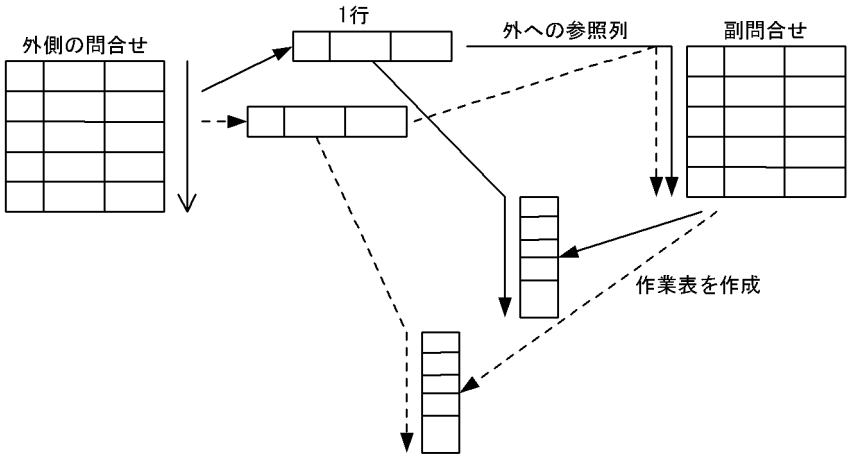
限定述語の右側，及び IN 述語の右側の表副問合せに適用します。

まず，外側の問合せを実行します。このとき，外側の問合せを 1 行取り出すごとに外への参照列の値を使用して副問合せを実行し，副問合せの選択式の値を求めて作業表を作成します。次に，副問合せから作成した作業表を使用して，外側の副問合せを含む条件を評価します。

外側の問合せは 1 行ずつ処理するため，同時に複数の作業表領域を作成することはありません。外側の問合せの行数分，副問合せを実行するため，外側の問合せの行数が多い場合は，性能が悪くなります。

NESTED LOOPS WORK TABLE SUBQ の処理方式を次の図に示します。

図 4–51　NESTED LOOPS WORK TABLE SUBQ の処理方式



例：

```
SELECT C1 FROM T1
  WHERE C1=ANY(SELECT C1 FROM T2 WHERE C2=T1.C2)
```

注　下線部が外への参照列となります。

外側の問合せを実行します。外側の問合せのすべての行に対して，外への参照列（T1.C2）の値を使用して副問合せを実行し，T2.C1 の値から作業表を作成します。次に，T1.C1 を T2.C1 の作業表と突き合わせて，副問合せを含む条件を評価します。

(b) ネストループ行値実行

NESTED LOOPS ROW VALUE SUBQ

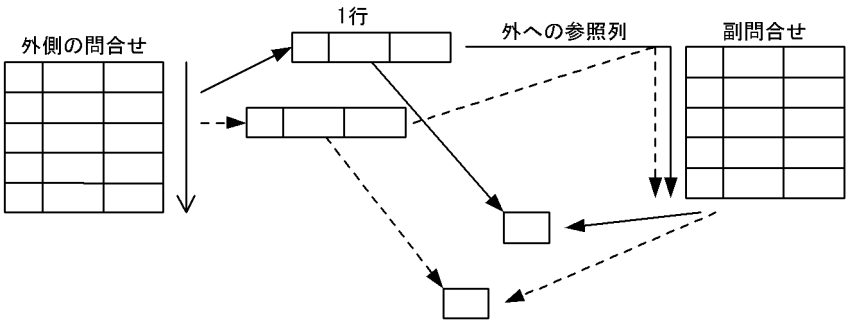
行副問合せ，スカラ副問合せ，及び EXISTS 述語に対して適用します。

まず，外側の問合せを実行します。このとき，外側の問合せを 1 行取り出すごとに外への参照列の値を使用して副問合せを実行し，副問合せの選択式の値を求めます（作業表は作成しません）。次に。副問合せ結果の値を使用して，外側の問合せの副問合せを含む条件を評価します

外側の問合せの行数分，副問合せを実行するため，外側の問合せの行数が多い場合は性能が悪くなります。

NESTED LOOPS ROW VALUE SUBQ の処理方式を次の図に示します。

図 4–52　NESTED LOOPS ROW VALUE SUBQ の処理方式



例：

```
SELECT C1 FROM T1
  WHERE C1=(SELECT MAX(C1) FROM T2 WHERE C2=T1.C2)
```

注　下線部が外への参照列となります。

外側の問合せを実行します。外側の問合せのすべての行に対して，外への参照列（T1.C2）の値を使用して副問合せを検索し，MAX(T2.C1)の値を取り出します（作業表は作成しません）。次に，外側の問合せ中の副問合せを含む条件を評価します。

(c)　ハッシュ実行

HASH SUBQ

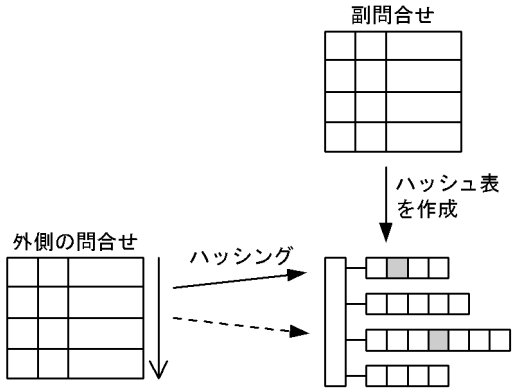
EXISTS 述語，比較述語の右側，限定述語の右側，及び IN 述語の右側の表副問合せに適用します。

まず，外への参照列を含む条件を除いて副問合せを実行し，問合せの選択式の値を求めます。このとき，副問合せ中の=で比較している探索条件から，外への参照列で絞り込んでいる列を使用してハッシュ表を作成します（=ANY，=SOME，IN 述語の場合は，更に選択式の列も使用してハッシュ表を作成します）。

次に，外側の問合せを実行し，1 行取り出すごとに外への参照列となる値でハッシングして（=ANY，=SOME，IN 述語の場合は，更に述語の左辺の列も使用してハッシングします），副問合せから作成したハッシュ表と突き合わせて検索します。

HASH SUBQ の処理方式を次の図に示します。

図 4–53　HASH SUBQ の処理方式



EXISTS 述語と比較述語の例を次に示します。

例１：EXISTS 述語の場合

```
SELECT T1.C1 FROM T1
  WHERE EXISTS(SELECT * FROM T2 WHERE C1='a' AND C2=T1.C2)
```

注　下線部が外への参照列となります。

まず，外への参照列が含まれる条件を除いて副問合せを評価し，外への参照列を使用して絞り込んでいる副問合せの列（T2.C2）の値でハッシュ表を作成します。次に，外側の問合せを実行し，外への参照列（T1.C2）の値でハッシングして，副問合せから作成したハッシュ表と突き合わせて EXISTS 述語を評価します。

例２：比較述語の場合

```
SELECT T1.C1 FROM T1
  WHERE T1.C3<(SELECT T2.C3 FROM T2 WHERE C1='a' AND C2=T1.C2)
```

注　下線部が外への参照列となります。

まず，外への参照列が含まれる条件を除いて副問合せを評価し，外への参照列で絞り込んでいる副問合せの列（T2.C2）からハッシュ表を作成します。次に，外側の問合せを実行し，外への参照列（T1.C2）の値でハッシングして，副問合せから作成したハッシュ表と突き合わせて外への参照列を含む条件を評価し，真の場合は更に比較述語（<）を評価します。

4.5.10　ハッシュジョイン，副問合せのハッシュ実行を適用する場合の準備

SQL 拡張最適化オプションで「ハッシュジョイン，副問合せのハッシュ実行」を適用する場合に，設定しておく項目について説明します。

(1)　事前に設定する項目

ハッシュジョイン，副問合せのハッシュ実行を適用する場合，次の項目を設定しておく必要があります。

- **ハッシュ表サイズ**

- **作業表用バッファの確保方式**
- **作業表用バッファサイズ**

(a)　ハッシュ表サイズ

システム定義の pd_hash_table_size オペランド，又はクライアント環境定義の PDHASHTBLSIZE でハッシュ表サイズを設定します。ハッシュ表サイズは，ハッシュ表最大行長を算出後，次の計算式で求めた値以上を設定してください。

また，ハッシュ表サイズはシステム定義の pd_work_buff_size オペランド，又は pd_work_buff_expand_limit オペランドより小さい値を設定してください。同じ値，又は大きい値を指定すると，ハッシュジョイン，副問合せのハッシュ実行時にエラーになります。

```
ハッシュ表サイズ（単位：キロバイト）
　　≧↑（ハッシュ表最大行長(単位：バイト)×2＋32）÷128↑×128
```

ハッシュ表最大行長：

個々の SELECT 文について，次の単位でハッシュ表行長を算出します。その中で最大のもの（ハッシュ表最大行長）を求めます。

- 複数の表を＝で結合している問合せ指定
- 副問合せ（次のどれかに該当する場合）
  - ＝ ANY の限定述語の，右側の表副問合せ
  - ＝ SOME の限定述語の，右側の表副問合せ
  - IN 述語の右側の表副問合せ
  - 上記以外で，探索条件中に＝で外への参照列を指定している副問合せ

ハッシュ表行長の算出方法を次に示します。

複数の表を＝で結合している問合せ指定

1. =で結合している表の選択式及び探索条件中に指定した列について，表ごとに次の計算式から行長を求めます。

$$\text{表ごとの行長}=\sum_{i=1}^{n}(ai)+2\times n$$

n：該当する表の選択式及び探索条件中の列数

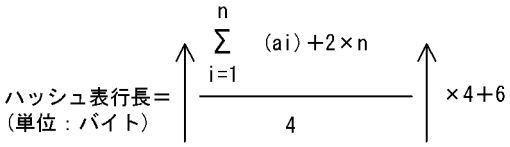
2. 1 で求めた表ごとの行長の中で，最小のもの以外を使用し，次の計算式でハッシュ表行長を求めます。

$$\text{ハッシュ表行長（単位：バイト）}=\uparrow\frac{\sum_{j=1}^{m-1}\text{表ごとの行長}j}{4}\uparrow\times 4+6$$

m：FROM句中の＝で結合している表数

副問合せ

副問合せの選択式中に指定した列，及び探索条件中の外への参照列を含む述語に指定した列について，次の計算式からハッシュ表行長を求めます。



ai：

i 番目のデータ長です。データ長については，マニュアル「HiRDB システム導入・設計ガイド」を参照してください。ただし，＝で結合している表の選択式にだけ指定した，定義長 256 バイト以上の文字データ（各国文字，混在文字含む）の場合は 12 となります。

上記から求めたサイズのハッシュ表には，1,500～2,000 行を格納できます。この行数に比べて，ジョインの内表件数や副問合せの検索件数が多い場合は，バケット分割が複数回実行されて，性能が良くならないことがあります。この場合，次の一括ハッシュジョインのハッシュ表サイズを算出して設定するか，又は「ハッシュ表サイズのチューニング方法」を参照してチューニングしてください。

```
一括ハッシュジョインのハッシュ表サイズ（単位：キロバイト）
  ＝↑(ハッシュ表データページ数＋ハッシュ表管理テーブルページ数)
        ÷1セグメントページ数↑×128
```

```
ハッシュ表データページ数
  ＝↑ハッシュ表行数÷MIN｛↓(ハッシュ表ページ長-48)÷ハッシュ表行長↓，255｝↑
    ＋63
```

```
ハッシュ表管理テーブルページ数
  ＝↑(16×ハッシュ表行数
        ＋(↑(ハッシュ表データページ数×ハッシュ表ページ長＋16×ハッシュ表行数)
              ÷(1セグメントページ数×ハッシュ表ページ長)↑×8)＋8)
        ÷ハッシュ表ページ長↑
```

```
1セグメントページ数
  ＝↓(128×1024)÷ハッシュ表ページ長↓
```

ハッシュ表ページ長は，次の表のハッシュ表行長から求めてください。

ハッシュ表行長	ハッシュ表ページ長
0～1012	4096
1013～2036	8192
2037～4084	16384
4085～16360	32768
16361～32720	↑(ハッシュ表行長＋ 48)÷2048↑×2048 ハッシュ表行長：

ハッシュ表行長	ハッシュ表ページ長
	一括ハッシュジョインの対象がジョインの場合は，ジョインの内表の件数です。対象が副問合せの場合は，探索条件中の外への参照列を含む述語を除いた副問合せの検索件数です。

(b)　作業表用バッファの確保方式

作業表用バッファの確保方式を，サーバプロセス単位でバッファ一括確保（pool 又は pool2）に設定しておく必要があるため，システム定義の pd_work_buff_mode オペランドに pool 又は pool2 を指定します。

(c)　作業表用バッファサイズ

ハッシュ表は，作業表用バッファ内に確保されます。指定したハッシュ表サイズより作業表用バッファサイズ，又は作業表用バッファの増分確保の上限サイズが小さいと，作業表用バッファ不足でエラーとなります。したがって，システム定義の pd_work_buff_size オペランド又は pd_work_buff_expand_limit オペランドには，次の計算式で求めた値以上を設定してください。

```
作業表用バッファサイズ（単位：キロバイト）
  ≧（ハッシュ表サイズ(単位：キロバイト)×2＋256）
    ×SELECT文のハッシュジョイン最大数＋128
```

SELECT 文のハッシュジョイン最大数：

個々の SELECT 文のハッシュジョイン数を次の計算式から求めて，その中で最大のものを SELECT 文のハッシュジョイン最大数とします。なお，ハッシュジョイン数は，アクセスパス表示ユティリティ（pdvwopt）で出力される結合処理情報の Join Type が"HASH JOIN"となっている項目をカウントすることでも求められます。

```
SELECT文でのハッシュジョイン数
  ＝((＝で結合している表数)-(＝で結合している問合せ指定数))
    ＋(＝ANYの限定述語の数)
    ＋(＝SOMEの限定述語の数)
    ＋(IN(副問合せ)の指定数)
    ＋(そのほかの副問合せで，探索条件中に＝で
          外への参照列を指定している副問合せ数)
```

なお，複数のカーソルを同時に開いて検索する場合は，カーソルごとに算出した値を合計します。

```
(例)
  SELECT A.A1,B.B2,C.C3 FROM A,B,C  .....................3-1
    WHERE A.A1=B.B1 AND A.A2=B.B2
      AND B.B3=C.C3
      AND A.A1=C.C1
      AND A.A4=ANY(SELECT D.D4 FROM D)  .................1
      AND A.A5=SOME(SELECT E.E5 FROM E)  ................1
      AND A.A6 IN(SELECT F.F6 FROM F WHERE F.F1=A.A1)  ...1
      AND EXISTS(SELECT G.G1 FROM G WHERE G.G1=B.B1)   ...1

この例の場合，(3-1)＋1＋1＋1＋1となるので，このSELECT文のハッシュジョイン数は6になります。
```

なお，上記の作業表用バッファサイズの計算式で求めた値に，更に 4,096 キロバイト程度の余裕があれば，バケット分割のときの入出力単位が大きくなるため，性能が良くなります。

すべてバケット分割をしない一括ハッシュジョインとなる場合，次の計算式を満たしていれば実行できます。

```
作業表用バッファサイズ（単位：キロバイト）
  ≧ハッシュ表サイズ(単位：キロバイト)
    ×SELECT文のハッシュジョイン最大数＋384
```

(2)　ハッシュ表サイズのチューニング方法

(a)　利用するチューニング情報

ハッシュ表サイズは，次のどちらかのチューニング情報を基にチューニングできます。

- UAP 統計レポート（クライアント環境定義 PDUAPREPLVL を指定）
- 統計解析ユティリティの UAP に関する統計情報

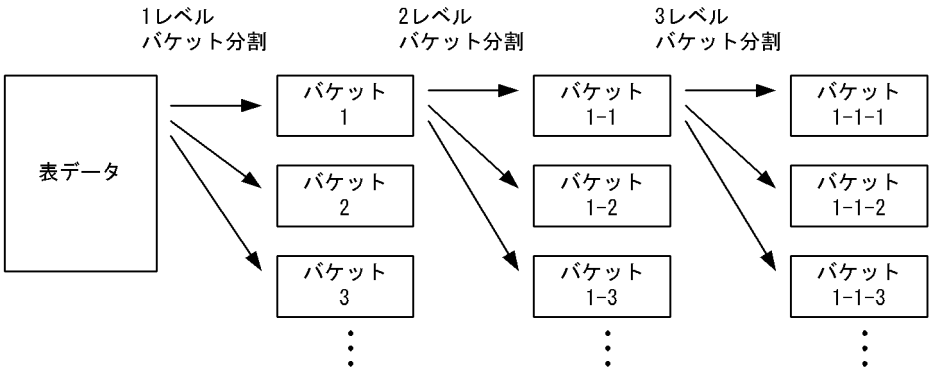
UAP 統計レポートについては「UAP 統計レポート機能」を，統計解析ユティリティについてはマニュアル「HiRDB コマンドリファレンス」を参照してください。

(b)　チューニング情報から分かる項目

ハッシュ表サイズのチューニング情報を取得すると，次のことが分かります。

- すべてのデータを一括してハッシュ表に展開する一括ハッシュジョインになっているか，又はバケット単位にハッシュ表に展開するバケット分割ハッシュジョインになっているか
- バケット分割ハッシュジョインの場合に，バケットの再分割をしているか
- バケット分割ハッシュジョインの場合に，一括ハッシュジョインにするにはハッシュ表サイズをどのくらいにしたらよいか
- バケット分割ハッシュジョインの場合に，バケットの再分割をしないようにするためにはハッシュ表サイズをどのくらいにしたらよいか

なお，バケットの再分割とは，バケットの大きさがハッシュ表サイズを超える場合に，最大 3 レベルまでバケット分割を再帰的に繰り返すことをいいます。例を次に示します。



1 回のバケット分割での分割数は，次の計算式から決まります。

```
バケット分割数
  ＝MIN｛↓(ハッシュ表サイズ÷2)÷ハッシュ表ページ長↓，64｝
```

また，一括ハッシュジョインの場合でも，ジョインの内表については 1 レベルバケット分割をします。

(c)　チューニング方法

ハッシュ表サイズのチューニング方法を次の表に示します。

表 4–16　ハッシュ表サイズのチューニング方法

チューニング情報 （単位：キロバイト）	チューニング方法
最大一括ハッシュ表サイズ	ハッシュ表サイズにこの値以上が設定されていれば，すべてバケット分割をしない一括ハッシュジョインになります。※1 また，この値が，ハッシュ表サイズに指定できる上限を超えている場合は，一括ハッシュジョインにはできません。 この値が 0 の場合は，ハッシュジョイン，副問合せのハッシュ実行がされていません。
1 レベル最大バケットサイズ	ハッシュ表サイズにこの値以上が設定されていれば，バケット分割が 1 レベルで完了しています。また，バケット分割が 2 レベル以上の場合は，ハッシュ表サイズにこの値を指定することで，バケット分割が 1 レベルで完了するようになります。※2 すべてバケット分割をしない一括ハッシュジョインの場合は，この値には 0 が表示されます。
2 レベル最大バケットサイズ	ハッシュ表サイズにこの値以上が設定されていれば，バケット分割が 2 レベルで完了しています。また，バケット分割が 3 レベル以上の場合は，ハッシュ表サイズにこの値を指定することで，バケット分割が 2 レベルで完了するようになります。※2 2 レベルバケット分割が一度もされなかった場合は，この値には 0 が表示されます。
3 レベル最大バケットサイズ	ハッシュ表サイズにこの値以上が設定されていれば，バケット分割が 3 レベルで完了しています。 ハッシュ表サイズがこの値より小さい場合，1 バケットを部分的にハッシュ表展開していく処理となり，処理効率が悪くなります。この場合，ハッシュ表サイズにこの値以上を設定するようにしてください。※2 また，ハッシュジョイン，副問合せのハッシュ実行を適用しないようにした方が，性能が良くなる場合もあります。 2 レベルバケット分割が一度もされなかった場合は，この値には 0 が表示されます。

注※1

ハッシュ表サイズを大きくすると，バケット分割数を求める計算式に従って，1 回のバケット分割数が増加することがあります。このため，チューニング情報取得時よりも，大きなハッシュ表サイズが必要となることがあります。

チューニング情報を利用してハッシュ表サイズを大きくした場合，再度チューニング情報を取得し，意図したとおりになっていなければ，新たに取得したチューニング情報に合わせて，再びハッシュ表サイズを大きくする必要があります。

注※2

ハッシュ表サイズを大きくすると，バケット分割数を求める計算式に従って，1 回のバケット分割数が増加することがあります。このため，チューニング情報取得時よりも，小さなハッシュ表サイズでも，意図したレベルでバケット分割が完了することがあります。

これに対して，ハッシュ表サイズを小さくすると，1 回のバケット分割数を減少させることがあるため，チューニング情報取得時と同じレベルでバケット分割が完了しなくなることがあります。したがって，このチューニング情報は，ハッシュ表サイズを大きくしていくときに利用してください。

4.5.11　探索高速化条件の導出

探索高速化条件とは，WHERE 句の探索条件，FROM 句の ON 探索条件から，CNF 変換又は条件推移で新たに導出される条件のことをいいます。探索高速化条件を導出すると，検索する行が早い段階で絞り込まれるため，検索性能が向上します。

探索高速化条件を導出する場合，導出の元となる探索条件は残すため，絞り込みに使用できない条件は生成しないで，導出した条件の中で最適な条件だけを生成できます。

探索高速化条件を導出すると，HiRDB がアクセスパス（表の検索方法，結合方法，結合順序など）を決定する場合に，新しく導出した探索高速化条件も考慮して最適化をします。そのため，探索高速化条件を導出すると，アクセスパスは次のように変わることがあります。

- 検索する行が早い段階で絞り込まれると判断して，インデクスを使用した検索が選ばれやすくなります。
- 結合条件に OR を指定している場合に，CNF 変換と OR の簡約化で結合条件を OR の外側に抜き出せるときは，直積以外にもネストループジョイン，マージジョイン，及びハッシュジョインができるようになります。
- 結合する場合に，片方の表だけに制限条件が付くときはネストループジョインが選ばれやすくなり，両方の表に制限条件が付くときはマージジョイン又はハッシュジョインが選ばれやすくなります。

なお，複雑な条件から探索高速化条件を導出すると，探索高速化条件を生成する時間，及び実行時の評価時間が長くなるため，SQL によっては性能が低下することがあります。

(1)　探索高速化条件の適用範囲

探索高速化条件を導出するかどうかは，SQL 最適化オプション及び SQL 拡張最適化オプションの指定値によって変わります。SQL 最適化オプション及び SQL 拡張最適化オプションと，探索高速化条件の導出の関係を次の表に示します。

表 4–17　SQL 最適化オプション及び SQL 拡張最適化オプションと探索高速化条件の導出の関係

種類	導出元となる条件	導出される条件	SQL 最適化オプション及び SQL 拡張最適化オプションの指定	
			探索高速化条件の導出をしない	探索高速化条件の導出をする※1
CNF 変換	1 表の OR の条件	1 表条件	×	×
	2 表以上にわたる OR の条件	1 表条件	○※2	○
		OR の簡約化で結合条件(列=列)が抜き出せる場合	○※2	○
		上記以外の 2 表以上の条件	×	×
条件推移	表 A，B の結合条件と，表 A の条件	表 B の 1 表条件	×	○
	表 A，B の結合条件と，表 A，C の結合条件	表 B，C の結合条件	×	×

（凡例）

○：探索高速化条件を生成します。

×：探索高速化条件を生成しません。

注※1

SQL 最適化オプションに「探索高速化条件の導出」を指定します。

注※2

探索高速化条件の導出元の検索に直積が指定されている場合，探索高速化条件を生成しますが，次に示す条件によって異なります。探索高速化条件の導出の可否とその条件を次に示します。

導出元の検索		導出の条件		導出される探索高速化条件	導出可否
2 表検索	直積となる	OR の簡約化によって結合条件(列=列)が抜き出せない場合	導出元の探索条件に導出する 1 表条件と同じ表の 1 表条件がある場合	なし	×
			導出元の探索条件に導出する 1 表条件と同じ表の 1 表条件がない場合	1 表条件	○
			−	結合条件(列=列)	−
				上記以外の 2 表条件	×

導出元の検索		導出の条件		導出される探索高速化条件	導出可否
		OR の簡約化によって結合条件(列=列)が抜き出せる場合	導出元の探索条件に導出する 1 表条件と同じ表の 1 表条件がある場合	なし	×
			導出元の探索条件に導出する 1 表条件と同じ表の 1 表条件がない場合	1 表条件	○
			−	結合条件(列=列)	○
				上記以外の 2 表条件	×
	直積とならない	−			×
3 表以上の検索	すべて直積となる	OR の簡約化によって結合条件(列=列)が抜き出せない場合	導出元の探索条件に導出する 1 表条件と同じ表の 1 表条件がある場合	なし	×
			導出元の探索条件に導出する 1 表条件と同じ表の 1 表条件がない場合	1 表条件	○
			−	結合条件(列=列)	−
				上記以外の 2 表条件	×
		一部が OR の簡約化によって結合条件(列=列)が抜き出せる場合	導出元の探索条件に導出する 1 表条件と同じ表の 1 表条件がある場合	なし	×
			導出元の探索条件に導出する 1 表条件と同じ表の 1 表条件がない場合	1 表条件	○
			−	結合条件(列=列)	○
				上記以外の 2 表条件	×
		OR の簡約化によって結合条件(列=列)が抜き出せる場合	導出元の探索条件に導出する 1 表条件と同じ表の 1 表条件がある場合	なし	×
			導出元の探索条件に導出する 1 表条	1 表条件	○

導出元の検索		導出の条件		導出される探索高速化条件	導出可否
			件と同じ表の 1 表条件がない場合		
			–	結合条件(列=列)	○
				上記以外の 2 表条件	×
	一部が直積，その他が直積にならない	OR の簡約化によって結合条件(列=列)が抜き出せない場合	導出元の探索条件に導出する 1 表条件と同じ表の 1 表条件がある場合	なし	×
			導出元の探索条件に導出する 1 表条件と同じ表の 1 表条件がない場合	1 表条件	○
			–	結合条件(列=列)	–
				上記以外の 2 表条件	×
		一部が OR の簡約化によって結合条件(列=列)が抜き出せる場合	導出元の探索条件に導出する 1 表条件と同じ表の 1 表条件がある場合	なし	×
			導出元の探索条件に導出する 1 表条件と同じ表の 1 表条件がない場合	1 表条件	○
			–	結合条件(列=列)	○
				上記以外の 2 表条件	×
		OR の簡約化によって結合条件(列=列)が抜き出せる場合	導出元の探索条件に導出する 2 表条件と同じ表の 1 表条件がある場合	なし	×
			導出元の探索条件に導出する 2 表条件と同じ表の 1 表条件がない場合	1 表条件	○
			–	結合条件(列=列)	○
				上記以外の 2 表条件	×
	直積とならない	–			×

（凡例）

○：導出する

×：導出しない

－：該当しない

(2)　CNF 変換での探索高速化条件の導出

CNF 変換とは，OR で結合した条件（DNF 形式(Disjunctive Normal Form：選言標準形)）を，AND で結合された等価な条件（CNF 形式(Conjunctive Normal Form：連言標準形)）に変換することをいいます。WHERE 句の探索条件，及び FROM 句の ON 探索条件に対して CNF 変換をすることで，探索高速化条件を導出します。

(a)　CNF 変換で導出される探索条件

CNF 変換で導出される探索条件を次に示します。

- OR で結合した 2 表以上にわたる条件に対して，CNF 変換することで 1 表条件が生成できる場合に，この条件を探索高速化条件として導出します。1 表条件を導出することで結合する件数を絞り込めます。
- OR で結合したすべての条件に 2 表の同じ結合条件（列＝列だけ）が含まれる場合，この条件を CNF 変換することで（結合条件 OR ... OR 結合条件）が導出できます。そして，OR で結合した結合条件がすべて同じ条件で重複排除できる場合（OR の簡約化），結合条件を探索高速化条件として導出します。結合条件を導出することで，直積処理がなくなり，性能が向上します。

なお，SQL 最適化オプション及び SQL 拡張最適化オプションの指定によって，探索高速化条件を導出するかどうかが決まります。SQL 最適化オプション及び SQL 拡張最適化オプションと，探索高速化条件の導出の関係については，「探索高速化条件の適用範囲」を参照してください。

(b)　CNF 変換をしない条件

次のどれかに該当する場合，CNF 変換での探索高速化条件の導出はしません。

- 導出後の探索条件に副問合せが含まれる場合
- 外結合で，ON 探索条件に指定した導出元の条件に対して，導出後の条件が外表に閉じる条件となる場合
- 外結合で，WHERE 句に指定した導出元の条件に対して，導出後の条件が内表に閉じる条件となる場合
- 外結合で，WHERE 句に指定した導出元の条件に対して，導出後の条件が 2 表以上の条件となる場合
- 探索高速化条件を導出した結果，論理演算の最大ネスト数が 255 を超える場合
- HAVING 句に探索条件を指定した場合
- 導出後の探索条件が直積後のジョインの結合条件となる場合

(3)　条件推移での探索高速化条件の導出

条件推移とは，二つ以上の条件から新たな条件を導き出すことをいいます。

条件推移で導出される探索条件を次に示します。

- 結合条件を介した 1 表条件の推移による探索高速化条件の導出
- 結合条件の推移での探索高速化条件の導出（UNIX 版限定）

なお，SQL 最適化オプション及び SQL 拡張最適化オプションの指定によって，探索高速化条件を導出するかどうかが決まります。SQL 最適化オプション及び SQL 拡張最適化オプションと，探索高速化条件の導出の関係については，「探索高速化条件の適用範囲」を参照してください。

(a)　結合条件を介した 1 表条件の推移による探索高速化条件の導出

2 表の結合条件（列＝列だけ）と，結合列を含む 1 表条件がある場合に，結合相手の列に対して 1 表条件を導出します。例を次に示します。

```
T1.C1 = T2.C1 AND T1.C1 > 10
→T1.C1 = T2.C1 AND T1.C1 > 10 AND T2.C1 > 10
```

下線部分が，導出した探索高速化条件となります。

- **条件推移の対象となる 1 表条件**

  条件推移の対象となる 1 表条件を次に示します。

  - 列指定 比較演算子 ｛値指定｜外への参照列｝

    比較演算子（=，<>，^=，!=，<，<=，>，>=）の左右が逆の場合にも条件推移をします。
  - 列指定 IS 〔NOT〕 NULL
  - 列指定 〔NOT〕 IN （値指定〔，値指定〕...）
  - 列指定 〔NOT〕 LIKE パターン文字列 〔ESCAPE エスケープ文字〕

    結合列のデータ長が異なる場合，パターン文字列が定数で，かつ前方一致のときにだけ条件推移をします。
  - 列指定 〔NOT〕 XLIKE パターン文字列 〔ESCAPE エスケープ文字〕

    結合列のデータ長が異なる場合には条件推移をしません。
  - 列指定 BETWEEN ｛値指定｜外への参照列｝ AND ｛値指定｜外への参照列｝
  - 列指定 〔NOT〕 SIMILAR TO パターン文字列 〔ESCAPE エスケープ文字〕

    結合列のデータ長が異なる場合，パターン文字列が定数で，かつ LIKE 述語と等価となる前方一致のときにだけ条件推移をします。
- **条件推移をしない条件**

  次のどれかに該当する場合，条件推移はしません。

  - 外結合の場合
  - 内結合の場合で，WHERE 句の探索条件と FROM 句の ON 探索条件との間での条件推移の場合（3 表以上の内結合をしている場合，複数の ON 探索条件間での条件推移はします）
  - 結合列のデータ型が固定長と可変長の比較の場合

- 結合列のデータ型が FLOAT 又は SMALLFLT の場合
- 繰返し列を使用して結合している場合
- 探索高速化条件を導出した結果，論理演算の最大ネスト数が 255 を超える場合
- HAVING 句に探索条件を指定した場合

(b) 結合条件の推移での探索高速化条件の導出（UNIX 版限定）

2 表の結合条件（列＝列だけ）と，どちらか片方の列と別の表の列との結合条件（列＝列だけ）がある場合に，結合条件がない列同士から新たな結合条件を導出します。相関名の指定がある場合は，相関名が異なれば別の表とみなします。例を次に示します。

```
T1.C1 = T2.C1 AND T2.C1 = T3.C1
→T1.C1 = T2.C1 AND T2.C1 = T3.C1 AND T1.C1 = T3.C1

下線部分が，導出した探索高速化条件となります。
```

- **結合条件の推移をしない条件**

  次のどれかに該当する場合，結合条件の推移をしません。

  - 外結合の場合
  - 内結合の場合で，WHERE 句の探索条件と FROM 句の ON 探索条件との間での条件推移（3 表以上の内結合をしている場合，複数の ON 探索条件間での条件推移はします）
  - 結合列のデータ型が固定長と可変長の比較の場合
  - 結合列のデータ型が FLOAT 又は SMALLFLT の場合
  - 探索高速化条件を導出した結果，論理演算の最大ネスト数が 255 を超える場合
  - HAVING 句に探索条件を指定した場合

4.6　データ保証レベル

データ保証レベルとは，検索したデータをトランザクションのどの時点まで保証するのかを指定するものです。データ保証レベルには，0〜2 のレベルがあります。データをほかのユーザに更新させない場合，ほかのユーザが更新中のデータを参照したい場合など，目的に応じて指定します。

4.6.1　データ保証レベルの指定方法

システム単位でデータ保証レベルを指定したい場合は，システム定義の pd_isolation_level オペランドを指定します。

UAP ごと，又はルーチンごとにデータ保証レベルを指定したい場合は，次の箇所のデータ保証レベルを指定します。

- クライアント環境定義の PDISLLVL
- ALTER PROCEDURE の SQL コンパイルオプション
- ALTER ROUTINE の SQL コンパイルオプション
- ALTER TRIGGER の SQL コンパイルオプション
- CREATE PROCEDURE の SQL コンパイルオプション
- CREATE TRIGGER の SQL コンパイルオプション
- CREATE TYPE の手続き本体の SQL コンパイルオプション

SQL ごとにデータ保証レベルを指定したい場合は，その SQL 文に排他オプションを指定します。カーソル宣言時の排他オプションと表操作時の排他オプションの関係については「カーソル宣言と排他の関係」を参照してください。

データ保証レベルと排他オプションを同時に指定した場合，排他オプションの指定が有効となります。データ保証レベルと排他オプションとの関係を次の表に示します。

表 4–18　データ保証レベルと排他オプションとの関係

データ保証レベル	排他オプション
0	WITHOUT LOCK NOWAIT※1※3
1	WITHOUT LOCK WAIT※3
2	WITH SHARE LOCK 又は EXCLUSIVE LOCK※2

注※1

FOR UPDATE 句を指定したカーソル宣言，及び FOR UPDATE 句を指定した動的 SELECT 文に対しては，データ保証レベルに 0 を指定しても無視され，1 が仮定されます。

注※2

FOR UPDATE 句を指定したカーソル宣言，及び FOR UPDATE 句を指定した動的 SELECT 文に対しては，WITH EXCLUSIVE LOCK が仮定されます。

注※3

次の場合，WITH EXCLUSIVE LOCK が仮定されます。

- FOR UPDATE 句を指定したカーソル宣言，及び FOR UPDATE 句を指定した動的 SELECT 文の実行時，クライアント環境定義 PDFORUPDATEEXLOCK に YES を指定している。
- ルーチン定義時に指定する SQL コンパイルオプションのデータ保証レベルの直後に，FOR UPDATE EXCLUSIVE を指定している。

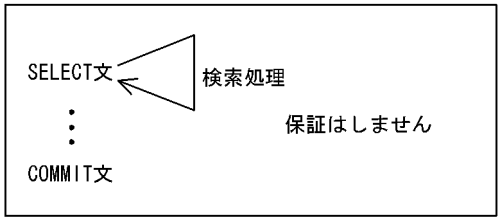
4.6.2　データ保証レベルの種類

(1)　データ保証レベル 0

データ保証レベル 0 は，ほかのユーザが更新中のデータでも，更新完了を待たないで参照する場合に指定します。ほかの保証レベルよりも同時実行性を向上できますが，同一トランザクション中で同じ行を 2 度検索した場合，1 回目と 2 回目の検索結果は同じにならないときもあります。

データ保証レベル 0 のデータの保証範囲を次の図に示します。

図 4–54　データ保証レベル 0 のデータの保証範囲

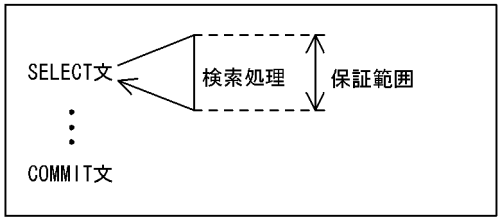


(2)　データ保証レベル 1

データ保証レベル 1 は，検索処理が終了するまで（HiRDB がページ又は行を見終わるまで），一度検索したデータをほかのユーザに更新させない場合に指定します。そのため，より同時実行性を向上できますが，同一トランザクション中で同じ行を 2 度検索した場合，1 回目と 2 回目の検索結果は同じにならないときもあります。

データ保証レベル 1 のデータの保証範囲を次の図に示します。

図 4–55　データ保証レベル 1 のデータの保証範囲

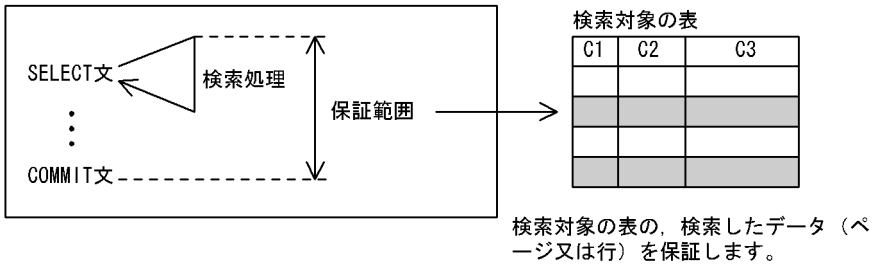


(3)　データ保証レベル 2

データ保証レベル 2 は，トランザクションが終了するまで，一度検索したデータをほかのユーザに更新させない場合に指定します。したがって，検索したデータについてはトランザクション終了まで保証されますが，検索していないデータについては保証されません。同一トランザクション中で同じ行を 2 度検索した場合，追加された行があると 1 回目と 2 回目の検索結果は同じにならないときもあります。

データ保証レベル 2 のデータの保証範囲を次の図に示します。

図 4–56　データ保証レベル 2 のデータの保証範囲



(4)　注意事項

更新を伴うカーソル宣言に対しては，データ保証レベルに 0 を指定しても無視され，1 が仮定されます。

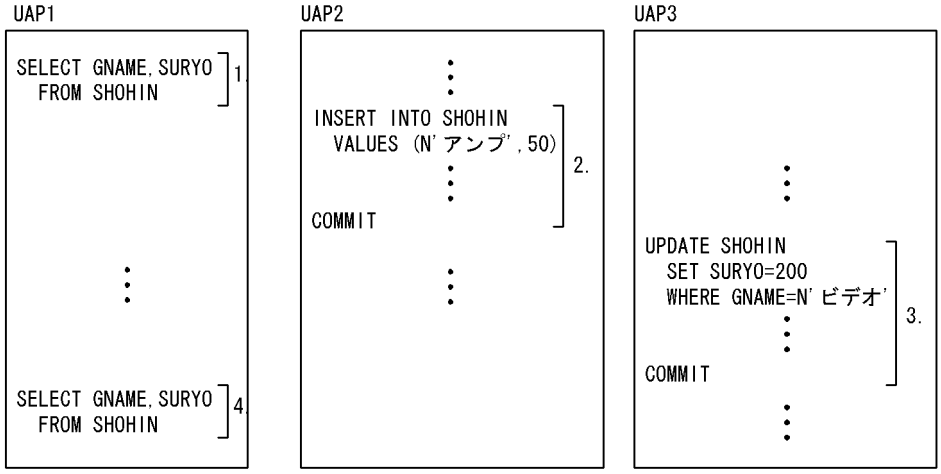
4.6.3　データ保証レベルを指定した場合の検索結果の例

データ保証レベルを指定した場合の検索結果の例を次の図に示します。UAP1 は SHOHIN 表を検索する UAP，UAP2 は SHOHIN 表にデータを挿入する UAP，UAP3 は SHOHIN 表のデータを更新する UAP です。UAP1～UAP3 の 1.～4.は実行順序を表します。

図 4–57　データ保証レベルを指定した場合の検索結果の例

SHOHIN

GNAME	SURYO
テレビ	100
ビデオ	80



上記のように UAP を実行した場合，UAP1 の 1.と 4.の検索結果は次のようになります。なお，UAP1～UAP3 のデータ保証レベルはすべて同じレベルで実行したものとします。

データ保証レベル	UAP1 の検索結果		説　　明
	1.	4.	
0	テレビ 100 ビデオ 80	テレビ 100 ビデオ 200 アンプ 50	同一トランザクション中で同じデータを 2 度検索していますが，1.と 4.の検索結果は異なります。 データは保証されないため，4.の検索結果には，2.と 3.の処理が反映されます。
1	テレビ 100 ビデオ 80	テレビ 100 ビデオ 200 アンプ 50	同一トランザクション中で同じデータを 2 度検索していますが，1.と 4.の検索結果は異なります。 検索処理中のデータは保証されますが，検索処理が終了するとデータは保証されなくなります。そのため，4.の検索結果には，2.と 3.の処理が反映されます。
2	テレビ 100 ビデオ 80	テレビ 100 ビデオ 80 アンプ 50	• 同一トランザクション中で同じデータを 2 度検索していますが，1.と 4.の検索結果は異なります。 • 1.で検索したとき，テレビとビデオの行は保証されますが，2.についてはデータが保証されないため，2.の処理は反映されます。ただし，3.についてはデータが保証されるため，3.の処理は待ち状態となり，4.の検索結果には反映されません。

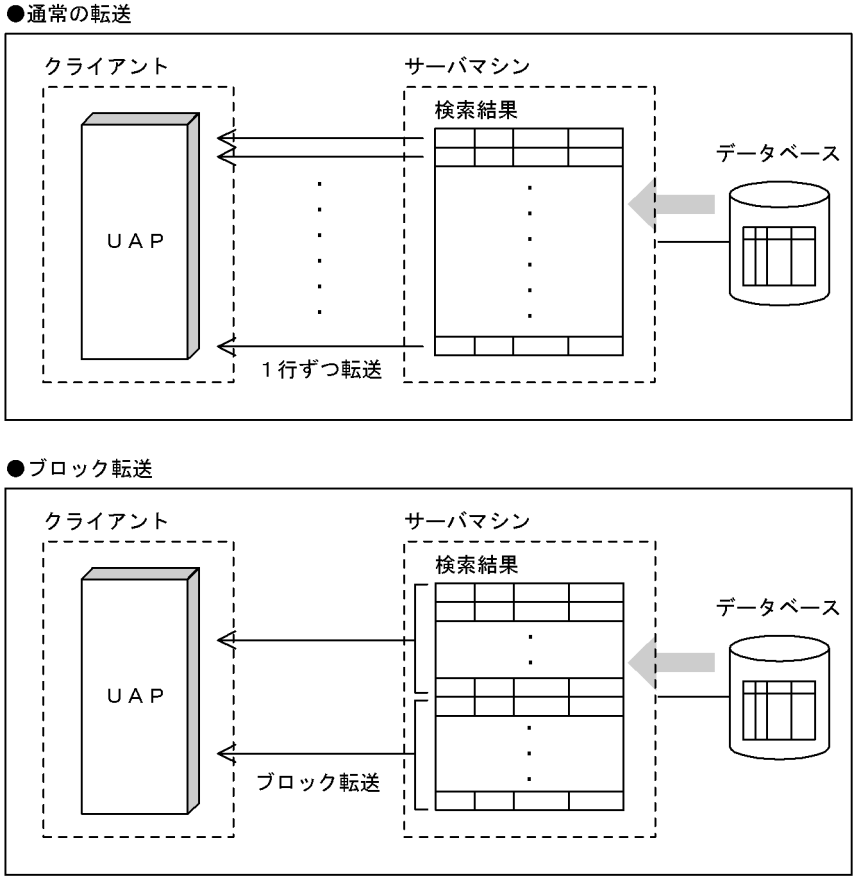
4.7　ブロック転送機能

4.7.1　機能概要

ブロック転送は，HiRDB システムから HiRDB クライアントにデータを転送するときに，任意の行数単位で転送する機能です。HiRDB クライアントから HiRDB システムにアクセスし，大量のデータを検索する場合に有効です。

ブロック転送機能の概要を次の図に示します。

図 4–58　ブロック転送機能の概要



4.7.2　使用方法

ブロック転送機能は，次の条件を両方とも満たす場合に実行されます。

1. クライアント環境定義 PDBLKF に 2 以上の値を指定している場合，又は PDBLKBUFFSIZE に 1 以上の値を指定している場合
2. FETCH 文を指定している場合（ただし，次のどれかの条件に該当する場合を除きます）
   - カーソルを使用した更新

- BLOB 型の選択式がある検索
- クライアント環境定義 PDBINARYBLKF が NO で，かつ定義長が 32,001 バイト以上の BINARY 型の選択式がある検索
- BLOB 位置付け子型，又は BINARY 位置付け子型の変数を使用して結果を受け取る検索で，かつホールダブルカーソルを使用した検索

4.7.3　サーバ，クライアント間の通信バッファサイズの指定

クライアント環境定義 PDBLKBUFFSIZE で，サーバ，クライアント間で使用する通信バッファのサイズを指定できます。

取り出す行数（PDBLKF の指定値）が大きい場合の検索で，PDBLKBUFFSIZE を指定すると，サーバ側が指定値以上に通信バッファメモリを割り当てることを抑止します。ただし，1 行転送する分の通信バッファメモリは確保します。

サーバ，クライアント間の通信バッファのメモリ計算式については，マニュアル「HiRDB システム導入・設計ガイド」の「ブロック転送又は配列 FETCH で必要なメモリ所要量の計算式」を参照してください。

4.7.4　1 回の通信で転送する行数

ブロック転送機能を使用した場合，1 回の通信で転送する行数を次に示します。

PDBLKF の指定値	PDBLKBUFFSIZE の指定値	
	0	1 以上
1	ブロック転送機能は適用されません。	行数=MIN（X，4096）※ X： 次の条件式を満たす n の最大値（指定したバッファサイズに格納できる行の数）となります。ただし，(a-b)＜ ci の場合は 1 となります（i は 1）。 $(a-b) \geqq \sum_{i=1}^{n} ci$ （単位：バイト） **ci**：FETCH 文で受け取る検索結果中の，i 行目のデータ長 **a**：指定したバッファサイズ（PDBLKBUFFSIZE の値×1024） **b**：ヘッダ情報など（864 ＋ 22×d ＋ 2×e） b の計算式中の d 及び e の内容を次に示します。 **d**：SELECT 句で指定する検索項目数 **e**：SELECT 句で指定する検索項目での，BINARY 型の選択式の数
2 以上	行数=PDBLKF の値	行数=MIN（X，Y）※

PDBLKF の指定値	PDBLKBUFFSIZE の指定値	
	0	1 以上
		X：指定したバッファサイズに格納できる行の数（上記の X と同じ） Y：PDBLKF の値

注※

実行する SQL によっては，算出した行数より多く転送することがあります。

4.7.5 注意事項

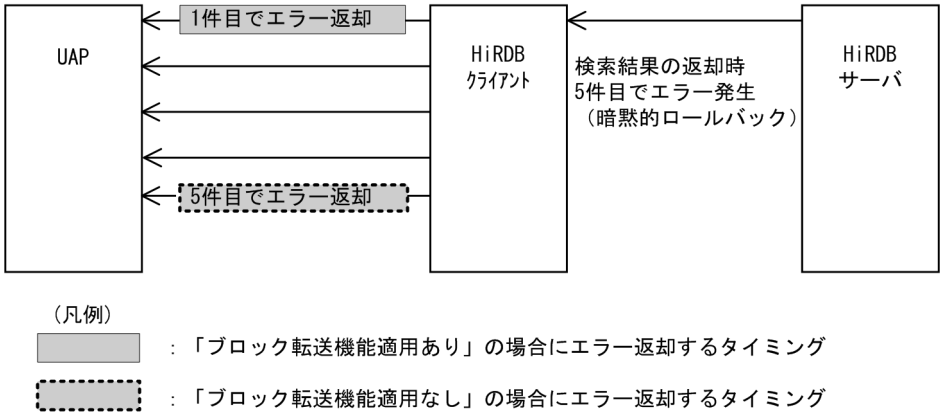
1. ブロック転送機能では，1 回の転送行数を多くすることで通信オーバヘッドが減少するので検索時間を短縮できますが，所要メモリが増加するので使用時には注意が必要です。クライアント環境定義 PDBLKBUFFSIZE を指定すると，通信バッファに使用するメモリサイズを一定値以下に抑えられます。ただし，値が小さ過ぎると，通信回数が削減されないため，ブロック転送機能の効果がなくなります。
2. 問合せ式での検索結果のうち，先頭から n 行だけの検索結果を必要とする場合にブロック転送機能を適用すると，n 行が「1 回の通信で転送する行数」に示す行数未満の場合は，不要な検索結果も取得するため，検索性能が劣化するおそれがあります。その場合は，LIMIT 句を指定して，必要な行数だけを取得するようにしてください。詳細は，「先頭から n 行の検索結果を取得する機能」を参照してください。
3. 次の事象が発生した場合，検索処理を中断して，それまでに検索したデータを返します。
   - 検索中に警告エラーが発生した場合（警告情報とそれまでに検索したデータを返します）※
   - リストを介した検索で，リスト作成時にあった行が削除，又は属性値が削除，更新された場合（該当する事象を示すリターンコードの情報（SQLCODE=+110）とそれまでに検索したデータを返します）

   注※

   警告エラーが発生しても検索処理を中断しない場合があります。中断しない場合，指定行数まで検索処理を続けて，検索中に発生したすべての警告エラーの情報と検索データを返します。
4. ブロック転送機能を使用していて，一つのカーソルの検索結果を複数の FETCH 文で受け取る場合，それぞれの FETCH 文には同じ埋込み変数を指定するか，又は同じ属性の埋込み変数を指定してください。異なる属性の埋込み変数で検索結果を受け取ろうとするとエラーになります。
5. ブロック転送機能適用時，HiRDB サーバから複数行取得した場合に暗黙的ロールバックとなるエラーが発生すると，アプリケーションには HiRDB サーバからの複数行取得終了後の初回検索結果取得時にエラーを返却します。アプリケーションのデバックなどでエラー発生行を正確に受け取りたい場合は，ブロック転送機能を適用しないでアプリケーションを実行してください。

   ブロック転送機能の適用有無による暗黙的ロールバックエラーの返却タイミングを次の図に示します。

図 4–59　ブロック転送機能の適用有無による暗黙的ロールバックエラーの返却タイミング



（凡例）

：「ブロック転送機能適用あり」の場合にエラー返却するタイミング

：「ブロック転送機能適用なし」の場合にエラー返却するタイミング

4.8　配列を使用した機能

4.8.1　配列を使用した FETCH 機能

(1)　概要

FETCH 文で，INTO 句に配列型の埋込み変数を指定するか，又は BY 句の埋込み変数に検索行数を指定すると，検索結果を一度に複数行取得できます。HiRDB クライアントから HiRDB システムにアクセスし，大量のデータを検索する場合に有効です。ブロック転送機能とは異なり，複数行の検索結果を取得することをプログラム内で明示的に記述します。

(2)　使用方法

(a)　静的に実行する場合

FETCH 文の INTO 句に指定した埋込み変数と標識変数を，すべて配列型の変数にしてください。一括して検索する行数は，指定した埋込み変数の最小配列要素数となります。

(b)　動的に実行する場合

次に示す手順で実行してください。

1. PREPARE 文で SELECT 文を前処理します。
2. DESCRIBE 文で前処理した SELECT 文の SQL 記述領域の情報を取得します。
3. SQL 記述領域の SQLDATA に，各データの受け取り領域を設定してください。また，可変長データ型の場合は，1 要素のサイズを SQLSYS に設定してください。
4. FETCH 文の USING DESCRIPTOR 句に SQL 記述領域を指定し，かつ BY 句に埋込み変数を指定してください。一括して検索する行数は，BY 句に指定した埋込み変数で指定します。

(3)　注意事項

1. 配列を使用した FETCH 機能で指定するカーソルは，配列を使用した FETCH 機能専用のカーソルとなります。そのカーソルを使用した場合，ブロック転送機能は無効となります。また，そのカーソルを指定して通常の FETCH を実行する場合でも，注意事項 4.が適用されます。同一モジュール（プリプロセス単位）で，配列を使用した FETCH 機能と通常の FETCH を使用する場合，それぞれ異なるカーソルを使用してください。
2. 配列を使用した FETCH 機能では，通常の FETCH とは異なり，検索の途中で取得する行がなくなって NOT FOUND が発生しても，その直前の行まではデータが取得されているので注意が必要です。同様に，エラーが発生した場合も，エラーが発生した行までのデータは取得されています。
3. 動的に実行する場合，BY 句の埋込み変数に受け取り領域以上の行数を指定すると，UAP 側の領域を破壊するおそれがあります。

4. 次のどれかに該当する場合，配列を使用した FETCH 機能は利用できません。
   - BLOB 型の選択式が問合せ指定にある場合
   - 問合せ指定に BINARY 型の選択式があり，かつ BINARY 型の選択式受け取り領域の 1 要素分の定義長が 4 の倍数でない場合
   - 定義長が 32,001 バイト以上の BINARY 型の選択式を含む検索で，HiRDB サーバ，又は HiRDB クライアントライブラリのバージョンのどちらかが 07-00 以前の場合

(4) 使用例

配列を使用した FETCH 機能のコーディング例について説明します。

例１：

FETCH 文形式 3 を使用します。対象となる表は SCODE(CHAR(4))，SNAME(VARCHAR(17))，COL(NCHAR(1))，TANKA(INTEGER)，及び ZSURYO(INTEGER)で構成されているものとします。

```
long sel_cnt;
long data_cnt;
short i;
char work[17];

/* 配列型の埋込み変数の宣言 */
EXEC SQL BEGIN DECLARE SECTION;
    char    xscode[50][5];
    SQL  TYPE  IS  VARCHAR(17)  xsname[50];
    char    xcol[50][3];
    long    xtanka[50];
    long    xzsuryo[50];
EXEC SQL END DECLARE SECTION;

EXEC SQL
    DECLARE CR3 CURSOR FOR
       SELECT SCODE,SNAME,COL,TANKA,ZSURYO
       FROM ZAIKO;

EXEC SQL WHENEVER SQLERROR GOTO FIN;

EXEC SQL OPEN CR3;

/*  見出し */

printf("    *****  在庫表 リスト  *****¥n¥n");
printf("    商品コード  商品名           色  単価      現在庫量¥n");
printf("    ----        ---------------  --  --------  --------¥n");

EXEC SQL WHENEVER SQLERROR GOTO OWARI;
EXEC SQL WHENEVER NOT FOUND GOTO OWARI;

/* FETCH */
sel_cnt = 0;
for(;;){
    EXEC SQL
        FETCH CR3 INTO :xscode,:xsname,:xcol,:xtanka,:xzsuryo;
```

```
      /* SQLERRD2にはこのカーソルで検索したトータル行数を格納 */
      data_cnt = SQLERRD2 - sel_cnt;       /* 取得した行数を求める */
      for(i=0; i < data_cnt; i++){
          memcpy(work, xsname[i].str, xsname[i].len);
          work[xsname[i].len] =  '¥0' ;
          printf("    %4s      %-16s  %2s  %8d  %8d¥n",
                  xscode[i], work, xcol[i], xtanka[i], xzsuryo[i]);
      }
      sel_cnt = SQLERRD2;
  }

OWARI:
/*                                                                  */
/* エラー発生時，及びNOT FOUND発生時でも，データが読み込まれて    */
/* いるため，残りのデータを表示する                                 */
/*                                                                  */
  if(sel_cnt  !=  SQLERRD2 ){
      data_cnt = SQLERRD2 - sel_cnt;
      for(i=0; i < data_cnt; i++){
          memcpy(work, xsname[i].str, xsname[i].len);
          work[xsname[i].len] =  '¥0' ;
          printf("    %4s      %-16s  %2s  %8d  %8d¥n",
                  xscode[i], work, xcol[i], xtanka[i], xzsuryo[i]);
      }
  }
FIN:
  EXEC SQL WHENEVER SQLERROR CONTINUE;
  EXEC SQL CLOSE CR3;
  EXEC SQL COMMIT;
```

例２：

FETCH 文形式 2 を使用します。対象となる表は SCODE(CHAR(4))，SNAME(VARCHAR(17))，COL(NCHAR(1))，TANKA(INTEGER)，及び ZSURYO(INTEGER)で構成されているものとします。

```
#include <pdbsqlda.h>              /* ユーザ定義のSQLDAを        */
                                   /* 使用するためにincludeする。*/

  long sel_cnt;
  long data_cnt;
  short i;
  char work[17];

  /* ユーザ定義SQLDAの宣言 */
  PDUSRSQLDA(5)  xsqlda;

  /* 配列型の埋込み変数の宣言 */
  EXEC SQL BEGIN DECLARE SECTION;
      char    xscode[50][5];
      SQL  TYPE  IS  VARCHAR(17)  xsname[50];
      char    xcol[50][3];
      long    xtanka[50];
      long    xzsuryo[50];
      short    arry_num;
  EXEC SQL END DECLARE SECTION;

  EXEC SQL WHENEVER SQLERROR GOTO FIN;
```

```
/* 検索SQLの前処理実行 */
EXEC SQL PREPARE SEL1 FROM
        'SELECT * FROM ZAIKO'  ;

/* 検索SQLの出力情報取得 */
PDSQLN(xsqlda) = 5 ;      /* SQLVAR数を設定 */
EXEC SQL DESCRIBE SEL1 INTO :xsqlda ;

EXEC SQL
    DECLARE CR3 CURSOR FOR SEL1 ;

EXEC SQL OPEN CR3;

/*  SQLVARの設定：本来であれば，I/O領域をSQLDAから動的に確保した */
/*  方がよいが，例題のため省略する                              */
/*  SQLLEN,SQLXDIM,SQLSYSはDESCRIBE時に設定される値を使用する。  */
/* SCODE列情報設定 */
PDSQLDATA(xsqlda, 0) = (void *)xscode ;    /* アドレス設定 */
PDSQLIND(xsqlda, 0) = NULL ;               /* 標識変数NULLクリア */
PDSQLCOD(xsqlda, 0) = PDSQL_CHAR ;         /* データコード設定 */
/* SNAME列情報設定 */
PDSQLDATA(xsqlda, 1) = (void *) xsname;    /* アドレス設定 */
PDSQLIND(xsqlda, 1) = NULL ;               /* 標識変数NULLクリア */
PDSQLCOD(xsqlda, 1) = PDSQL_VARCHAR ;      /* データコード設定 */
PDSQLSYS(xsqlda, 1) = sizeof(xsname[0]) ;  /* 可変長なのでSQLSYSを*/
                                           /* 設定                */
/* COL列情報設定 */
PDSQLDATA(xsqlda, 2) = (void *) xcol;      /* アドレス設定 */
PDSQLIND(xsqlda, 2) = NULL ;               /* 標識変数NULLクリア */
PDSQLCOD(xsqlda, 2) = PDSQL_NCHAR ;        /* データコード設定 */
/* TANKA列情報設定 */
PDSQLDATA(xsqlda, 3) = (void *) xtanka;    /* アドレス設定 */
PDSQLIND(xsqlda, 3) = NULL ;               /* 標識変数NULLクリア */
PDSQLCOD(xsqlda, 3) = PDSQL_INTEGER ;      /* データコード設定 */
/* GSURYO列情報設定 */
PDSQLDATA(xsqlda, 4) = (void *) xzsuryo;   /* アドレス設定 */
PDSQLIND(xsqlda, 4) = NULL ;               /* 標識変数NULLクリア */
PDSQLCOD(xsqlda, 4) = PDSQL_ INTEGER;      /* データコード設定 */

/*  見出し */

printf("    *****  在庫表 リスト  *****¥n¥n");
printf("    商品コード  商品名           色  単価      現在庫量¥n");
printf("    ----        ---------------  --  --------  --------¥n");

EXEC SQL WHENEVER SQLERROR GOTO OWARI;
EXEC SQL WHENEVER NOT FOUND GOTO OWARI;

/* FETCH */
sel_cnt = 0;
for(;;){
    arry_num = 50 ;
    EXEC SQL
        FETCH CR3 USING DESCRIPTOR :xsqlda BY :arry_num ROWS ;
    /* SQLERRD2にはこのカーソルで検索したトータル行数を格納 */
    data_cnt = SQLERRD2 - sel_cnt;       /* 取得した行数を求める */
    for(i=0; i < data_cnt; i++){
```

```
          memcpy(work, xsname[i].str, xsname[i].len);
          work[xsname[i].len] =  '¥0' ;
          printf("    %4s      %-16s  %2s  %8d  %8d¥n",
                  xscode[i], work, xcol[i], xtanka[i], xzsuryo [i]);
      }
      sel_cnt = SQLERRD2;
  }

OWARI:
/*                                                              */
/* エラー発生時，及びNOT FOUND発生時でも，データが読み込まれて */
/* いるため，残りのデータを表示する                             */
/*                                                              */
  if(sel_cnt  !=  SQLERRD2 ){
      data_cnt = SQLERRD2 - sel_cnt;
      for(i=0; i < data_cnt; i++){
          memcpy(work, xsname[i].str, xsname[i].len);
          work[xsname[i].len] =  '¥0' ;
          printf("    %4s      %-16s  %2s  %8d  %8d¥n",
                  xscode[i], work, xcol[i], xtanka[i], xzsuryo [i]);
      }
  }
FIN:
  EXEC SQL WHENEVER SQLERROR CONTINUE;
  EXEC SQL CLOSE CR3;
  EXEC SQL COMMIT;
```

例３：

FETCH 文形式 3 を使用します。対象となる表は XCODE(INTEGER)，及び ROW_DATA(BINARY(3002))で構成されているものとします。

```
  long sel_cnt;
  long data_cnt;
  short i;

  /* 配列型の埋込み変数の宣言 */
  EXEC SQL BEGIN DECLARE SECTION;
      long    xcode[50];
      /* BINARY型の配列を使用したFETCHを行う場合は,     */
      /* 領域長を4の倍数で定義すること                  */
      SQL  TYPE  IS  BINARY(3004)  xrow_data[50];
  EXEC SQL END DECLARE SECTION;

  EXEC SQL
      DECLARE CR3 CURSOR FOR
         SELECT * FROM T_BINARY;

  EXEC SQL WHENEVER SQLERROR GOTO FIN;

  EXEC SQL OPEN CR3;

  /*  見出し */

  printf("    *****  バイナリデータ表  *****¥n¥n");

  EXEC SQL WHENEVER SQLERROR GOTO OWARI;
```

```
  EXEC SQL WHENEVER NOT FOUND GOTO OWARI;

  /* FETCH */
  sel_cnt = 0;
  for(;;){

      EXEC SQL
          FETCH CR3 INTO : xcode,: xrow_data;
      /* SQLERRD2にはこのカーソルで検索したトータル行数を格納 */
      data_cnt = SQLERRD2 - sel_cnt;       /* 取得した行数を求める */
      for(i=0; i < data_cnt; i++){
          printf("    CODE=%8d¥n",xcode[i]);
          printf("    DATA_LENGTH=%d¥n", xrow_data [i].len);
          /* BINARYデータ部の表示は例題のため行わない。    */
          /* xrow_data[i].strを各UAP固有の形式に変換すること */
      }
      sel_cnt = SQLERRD2;
  }

OWARI:
/*                                                              */
/* エラー発生時，及びNOT FOUND発生時でも，データが読み込まれて */
/* いるため，残りのデータを表示する                             */
/*                                                              */
  if(sel_cnt  !=  SQLERRD2 ){
      data_cnt = SQLERRD2 - sel_cnt;
      for(i=0; i < data_cnt; i++){
          printf("    CODE=%8d¥n",xcode[i]);
          printf("    DATA_LENGTH=%d¥n", xrow_data [i].len);
          /* BINARYデータ部の表示は例題のため行わない。    */
          /* xrow_data[i].strを各UAP固有の形式に変換すること */
      }
  }
FIN:
  EXEC SQL WHENEVER SQLERROR CONTINUE;
  EXEC SQL CLOSE CR3;
  EXEC SQL COMMIT;
```

例４：

FETCH 文形式 2 を使用します。対象となる表は XCODE(INTEGER)，及び ROW_DATA(BINARY(3002))で構成されているものとします。

```
#include <pdbsqlda.h>       /* ユーザ定義のSQLDAを使用するために */
                            /* includeする。                     */

  long sel_cnt;
  long data_cnt;
  short i;

  /* ユーザ定義SQLDAの宣言 */
  PDUSRSQLDA(2)  xsqlda;

  /* 配列型の埋込み変数の宣言 */
  EXEC SQL BEGIN DECLARE SECTION;
      long    xcode[50];
      /* BINARY型の配列を使用したFETCHを行う場合は,     */
```

```
    /* 領域長を4の倍数で定義すること                    */
    SQL  TYPE  IS  BINARY(3004)  xrow_data[50];
    short    arry_num;
EXEC SQL END DECLARE SECTION;

EXEC SQL WHENEVER SQLERROR GOTO FIN;

/* 検索SQLの前処理実行 */
EXEC SQL PREPARE SEL1 FROM
        'SELECT * FROM T_BINARY ;

/* 検索SQLの出力情報取得 */
PDSQLN(xsqlda) = 2 ;      /* SQLVAR数を設定 */
EXEC SQL DESCRIBE SEL1 INTO :xsqlda ;

EXEC SQL
    DECLARE CR3 CURSOR FOR SEL1 ;

EXEC SQL OPEN CR3;

/*  SQLVARの設定：本来であれば，I/O領域をSQLDAから動的に確保した*/
/*  方がよいが，例題のため省略する                          */
/*  SQLLEN,SQLXDIM,SQLSYSはDESCRIBE時に設定される値を使用する。 */
/* XCODE列情報設定 */
PDSQLDATA(xsqlda, 0) = (void *)xcode ;    /* アドレス設定 */
PDSQLIND(xsqlda, 0) = NULL ;              /* 標識変数NULLクリア */
PDSQLCOD(xsqlda, 0) = PDSQL_INTEGER ;     /* データコード設定 */
/* R_DATA列情報設定 */
PDSQLDATA(xsqlda, 1) = (void *) xrow_data; /* アドレス設定 */
PDSQLIND(xsqlda, 1) = NULL ;               /* 標識変数NULLクリア */
PDSQLCOD (xsqlda, 1) = PDSQL_BINARY ;      /* データコード設定*/
PDSQLLEN (xsqlda, 1) = 3004 ;              /* 定義長が４の倍数で */
                                           /* ないため再設定する */

/*  見出し */

printf("    *****  バイナリデータ表  *****¥n¥n");

EXEC SQL WHENEVER SQLERROR GOTO OWARI;
EXEC SQL WHENEVER NOT FOUND GOTO OWARI;

/* FETCH */
sel_cnt = 0;
for(;;){
    arry_num = 50 ;
    EXEC SQL
        FETCH CR3 USING DESCRIPTOR :xsqlda BY :arry_num ROWS ;
    /* SQLERRD2にはこのカーソルで検索したトータル行数を格納 */
    data_cnt = SQLERRD2 - sel_cnt;       /* 取得した行数を求める */
    for(i=0; i < data_cnt; i++){
        printf("    CODE=%8d¥n",xcode[i]);
        printf("    DATA_LENGTH=%d¥n", xrow_data [i].len);
        /* BINARYデータ部の表示は例題のため行わない。    */
        /* xrow_data[i].strを各UAP固有の形式に変換すること */
    }
    sel_cnt = SQLERRD2;
}
```

```
OWARI:
/*                                                              */
/* エラー発生時，及びNOT FOUND発生時でも，データが読み込まれて */
/* いるため，残りのデータを表示する                             */
/*                                                              */
  if(sel_cnt  !=  SQLERRD2 ){
      data_cnt = SQLERRD2 - sel_cnt;
      for(i=0; i < data_cnt; i++){
          printf("    CODE=%8d¥n",xcode[i]);
          printf("    DATA_LENGTH=%d¥n", xrow_data [i].len);
          /* BINARYデータ部の表示は例題のため行わない。    */
          /* xrow_data[i].strを各UAP固有の形式に変換すること */
      }
  }
FIN:
  EXEC SQL WHENEVER SQLERROR CONTINUE;
  EXEC SQL CLOSE CR3;
  EXEC SQL COMMIT;
```

4.8.2　配列を使用した INSERT 機能

(1)　概要

複数行分のデータを設定した配列型の変数を指定することで，一つの SQL 文で複数行分のデータを挿入できます。配列を使用した INSERT 機能を使用すると，HiRDB クライアントと HiRDB サーバとの間の通信回数を削減できます。また，HiRDB/パラレルサーバの場合には，更に HiRDB サーバ内のサーバ間の通信回数も削減できます。そのため，HiRDB クライアントから HiRDB サーバにアクセスし，大量のデータを高速に挿入したい場合に有効となります。

(2)　使用方法

(a)　静的に実行する場合

INSERT 文で FOR 句に埋込み変数を指定し，かつ指定する埋込み変数と標識変数をすべて配列型の変数にしてください。一括して挿入する行数は，FOR 句に指定した埋込み変数で制御します。

(b)　動的に実行する場合

次に示す手順で実行してください。

1. PREPARE 文で，INSERT 文（一つ以上の？パラメタを指定）を前処理します。
2. EXECUTE 文の USING 句に，前処理した INSERT 文の入力？パラメタに与える値を配列で指定し，かつ BY 句に埋込み変数を指定してください。一括して挿入する行数は，BY 句に指定した埋込み変数で制御します。

   USING 句で埋込み変数を指定する場合は，埋込み変数及び標識変数をすべて配列型の変数にしてください。

USING 句で SQL 記述領域を指定する場合は，SQLDATA が指すすべての領域に，データを配列形式で指定してください。また，SQLSYS 領域にデータ型に応じた値を設定してください。

(3) 注意事項

INSERT 文の FOR 句，又は EXECUTE 文の BY 句の埋込み変数に，書き込み領域以上の行数を指定すると，DB 破壊，又は UAP 側の領域破壊となるおそれがあります。

(4) 使用例

配列を使用した INSERT 機能のコーディング例について説明します。

例１：

INSERT 文形式 3 を使用して，ファイルから読み出したデータを配列形式の埋込み変数に設定し，在庫表（ZAIKO）へ一括して 50 行分ずつ挿入します。

対象となる表は SCODE(CHAR(4))，SNAME(VARCHAR(17))，COL(NCHAR(1))，TANKA(INTEGER)，及び ZSURYO(INTEGER)で構成されているものとします。

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#define MAXCOLUMN 80
#define INFILE   ”inputf1”

void abnormalend();

FILE *input ;

main() {
  char indata[MAXCOLUMN];
  char in_scode[5];
  short in_sname_len;
  char in_sname[17];
  char in_col[3];
  int in_tanka;
  int i;

  EXEC SQL BEGIN DECLARE SECTION;
    short  xinsert_num;
    /* 配列型の埋込み変数の宣言 */
    char    xscode[50][5];            /* SCODE (CHAR(4)型の列) */
                                      /* への挿入値指定用      */
    SQL TYPE IS VARCHAR(17) xsname[50];
                                      /* SNAME(VARCHAR(17)型の列) */
                                      /* への挿入値指定用         */
    char    xcol[50][3];              /* COL(NCHAR(1)型の列)      */
                                      /* への挿入値指定用         */
    long    xtanka[50];               /* TANKA (INTEGER型の列)   */
                                      /* への挿入値指定用         */
    EXEC SQL END DECLARE SECTION;
```

```
        -------(HiRDBへのCONNECT処理(省略))-------

  input = fopen(INFILE, "r");
  if (input == NULL) {
  fprintf(stderr, "can't open %s.", INFILE);
  goto FIN;
  }

    EXEC SQL WHENEVER SQLERROR PERFORM abnormalend;

    /* 一括挿入行数設定（最大50行ごと）*/
    xinsert_num=50;
    while (!feof(input)) {
    /* 配列変数に50行分(ファイルの最終データの場合はその行まで）*/
    /* の入力データを設定する                                    */
    for ( i = 0; i < 50; i++) {
      /* ファイルからデータを読み込む */
      fgets(indata, MAXCOLUMN, input);
      if (feof(input)){
        /* ファイルの最終データの場合は，一括挿入行数にここまでの */
        /* 行数を設定してfor文を抜ける                          */
        xinsert_num= i;
        break;
      }
      sscanf(indata, "%4s %hd %16s %2s %8d",
      in_scode, &in_sname_len, in_sname, in_col, &in_tanka);
      /* 配列変数の要素に入力データを設定 */
      strncpy(xscode[i], in_scode, 5);
      xsname[i].len = in_sname_len;
      strncpy(xsname[i].str, in_sname, 17);
      strncpy(xcol[i], in_col, 3);
      xtanka[i] = in_tanka;
    }
    /* INSERT実行 */
    EXEC SQL FOR :xinsert_num
      INSERT INTO ZAIKO (SCODE, SNAME, COL, TANKA)
      VALUES (:xscode, :xsname, :xcol, :xtanka);
  }

  EXEC SQL COMMIT;
  printf(" *** normal ended ***¥n");
FIN:
  if (input != NULL) {
  fclose(input);
  }
  EXEC SQL WHENEVER SQLERROR CONTINUE;
  EXEC SQL WHENEVER SQLWARNING CONTINUE;
  EXEC SQL DISCONNECT;
  return(0);
}
void abnormalend()
{
  int  wsqlcode;

  if (input != NULL) {
    fclose(input);
  }
  wsqlcode = -SQLCODE;
```

```
  printf("¥n*** HiRDB SQL ERROR SQLCODE = %d ¥n", wsqlcode);
  printf("SQLERRMC = %s¥n", SQLERRMC);
  EXEC SQL ROLLBACK;
  EXEC SQL DISCONNECT;
  exit(1);
}
```

例２：

INSERT 文形式 3 を使用して，データ読み込み関数から読み込んだデータを，配列形式の埋込み変数に設定し，在庫表（ZAIKO）へ一括して 50 行分ずつ挿入します。

対象となる表は SCODE(CHAR(4))，及び ROW_DATA(BINARY(3002))で構成されているものとします。

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

void abnormalend();

main() {
  int i,rc;

  EXEC SQL BEGIN DECLARE SECTION;
    short  xinsert_num;
    /* 配列型の埋込み変数の宣言 */
    char  xscode[50][5];         /* SCODE (CHAR(4)型の列) */
                                 /* への挿入値指定用      */
    SQL TYPE IS BINARY(3004) xrow_data[50];
                    /* ROW_DATA(BINARY(3002)型の列)への挿入値指定用 */
                    /* ただし，データ長は4の倍数にする             */
    EXEC SQL END DECLARE SECTION;

        -------(HiRDBへのCONNECT処理(省略))-------

    EXEC SQL WHENEVER SQLERROR PERFORM abnormalend;

  rc = 0 ;
  /* 一括挿入行数設定（最大50行ごと）*/
  xinsert_num=50;
  while (0==rc) {
    /* 配列変数に50行分(ファイルの最終データの場合はその行まで）*/
    /* の入力データを設定する                                    */
    for ( i = 0; i < 50; i++) {
      /* BINARYデータを読み込む：関数の詳細については省略 */
      rc = get_binarydata(&xscode[i],&xrow_data[i]);
      if (0 != rc){
        /* 入力データがなくなったら，一括挿入行数にここまでの行数を*/
        /* 設定してfor文を抜ける                                   */
        xinsert_num= i;
        break;
      }
    }
    /* INSERT実行 */
    EXEC SQL FOR :xinsert_num
      INSERT INTO ZAIKO (SCODE, ROW_DATA)
      VALUES (:xscode, :xrow_data);
```

```
  }

  EXEC SQL COMMIT;
  printf(" *** normal ended ***¥n");
FIN:
  EXEC SQL WHENEVER SQLERROR CONTINUE;
  EXEC SQL WHENEVER SQLWARNING CONTINUE;
  EXEC SQL DISCONNECT;
  return(0);
}
void abnormalend()
{
  int  wsqlcode;

  wsqlcode = -SQLCODE;
  printf("¥n*** HiRDB SQL ERROR SQLCODE = %d ¥n", wsqlcode);
  printf("SQLERRMC = %s¥n", SQLERRMC);
  EXEC SQL ROLLBACK;
  EXEC SQL DISCONNECT;
  exit(1);
}
```

例３：

EXECUTE 文形式 2 を使用して，ファイルから読み出したデータを配列形式の埋込み変数に設定し，在庫表（ZAIKO）へ一括して 50 行分ずつ挿入します。

対象となる表は SCODE(CHAR(4))，SNAME(VARCHAR(17))，COL(NCHAR(1))，TANKA(INTEGER)，及び ZSURYO(INTEGER)で構成されているものとします。

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#define MAXCOLUMN 80
#define INFILE   "inputf1"

void abnormalend();

FILE *input ;

main() {
  char indata[MAXCOLUMN];
  char in_scode[5];
  short in_sname_len;
  char in_sname[17];
  char in_col[3];
  int in_tanka;
  int i;

  EXEC SQL BEGIN DECLARE SECTION;
    short  xinsert_num;
    /* 配列型の埋込み変数の宣言 */
    char  xscode[50][5];     /* SCODE (CHAR(4)型の列) */
                             /* への挿入値指定用      */
    SQL TYPE IS VARCHAR(17) xsname[50];
                         /* SNAME(VARCHAR(17)型の列)への挿入値指定用 */
```

```
    char  xcol[50][3];  /* COL(NCHAR(1)型の列）への挿入値指定用 */
    long  xtanka[50];   /* TANKA（INTEGER型の列）への挿入値指定用 */
  EXEC SQL END DECLARE SECTION;

      -------(HiRDBへのCONNECT処理(省略))-------

  input = fopen(INFILE, "r");
  if (input == NULL) {
    fprintf(stderr, "can't open %s.", INFILE);
    goto FIN;
  }

  EXEC SQL WHENEVER SQLERROR PERFORM abnormalend;

  /* SQL前処理実行 */
  EXEC SQL PREPARE INS1 FROM
     'INSERT INTO ZAIKO(SCODE, SNAME, COL, TANKA) VALUES(?,?,?,?)' ;

  /* 一括挿入行数設定（最大50行ごと）*/
  xinsert_num=50;
  while (!feof(input)) {
    /* 配列変数に50行分(ファイルの最終データの場合はその行まで）*/
    /* の入力データを設定する                                    */
    for ( i = 0; i < 50; i++) {
      /* ファイルからデータを読み込む */
      fgets(indata, MAXCOLUMN, input);
      if (feof(input)){
        /* ファイルの最終データの場合，一括挿入行数にここまでの */
        /* 行数を設定してfor文を抜ける                          */
        xinsert_num= i;
        break;
      }
      sscanf(indata, "%4s %hd %16s %2s %8d",
      in_scode, &in_sname_len, in_sname, in_col, &in_tanka);
      /* 配列変数の要素に入力データを設定 */
      strncpy(xscode[i], in_scode, 5);
      xsname[i].len = in_sname_len;
      strncpy(xsname[i].str, in_sname, 17);
      strncpy(xcol[i], in_col, 3);
      xtanka[i] = in_tanka;
    }
    /* EXECUTE実行 */
    EXEC  SQL  EXECUTE  INS1
      USING  :xscode, :xsname, :xcol, :xtanka
      BY :xinsert_num ROWS ;
  }

  EXEC SQL COMMIT;
  printf(" *** normal ended ***¥n");
FIN:
  if (input != NULL) {
  fclose(input);
  }
  EXEC SQL WHENEVER SQLERROR CONTINUE;
  EXEC SQL WHENEVER SQLWARNING CONTINUE;
  EXEC SQL DISCONNECT;
  return(0);
}
```

```
void abnormalend()
{
  int  wsqlcode;

  if (input != NULL) {
  fclose(input);
  }
  wsqlcode = -SQLCODE;
  printf("¥n*** HiRDB SQL ERROR SQLCODE = %d ¥n", wsqlcode);
  printf("SQLERRMC = %s¥n", SQLERRMC);
  EXEC SQL ROLLBACK;
  EXEC SQL DISCONNECT;
  exit(1);
}
```

例４：

EXECUTE 文形式 2 を使用して，ファイルから読み出したデータを配列形式の埋込み変数に設定し，ユーザ定義の SQLDA を使用して在庫表（ZAIKO）へ一括して 50 行分ずつ挿入します。

対象となる表は SCODE(CHAR(4))，SNAME(VARCHAR(17))，COL(NCHAR(1))，TANKA(INTEGER)，及び ZSURYO(INTEGER)で構成されているものとします。

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <pdbsqlda.h>          /* ユーザ定義のSQLDAを使用するために */
                               /* includeする。                     */

#define MAXCOLUMN 80
#define INFILE   "inputf1"

void abnormalend();

FILE *input ;

main() {
  char indata[MAXCOLUMN];
  char in_scode[5];
  short in_sname_len;
  char in_sname[17];
  char in_col[3];
  int in_tanka;
  int i;

  /* ユーザ定義SQLDAの宣言 */
  PDUSRSQLDA(4)  xsqlda;

  EXEC SQL BEGIN DECLARE SECTION;
    short  xinsert_num;
    /* 配列型の埋込み変数の宣言 */
    char  xscode[50][5];       /* SCODE (CHAR(4)型の列) */
                               /* への挿入値指定用      */
    SQL TYPE IS VARCHAR(17) xsname[50];
                      /* SNAME(VARCHAR(17)型の列)への挿入値指定用 */
    char  xcol[50][3];      /* COL(NCHAR(1)型の列)への挿入値指定用 */
    long  xtanka[50];      /* TANKA(INTEGER型の列)への挿入値指定用 */
```

```
EXEC SQL END DECLARE SECTION;

     -------(HiRDBへのCONNECT処理(省略))-------

input = fopen(INFILE, "r");
if (input == NULL) {
  fprintf(stderr, "can't open %s.", INFILE);
  goto FIN;
}

EXEC SQL WHENEVER SQLERROR PERFORM abnormalend;

/* SQL前処理実行 */
EXEC SQL PREPARE INS1 FROM
   'INSERT INTO ZAIKO(SCODE, SNAME, COL, TANKA) VALUES(?,?,?,?)' ;

/* SQLVARの設定 */
PDSQLN(xsqlda) = 4 ;       /* SQLVAR数を設定 */
PDSQLD(xsqlda) = 4 ;       /* ？パラメタ数を設定 */
/* SCODE列情報設定 */
PDSQLCOD(xsqlda, 0) = PDSQL_CHAR ;  /* データコード設定 */
PDSQLXDIM(xsqlda, 0) = 1 ;          /* 繰り返し構造の要素数設定 */
PDSQLSYS(xsqlda, 0) = 0 ;           /* 1要素の長さ(可変長文字列 */
                                    /* 以外は0固定)           */
PDSQLLEN(xsqlda, 0) = 4 ;           /* データ定義長設定 */
PDSQLDATA(xsqlda, 0) = (void *)xscode ; /* データ領域アドレス設定 */
PDSQLIND(xsqlda, 0) = NULL ;        /* 標識変数NULLクリア */
/* SNAME列情報設定 */
PDSQLCOD(xsqlda, 1) = PDSQL_VARCHAR ;  /* データコード設定 */
PDSQLXDIM(xsqlda, 1) = 1 ;          /* 繰り返し構造の要素数設定 */
PDSQLLEN(xsqlda, 1) = 17 ;          /* データ定義長設定 */
PDSQLSYS(xsqlda, 1) = sizeof(xsname[0]) ;  /* 1要素の長さ */
PDSQLDATA(xsqlda, 1) = (void *) xsname;  /* データ領域アドレス設定 */
PDSQLIND(xsqlda, 1) = NULL ;        /* 標識変数NULLクリア */
/* COL列情報設定 */
PDSQLCOD(xsqlda, 2) = PDSQL_NCHAR ;  /* データコード設定 */
PDSQLXDIM(xsqlda, 2) = 1 ;           /* 繰り返し構造の要素数設定 */
PDSQLSYS(xsqlda, 2) = 0 ;            /* 1要素の長さ(可変長文字列 */
                                     /* 以外は0固定)           */
PDSQLLEN(xsqlda, 2) = 1 ;            /* データ定義長設定 */
PDSQLDATA(xsqlda, 2) = (void *) xcol;  /* データ領域アドレス設定 */
PDSQLIND(xsqlda, 2) = NULL ;         /* 標識変数NULLクリア */
/* TANKA列情報設定 */
PDSQLCOD(xsqlda, 3) = PDSQL_INTEGER ;  /* データコード設定 */
PDSQLXDIM(xsqlda, 3) = 1 ;           /* 繰り返し構造の要素数設定 */
PDSQLSYS(xsqlda, 3) = 0 ;            /* 1要素の長さ(可変長文字列 */
                                     /* 以外は0固定)           */
PDSQLLEN(xsqlda, 3) = 4 ;            /* データ定義長設定 */
PDSQLDATA(xsqlda, 3) = (void *) xtanka;  /* データ領域アドレス設定 */
PDSQLIND(xsqlda, 3) = NULL ;         /* 標識変数NULLクリア */

/* 一括挿入行数設定（最大50行ごと）*/
xinsert_num=50;
while (!feof(input)) {
  /* 配列変数に50行分(ファイルの最終データの場合はその行まで) */
  /* の入力データを設定する                                   */
  for ( i = 0; i < 50; i++) {
    /* ファイルからデータを読み込む */
```

```
        fgets(indata, MAXCOLUMN, input);
        if (feof(input)){
          /* ファイルの最終データの場合，一括挿入行数にここまでの */
          /* 行数を設定してfor文を抜ける                          */
          xinsert_num= i;
          break;
        }
        sscanf(indata, "%4s %hd %16s %2s %8d",
        in_scode, &in_sname_len, in_sname, in_col, &in_tanka);
        /* 配列変数の要素に入力データを設定 */
        strncpy(xscode[i], in_scode, 5);
        xsname[i].len = in_sname_len;
        strncpy(xsname[i].str, in_sname, 17);
        strncpy(xcol[i], in_col, 3);
        xtanka[i] = in_tanka;
      }
      /* EXECUTE実行 */
      EXEC  SQL  EXECUTE  INS1
        USING  DESCRIPTOR  :xsqlda
        BY  :xinsert_num  ROWS ;
  }

  EXEC SQL COMMIT;
  printf(" *** normal ended ***¥n");
FIN:
  if (input != NULL) {
    fclose(input);
  }
    EXEC SQL WHENEVER SQLERROR CONTINUE;
  EXEC SQL WHENEVER SQLWARNING CONTINUE;
  EXEC SQL DISCONNECT;
  return(0);
}
void abnormalend()
{
  int  wsqlcode;

  if (input != NULL) {
  fclose(input);
  }
  wsqlcode = -SQLCODE;
  printf("¥n*** HiRDB SQL ERROR SQLCODE = %d ¥n", wsqlcode);
  printf("SQLERRMC = %s¥n", SQLERRMC);
  EXEC SQL ROLLBACK;
  EXEC SQL DISCONNECT;
  exit(1);
}
```

例５：

EXECUTE 文形式 2 を使用して，データ読み込み関数から読み込んだデータを，配列形式の埋込み変数に設定し，ユーザ定義の SQLDA を使用して在庫表（ZAIKO）へ一括して 50 行分ずつ挿入します。

対象となる表は SCODE(CHAR(4))，及び ROW_DATA(BINARY(3002))で構成されているものとします。

```
#include <stdio.h>
#include <string.h>
```

```
#include <stdlib.h>
#include <pdbsqlda.h>          /* ユーザ定義のSQLDAを使用するために */
                               /* includeする。                      */

void abnormalend();

main() {
  int i,rc;

  /* ユーザ定義SQLDAの宣言 */
  PDUSRSQLDA(4)  xsqlda;

  EXEC SQL BEGIN DECLARE SECTION;
    short  xinsert_num;
    /* 配列型の埋込み変数の宣言 */
    char  xscode[50][5];    /* SCODE (CHAR(4)型の列) */
                            /* への挿入値指定用      */
    SQL TYPE IS BINARY(3004) xrow_data[50];
                  /* ROW_DATA(BINARY(3002)型の列)への挿入値指定用 */
                  /* ただし，データ長は4の倍数にする              */
  EXEC SQL END DECLARE SECTION;

      -------(HiRDBへのCONNECT処理(省略))-------

  EXEC SQL WHENEVER SQLERROR PERFORM abnormalend;

  /* SQL前処理実行 */
  EXEC SQL PREPARE INS1 FROM
    'INSERT INTO ZAIKO(SCODE, ROW_DATA) VALUES(?,?)' ;

  /* SQLVARの設定 */
  PDSQLN(xsqlda) = 2 ;                /* SQLVAR数を設定 */
  PDSQLD(xsqlda) = 2 ;                /* ？パラメタ数を設定 */
  /* SCODE列情報設定 */
  PDSQLCOD(xsqlda, 0) = PDSQL_CHAR ;  /* データコード設定 */
  PDSQLXDIM(xsqlda, 0) = 1 ;       /* 繰り返し構造の要素数設定 */
  PDSQLSYS(xsqlda, 0) = 0 ;        /* 1要素の長さ(可変長文字列 */
                                   /* 以外は0固定)             */
  PDSQLLEN(xsqlda, 0) = 4 ;        /* データ定義長設定 */
  PDSQLDATA(xsqlda, 0) = (void *)xscode ;  /* データ領域アドレス設定 */
  PDSQLIND(xsqlda, 0) = NULL ;     /* 標識変数NULLクリア */
  /* ROW_DATA列情報設定 */
  PDSQLCOD(xsqlda, 1) = PDSQL_BINARY ;  /* データコード設定 */
  PDSQLXDIM(xsqlda, 1) = 1 ;       /* 繰り返し構造の要素数設定 */
  PDSQLLOBLEN(xsqlda, 1) = 3004 ;  /* データ定義長設定 */
  PDSQLDATA(xsqlda, 1) = (void *) xrow_data;  /* データ領域   */
                                              /* アドレス設定 */
  PDSQLIND(xsqlda, 1) = NULL ;     /* 標識変数NULLクリア */

  rc = 0 ;
  /* 一括挿入行数設定（最大50行ごと）*/
  xinsert_num=50;
  while (0==rc) {
    /* 配列変数に50行分(ファイルの最終データの場合はその行まで）*/
    /* の入力データを設定する                                    */
    for ( i = 0; i < 50; i++) {
      /* BINARYデータを読み込む：関数の詳細については省略 */
      rc = get_binarydata(&xscode[i],&xrow_data[i]);
```

```
      if (0 != rc){
        /* 入力データがなくなった場合，一括挿入行数にここまでの */
        /* 行数を設定してfor文を抜ける                          */
        xinsert_num= i;
        break;
      }
    }
    /* EXECUTE実行 */
    EXEC  SQL  EXECUTE  INS1
      USING  DESCRIPTOR  :xsqlda
      BY  :xinsert_num  ROWS ;
  }

  EXEC SQL COMMIT;
  printf(" *** normal ended ***¥n");
FIN:
  EXEC SQL WHENEVER SQLERROR CONTINUE;
  EXEC SQL WHENEVER SQLWARNING CONTINUE;
  EXEC SQL DISCONNECT;
  return(0);
}
void abnormalend()
{
  int  wsqlcode;

  wsqlcode = -SQLCODE;
  printf("¥n*** HiRDB SQL ERROR SQLCODE = %d ¥n", wsqlcode);
  printf("SQLERRMC = %s¥n", SQLERRMC);
  EXEC SQL ROLLBACK;
  EXEC SQL DISCONNECT;
  exit(1);
}
```

4.8.3　配列を使用した UPDATE 機能

(1)　概要

複数回分のデータを設定した配列型の変数を指定することで，一つの SQL 文で複数回分の表の列の更新ができます。

HiRDB クライアントと HiRDB サーバとの間の通信回数を削減できるため，HiRDB クライアントから HiRDB サーバにアクセスし，大量データを高速に更新する場合に有効です。

(2)　使用方法

(a)　静的に実行する場合

UPDATE 文で，FOR 句に埋込み変数を指定し，かつ探索条件中に指定した埋込み変数と標識変数をすべて配列型の変数にしてください。一括して更新する回数は，FOR 句に指定した埋込み変数で制御します。

(b)　動的に実行する場合

次に示す手順で実行してください。

1. PREPARE 文で，UPDATE 文（更新値や探索条件中に？パラメタを指定）を前処理します。
2. EXECUTE 文の USING 句に，前処理した UPDATE 文の入力？パラメタに与える値を配列で指定し，かつ BY 句に埋込み変数を指定してください。一括して更新する回数は，BY 句に指定した埋込み変数で制御します。

動的に実行する場合の注意事項を次に示します。

- USING 句で埋込み変数を指定する場合，埋込み変数及び標識変数をすべて配列型の変数にしてください。
- USING 句で SQL 記述領域を指定する場合，SQLDATA が指すすべての領域に配列形式でデータを指定してください。また，SQLSYS 領域にデータ型に応じた値を設定してください。

(3)　注意事項

1. UPDATE 文の FOR 句，又は EXECUTE 文の BY 句の埋込み変数に，書き込み領域以上の回数を指定すると，DB 破壊，又は UAP 側の領域破壊を起こすおそれがあります。

(4)　使用例

例：

ファイルから読んだデータを配列形式の埋込み変数に設定し，在庫表（ZAIKO）に対して，一括で複数回の削除をします。

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#define MAXCOLUMN 80
#define INFILE   ”inputf1”

void abnormalend();

FILE *input ;

main() {
  char indata[MAXCOLUMN];
  char in_scode[5];
  int in_suryo;
  int i;

  EXEC SQL BEGIN DECLARE SECTION;
    short   xupdate_num;
    /* 配列型の埋込み変数の宣言 */
    char    xscode[50][5]; /* SCODE(CHAR(4)型の列) への探索条件用    */
    long    xsuryo[50];    /* ZSURYO(INTEGER型の列)への更新値指定用 */
  EXEC SQL END DECLARE SECTION;
```

```
      -------(HiRDBへのCONNECT処理(省略))-------

  input = fopen(INFILE, "r");
  if (input == NULL) {
    fprintf(stderr, "can't open %s.", INFILE);
    goto FIN;
  }

  EXEC SQL WHENEVER SQLERROR PERFORM abnormalend;

  /* 一括更新回数設定（最大50回に設定）*/
  xupdate_num=50;
  while (!feof(input)) {
    /* 配列変数に50回分(ファイルの終わりに到達した場合はその行まで）*/
    /* の更新/探索条件データを設定する                                    */
    for ( i = 0; i < 50; i++) {
      /* ファイルからデータを読み込む */
      fgets(indata, MAXCOLUMN, input);
      if (feof(input)){
        /* ファイルの終わりに到達したら，ここまでの配列要素数を */
        /* 一括更新回数に設定してfor文を抜ける                  */
        xupdate_num= i;
        break;
      }
      sscanf(indata, "%4s %8d", in_scode, &in_suryo);
      /* 配列変数の要素に更新/探索条件データを設定 */
      strncpy(xscode[i], in_scode, 5);
      xsuryo[i] = in_suryo;
    }
    /* UPDATE実行 */
    EXEC SQL FOR :xupdate_num
      UPDATE ZAIKO SET ZSURYO = :xsuryo WHERE SCODE = :xscode ;
  }

  EXEC SQL COMMIT;
  printf(" *** normal ended ***¥n");
FIN:
  if (input != NULL) {
  fclose(input);
  }
  EXEC SQL WHENEVER SQLERROR CONTINUE;
  EXEC SQL WHENEVER SQLWARNING CONTINUE;
  EXEC SQL DISCONNECT;
  return(0);
}
void abnormalend()
{
  int  wsqlcode;

  if (input != NULL) {
    fclose(input);
  }
  wsqlcode = -SQLCODE;
  printf("¥n*** HiRDB SQL ERROR SQLCODE = %d ¥n", wsqlcode);
  printf("SQLERRMC = %s¥n", SQLERRMC);
  EXEC SQL ROLLBACK;
  EXEC SQL DISCONNECT;
```

```
  exit(1);
}
```

4.8.4　配列を使用した DELETE 機能

(1)　概要

複数回分のデータを設定した配列型の変数を指定することで，一つの SQL 文で複数回分の行の削除ができます。

HiRDB クライアントと HiRDB サーバとの間の通信回数を削減できるため，HiRDB クライアントから HiRDB サーバにアクセスし，大量データを高速に削除する場合に有効です。

(2)　使用方法

(a)　静的に実行する場合

DELETE 文で，FOR 句に埋込み変数を指定し，かつ探索条件中に指定した埋込み変数と標識変数をすべて配列型の変数にしてください。一括して削除する回数は，FOR 句に指定した埋込み変数で制御します。

(b)　動的に実行する場合

次に示す手順で実行してください。

1. PREPARE 文で，DELETE 文（探索条件中に？パラメタを指定）を前処理します。
2. EXECUTE 文の USING 句に，前処理した DELETE 文の入力？パラメタに与える値を配列で指定し，かつ BY 句に埋込み変数を指定してください。一括して削除する回数は，BY 句に指定した埋込み変数で制御します。

動的に実行する場合の注意事項を次に示します。

- USING 句で埋込み変数を指定する場合，埋込み変数及び標識変数をすべて配列型の変数にしてください。
- USING 句で SQL 記述領域を指定する場合，SQLDATA が指すすべての領域に配列形式でデータを指定してください。また，SQLSYS 領域にデータ型に応じた値を設定してください。

(3)　注意事項

1. EXECUTE 文の BY 句の埋込み変数に，書き込み領域以上の回数を指定すると，DB 破壊，又は UAP 側の領域破壊を起こすおそれがあります。

(4) 使用例

例：

ファイルから読んだデータを配列形式の埋込み変数に設定し，在庫表（ZAIKO）を一括して複数回分更新します。

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#define MAXCOLUMN 80
#define INFILE   "inputf1"

void abnormalend();

FILE *input ;

main() {
  char indata[MAXCOLUMN];
  char in_scode[5];
  int i;

  EXEC SQL BEGIN DECLARE SECTION;
    short   xdelete_num;
    /* 配列型の埋込み変数の宣言 */
    char    xscode[50][5]; /* SCODE(CHAR(4)型の列）への探索条件用 */
  EXEC SQL END DECLARE SECTION;

      -------(HiRDBへのCONNECT処理(省略))-------

  input = fopen(INFILE, "r");
  if (input == NULL) {
    fprintf(stderr, "can't open %s.", INFILE);
    goto FIN;
  }

  EXEC SQL WHENEVER SQLERROR PERFORM abnormalend;

  /* 一括削除回数設定（最大50回に設定）*/
  xdelete_num=50;
  while (!feof(input)) {
    /* 配列変数に50回分(ファイルの終わりに到達した場合はその行まで）*/
    /* の探索条件データを設定する                                    */
    for ( i = 0; i < 50; i++) {
      /* ファイルからデータを読み込む */
      fgets(indata, MAXCOLUMN, input);
      if (feof(input)){
        /* ファイルの終わりに到達したら，ここまでの配列要素数を  */
        /* 一括削除回数に設定してfor文を抜ける                  */
        xdelete_num= i;
        break;
      }
      sscanf(indata, "%4s", in_scode);
      /* 配列変数の要素に探索条件データを設定 */
      strncpy(xscode[i], in_scode, 5);
```

```
    }
    /* DELETE実行 */
    EXEC SQL FOR :xdelete_num
      DELETE FROM ZAIKO WHERE SCODE = :xscode ;
  }

  EXEC SQL COMMIT;
  printf(" *** normal ended ***¥n");
FIN:
  if (input != NULL) {
    fclose(input);
  }
  EXEC SQL WHENEVER SQLERROR CONTINUE;
  EXEC SQL WHENEVER SQLWARNING CONTINUE;
  EXEC SQL DISCONNECT;
  return(0);
}
void abnormalend()
{
  int  wsqlcode;
  if (input != NULL) {
    fclose(input);
   }
  wsqlcode = -SQLCODE;
  printf("¥n*** HiRDB SQL ERROR SQLCODE = %d ¥n", wsqlcode);
  printf("SQLERRMC = %s¥n", SQLERRMC);
  EXEC SQL ROLLBACK;
  EXEC SQL DISCONNECT;
  exit(1);
}
```

4.9　グループ分け高速化機能

4.9.1　概要

SQL の GROUP BY 句を指定してグループ分け処理をする場合，ソートしてからグループ分けをしています。これにハッシングを組み合わせてグループ分けすることで高速なグループ分け処理が実現できます。この機能は，グループ分けのグループ数が少なく，元の行数が多いほど，グループ分けの処理時間が短縮できます。

HiRDB/パラレルサーバの場合は，フロータブルサーバの使用方法が性能に影響するため，グループ分け処理方式も考慮する必要があります。グループ分け処理方式については，「グループ分け処理方式（HiRDB/パラレルサーバ限定）」を参照してください。

4.9.2　適用条件

グループ分け高速化機能は，次に示す条件をすべて満たす SQL を実行する場合に適用されます。

- **HiRDB/パラレルサーバの場合**
  - GROUP BY 句を指定している。
  - システム共通定義，フロントエンドサーバ定義，クライアント環境定義，又はルーチン定義の SQL 最適化オプションでグループ分け高速化機能を利用することを定義している。
  - 選択式列長が 4,096 以下である。
  - 集合演算（UNION,EXCEPT）の入力となる問合せ指定のグループ分け処理ではない。
  - 集合関数内に DISTINCT 指定がない。
  - 集合関数内に，定義長が 256 バイト以上の文字列型，若しくは BINARY 型，又は BLOB 型の列の指定がない。
  - GROUP BY 句が指定されている問合せ指定中に，HAVING 句が指定されているならば，その HAVING 句に副問合せの指定がない。
  - 選択式に副問合せの指定がない。

  ただし，次の場合は SQL 最適化オプションの指定にかかわらず高速に処理をしています。
  - グループ化列のインデクスを利用することで，ソート処理をしないでグループ分けを実行できる。

  また，グループ分け高速化機能を使用する場合，次の機能は適用できません。
  - 複数のオブジェクトを作成する機能
  - AND の複数インデクス利用（ただし，構造化繰返し述語，又はインデクス型プラグイン専用関数の条件については適用します）
- **HiRDB/シングルサーバの場合**

- GROUP BY 句を指定している。
- システム共通定義，クライアント環境定義，又はルーチン定義の SQL 最適化オプションでグループ分け高速化機能を利用することを定義している。
- 集合演算（UNION，EXCEPT）の入力となる問合せ指定のグループ分け処理ではない。
- 集合関数内に DISTINCT 指定がない。
- 集合関数内に，定義長が 256 バイト以上の文字列型，若しくは BINARY 型，又は BLOB 型の列の指定がない。

ただし，次の場合は SQL 最適化オプションの指定にかかわらず高速に処理をしています。

- グループ化列のインデクスを利用する，又はソートマージ結合のための結合列のソートによって，ソート処理をしないでグループ分けを実行できる。

また，グループ分け高速化機能を使用する場合，次の機能は適用しません。

- AND の複数インデクス利用（ただし，構造化繰返し述語，又はインデクス型プラグイン専用関数の条件については適用します）

4.9.3 指定方法

グループ分け高速化機能を使用するには，SQL 最適化オプションに「RAPID_GROUPING」，又は 1,024 を加算した値を指定します。SQL 最適化オプションの指定方法については，次の箇所を参照してください。

- マニュアル「HiRDB システム定義」の pd_optimize_level
- マニュアル「HiRDB SQL リファレンス」の SQL 最適化オプション
- 「クライアント環境定義の設定内容」の PDSQLOPTLVL

4.9.4 チューニング方法

グループ分けするグループ数が多い場合，グループ分け高速化機能の効果が出ないことがあります。この場合，クライアント環境定義の PDAGGR オペランドに必要な大きさの値（グループ数以上の値）を指定します。ただし，プロセス固有メモリの使用量が多くなるので注意が必要です。メモリ使用量が多くなり，必要な大きさの値が指定できない場合は，グループ数以下の値で指定できる最大値にしてください。また，グループ数よりも大きな値を指定しても，グループ数と同じ値を指定した場合と比べて効果は変わりません。PDAGGR オペランドについては，「クライアント環境定義の設定内容」を参照してください。

PDAGGR に指定するグループ数の求め方を次に示します。

- **SQL を使用して求める方法**
  グループ化したい列を GROUP BY 句に指定し，かつ選択式に COUNT(*)を指定した SELECT 文を実行してください。
  実行した SELECT 文の結果の値がグループ数になります。

- **SQL 実行時の中間結果情報を使用して求める方法**

  SQL ごとに「SQL 実行時の中間結果情報」を取得してください。

  「SQL 実行時の中間結果情報」中にある「グループ分け処理の出力行数」の最大値がグループ数になります。

  「SQL 実行時の中間結果情報」の取得方法と，「グループ分け処理の出力行数」の詳細は，「UAP 統計レポート機能」を参照してください。

4.10　複数接続機能

4.10.1　機能概要

(1)　複数接続機能とは

複数接続機能は，HiRDB クライアントで一つの UAP から，HiRDB サーバに対して複数の CONNECT を別々に行えるようにする機能です。

複数接続機能のそれぞれの接続は独立していて，サーバプロセスは接続ごとに割り当てられて別々のトランザクションとして処理されるので，UAP は複数の SQL 文を同時に実行できます。一つの UAP から複数接続できるので，実行する UAP の数を削減でき，全体としての UAP のメモリ所要量を削減できます。

サーバの最大接続数は，接続ごとに別ユーザとしてカウントされるので，ユーザ数の上限ではなく，同時に CONNECT する接続数の上限となります。

複数接続機能の特長を次に示します。

- 接続ごとに，異なる認可識別子，パスワードを使用できます。
- 接続ごとに，異なるサーバマシンのサーバへ接続できるので，一つの UAP から複数のサーバマシンのサーバに同時に接続して，SQL 文を実行できます。
- 複数接続機能は，クライアントライブラリを接続できるすべてのサーバに対して使用できます。

(2)　X/Open XA インタフェース環境下での複数接続機能

X/Open XA インタフェース環境下で複数接続機能を使用すると，一つのトランザクションマネジャ(OpenTP1など）下の UAP から，XA インタフェースを使用して複数の HiRDB にアクセスできます。XA インタフェースを使用するので，同期をとって複数の HiRDB にアクセスしたトランザクション間の処理を制御できます。

xa_open()関数に指定するオープン文字列に，環境変数（クライアント環境定義）を設定したファイルの名称を指定します。xa_open()関数では，設定された環境変数に従って HiRDB に接続します。なお，SQL の発行先は，xa_open()関数によって接続した接続先の中から選択できます。

X/Open XA インタフェース環境下での複数接続機能は，クライアントのプラットフォームが次の場合にだけ使用できます。

- AIX
- Linux（シングルスレッド）
- Windows

4.10.2　処理概要

複数接続機能の処理概要を次の図に示します。

- 図「複数接続機能の処理概要（マルチスレッドを使用しない場合）」
- 図「複数接続機能の処理概要（マルチスレッドを使用した場合）」
- 図「複数接続機能の処理概要（スレッド間で接続を共有する場合）」
- 図「複数接続機能の処理概要（シングルスレッドの OLTP 下で，X/Open に従った API を使用した AP の場合）」
- 図「複数接続機能の処理概要（マルチスレッドの OLTP 下で，X/Open に従った API を使用した AP の場合）」

図 4–60　複数接続機能の処理概要（マルチスレッドを使用しない場合）

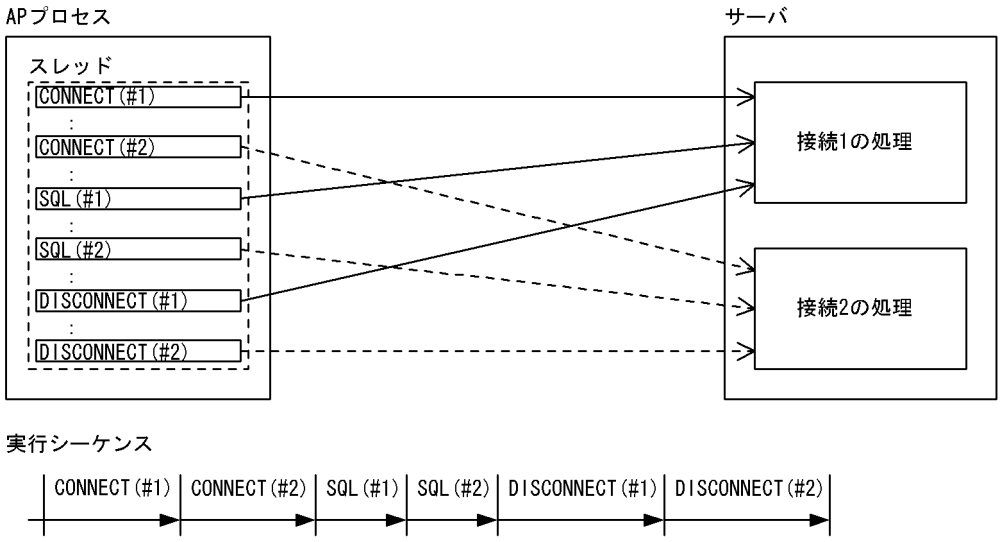
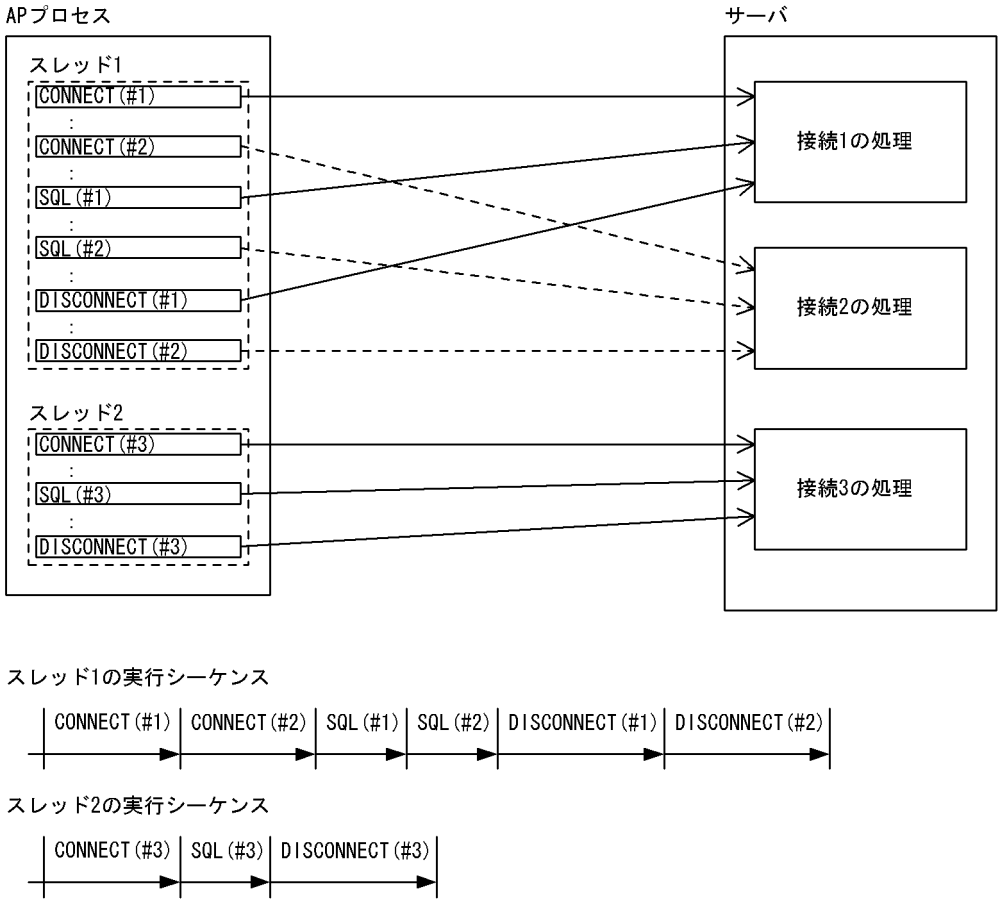


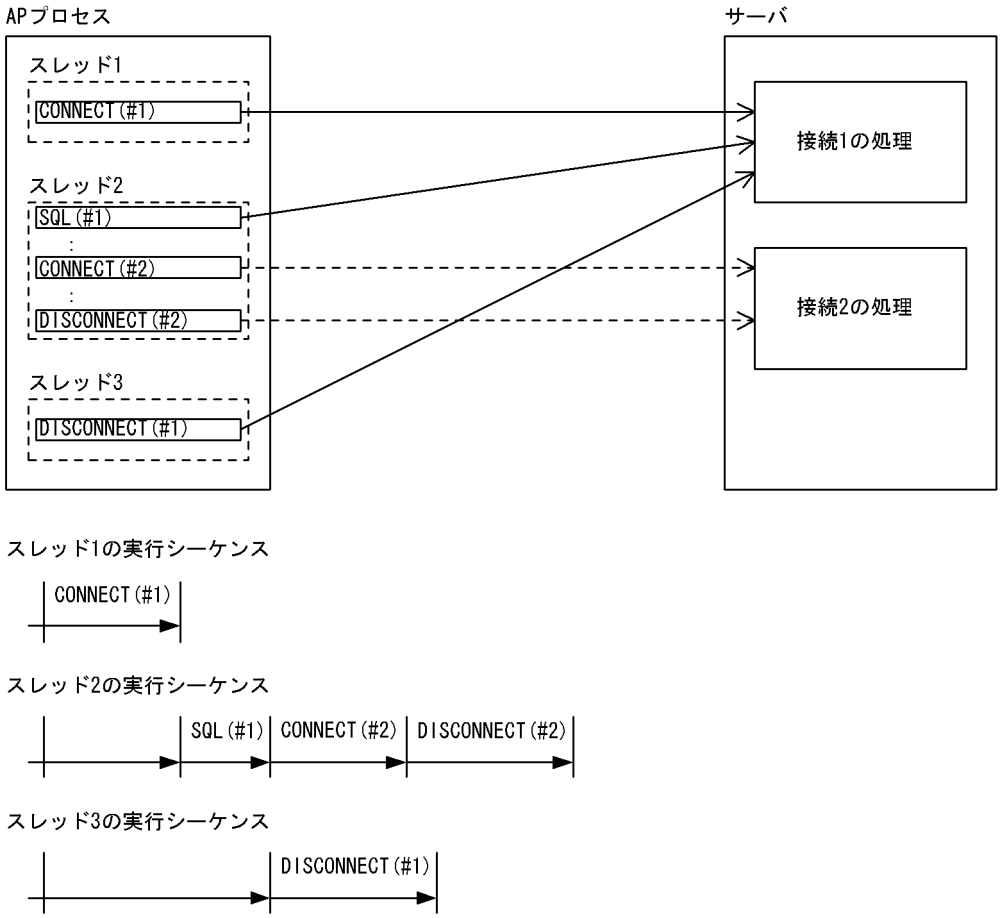
図 4–61　複数接続機能の処理概要（マルチスレッドを使用した場合）



注

それぞれの接続が独立しているので，スレッドごとに同時に SQL を実行できます。

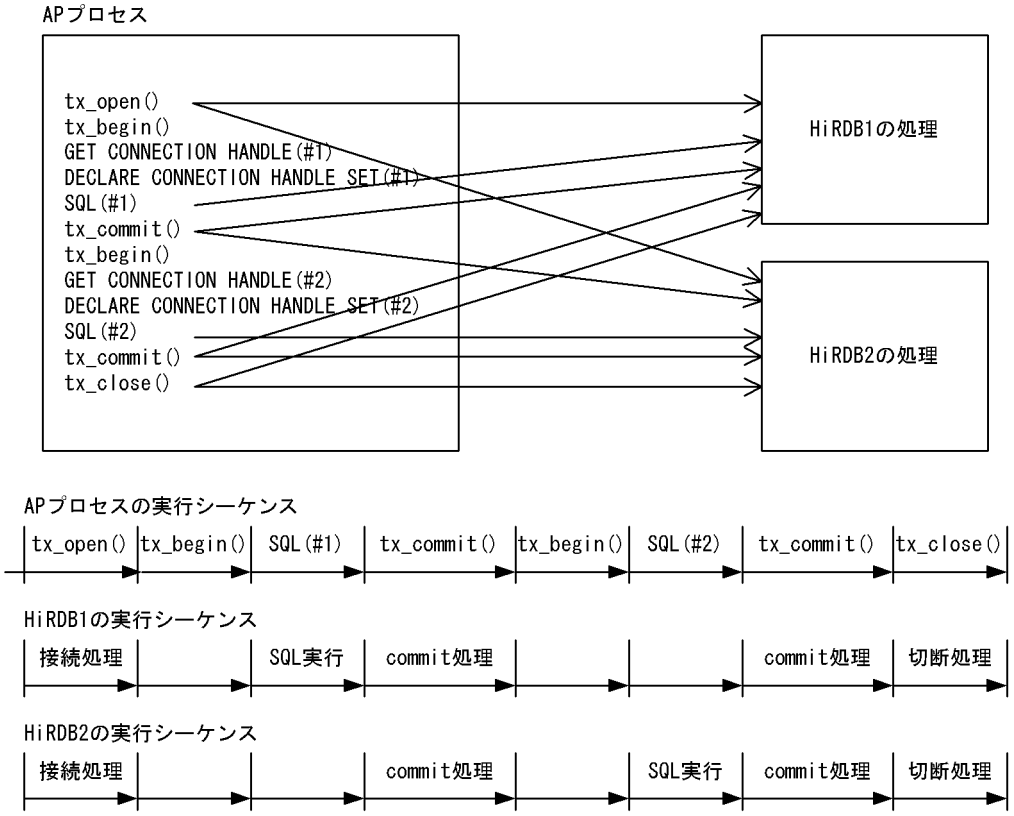
図 4–62　複数接続機能の処理概要（スレッド間で接続を共有する場合）



注

このような例の場合，スレッド 1 が CONNECT(#1)を実行する前に，スレッド 2 が SQL(#1)を実行したり，スレッド 3 が DISCONNECT(#1)を実行したりしないように，ユーザがスレッド間の処理の同期をとる必要があります。

図 4–63　複数接続機能の処理概要（シングルスレッドの OLTP 下で，X/Open に従った API を使用した AP の場合）

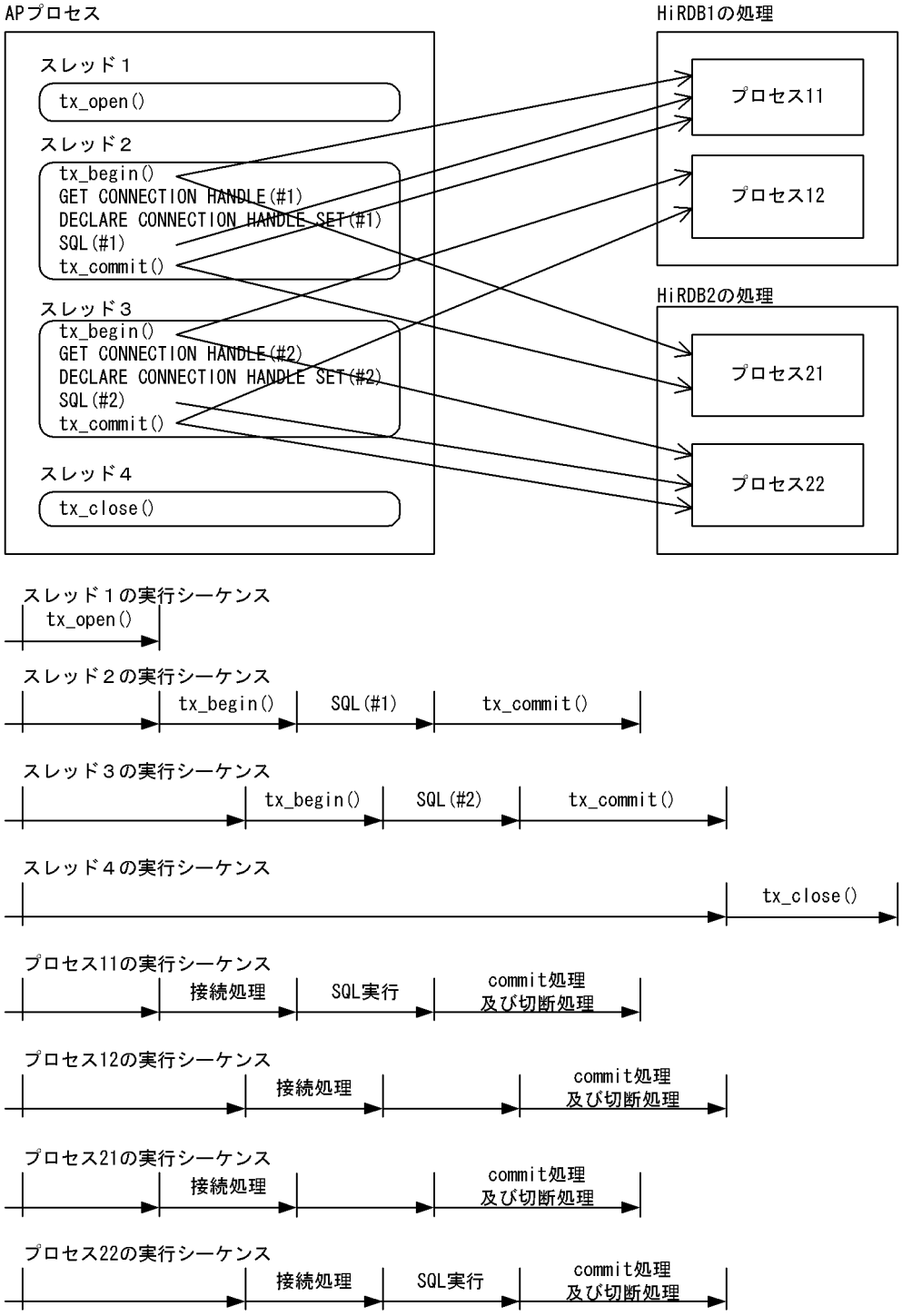


[説明]

あらかじめ，OLTP システムには HiRDB1 と HiRDB2 を登録しておきます。

OLTP システムは tx_open()時に登録してあるすべての HiRDB に接続します。SQL を実行するときに接続先を選択してください。

図 4-64　複数接続機能の処理概要（マルチスレッドの OLTP 下で，X/Open に従った API を使用した AP の場合）



[説明]

あらかじめ，OLTP システムには HiRDB1 と HiRDB2 を登録しておきます。

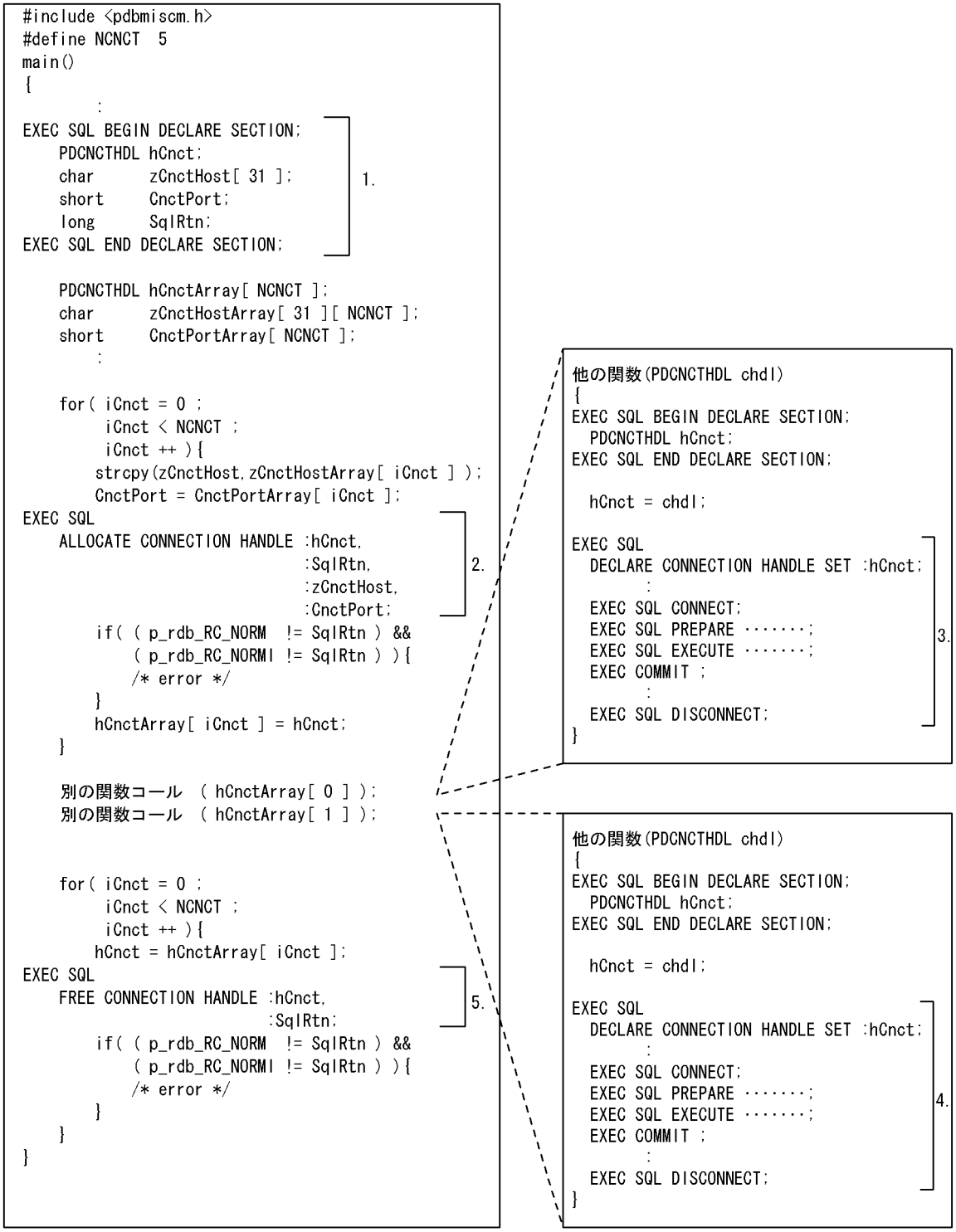
OLTP システムは tx_begin()時に登録してあるすべての HiRDB に接続します。SQL を実行するときに接続先を選択してください。それぞれのトランザクションが独立しているので，スレッドごとに SQL を同時実行できます。

4.10.3　コーディング例

(1)　通常の UAP

複数接続機能を使用した UAP のコーディング例（C 言語の場合）を次の図に示します。

図 4–65　複数接続機能を使用した UAP のコーディング例（C 言語の場合）

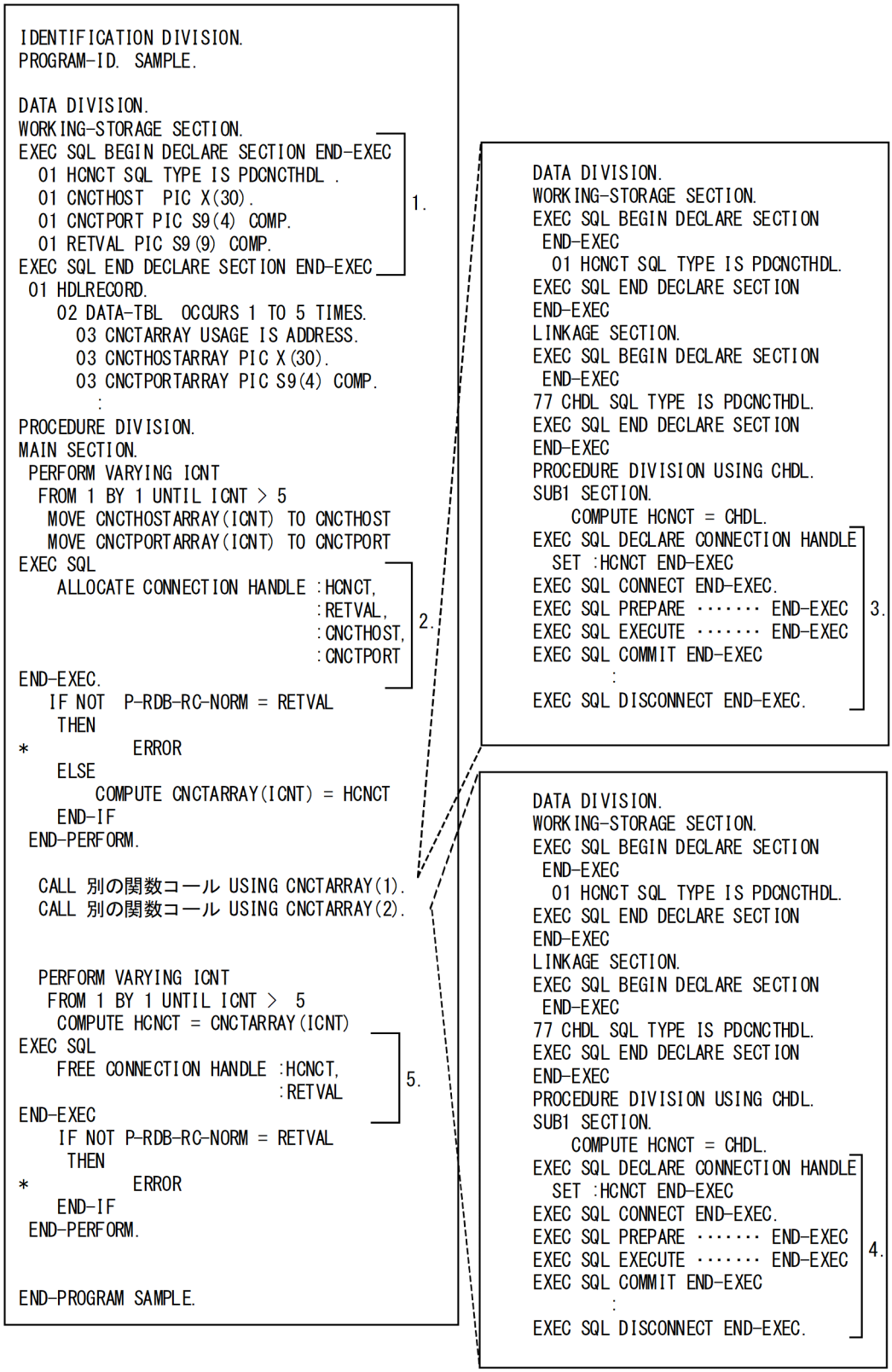


［説明］

1. 接続ハンドルの定義
2. 接続ハンドルの取得
3. 接続 1 の HiRDB の処理
4. 接続 2 の HiRDB の処理
5. 接続ハンドルの解放

複数接続機能を使用した UAP のコーディング例（COBOL 言語の場合）を次の図に示します。

図 4-66　複数接続機能を使用した UAP のコーディング例（COBOL 言語の場合）



```
IDENTIFICATION DIVISION.
PROGRAM-ID.  SAMPLE.

DATA DIVISION.
WORKING-STORAGE SECTION.
EXEC SQL BEGIN DECLARE SECTION END-EXEC
  01 HCNCT SQL TYPE IS PDCNCTHDL .
  01 CNCTHOST  PIC X(30).
  01 CNCTPORT PIC S9(4) COMP.
  01 RETVAL PIC S9(9) COMP.
EXEC SQL END DECLARE SECTION END-EXEC
 01 HDLRECORD.
    02 DATA-TBL  OCCURS 1 TO 5 TIMES.
      03 CNCTARRAY USAGE IS ADDRESS.
      03 CNCTHOSTARRAY PIC X(30).
      03 CNCTPORTARRAY PIC S9(4) COMP.
        :
PROCEDURE DIVISION.
MAIN SECTION.
 PERFORM VARYING ICNT
  FROM 1 BY 1 UNTIL ICNT > 5
   MOVE CNCTHOSTARRAY(ICNT) TO CNCTHOST
   MOVE CNCTPORTARRAY(ICNT) TO CNCTPORT
EXEC SQL
    ALLOCATE CONNECTION HANDLE :HCNCT,
                               :RETVAL,
                               :CNCTHOST,
                               :CNCTPORT
END-EXEC.
   IF NOT  P-RDB-RC-NORM = RETVAL
    THEN
*          ERROR
    ELSE
        COMPUTE CNCTARRAY(ICNT) = HCNCT
    END-IF
 END-PERFORM.

  CALL 別の関数コール USING CNCTARRAY(1).
  CALL 別の関数コール USING CNCTARRAY(2).


  PERFORM VARYING ICNT
   FROM 1 BY 1 UNTIL ICNT >  5
    COMPUTE HCNCT = CNCTARRAY(ICNT)
EXEC SQL
    FREE CONNECTION HANDLE :HCNCT,
                           :RETVAL
END-EXEC
    IF NOT P-RDB-RC-NORM = RETVAL
     THEN
*          ERROR
    END-IF
 END-PERFORM.


END-PROGRAM SAMPLE.
```

1.

2.

5.

```
DATA DIVISION.
WORKING-STORAGE SECTION.
EXEC SQL BEGIN DECLARE SECTION
 END-EXEC
  01 HCNCT SQL TYPE IS PDCNCTHDL.
EXEC SQL END DECLARE SECTION
END-EXEC
LINKAGE SECTION.
EXEC SQL BEGIN DECLARE SECTION
 END-EXEC
77 CHDL SQL TYPE IS PDCNCTHDL.
EXEC SQL END DECLARE SECTION
END-EXEC
PROCEDURE DIVISION USING CHDL.
SUB1 SECTION.
    COMPUTE HCNCT = CHDL.
EXEC SQL DECLARE CONNECTION HANDLE
  SET :HCNCT END-EXEC
EXEC SQL CONNECT END-EXEC.
EXEC SQL PREPARE ····· END-EXEC
EXEC SQL EXECUTE ····· END-EXEC
EXEC SQL COMMIT END-EXEC
        :
EXEC SQL DISCONNECT END-EXEC.
```

3.

```
DATA DIVISION.
WORKING-STORAGE SECTION.
EXEC SQL BEGIN DECLARE SECTION
 END-EXEC
  01 HCNCT SQL TYPE IS PDCNCTHDL.
EXEC SQL END DECLARE SECTION
END-EXEC
LINKAGE SECTION.
EXEC SQL BEGIN DECLARE SECTION
 END-EXEC
77 CHDL SQL TYPE IS PDCNCTHDL.
EXEC SQL END DECLARE SECTION
END-EXEC
PROCEDURE DIVISION USING CHDL.
SUB1 SECTION.
    COMPUTE HCNCT = CHDL.
EXEC SQL DECLARE CONNECTION HANDLE
  SET :HCNCT END-EXEC
EXEC SQL CONNECT END-EXEC.
EXEC SQL PREPARE ····· END-EXEC
EXEC SQL EXECUTE ····· END-EXEC
EXEC SQL COMMIT END-EXEC
        :
EXEC SQL DISCONNECT END-EXEC.
```

4.

注

SQL は，SQL 先頭子，及び SQL 終了子も含めて，すべて B 領域（第 12 欄～72 欄）に記述してください。

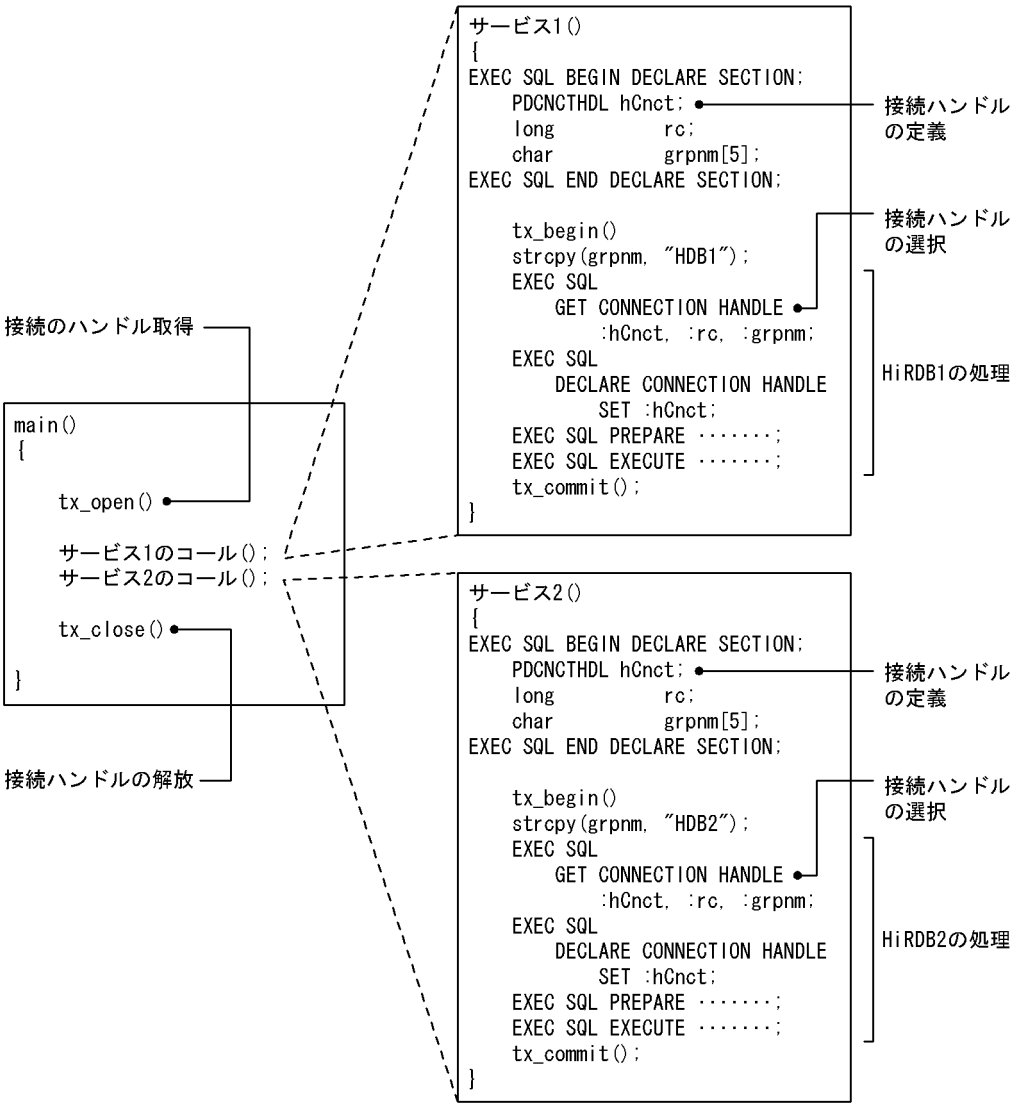
[説明]

1. 接続ハンドルの定義
2. 接続ハンドルの取得
3. 接続 1 の HiRDB の処理
4. 接続 2 の HiRDB の処理
5. 接続ハンドルの解放

(2) OLTP 下で X/Open に従った API を使用した UAP

OLTP 下で X/Open に従った API を使用した UAP で複数接続機能を使用したコーディングの例（C 言語の場合）を次の図に示します。

図 4–67 OLTP 下で X/Open に従った API を使用した UAP で複数接続機能を使用したコーディングの例（C 言語の場合）

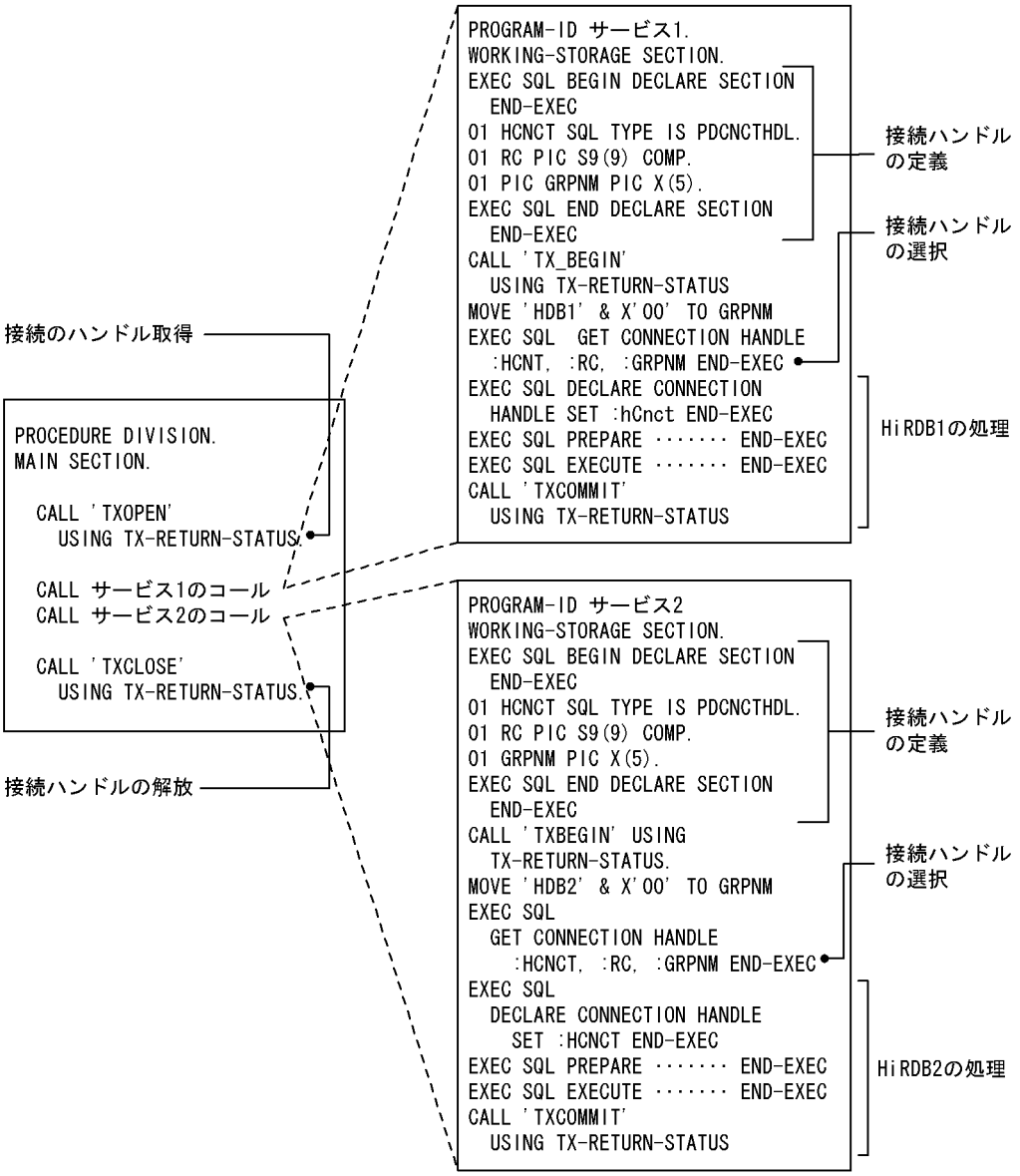


［説明］

あらかじめ，OLTP システムには HiRDB1(環境変数グループ識別子 HDB1)と HiRDB2(環境変数グループ識別子 HDB2)を登録しておきます。HiRDB のトランザクションマネジャへの登録については，マニュアル「HiRDB システム導入・設計ガイド」を参照してください。

OLTP 下で X/Open に従った API を使用した UAP で複数接続機能を使用したコーディングの例（COBOL 言語の場合）を次の図に示します。

図 4–68　OLTP 下で X/Open に従った API を使用した UAP で複数接続機能を使用したコーディングの例（COBOL 言語の場合）



注

SQL は，SQL 先頭子，及び SQL 終了子も含めて，すべて B 領域（第 12 欄～72 欄）に記述してください。

［説明］

あらかじめ，OLTP システムには HiRDB1(環境変数グループ識別子 HDB1)と HiRDB2(環境変数グループ識別子 HDB2)を登録しておきます。HiRDB のトランザクションマネジャへの登録については，マニュアル「HiRDB システム導入・設計ガイド」を参照してください。

4.10.4 規則

1. 複数接続機能を使用する場合，専用のライブラリをリンクする必要があります。詳細については，「複数接続機能を使用する場合のコンパイルとリンケージ」を参照してください。
2. 複数接続機能用ライブラリを使用する UAP では，一つの接続を保持したままスレッドを分岐し，そのスレッドが SQL を実行する場合，ほかの SQL を発行するスレッドとの間で処理のシリアライズを必要とします。したがって，同一接続に対する SQL は同時に発行できません。異なる接続に対する SQL は同時に発行できます。
3. ALLOCATE CONNECTION HANDLE，FREE CONNECTION HANDLE のエラー情報を参照する場合，SQLCODE，SQLERRM ではなく，リターンコード受け取り変数の値を参照してください。リターンコード受け取り変数については，マニュアル「HiRDB SQL リファレンス」を参照してください。
4. SQL 連絡領域を参照する場合，DECLARE CONNECTION HANDLE SET 文で参照する SQL 連絡領域に対応した接続ハンドルを宣言しておく必要があります。
5. COBOL 言語の場合，複数接続機能を使用する SQL 文を含む UAP で，DECLARE CONNECTION HANDLE SET の記述前（有効範囲外）に接続ハンドルの割り当て，取得以外の SQL 文を記述できません。
6. COBOL 言語の場合，DECLARE CONNECTION HANDLE UNSET は使用できません。
7. 複数接続機能は，マルチスレッド（DCE スレッド，リアルスレッド），及びシングルスレッド対応の UAP で使用できます。マルチスレッド対応の UAP で複数接続機能を使用する場合，HiRDB での UAP の開発知識のほかに，DCE スレッド，リアルスレッドを用いた UAP の開発知識を必要とします。
8. Windows 版での複数接続機能は，X/Open XA インタフェースを使用しない場合，マルチスレッド対応 UAP でだけ使用できます。したがって，Visual Studio を用いた UAP のコンパイルで使用する，C のランタイムライブラリの指定は，マルチスレッド・DLL を選択してください（「コンパイルオプション：コード生成」で，「マルチスレッド・DLL」を指定します）。
9. Windows 版での複数接続機能は，X/Open XA インタフェースを使用する場合，シングルスレッド対応 UAP でも使用できます。なお，シングルスレッド対応 UAP の場合でも，Visual Studio を用いた UAP のコンパイルで使用する，C のランタイムライブラリの指定は，マルチスレッド・DLL を選択してください（「コンパイルオプション：コード生成」で，「マルチスレッド・DLL」を指定します）。
10. C，及び C++言語で SQL 連絡領域を参照する場合，直接 SQLCA の構造体を参照しないで，SQL で始まるマクロ名を使用して参照してください。使用するマクロ名については「C 言語の場合」を参照してください。

4.11　絞込み検索

4.11.1　絞込み検索とは

段階的に対象レコードを絞り込む検索のことを**絞込み検索**といいます。

絞込み検索をする場合，操作系 SQL の ASSIGN LIST 文でリストを作成します。リストとは，適当な件数になるまで条件を指定して段階的にデータを絞り込んでいくような情報検索をするために，その途中段階のデータの集合を一時的に名前（リスト名）を付けて保存したもの，又は保存したデータの集合を意味します。

ある条件で作成したリストがあれば，そのリストを使用することで処理速度の向上が図れます。また，複数の条件を指定する場合は，複数のリストを組み合わせた検索もできます。

4.11.2　絞込み検索をするためには

絞込み検索をするためには，あらかじめ次の準備をしておく必要があります。

- システム定義の指定
- リスト用 RD エリアの作成

システム定義を指定し，リスト用 RD エリアを作成すると，絞込み検索ができるようになります（リストを作成できるようになります）。

(1)　システム定義の指定

絞込み検索をする場合，システム定義の絞込み検索に関するオペランドを指定する必要があります。絞込み検索をする場合に必ず指定するオペランドを次に示します。

- pd_max_list_users（最大リスト作成ユーザ数）
- pd_max_list_count（1 ユーザ当たりの最大作成可能リスト数）

また，必要に応じて次のオペランドも指定できます。

- pd_max_list_users_wrn_pnt（pd_max_list_users の指定値に対する使用率警告メッセージの出力契機の指定）
- pd_max_list_count_wrn_pnt（pd_max_list_count の指定値に対する使用率警告メッセージの出力契機の指定）
- pd_rdarea_list_no_wrn_pnt（サーバ内に作成できる最大リスト数に対する使用率警告メッセージの出力契機の指定）

これらのシステム定義のオペランドについては，マニュアル「HiRDB システム定義」を参照してください。

(2)　リスト用 RD エリアの作成

データベース初期設定ユティリティ（pdinit），又はデータベース構成変更ユティリティ（pdmod）で，リスト用 RD エリアを作成します。データベース初期設定ユティリティ，及びデータベース構成変更ユティリティについては，マニュアル「HiRDB コマンドリファレンス」を参照してください。

また，リスト用 RD エリアに指定する HiRDB ファイルシステム領域は，使用目的を WORK にしてください。リスト用 RD エリアの設計については，マニュアル「HiRDB システム導入・設計ガイド」を参照してください。

4.11.3　リストを使用した検索

リストを使用した検索方法について説明します。

リストを使用した検索の例を次の図に示します。

図 4-69　リストを使用した検索の例

ZAIKO（在庫表）

商品コード (SCODE)	商品名 (SNAME)	色 (COL)	単価 (TANKA)	在庫量 (ZSURYO)
101L	ブラウス	青	3500	62
101M	ブラウス	白	3500	85
201M	ポロシャツ	白	3640	29
202M	ポロシャツ	赤	3640	67
302S	スカート	白	5110	65
353L	スカート	赤	4760	18
353M	スカート	緑	4760	56
411M	セーター	青	8400	12
412M	セーター	赤	8400	22
591L	ソックス	赤	250	300
591M	ソックス	青	250	90
591S	ソックス	白	250	280

商品名がスカートの行を選択したLIST1を作成する

SQL文

```
ASSIGN LIST LIST1
  FROM (ZAIKO) WHERE SNAME = N'スカート'
```

リスト名：LIST1

商品コード (SCODE)	商品名 (SNAME)	色 (COL)	単価 (TANKA)	在庫量 (ZSURYO)
302S	スカート	白	5110	65
353L	スカート	赤	4760	18
353M	スカート	緑	4760	56

単価が5000円以上の行を選択したLIST2を作成する

SQL文

```
ASSIGN LIST LIST2
  FROM (ZAIKO) WHERE TANKA >= 5000
```

リスト名：LIST2

商品コード (SCODE)	商品名 (SNAME)	色 (COL)	単価 (TANKA)	在庫量 (ZSURYO)
302S	スカート	白	5110	65
411M	セーター	青	8400	12
412M	セーター	赤	8400	22

LIST1とLIST2の集合積を求めたLIST3を作成する（商品名がスカートで，かつ単価が5000円以上の行を求める）

SQL文

```
ASSIGN LIST LIST3 FROM LIST1 AND LIST2
```

リスト名：LIST3

商品コード (SCODE)	商品名 (SNAME)	色 (COL)	単価 (TANKA)	在庫量 (ZSURYO)
302S	スカート	白	5110	65

在庫表（ZAIKO）から，商品名がスカートで，かつ単価が5000円以上のデータを検索したい場合は，リスト（LIST3）を検索します。在庫表から条件を指定して検索するより，LIST3を検索した方が処理は速くなります。

SQL文

```
SELECT * FROM LIST LIST3
```

商品コード (SCODE)	商品名 (SNAME)	色 (COL)	単価 (TANKA)	在庫量 (ZSURYO)
302S	スカート	白	5110	65

4.11.4　リストを使用するトランザクションでロールバックが発生した場合の処置

SQL の ROLLBACK 文，又はエラーによってトランザクションが取り消された場合，そのトランザクションで作成，又は削除していたリストについては，必要に応じて再作成しなければならない場合があります。トランザクションが取り消されたときに作成，又は削除していたリストの状態とユーザの処置を次に示します。

取り消されたトランザクションでのリストの操作		リストの状態	ユーザの処置
トランザクション内で ASSIGN LIST 文で作成していたリスト	トランザクション開始前にはないリスト名を指定して，リストを作成した場合	作成したリストはなくなります。	トランザクション処理を再度実行してください。
	トランザクション開始前にあったリスト名を指定して，リストを作成した場合	トランザクション開始前にあったリスト名と同じ名前のリストは，使用できなくなります（検索するとエラー)。※	トランザクション開始前にあったリスト名と同じ名前のリストを，トランザクションで使用したい場合，そのリストを再作成してください。その後，トランザクションを再度実行してください。
トランザクション内で DROP LIST 文の削除対象となっていたリスト	トランザクション開始前に削除対象リストがなかった場合	削除したリストはなくなります。	トランザクション処理を再度実行してください。
	トランザクション開始前に削除対象リストがあった場合	トランザクション開始前になかったリストはなくなります。トランザクション開始前にあった削除対象リストは使用できなくなります(検索するとエラー)。※	トランザクション開始前にあった削除対象リストをトランザクションで使用したい場合，そのリストを再作成してください。その後，トランザクションを再度実行してください。

注※

トランザクションの取り消しのタイミングによっては，正常に使用できる場合もあります。

4.11.5　HiRDB の起動と停止時のリストの自動削除

HiRDB を終了し，開始した場合，開始モードに関係なく作成済みのリストはすべて削除されます。

HiRDB/パラレルサーバの場合，ユニット単位の終了，開始をすると，そのユニットにあるリスト用 RD エリア内のリストがすべて削除されます。また，サーバ単位の終了，開始をすると，そのサーバにあるリスト用 RD エリア内のリストがすべて削除されます。削除されたリストを検索するとエラーとなります。

なお，HiRDB がユニット単位に異常終了した場合，及び HiRDB を構成するすべてのユニットが停止した場合，HiRDB を開始すると，作成済みのリストがすべて削除されます。一部のユニットが異常終了した場

合にそのユニットを再開始すると，ユニット下のリスト用 RD エリア内のリストがすべて削除されます。削除されたリストを検索するとエラーとなります。

このようにエラーとなる場合，次のどれかの方法でリストの削除，又はリストの再作成をしてください。

検索するとエラーになるリストを使用したい場合

ASSIGN LIST 文で，以前と同じリスト名のリストを作成してください。

検索するとエラーになるリストを使用しない場合

検索するとエラーになるリストを，DROP LIST 文で削除するか，又は HiRDB をいったん終了させ開始することで，作成済みのすべてのリストを削除してください。

4.11.6 リスト使用時の注意事項

(1) HiRDB から切り離しをした後のリスト

リストは，HiRDB から切り離しをしても削除されません。リストを削除する場合は，DROP LIST 文で削除してください。また，HiRDB システムを停止した場合は，すべてのリストが削除されます。

(2) 行の挿入，削除によるリストの状態

リストを使用した検索では，リスト作成時にあった行がリスト作成後に削除されると，その行は検索されません。また，リスト作成後に更新した行は，更新後のデータを取り出します。

(3) リスト作成後の行の削除，挿入

リストを使用した検索では，リスト作成後に，基表に対して行の削除をした後に，別の行を挿入した場合，挿入した行が検索されることがあります。

(4) 横分割表に対する ASSIGN LIST 文の実行

横分割した表に対して ASSIGN LIST 文を実行した場合，基表の RD エリアの一部が閉塞などの要因で検索できないときは，探索条件中の分割列に対する条件で検索できる RD エリアのデータを指定していてもエラーとなります。

(5) 同一ユーザのリストの操作

同一ユーザが，複数同時に HiRDB と接続してリストを操作できません。

(6) ディクショナリサーバ，又はディクショナリサーバがあるユニットの停止

HiRDB／パラレルサーバの場合，ディクショナリサーバ，又はディクショナリサーバがあるユニットを停止すると，リスト管理情報が失われます。この結果，それまで作成していたすべてのリストが操作（検索，

削除，変更）できなくなります（操作するとエラーになります）。操作するとエラーになるリストを使用したい場合は，ASSIGN LIST 文で以前と同じリスト名のリストを作成してください。

また，ディクショナリサーバを再起動した場合，停止以前に開始した他ユーザのリストを使用したトランザクションの回復が完了するまでの間，リストを使用した処理が KFPA11998-E エラー（トランザクション未決着状態でのリスト操作）となることがあります。

(7) データベース回復ユティリティでのリストの基表の回復

ログを使用してリストの基表を最新の状態まで回復した場合，作成済みのリストをそのまま使用できます。ただし，バックアップだけを使用した回復，ログを使用した時間指定の回復，又は最新のログを使用しない回復のどれかの場合は，必ず次のどれかの方法で回復した表を基にしたリストをすべて削除，又は再作成してください。

リストを使用する場合

ASSIGN LIST 文で，以前と同じリスト名のリストを作成してください。

リストを使用しない場合

DROP LIST 文でリストを削除するか，又は HiRDB をいったん終了させ開始することで，作成済みのすべてのリストを削除してください。

(8) リストの基表を格納している RD エリアの再初期化

必ず次のどれかの方法で，再初期化した RD エリアに格納されている表を基にしたリストを，すべて削除，又は再作成してください。

リストを使用する場合

ASSIGN LIST 文で，以前と同じリスト名のリストを作成してください。

リストを使用しない場合

DROP LIST 文でリストを削除するか，又は HiRDB をいったん終了させ開始することで，作成済みのすべてのリストを削除してください。

(9) リストの基表に対する再編成，作成モードのデータロード，又は PURGE TABLE 文の実行

リストの基表に対して再編成，作成モードのデータロード，又は PURGE TABLE 文を実行した場合，以前にその表を基に作成したリストの検索結果が不正になります。このリストを使用するためには，ASSIGN LIST 文でリストを再作成する必要があります。

(10) インナレプリカ機能を使用した場合の絞込み検索

インナレプリカ機能を使用して，pddbchg コマンド，UAP 環境定義 PDDBACCS，又はクライアント環境定義の PDDBACCS でアクセス対象 RD エリアを切り替えた場合，次のどちらかの条件を満たしていないと検索結果が不正になります。

- リスト検索時のアクセス対象 RD エリアとリスト作成時のアクセス対象 RD エリアが一致している
- リスト検索時のアクセス対象 RD エリアがリスト作成時のアクセス対象 RD エリアからデータ複製されている

リストを使用する場合は，次のどれかの対処をしてください。

- リスト作成時のアクセス対象 RD エリアを使用する
- リスト作成時のアクセス対象 RD エリアからデータ複製されたアクセス対象 RD エリアを使用する
- 現在のアクセス対象 RD エリアでリストを再作成する

4.12　BLOB データのファイル出力機能

4.12.1　BLOB データのファイル出力機能とは

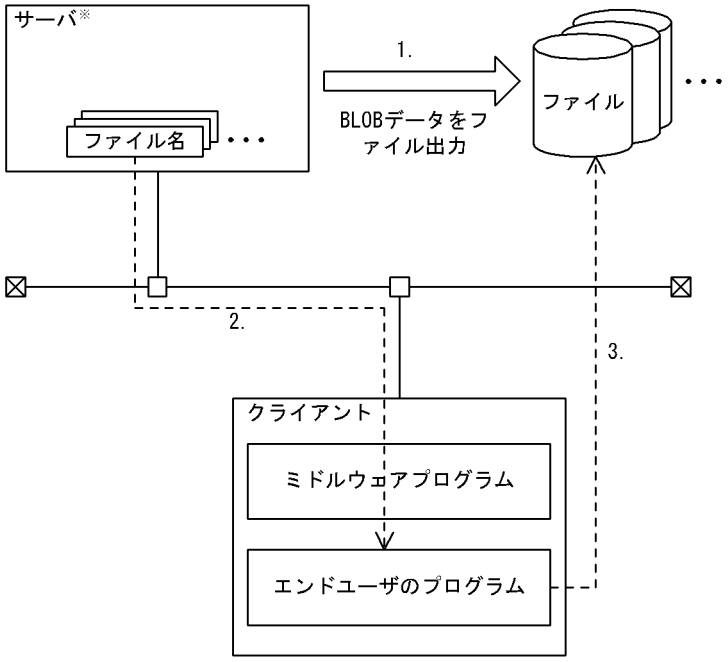
BLOB データを検索する場合，BLOB データを格納するためのメモリ領域をクライアント側で用意する必要があります。さらに，サーバ側では BLOB データ返却の送信バッファ，クライアントライブラリでの BLOB データ受け取りの受信バッファのメモリも必要となります。そのため，BLOB データに合わせた長大なメモリ確保が必要になり，メモリ資源を圧迫します。

また，エンドユーザのプログラムと HiRDB の間に，HiRDB のクライアントとして動作するミドルウェアプログラムが介在する構成が増えてきているため，これらのプログラム間の BLOB データの受け渡しで更にメモリが増加する傾向にあります。

この BLOB データ検索時のメモリ増大を防ぐために，検索した BLOB データをクライアントに返却しないで，シングルサーバ，又はフロントエンドサーバがあるユニットのファイルに直接出力し，そのファイル名をクライアントに返却するのが **BLOB データのファイル出力機能**です。

BLOB データのファイル出力機能の概要を次の図に示します。

図 4–70　BLOB データのファイル出力機能の概要



注※　シングルサーバ，又はフロントエンドサーバがあるユニット
［説明］
1．クライアントからBLOBデータを検索した場合，そのBLOBデータを1行1列ごとにファイル出力します。
2．1で出力したBLOBデータのファイル名をクライアントに返却します。
3．返却されたファイル名を基に，サーバ側にあるBLOBデータのファイルをアクセスします。

4.12.2　適用基準

BLOB データ検索時に，メモリ所要量を削減したいときに適用してください。

ただし，クライアントプログラムのメモリ削減，及びサーバ，クライアント間の通信バッファのメモリ削減に効果はありますが，ファイル出力のためのディスク入出力は増加するため，メモリ所要量とディスク入出力の兼ね合いを考慮して利用してください。

4.12.3　指定方法

BLOB データのファイル出力機能は，SQL の WRITE 指定で指定します。WRITE 指定は，カーソル指定，及び問合せ指定の中に指定できます。

WRITE 指定については，マニュアル「HiRDB SQL リファレンス」を参照してください。

4.12.4　BLOB データのファイル出力機能を使用する場合の留意点

1. 作成した BLOB データのファイルが不要になった場合，ユーザが削除する必要があります。ファイルを削除する場合は，次の点に注意してください。なお，カーソルクローズ後，トランザクション解決後は，無条件に削除できます。
   - FETCH 直後に削除する場合，同じカーソル検索の直前の FETCH 結果と BLOB 値が同じときは，同じファイル名でファイルを再作成しないことがあります。この場合，直前のファイル名を記憶しておいて，ファイル名が変わったときに削除するように制御してください。
2. 障害，又はロールバックが発生しても，作成した BLOB データのファイルは削除されません。また，ファイルを削除しないでいると，ディスク容量など OS の資源を圧迫することになるので注意してください。
3. 次の機能を使用する場合，事前にディスク容量に空きがあるか確認しておいてください。
   - 配列を使用した FETCH 機能を使用する場合

     1 回の FETCH で配列用要素数分のファイルが作成されます。
   - ブロック転送機能を使用する場合

     最初の FETCH でブロック転送行数分のファイルを作成し，以降ブロック転送行数分の FETCH 終了後の，次の FETCH のたびにブロック転送行数分のファイル作成を繰り返します。
4. ほかのトランザクションやカーソル検索とファイル名が重複した場合，ファイルを互いに破壊する可能性があります。この場合，トランザクションごと，又はカーソルごとにファイル接頭辞のディレクトリ名やファイル名を変えて，名称が重複しないようにすることをお勧めします。

4.12.5　BLOB データのファイル出力機能を使用した例

BLOB データのファイル出力機能を使用した検索例を次に示します。

(1)　BLOB 列を検索する場合

表 T1 から，列 C1，C2 を検索します。このとき，C1 の BLOB データをファイル出力し，そのファイル名を取得します。

表T1

C1	C2
BLOB値１	10
BLOB値２	20
BLOB値３	30
BLOB値４	40

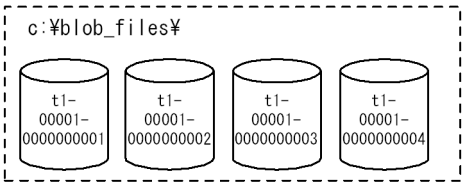
SQL文

```
SELECT WRITE(C1,'c:¥blob_files¥t1',0),C2 FROM T1
```

検索結果

C1	C2
172.16.202.5:c:¥blob_files¥t1-00001-0000000001	10
172.16.202.5:c:¥blob_files¥t1-00001-0000000002	20
172.16.202.5:c:¥blob_files¥t1-00001-0000000003	30
172.16.202.5:c:¥blob_files¥t1-00001-0000000004	40

サーバ側に出力されたBLOBデータ
・IPアドレス：172.16.202.5



(2)　BLOB 属性の抽象データ型を検索する場合

表 T2 から，CONTAINS()が真となる ADT1 列を検索します。このとき，列値を EXTRACTS()の引数に渡した結果の BLOB 値をファイルに出力し，ファイル名を取得します。なお，この例は全件ヒットした場合を示しています。

表T2

ADT1
抽象データ型値１
抽象データ型値２
抽象データ型値３
抽象データ型値４

SQL文

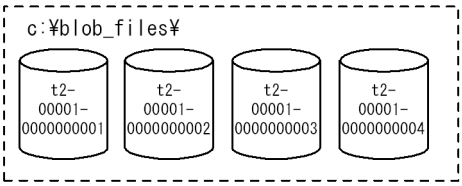
```
SELECT WRITE(EXTRACTS(ADT1,･･･),'c:¥blob_files¥t2',0) FROM T2
    WHERE CONTAINS(ADT1,･･･) IS TRUE
```

検索結果

ADT1
172.16.202.5:c:¥blob_files¥t2-00001-0000000001
172.16.202.5:c:¥blob_files¥t2-00001-0000000002
172.16.202.5:c:¥blob_files¥t2-00001-0000000003
172.16.202.5:c:¥blob_files¥t2-00001-0000000004

サーバ側に出力されたBLOBデータ

・IPアドレス：172.16.202.5



4.13　BLOB データ，BINARY データの部分的な更新・検索

4.13.1　BLOB データ，BINARY データの部分的な更新・検索とは

登録されている BLOB データ又は BINARY データ※に対して，新たなデータを追加する場合に，データ全体を更新したり，BLOB データ又は BINARY データ※を検索する場合にデータ全体を取得したりすると，サーバ及びクライアントの双方で長大なデータに合わせてメモリを多量に確保する必要があり，メモリ資源を圧迫します。**BLOB データ，BINARY データの部分的な更新・検索**は，この問題を解決するために次の機能を提供します。

- BLOB データ，BINARY データの追加更新
- BLOB データ，BINARY データの部分抽出
- BLOB データ，BINARY データの後方削除更新

注※

定義長が 32,001 バイト以上の BINARY データを示します。

(1)　BLOB データ，BINARY データの追加更新

UPDATE 文の SET 句に連結演算を指定すると，登録されている BLOB データ又は BINARY データに対して新たなデータを追加できます。また，メモリ消費量も追加分だけに抑えられます。

(2)　BLOB データ，BINARY データの部分抽出

スカラ関数 SUBSTR を指定すると，BLOB データ又は BINARY データから，指定した部分だけを抽出できます。また，メモリ消費量も抽出分だけに抑えられます。

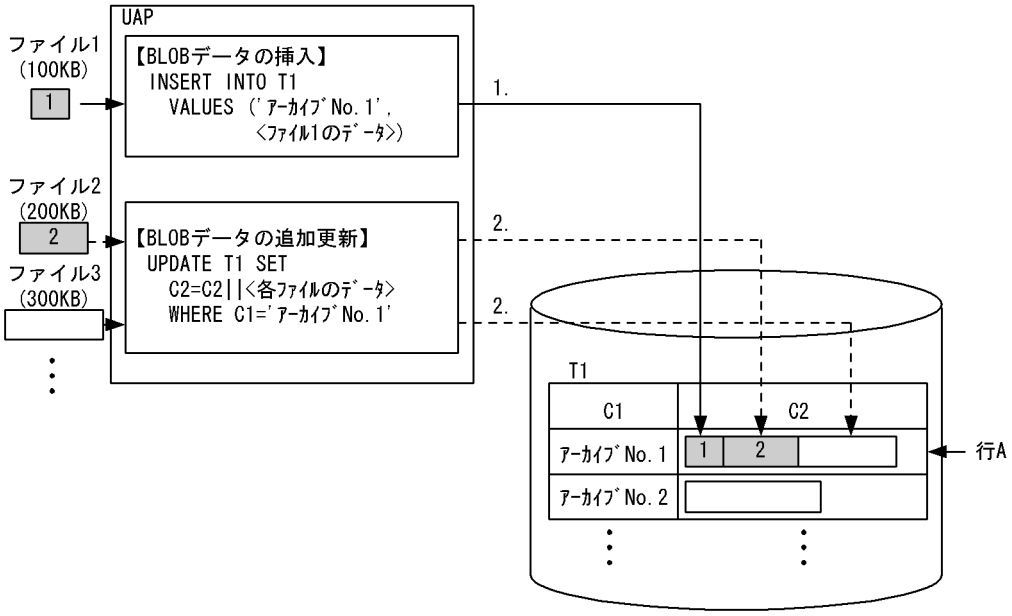
(3)　BLOB データ，BINARY データの後方削除更新

UPDATE 文の SET 句に，処理対象列，及び開始位置として定数 1 を指定したスカラ関数 SUBSTR を使用することで，BLOB データ又は BINARY データの後方部分だけを削除できます。更新データ分のメモリを取得しないで更新できるため，メモリ消費量，及びログ量を抑えられます。

4.13.2　使用例

(1)　BLOB データの追加更新

複数のファイルを一つの BLOB データとして格納します。

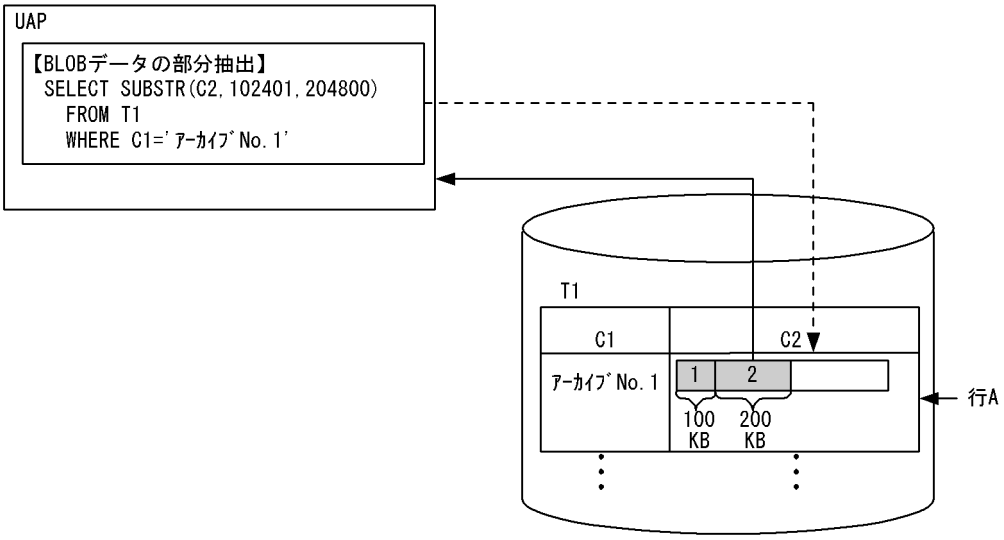


[説明]

1. 対象表（T1）の行 A の C2 列に，ファイル 1 の BLOB データを挿入します。
2. 行 A の C2 列に対して，ファイル 2 の BLOB データを連結することで追加更新されます。これ以降にデータを追加する場合も同様です。

(2) BLOB データの部分抽出

「BLOB データの追加更新」で格納した行 A の BLOB データ（C2 列）から，ファイル 2 の部分を抽出します。

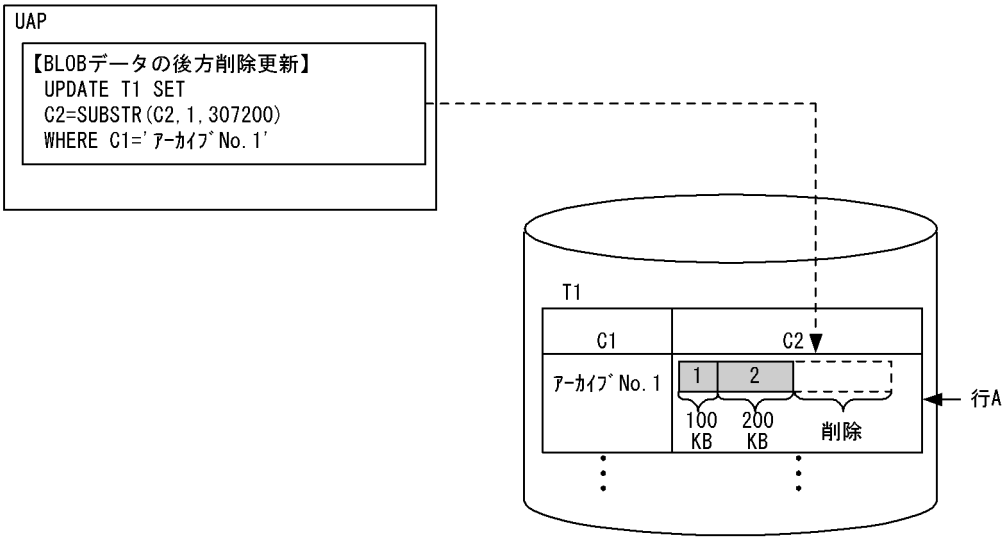


[説明]

スカラ関数 SUBSTR を使用して，ファイル 2 のデータ列の開始位置（100×1024 ＋ 1 ＝ 102401 バイト目）から，ファイル 2 のデータ列の長さ分（200×1024 ＝ 204800 バイト）だけ抽出します。

(3) BLOB データの後方削除更新

「BLOB データの追加更新」で格納した行 A の BLOB データ（C2 列）の後方部分を削除し，ファイル 1 とファイル 2 だけを残します。



[説明]

スカラ関数 SUBSTR を使用して，ファイル 1 のデータ列の開始位置 1 バイト目から，ファイル 1 とファイル 2 の長さ分（100×1024 ＋ 200×1024 ＝ 307200 バイト）だけを抽出したデータに置き換えて更新します。これによって，ファイル 1 とファイル 2 だけが残り，後方部分のデータは削除されます。

4.13.3 BLOB データ，BINARY データの部分的な更新・検索を行う場合の留意点

BLOB データ，BINARY データの部分的な更新・検索を行う場合は，次の点に留意してください。

1. BLOB データ又は BINARY データの連結演算を指定できるのは，UPDATE 文の SET 句の更新値だけです。また，連結演算の第 1 演算項に指定できるのは列指定だけで，第 2 演算項に指定できるのは埋込み変数，？パラメタ，SQL 変数，及び SQL パラメタです。

   連結演算を使用して BLOB 型又は BINARY 型の列を更新する場合の規則については，マニュアル「HiRDB SQL リファレンス」を参照してください。

2. 追加更新をする場合，ユニークなキーを格納する列を作成し，探索条件に指定して更新行を特定してください。また，行の特定を高速にするため，インデクスをその列に作成してください。

3. BLOB データの最小入出力単位は RD エリアのページ長になり，128 キロバイトまでは一括して入出力処理をします。したがって，BLOB データの挿入，追加更新，部分抽出処理の性能を良くするために，データの長さを 128×1024×n バイト（n は 0 以上の整数）単位にすることを推奨します。

4. システム共通定義の pd_rpl_func_control オペランドに BACKWARD_CUTOFF_UPDATE を指定していない場合は，後方削除更新が無効になります。この場合，SUBSTR に指定されたデータをメモリ上に抽出してから更新を行います。

4.14　先頭から n 行の検索結果を取得する機能

4.14.1　概要

先頭から n 行だけの検索結果を取得すると，SQL の検索性能を向上できることがあります。検索結果の行数が少ないほど，性能の向上も期待できます。

先頭から n 行の検索結果を取得する機能を使用すると，SQL の検索結果のうち，先頭（又は，指定した先頭からのオフセット行数分スキップした箇所）から n 行だけを受け取ります。この場合，SQL の最適化が選択するアクセスパスが変わります。その結果，次のように SQL の検索性能が向上することがあります。

- 探索条件を満たしたすべての行を対象とするソート処理が不要となるため，ソート処理の対象行が少なくなる場合があります。
- ORDER BY だけのために，HiRDB が作成していた作業表が不要となる場合があります。
- 各サーバプロセスで，検索結果の先頭 n 行に入らない行を読み込まないことで，サーバプロセス間の通信量が削減できる場合があります。

先頭から n 行の検索結果を取得する機能を使用する場合，LIMIT を指定します。LIMIT については，マニュアル「HiRDB SQL リファレンス」を参照してください。

4.14.2　留意事項

次の場合は，先頭から n 行の検索結果を取得する機能を使用しても検索性能が向上しない，又は逆に検索性能が劣化する可能性があります。

1. オフセット行数＋リミット行数の値が，LIMIT 句を指定しない場合と同じか，又は非常に近い場合
2. ORDER BY 句を指定しないで LIMIT 句を指定した場合，どの行が検索結果になるかは一意に決まりません。LIMIT 句を指定した場合は，ORDER BY 句も同時に指定することをお勧めします。ただし，ORDER BY 句を指定することで，SQL の最適化が異なるアクセスパスを選択し，検索処理の性能が劣化することがあります。SQL の最適化が選択したアクセスパスは，アクセスパス表示ユティリティ（pdvwopt）で確認ができます。
3. ORDER BY 句と LIMIT 句を指定した場合，オフセット行数を基に読み飛ばした最終行，又はリミット行数を基に取得した最終行と，同じ値のソートキーを持つ行が複数あると，ソートキーが同じ値である行のうち，どの行が検索結果になるかは一意に決まりません。この条件に該当する行と同じ値のソートキーを持つ特定の行を検索結果とする場合，ソートキーの構成列を増やすことで，特定の行を検索結果にできます。ただし，ソートキーの構成列を増やすことで，SQL の最適化が異なるアクセスパスを選択し，検索処理の性能が劣化することがあります。SQL の最適化が選択したアクセスパスは，アクセスパス表示ユティリティ（pdvwopt）で確認ができます。

上記のような場合には，先頭から n 行の検索結果を取得する機能は使用しないでください。

また，リミット行数が 1 以上で，かつオフセット行数＋リミット行数の値が 32,767 以下の場合，作業表を作成しない代わりに，オフセット行数＋リミット行数以内に入る行をメモリに保持します。このため，先頭から n 行の検索結果を取得する機能を使用しない場合に比べて，メモリ所要量が増加します。メモリ所要量については，マニュアル「HiRDB システム導入・設計ガイド」の「先頭から n 行の検索結果を取得する機能実行時に必要なメモリ所要量の求め方」を参照してください。

4.14.3　アクセスパスの確認方法

検索処理を高速化するため，先頭から n 行の検索結果を取得する機能を使用する場合としない場合とで，SQL の最適化が ORDER BY 処理方式の異なるアクセスパスを選択することがあります。ORDER BY 処理方式については，マニュアル「HiRDB コマンドリファレンス」の pdvwopt を参照してください。

4.15　自動再接続機能

自動再接続機能とは，サーバプロセスダウン，系切り替え，ネットワーク障害などの要因による HiRDB サーバとの接続障害※を検知した場合に，自動的に再接続する機能です。自動再接続機能を使用すると，ユーザは HiRDB サーバとの接続の切断を意識しないで，UAP の実行を継続できます。

自動再接続機能を使用する場合は，クライアント環境定義 PDAUTORECONNECT に YES を指定します。

注※

接続障害は SQL 実行時の SQL エラーが次の場合を示します。

- KFPA11722-E(SQLCODE：-722)
- KFPA11723-E(SQLCODE：-723)
- KFPA11728-E(SQLCODE：-728)
- KFPA11732-E(SQLCODE：-732)

4.15.1　適用基準

HiRDB サーバで次の処理を実行している場合，HiRDB クライアントはその処理が終わるまで待ち状態となります。

- システム定義の変更（pdchgconf コマンド）を実行している場合
- 修正版 HiRDB の入れ替え（pdprgcopy 及び pdprgrenew コマンド）を実行している場合
- トランザクションキューイング機能（pdtrnqing コマンド）を使用して，計画系切り替えを実行している場合

待ち状態となっている間，PDCWAITTIME の時間で待ち時間が監視されます。PDCWAITTIME の時間を超えた場合，待ち状態は解除されて UAP に PDCWAITTIME オーバーのエラーを返却します。

タイミングによっては，上記の処理が実行中であることを検知できないため，通信処理エラーになることがあります。あらかじめ上記の処理を実行することが分かっている場合は，自動再接続機能の適用を検討してください。自動再接続機能を使用すると，上記の処理を実行している場合でも，UAP にエラーを返却しないで処理を続行できます。

4.15.2　再接続する契機

再接続する契機を次に示します。

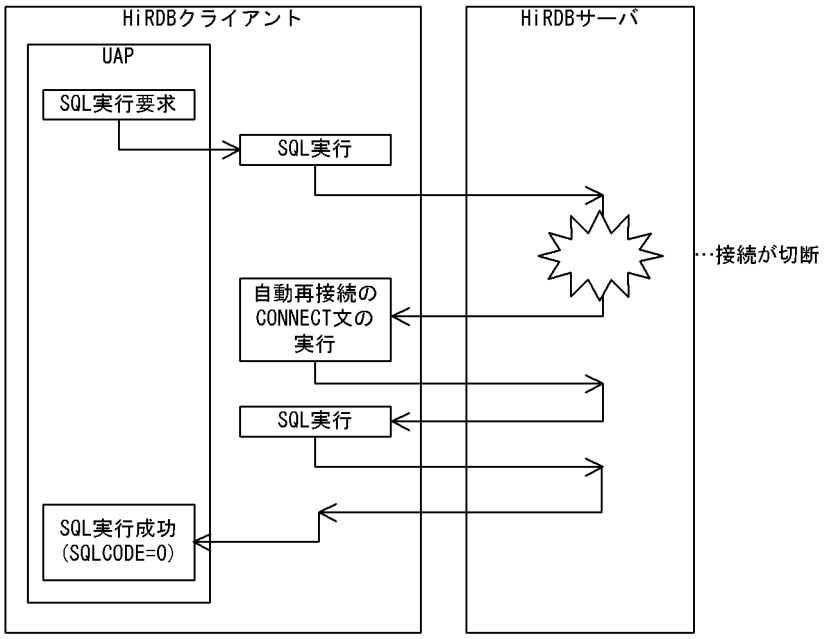
- CONNECT 文の実行直後，又は前回の SQL でトランザクション決着済みの場合に，SQL を実行したとき
- HiRDB サーバが前回の SQL のトランザクション処理中に，SQL を実行したとき

- CONNECT 文を実行したとき

(1) CONNECT 文の実行直後，又は前回の SQL でトランザクション決着済みの場合に，SQL を実行したとき

SQL を実行したときに，接続が切断されているのを検知します。検知した場合，再接続をして，再接続後に再度 SQL を実行します。自動再接続後の SQL 実行で接続の障害を検知した場合は，UAP へエラーを返却します。再接続する契機（CONNECT 文の実行直後，又は前回の SQL でトランザクション決着済みの場合に，SQL を実行したとき）を次の図に示します。

図 4–71　再接続する契機（CONNECT 文の実行直後，又は前回の SQL でトランザクション決着済みの場合に，SQL を実行したとき）

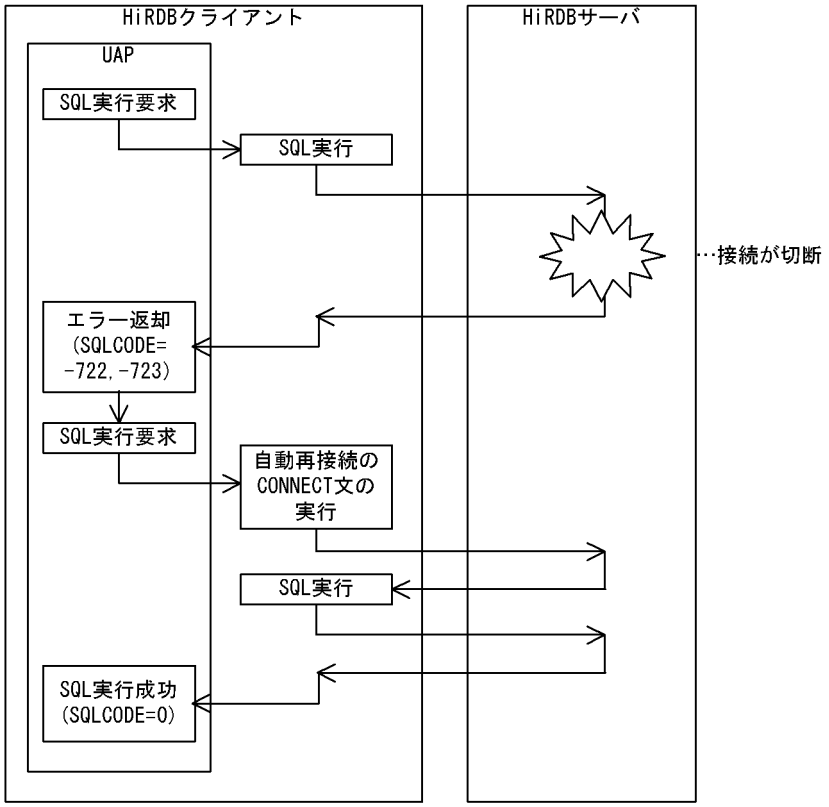


(2) HiRDB サーバが前回の SQL のトランザクション処理中に，SQL を実行したとき

SQL を実行したときに，接続が切断されているのを検知します。検知した場合，接続エラー（SQLCODE=-722，-723）を UAP に返却します。次回の SQL 実行時に再接続をして，再度 SQL を実行します。

自動再接続後の SQL 実行で接続の障害を検知した場合は，UAP へエラーを返却します。再接続する契機（HiRDB サーバが前回の SQL のトランザクション処理中に，SQL を実行したとき）を次の図に示します。なお，エラー返却された SQL までに実行していた未決着トランザクションはロールバックされます。

図 4–72　再接続する契機（HiRDB サーバが前回の SQL のトランザクション処理中に，SQL を実行したとき）

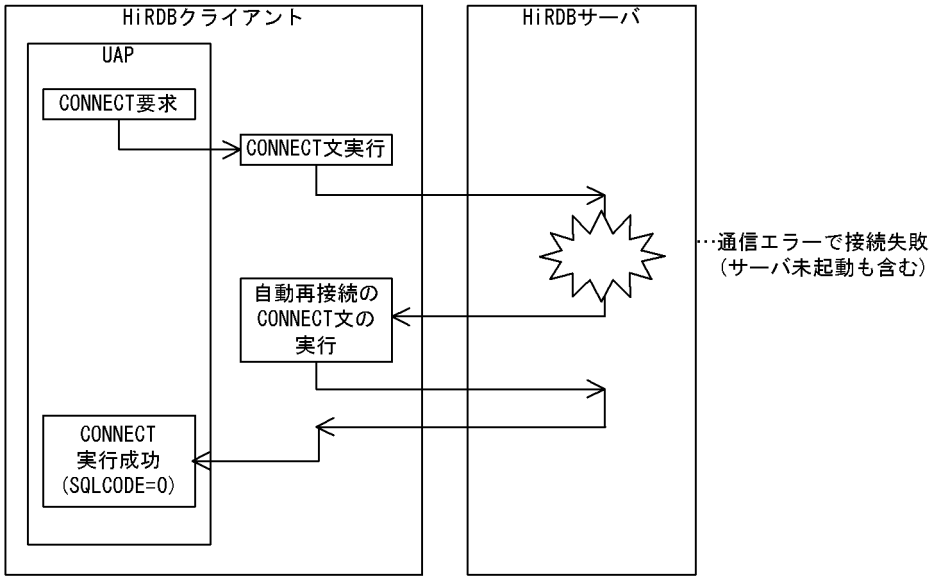


(3)　CONNECT 文を実行したとき

CONNECT 文実行時に，通信エラーなどで接続が失敗した場合，そのまま再接続をします。

再接続する契機（CONNECT 文を実行したとき）を次の図に示します。

図 4–73　再接続する契機（CONNECT 文を実行したとき）



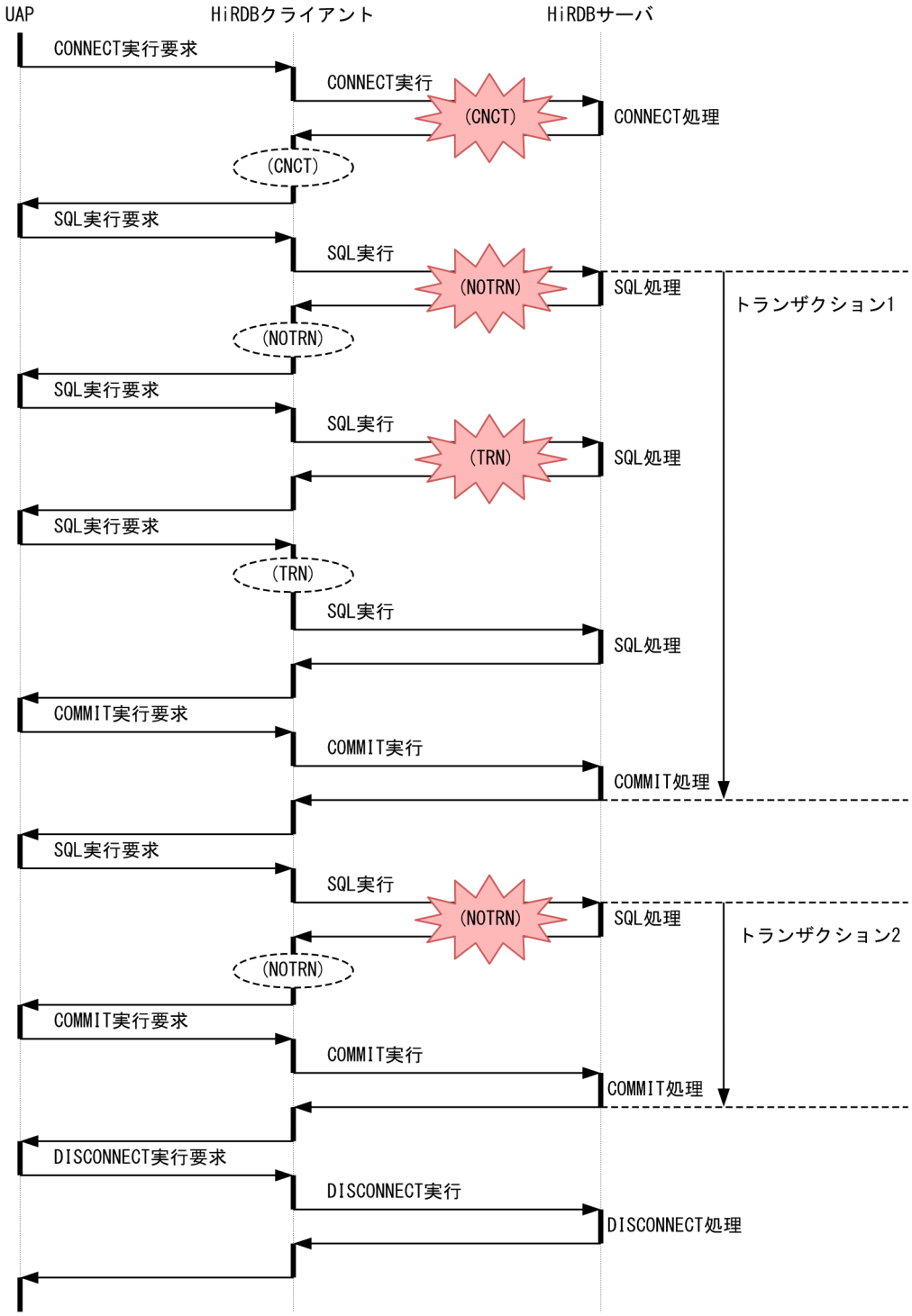
それぞれの再接続契機に自動再接続機能を適用するかどうかは，クライアント環境定義の PDRCTIMING で指定できます。再接続する契機と PDRCTIMING の指定値の関係を次に示します。

表 4–19　再接続する契機と PDRCTIMING の指定値の関係

再接続する契機	PDRCTIMING 指定値
CONNECT 文の実行直後，又は前回の SQL でトランザクション決着済みの場合に，SQL を実行したとき	NOTRN 又は ALL
HiRDB サーバが前回の SQL のトランザクション処理中に，SQL を実行したとき	TRN 又は ALL
CONNECT 文を実行したとき	CNCT 又は ALL

SQL 実行シーケンスでの自動再接続契機と PDRCTIMING 指定値の対応を次の図に示します。

図 4-74　SQL 実行シーケンスでの自動再接続契機と PDRCTIMING 指定値の対応



注　()内はPDRCTIMINGの指定値を示します。

（凡例）

：自動再接続契機となる接続障害

：自動再接続

4.15.3　自動再接続での CONNECT 処理

自動再接続では，内部的に 5 秒間隔で CONNECT 文を 5 回実行します。CONNECT 文の実行回数，及び実行間隔は，クライアント環境定義の PDRCCOUNT，及び PDRCINTERVAL で変更できます。

再接続契機が CONNECT 文の場合は PDCONNECTWAITTIME で監視されますが，CONNECT 文以外の場合は PDCWAITTIME で監視されます。自動再接続の処理時間が PDCONNECTWAITTIME 又は PDCWAITTIME の時間を超えた場合は，自動再接続の処理は打ち切られ，UAP にエラーを返却します。自動再接続機能を PDRCCOUNT の指定回数分リトライさせたい場合は，PDCWAITTIME の指定値を PDRCCOUNT×PDRCINTERVAL 以上に設定してください。

4.15.4　留意事項

1. UNTIL DISCONNECT 指定の LOCK 文を使用している UAP の場合，自動再接続機能は使用できません。
2. SQL セッション固有一時表（ON COMMIT PRESERVE ROWS 指定）を使用している UAP の場合，自動再接続機能は使用できません。
3. トランザクション処理中でない場合でも，ホールダブルカーソルを使用しているときは，UAP にいったんエラーを返却します。
4. JDBC ドライバ※1，又は DABroker for JAVA※2からのアクセスで，トランザクションをわたったステートメントが有効となっている場合，自動再接続機能で再接続後は，JDBC のステートメントが無効となります。この場合，再度 prepareStatement()メソッドの実行が必要となります。
5. トランザクションの最初の SQL で PDCWAITTIME の時間を超えた場合でも，UAP にエラーを返さないで CONNECT 文，及び SQL の再実行を行います。そのため，PDCWAITTIME の約 2 倍の時間でエラーを返すことがあります。
6. Java EE アプリケーションサーバ接続時はステートメントプーリング／ステートメントキャッシュ機能を有効にすると，HiRDB の自動再接続機能によってコネクションが再接続された後の SQL 実行で，KFPA11901-E メッセージを含む SQLException 例外が発生することがあります。ステートメントプーリング／ステートメントキャッシュ機能を使用する場合は，自動再接続機能を使用しないでください。なお，ステートメントプーリング／ステートメントキャッシュ機能については，マニュアル「HiRDB システム導入・設計ガイド」の「ステートメントプーリング／ステートメントキャッシュ」を参照してください。

注※1

JDBC ドライバでトランザクションをわたったステートメントが有効となるのは，COMMIT_BEHAVIOR に"CLOSE"，又は"PRESERVE"を設定している場合です。COMMIT_BEHAVIOR は，Driver クラスの connect メソッドの引数 Properties info，DriverManager.getConnection メソッドの引数 Properties info，又は URL 接続での COMMIT_BEHAVIOR キーで設定できます。

注※2

DABroker for JAVA でトランザクションをわたったステートメントが有効となるのは，DABroker 03-06 以降で，かつ DABroker for JAVA 02-10 以降の場合です。

4.16　位置付け子機能

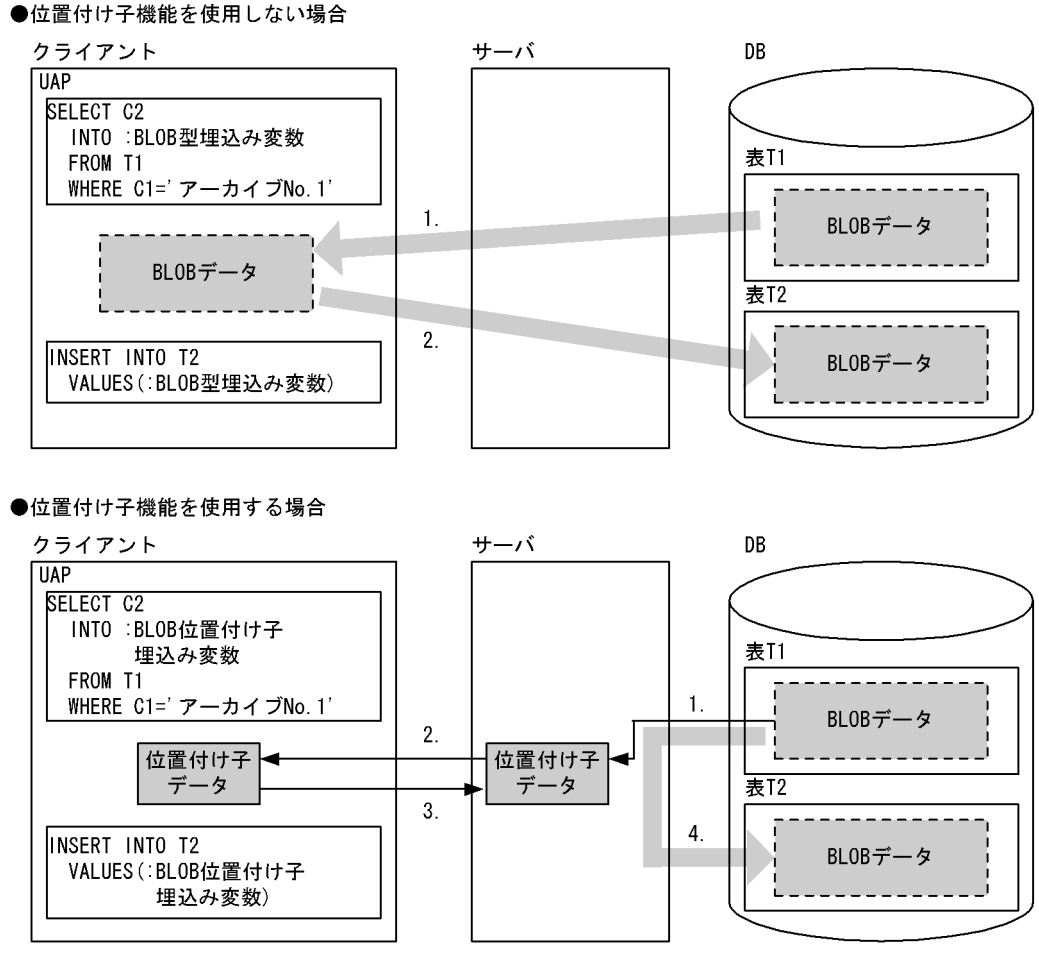
4.16.1　位置付け子機能とは

クライアントの UAP で，検索した BLOB データ又は BINARY データをそのデータ型の埋込み変数で受け取る場合，受け取ったデータを格納するためのメモリ領域をクライアント側で用意する必要があります。このため，長大なデータを検索する場合，クライアント側のメモリ資源を圧迫します。さらに，サーバからクライアントへのデータ転送量も大きくなります。しかし，必要なデータが一部だけであったり，受け取ったデータを変更しないでほかの SQL 文中に指定してサーバに送り返すだけであったりする場合，データをすべてクライアントに転送することはむだなことになります。

位置付け子機能は，これを解決するために使用する機能です。位置付け子は，サーバ上のデータを識別する 4 バイトの値のデータであり，1 行 SELECT 文や FETCH 文の INTO 句などに位置付け子の埋込み変数を指定することで，検索結果としてデータの実体ではなく，そのデータを識別する位置付け子の値を受け取ります。また，データを識別する位置付け子の埋込み変数をほかの SQL 文中に指定することで，位置付け子が識別するデータを扱う処理ができます。

位置付け子機能の概要を次の図に示します。

図 4–75　位置付け子機能の概要



［説明］

位置付け子機能を使用しない場合：

1. DB から検索した BLOB データを，サーバからクライアントに転送します。
2. クライアントからサーバに BLOB データを転送し，それを DB に格納します。

位置付け子機能を使用する場合：

1. サーバが，DB から検索したデータを識別する位置付け子データを作成します。
2. 位置付け子データを，サーバからクライアントに転送します。
3. クライアントからサーバに位置付け子データを転送します。
4. 位置付け子データが識別するサーバ上の BLOB データを DB に格納します。

4.16.2　適用基準

BLOB データ又は BINARY データを検索する場合，クライアント側のメモリ所要量を削減したいとき，及びサーバ，クライアント間のデータ転送量を少なくしたいときに適用してください。

位置付け子機能を使用すると，クライアント側で実際のデータの大きさ分だけメモリを確保する必要がなくなり，さらにサーバ，クライアント間のデータ転送を位置付け子で行うことができるため，データ転送量を少なくできます。

4.16.3　使用方法

位置付け子の値を受け取る場合，SQL 文中の BLOB 型又は BINARY 型のデータを受け取る埋込み変数を指定する箇所に，対応するデータ型の位置付け子の埋込み変数を指定します。また，位置付け子に割り当てられたデータを処理する場合は，SQL 文中に BLOB 型又は BINARY 型の埋込み変数を指定する代わりに，対応するデータ型の位置付け子の埋込み変数を指定します。

4.16.4　使用例

表 T1 の C1=1 である行の列 C2 データを，あるバイナリデータ列（search_data）から始まる 400 キロバイトの部分だけ，別のデータ（change_data）に置き換えます。それを C2 列とした新しい行（C1=2）として，表 T1 に挿入します。

表 T1 の各列のデータ型を次に示します。

- C1：INTEGER NOT NULL (INDEX)
- C2：BLOB (100M) NOT NULL

```
void abnormalend(void);

main()
{
  EXEC SQL BEGIN DECLARE SECTION;
    SQL TYPE IS BLOB AS LOCATOR alldata_loc; /* 全データを表す位置付け子 */
    long change_pos;                        /* 変更開始位置 */
    SQL TYPE IS BLOB(10) search_data;       /* 検索バイナリデータ列 */
    SQL TYPE IS BLOB(400K) change_data;     /* 変更バイナリデータ列 */
    SQL TYPE IS BLOB AS LOCATOR enddata_loc; /* 変更部分の後に続く     */
                                             /* データを表す位置付け子 */
    long pos;
  EXEC SQL END DECLARE SECTION;

         -------(HiRDBへのCONNECT処理(省略))     -------
         -------(検索バイナリデータ列の設定(省略))-------
         -------(変更バイナリデータ列の設定(省略))-------

  EXEC SQL WHENEVER SQLERROR PERFORM abnormalend;
  /* 列データを位置付け子で取得 */
  EXEC SQL SELECT C2 INTO :alldata_loc FROM T1 WHERE C1 = 1;
  /* 検索バイナリデータ列を含む開始位置を取得 */
  EXEC SQL SET :change_pos = POSITION(:search_data AS BLOB(10)
    IN :alldata_loc AS BLOB(100M));
  pos = change_pos + 409600;
  /* 変更部分の後に続くデータを位置付け子で取得 */
  EXEC SQL SET :enddata_loc = SUBSTR(:alldata_loc AS BLOB(100M), :pos);
  pos = change_pos -1;
  /* 変更部分より前のデータを位置付け子を用いてINSERT */
  EXEC SQL INSERT INTO T1 VALUES(2, SUBSTR
    (:alldata_loc AS BLOB(100M), 1, :pos));
  /* 全データを表す位置付け子は必要なくなったので無効化する */
  EXEC SQL FREE LOCATOR :alldata_loc;
  /* 変更部分のデータを連結してUPDATE */
  EXEC SQL UPDATE T1 SET C2 = C2 || :change_data WHERE C1 = 2;
  /* 変更部分の後に続くデータを位置付け子を用いて連結してUPDATE */
  EXEC SQL UPDATE T1 SET C2 = C2 || :enddata_loc WHERE C1 = 2;
  EXEC SQL COMMIT;
  printf(" *** normally ended ***¥n");
  EXEC SQL WHENEVER SQLERROR CONTINUE;
  EXEC SQL WHENEVER SQLWARNING CONTINUE;
  EXEC SQL DISCONNECT;
  return(0);
}
void abnormalend()
{
  int  wsqlcode;
  wsqlcode = -SQLCODE;printf("¥n*** HiRDB SQL ERROR SQLCODE
    = %d ¥n", wsqlcode);
  printf("SQLERRMC = %s¥n", SQLERRMC);
  EXEC SQL ROLLBACK;
  EXEC SQL DISCONNECT;
  exit(1);
}
```

4.16.5　留意事項

1. 位置付け子に対してサーバ上のデータを割り当てる場合，サーバ上に，位置付け子に割り当てたデータを保持しておくメモリが必要になることがあります。このため，1 トランザクションで多数のデータを位置付け子に割り当て，有効にしたままにすると，サーバのメモリを圧迫します。したがって，必要のなくなった位置付け子は，FREE LOCATOR 文で無効にしてください。

4.17　総ヒット件数返却機能

4.17.1　機能概要

通常，総ヒット件数とヒットした行の値を求める場合，二つの SQL 文を実行しますが，総ヒット件数返却機能を使用すると，総ヒット件数を求める SQL と，ヒットした行の値を求める SQL とを，一つの SQL に統合できます。これによって，二つの SQL 文を実行するための検索時間が，一つの SQL 文を実行する検索時間とほぼ等しくなります。

総ヒット件数返却機能は，ウィンドウ関数 COUNT(*) OVER()を選択式中に指定することで使用できます。ウィンドウ関数については，マニュアル「HiRDB SQL リファレンス」を参照してください。

4.17.2　使用例

在庫表（ZAIKO）から単価（TANKA）が 5000 円以上の商品の総数と，その商品名（SNAME）を求め，数量（ZSURYO）でソートする場合の例を次に示します。

- 総ヒット件数返却機能を使用しない場合

```
SELECT COUNT(*) FROM ZAIKO WHERE TANKA>=5000
SELECT SNAME    FROM ZAIKO WHERE TANKA>=5000 ORDER BY ZSURYO
```

[説明]

総ヒット件数返却機能を使用しない場合は，二つの SQL 文が必要となります。

- 総ヒット件数返却機能を使用する場合

```
SELECT COUNT(*) OVER(), SNAME
                FROM ZAIKO WHERE TANKA>=5000 ORDER BY ZSURYO
```

[説明]

二つの SQL 文の下線部が同じなので，総ヒット件数返却機能を使用することで一つの SQL 文に統合し，初回取り出し時に総ヒット件数を取得できます。

4.17.3　留意事項

次に示すケースでは，総ヒット件数返却機能を使用しても，検索性能の向上が期待できない，又は検索性能が低下するおそれがあります。この場合は，総ヒット件数返却機能を使用しないようにしてください。

- DISTINCT，ORDER BY 句，及び FOR READ ONLY 句のどれも指定しない場合。
- ORDER BY 句を指定し，ORDER BY のためのソートがキャンセルできるアクセスパスを選択している場合。

ORDER BY のためのソートがキャンセルできるかどうかは，アクセスパス表示ユティリティ（pdvwopt）で確認できます。アクセスパス表示ユティリティについては，マニュアル「HiRDB コマンドリファレンス」を参照してください。

- 射影長が短くて，COUNT(*) OVER()の列長 4 バイト分の通信量の増加が無視できない場合。
- 検索処理コストが小さい場合。

4.18　RD エリア名を指定した検索，更新，又は削除

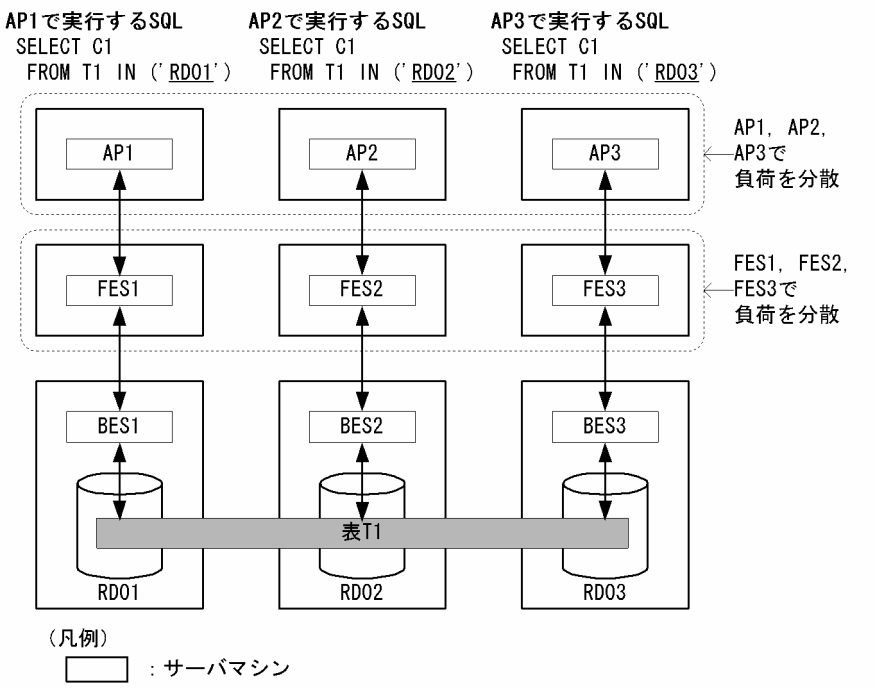
4.18.1　機能概要

マルチフロントエンドサーバで表の横分割を行っている場合，RD エリア名を指定して検索，更新，又は削除を行うと，アクセスする RD エリアを限定することができます。これによって，複数の RD エリアに対して並列にアクセスできるようになり，サーバマシンに対する負荷を分散させることができます。

4.18.2　使用例

RD エリア名を指定した検索をする場合の例を次の図に示します。

図 4–76　RD エリア名を指定した検索をする場合の例



[説明]

RD エリア RD01，RD02，RD03 に格納している横分割表 T1 に対して，検索を行います。このとき，AP1，AP2，AP3 という三つの UAP から，それぞれアクセスする RD エリアを指定した SELECT 文を発行します。これによって，RD エリアへのアクセスを，フロントエンドサーバ FES1，FES2，FES3 経由で並列に実行することができ，UAP 又はフロントエンドサーバがあるサーバマシンの負荷を分散させることができます。

4.18.3　留意事項

- 次に示す場合は，この機能が適用されません。
  - CREATE TABLE に WITHOUT ROLLBACK オプションを指定した表を検索，更新，又は削除する場合
  - ログレスモードで UAP を実行している場合
  - pdlbuffer オペランドの-i オプションでインデクス用のローカルバッファを割り当てている場合
- この機能が適用された場合，分割数が表の分割数と等しいインデクスだけが利用可能になります。RD エリア名を指定した検索，更新，又は削除しか行わない場合，分割数が表の分割数と異なるインデクスを定義していると，インデクスが利用されません。RD エリア名を指定した検索，更新，又は削除を行う場合は，分割数が表の分割数と等しいインデクスを定義してください。

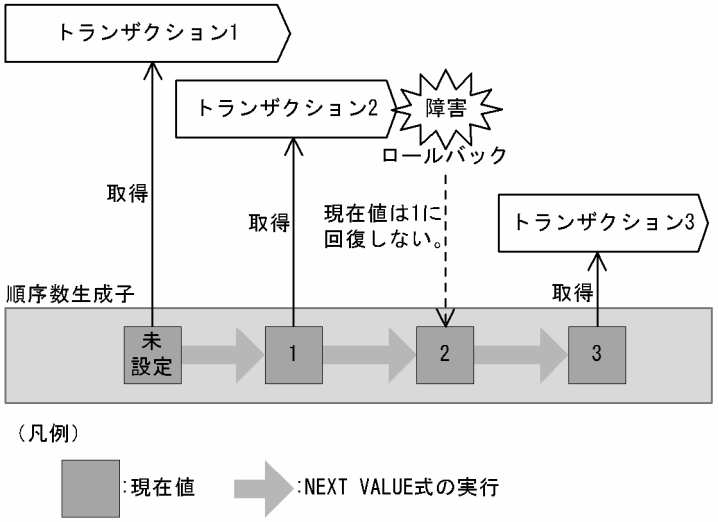
4.19　自動採番機能

自動採番機能とは，データベース中でデータを呼び出すごとに一連の整数値を返す機能です。この機能は，順序数生成子を定義することで使用できます。自動採番機能を使用すると，採番を行う UAP の開発効率が向上します。また，順序数生成子をサポートしている他 DBMS で作成した UAP からの移行性も向上します。このため，採番業務では自動採番機能を使用することを推奨します。

4.19.1　順序数生成子とは

順序数生成子は，ユーザやトランザクションの状態に関係なく，連続した番号（順序番号）を一度に一つ生成します。順序数生成子の概要を次の図に示します。

図 4–77　順序数生成子の概要



[説明]

順序数生成子が生成する順序番号を取得するためには，NEXT VALUE 式を使用します。NEXT VALUE 式は，順序数生成子が生成した最新の値（現在値）の次の値を取得し，現在値を次の値に更新します。

順序数生成子が定義されてから一度も NEXT VALUE 式を使用していない場合，現在値は未設定となります。現在値が未設定の状態で NEXT VALUE 式を使用すると，順序数生成子の開始値が返され，現在値には順序数生成子の開始値が格納されます。

注意事項

現在値はロールバックが発生しても回復されません。トランザクションの状態に関係なく，連続した番号を生成します。

4.19.2　順序数生成子の定義

順序数生成子の定義には，CREATE SEQUENCE を使用します。CREATE SEQUENCE については，マニュアル「HiRDB SQL リファレンス」を参照してください。

順序数生成子の定義例を次に示します。

```
CREATE SEQUENCE SEQ1.....1
  AS INTEGER.....2
  START WITH 10.....3
  INCREMENT BY 10.....4
  MAXVALUE 999.....5
  MINVALUE 10.....6
  CYCLE.....7
  LOG INTERVAL 3.....8
IN RD01.....9
```

[説明]

1. 2〜9 の条件で順序数生成子 SEQ1 を定義します。
2. データ型
3. 開始値
4. 増分値
5. 最大値
6. 最小値
7. 循環指定
   増分値 10，最小値 10，最大値 999 で，最大値の次の値が最小値となるように値を循環させます。
8. ログ出力間隔
9. 格納先 RD エリア名

注意事項

循環指定をした場合，順序数生成子が循環すると順序番号に重複が発生します。

(1) 順序数生成子格納 RD エリアの指定

順序数生成子の定義時，順序数生成子の格納先として，次に示す条件を満たした RD エリアを指定できます。

- 定義されている表と順序数生成子の合計が 500 未満の RD エリア
- 閉塞していない RD エリア
- インナレプリカ機能を使用していない RD エリア

注意事項

- HiRDB/パラレルサーバの場合，順序数生成子と順序数生成子を使用する表（サーバ間分割していない表）が別々のサーバに格納されているときは，順序数生成子を使用するたびに通信が発生して，処理性能が低下します。サーバ間分割している表の場合，順序数生成子と順序数生成子を使用する表を同じサーバの RD エリアに格納することで，通信回数を軽減できることがあります。そのため，順序数生成子と順序数生成子を使用する表は同じサーバの RD エリアに格納することを推奨します。
- 順序数生成子格納 RD エリアに対してインナレプリカ機能は使用できません。

(2)　ログ出力間隔の指定

順序数生成子の定義時，ログの出力間隔を指定することで，処理性能が向上します。

順序数生成子ログは，次の契機で出力されます。

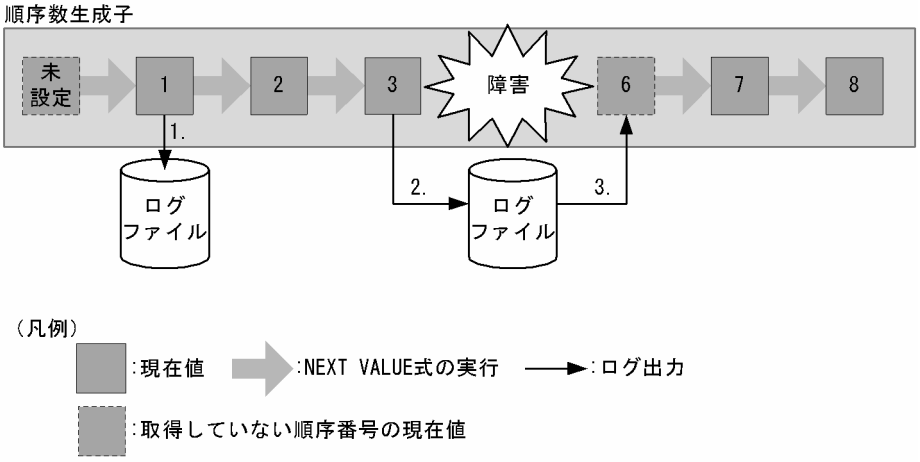
- 順序数生成子を定義してから最初に NEXT VALUE 式を使用したとき
- 順序番号の取得回数がログ出力間隔に達したとき

ログ出力間隔を指定した例を次の図に示します。

順序数生成子の定義

```
CREATE SEQUENCE SEQ1
  AS INTEGER
  START WITH 1
  INCREMENT BY 1
  MAXVALUE 999
  MINVALUE 1
  CYCLE
  LOG INTERVAL 3
IN RDO1
```

順序数生成子



（凡例）

［説明］
1. 順序数生成子を定義してから最初に順序番号を取得したとき，ログを出力します。
2. 順序番号の取得回数がログ出力間隔（3回）に達したとき，ログを出力します。
3. HiRDBにシステム障害が発生したとき，再開時のログ出力値（6）を現在値として
読み込みます。このとき，障害発生前のログ出力から障害発生までの順序番号（4，5，6）
は失われます。

注意事項

- 順序数生成子のログ出力間隔を小さく設定すると，システム障害発生時の欠番は少なくなりますが，ログの出力回数が多くなるので性能が低下します。ログ出力間隔を大きく設定すると，システム障害発生時の欠番が多くなりますが，ログの出力回数が少なくなるので性能は向上します。
- 欠番が発生した場合，最大でログ出力間隔に指定した値の分だけ欠番が発生します。

4.19.3　順序数生成子の削除

順序数生成子の削除には，DROP SEQUENCE を使用します。DROP SEQUENCE については，マニュアル「HiRDB SQL リファレンス」を参照してください。

4.19.4　順序数生成子が生成する順序番号の取得

順序数生成子が生成する順序番号の取得方法には次の 2 種類があります。

- 自動採番機能を使用したデータロード
- NEXT VALUE 式

自動採番機能を使用したデータロードについては，マニュアル「HiRDB コマンドリファレンス」を参照してください。NEXT VALUE 式については，マニュアル「HiRDB SQL リファレンス」を参照してください。

4.19.5　使用例

NEXT VALUE 式は，INSERT 文の問合せ式の選択式，INSERT 文の挿入値，又は UPDATE 文の更新値に指定できます。同一の行に対して，同じ順序数生成子を指定した NEXT VALUE 式を複数指定した場合，それらの NEXT VALUE 式はすべて同じ値を返します。

NEXT VALUE 式の使用例を次に示します。なお，順序数生成子は，「順序数生成子の定義」で定義した SEQ1 を使用します。

例

順序数生成子 SEQ1 の順序番号を一度も取得していない状態で NEXT VALUE 式を実行すると，順序数生成子の開始値（10）が返されます。

●実行SQL

```
INSERT INTO T1 VALUES(
  NEXT VALUE FOR SEQ1,
  104,
  204)
```

●実行結果



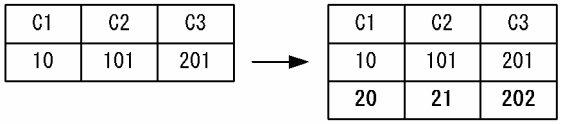
例

同一行に同じ順序数生成子を指定した NEXT VALUE 式を二つ以上指定すると，すべての NEXT VALUE 式は同じ値（20）が返されます。

●実行SQL

```
INSERT INTO T1 VALUES(
  NEXT VALUE FOR SEQ1,
  NEXT VALUE FOR SEQ1 + 1,
  202)
```

●実行結果



C1	C2	C3
10	101	201

→

C1	C2	C3
10	101	201
20	**21**	**202**

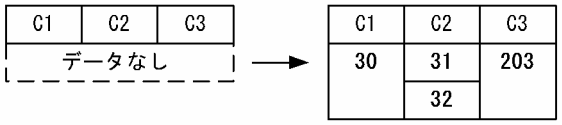
例

繰返し列を含む表に対しても，同一行に同じ順序数生成子を指定した NEXT VALUE 式を二つ以上指定すると，すべての NEXT VALUE 式は同じ値（30）が返されます。

●実行SQL

```
INSERT INTO T2 VALUES(
 NEXT VALUE FOR SEQ1,
  ARRAY [
   NEXT VALUE FOR SEQ1 + 1,
   NEXT VALUE FOR SEQ1 + 2
  ],
 203)
```

●実行結果



C1	C2	C3
データなし		

→

C1	C2	C3
30	**31**	**203**
	32	

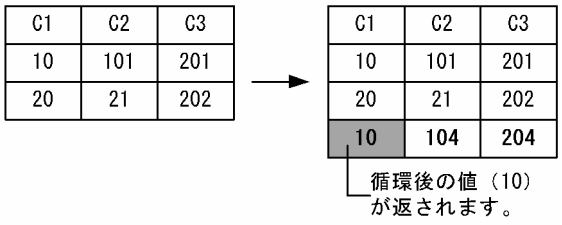
例

順序数生成子 SEQ1 の現在値が最大値（990）のときに NEXT VALUE 式を指定すると，循環後の値（10）が返されます。

●実行SQL

```
INSERT INTO T1 VALUES(
  NEXT VALUE FOR SEQ1,
  104,
  204)
```

●実行結果



C1	C2	C3
10	101	201
20	21	202

→

C1	C2	C3
10	101	201
20	21	202
10	**104**	**204**

循環後の値（10）が返されます。

4.19.6 留意事項

順序数生成子格納 RD エリアと，順序数生成子を使用する表を格納している RD エリアのバックアップ取得順序によっては，データベースの回復後，順序番号に重複や欠番が発生するおそれがあります。

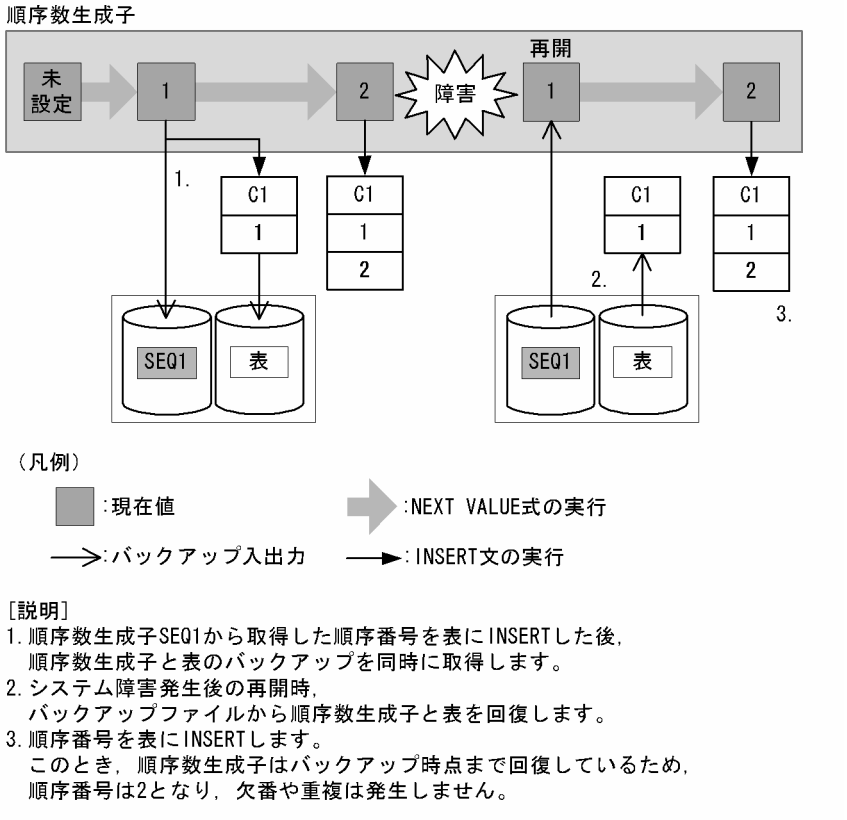
(1)　順序番号に重複や欠番が発生しないケース

順序数生成子格納 RD エリアと，順序数生成子を使用する表を格納している RD エリアのバックアップを同時に取得した場合，順序番号に重複や欠番が発生しません。

図 4–78　順序数生成子格納 RD エリアと，順序数生成子を使用する表を格納している RD エリアのバックアップを同時に取得した場合

順序数生成子の定義

```
CREATE SEQUENCE SEQ1
  START WITH   1
  INCREMENT BY 1
  MAXVALUE     999
  MINVALUE     1
  LOG INTERVAL 1
  CYCLE
```



[説明]
1. 順序数生成子SEQ1から取得した順序番号を表にINSERTした後，
   順序数生成子と表のバックアップを同時に取得します。
2. システム障害発生後の再開時，
   バックアップファイルから順序数生成子と表を回復します。
3. 順序番号を表にINSERTします。
   このとき，順序数生成子はバックアップ時点まで回復しているため，
   順序番号は2となり，欠番や重複は発生しません。

(2)　順序番号に重複や欠番が発生するおそれのあるケース

(a)　バックアップ取得順序が，順序数生成子→順序数生成子を使用する表の場合

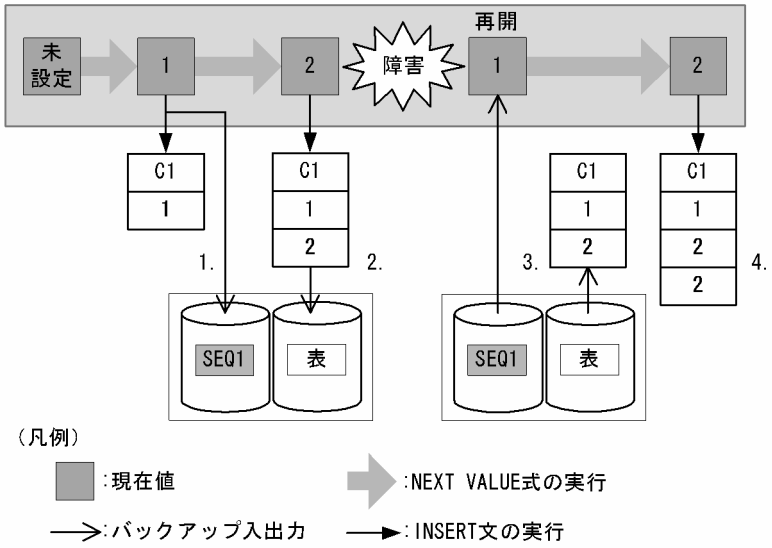
順序番号に重複のおそれがあります。詳細を次の図に示します。

図 4-79　バックアップ取得順序が，順序数生成子→順序数生成子を使用する表の場合

順序数生成子の定義

```
CREATE SEQUENCE SEQ1
  START WITH   1
  INCREMENT BY 1
  MAXVALUE     999
  MINVALUE     1
  LOG INTERVAL 1
  CYCLE
```

順序数生成子



（凡例）

:現在値　:NEXT VALUE式の実行

:バックアップ入出力　:INSERT文の実行

［説明］

1. 順序数生成子SEQ1から取得した順序番号（1）を表にINSERTした後，
   順序数生成子のバックアップを取得します。
2. 順序番号（2）を表にINSERTした後，表のバックアップを取得します。
3. システム障害発生後の再開時，
   バックアップファイルから順序数生成子と表を回復します。
4. 順序番号（2）を表にINSERTします。
   順序数生成子はバックアップ時点まで回復しているため，順序番号は2となります。
   このため，順序数生成子のバックアップを取得してから表のバックアップを取得する
   間に順序数生成子から取得した順序番号（2）が重複します。

(b)　レプリケーション機能を使用した場合

順序番号に重複のおそれがあります。

レプリケーション機能（HiRDB Dataextractor 及び HiRDB Datareplicator）では，順序数生成子を使用できません。抽出側と反映側を切り替えて，同じ名前の順序数生成子を使用して運用する場合，反映側に抽出側の順序数生成子の現在値を引き継げないため，順序番号が重複するおそれがあります。

(c)　バックアップ取得順序が，順序数生成子を使用する表→順序数生成子の場合

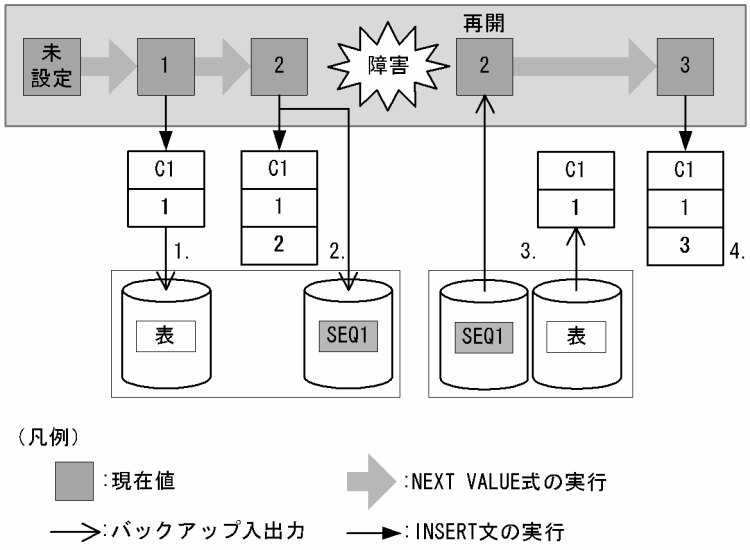
順序番号に欠番のおそれがあります。詳細を次の図に示します。

図 4-80　バックアップ取得順序が，順序数生成子を使用する表→順序数生成子の場合

順序数生成子の定義

```
CREATE SEQUENCE SEQ1
  START WITH   1
  INCREMENT BY 1
  MAXVALUE     999
  MINVALUE     1
  LOG INTERVAL 1
  CYCLE
```

順序数生成子



（凡例）

:現在値　:NEXT VALUE式の実行

:バックアップ入出力　:INSERT文の実行

［説明］

1. 順序数生成子SEQ1から取得した番号（1）を表にINSERTした後，
   表のバックアップを取得します。
2. 順序番号（2）を表にINSERTした後，順序数生成子のバックアップを取得します。
3. システム障害発生後の再開時，
   バックアップファイルから順序数生成子と表を回復します。
4. 順序番号（3）を表にINSERTします。
   順序数生成子はバックアップ時点まで回復しているため，順序番号は3となります。
   このため，表のバックアップを取得してから順序数生成子のバックアップを取得する
   間に順序数生成子から取得した番号（2）が欠番となります。

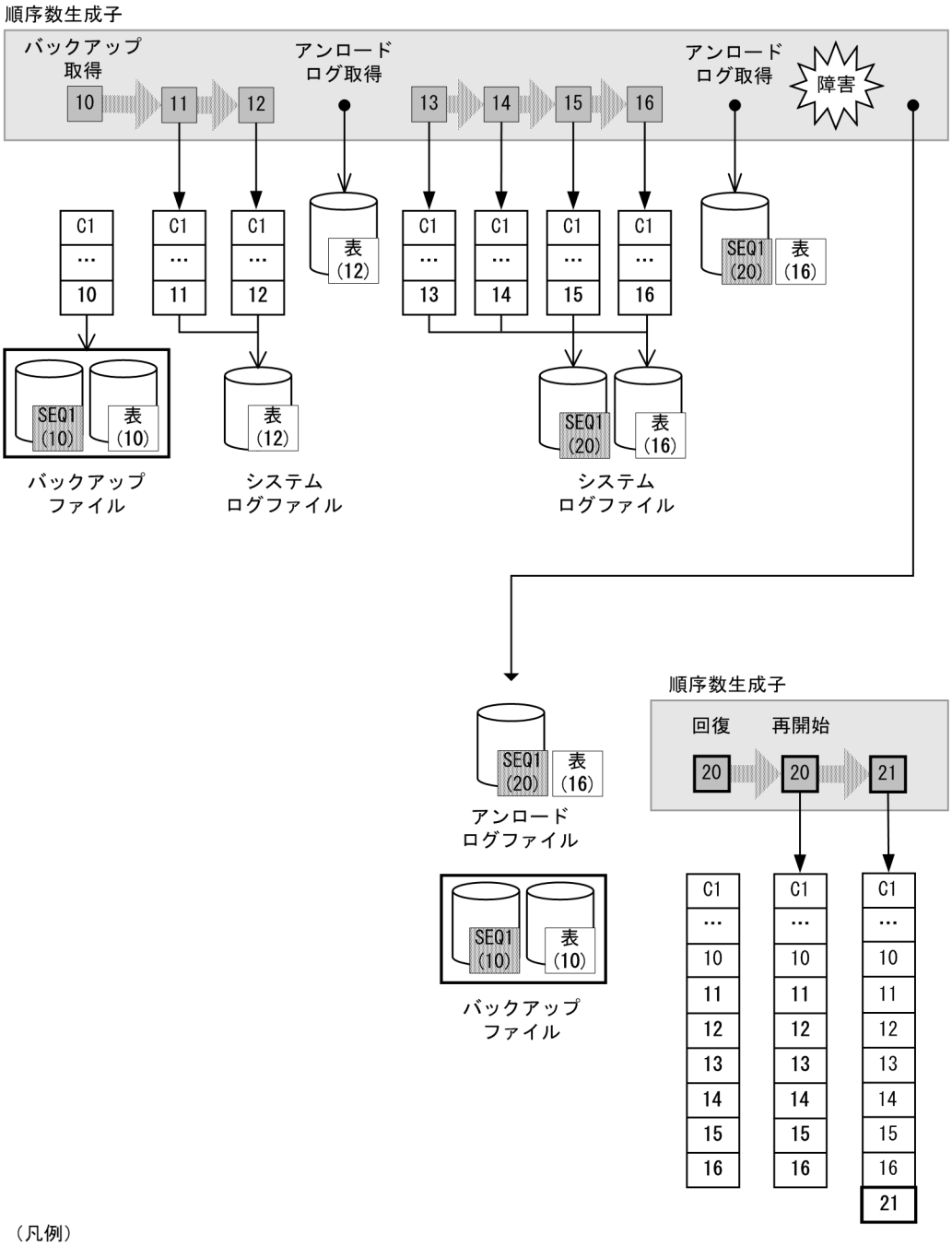
(d)　ログを使用した回復を行う場合

バックアップ取得時点以降の任意の同期点，又は障害発生直前の最新の同期点に回復する場合，バックアップファイルとシステムログファイル，又はアンロードログファイルをバックアップとしてデータベースを回復してください。このとき，順序数生成子を定義するときに指定したログ出力間隔の値の分だけ，順序番号に欠番のおそれがあります。詳細を次の図に示します。

図 4-81　ログを使用した回復を行う場合

順序数生成子の定義

```
CREATE SEQUENCE SEQ1
  START WITH   1
  INCREMENT BY 1
  MAXVALUE     999
  MINVALUE     1
  LOG INTERVAL 5
  CYCLE
```



（凡例）

:NEXT VALUE式の実行　:INSERT文の実行　:バックアップ，ログ入出力

［説明］

取得したアンロードログファイルを使用して，順序数生成子 SEQ1 と表を回復した場合，回復後に NEXT VALUE 式を実行したとき，順序数生成子 SEQ1 から取得した値は 21 となり，バックアップ取得時点から障害発生までの間に取得されていない順序数生成子 SEQ1 の値（17～20）は欠番となります。

4.20　プラグインインデクスでの他インデクス絞り込み結果の利用

4.20.1　概要

プラグインインデクスを使用した条件の絞り込み時に，ほかのインデクスを使用した条件による絞り込み結果の行識別子集合を利用する機能です。この機能の適用条件は，プラグインインデクスを使用した検索で，検索方式として AND の複数インデクス利用が適用できる場合です。この機能の適用によって，ほかの条件による絞り込み率がよい場合には，その絞り込み結果を利用してプラグインインデクスのサーチ範囲をさらに絞り込むので，高速に検索できます。ただし，ほかの条件による絞り込み率が悪い場合は，その絞り込み結果をプラグインインデクスのサーチに利用しません。

4.20.2　使用方法

次の表を参照して検索高速化インデクス（V2）を定義してください。

表 4–20　プラグインインデクスのデータ型に対応するマニュアル

プラグインインインデクスを定義するデータ型	対応するマニュアル
XML 型	「HiRDB XML Extension」の「XML データ検索プラグインのデータベースの作成」,「インデクスの定義」
SGML 型	「HiRDB Text Search Plug-in」の「文書検索プラグインのデータベースの作成」,「インデクスの定義」
FREEWORD 型	「HiRDB Text Search Plug-in」の「文字列検索プラグインのデータベースの作成」,「インデクスの定義」

検索高速化インデクス（V2）を使用する条件と，ほかのインデクスを使用した条件との AND 演算を指定してください。ほかの条件による絞り込み結果の行識別子集合を利用できるプラグインインデクスの条件の場合は，アクセスパスのサーチ条件種別が IS TRUE（AND）になります。詳細はマニュアル「HiRDB コマンドリファレンス」の「アクセスパス表示ユティリティ（pdvwopt）」の「サーチ条件」を参照してください。

4.20.3　留意事項

1. XMLEXISTS 述語の条件と，次に示す条件との AND 演算を指定する場合，XMLEXISTS 述語に対してインデクスを利用しないため，検索方式が AND の複数インデクス利用にはなりません。そのため，XMLEXISTS 述語に対してこの機能は適用されません。
   - CONTAINS 関数
   - CONTAINS_WITH_SCORE 関数

- CONCEPT_WITH_SCORE 関数
- CONCEPT_WITH_TERMS 関数
- 構造化繰返し述語

2. プラグインインデクスで他インデクス絞り込み結果を利用しないアクセスパスにする場合，SQL 拡張最適化オプションに，DETER_IS_TRUE_AND（2048）を指定してください。SQL 拡張最適化オプションの指定方法は，「クライアント環境定義の設定内容」，及びマニュアル「HiRDB システム定義」の「SQL の最適化に関するオペランド」を参照してください。

4.20.4 使用例

使用例を示します。

前提とする表定義及びインデクス定義

```
CREATE TABLE TBL1(ID INTEGER, REGDATE DATE, CHI XML);
CREATE INDEX IID ON TBL1(ID);
CREATE INDEX ICHI USING TYPE IXXML ON TBL1(CHI) IN ((RUL01)) PLUGIN 'INDEX_V2';
```

<プラグインインデクスで他インデクス絞り込み結果を利用する例>

例 1：

プラグインインデクスを利用できる条件と B-Tree インデクスを利用できる条件を AND 演算すると，プラグインインデクスで他インデクス絞り込み結果を利用できます。

```
SELECT ID FROM TBL1 WITH INDEX (IID, ICHI)
WHERE ID >= 100 AND CONTAINS(CHI,'root[elm{"1"}]') IS TRUE;
 （アンダーラインのサーチ条件種別: SearchCnd : IS TRUE (AND))
```

例 2：

プラグインインデクスを利用できる条件とプラグインインデクスを利用できる条件を AND 演算すると，2 番目に指定したプラグインインデクスを利用できる条件に対して，プラグインインデクスで他インデクス絞り込み結果を利用できます。

```
SELECT ID FROM TBL1
WHERE CONTAINS(CHI,'root[time{"180"}]') IS TRUE AND CONTAINS(CHI,'root[elm{"1"}]') IS TRU
E;
 （アンダーラインのサーチ条件種別: SearchCnd : IS TRUE (AND))
```

例 3：

プラグインインデクスを利用できる二つの条件と B-Tree インデクスを利用できる条件を AND 演算すると，プラグインインデクスを利用できる両方の条件に対して，プラグインインデクスで他インデクス絞り込み結果を利用できます。このとき，2 番目に指定したプラグインインデクスの利用時には，B-Tree インデクスを利用できる条件と 1 番目に指定したプラグインインデクスを利用できる条件との AND 演算の絞り込み結果の行識別子集合の利用ができます。

```
SELECT ID FROM TBL1 WITH INDEX (IID, ICHI, ICHI)
WHERE ID >= 100 AND CONTAINS(CHI,'root[time{"180"}]') IS TRUE
      AND CONTAINS(CHI,'root[elm{"1"}]') IS TRUE;
 （アンダーラインのサーチ条件種別: SearchCnd : IS TRUE (AND))
```

<プラグインインデクスで他インデクス絞り込み結果を利用しない例>

例 1：

インデクスを利用できる条件としてプラグインインデクスに対する条件を一つしか指定していないのでプラグインインデクスで他インデクス絞り込み結果を利用できません。

```
SELECT ID FROM TBL1 WHERE CONTAINS(CHI,'root[elm{"1"}]') IS TRUE;
 （アンダーラインが該当部分）
```

例 2：

AND 演算ではなく，OR 演算を指定しているのでプラグインインデクスで他インデクス絞り込み結果を利用できません。

```
SELECT ID FROM TBL1 WITH INDEX (IID, ICHI)
WHERE ID >= 100 OR CONTAINS(CHI,'root[elm{"1"}]') IS TRUE;
 （アンダーラインが該当部分）
```

例 3：

プラグインインデクスを利用できる条件とインデクスを利用できない条件を AND 演算しているのでプラグインインデクスで他インデクス絞り込み結果を利用できません。

```
SELECT ID FROM TBL1
WHERE REGDATE = '2000-01-01'
      AND CONTAINS(CHI,'root[elm{"1"}]') IS TRUE;
 （アンダーラインが該当部分）
```

5 オブジェクトリレーショナルデータベースをアクセスする UAP 作成時の注意事項

この章では，オブジェクトリレーショナルデータベースをアクセスする UAP 作成時の注意事項について説明します。

5.1　抽象データ型及びユーザ定義関数を使用する場合の注意事項

抽象データ型がある表をアクセスする UAP，及びユーザ定義関数を使用する UAP を記述するときの注意事項について説明します。

5.1.1　埋込み変数のデータ型

- 埋込み変数の宣言（埋込み SQL 宣言節）中には，ユーザ定義型は指定できません。
- 検索対象となる表の列のデータ型が抽象データ型の場合，SELECT 文の選択式に列指定は指定できません。
- 関数の引数に埋込み変数を指定する場合は，実際に使用する関数の各引数のデータ型と一致させる必要があります。埋込み変数の記述と，関数の引数のデータ型が一致しない場合，その関数は利用できません。

  関数の引数のデータ型は，ディクショナリ表の SQL_ROUTINE_PARAMS 表を検索すれば分かります。ディクショナリ表の検索方法については，「操作系 SQL によるデータディクショナリ表の参照」の「検索時の SQL の記述例」を参照してください。

  関数の引数に次のデータ型を指定した場合，埋込み変数は使用できません。

  - ROW

5.1.2　定数のデータ型

関数の引数に定数を使用する場合は，実際に使用する関数の各引数のデータ型と一致させる必要があります。例えば，関数の引数のデータ型が SMALLINT の場合，整数定数を指定してもデータ型は一致しません。このような場合に，同じ関数名，同じ引数の数，及び引数が INTEGER 型の関数があるときは，その関数が適用されることがあるので注意してください。

関数の引数に次のデータ型を指定した場合，定数は使用できません。

- SMALLINT
- SMALLFLT
- CHAR
- NCHAR
- MCHAR
- DATE
- TIME
- TIMESTAMP
- INTERVAL YEAR TO DAY

- INTERVAL HOUR TO SECOND
- ROW
- BLOB
- BINARY

5.2　プラグイン提供関数の制限

プラグイン提供関数とは，プラグインが提供する関数のことを指します。

5.2.1　プラグイン提供関数間の値の受け渡しに関する制限

(1)　プラグイン提供関数の種類

プラグイン提供関数の種類を次の表に示します。

表 5–1　プラグイン提供関数の種類

関数の種類	プラグインの処理	
	受け渡しする値を生成し，ほかのプラグイン提供関数に送信する処理	ほかのプラグイン提供関数から送信された，受け渡しする値を受信する処理
受け渡しする値がない関数	なし	なし
受け渡しする値を受信する関数※1	なし	あり
受け渡しする値を送信する関数※2	あり	なし
受け渡しする値を送受信する関数	あり	あり

注※1

例えば，HiRDB Text Search Plug-in の場合，score を示します。

注※2

例えば，HiRDB Text Search Plug-in の場合，contains_with_score を示します。

プラグイン提供関数の機能として，プラグイン提供関数間で値の受け渡しができます。プラグイン提供関数間の値の受け渡しは，HiRDB が自動的にするので，受け渡しする値をプラグイン提供関数の引数に指定する必要はありません。

プラグイン提供関数の種類によっては，SQL 中に記述できる場所が異なるので注意する必要があります。プラグイン提供関数の種類については，各種プラグインマニュアルを参照してください。

なお，これ以降の説明では用語を次のように略します。

- 受け渡しする値がない関数→受渡し値なし関数
- 受け渡しする値を受信する関数→受渡し値受信関数
- 受け渡しする値を送信する関数→受渡し値送信関数
- 受け渡しする値を送受信する関数→受渡し値送受信関数

(2)　受渡し値送信関数と受渡し値受信関数の対応

受渡し値送信関数と受渡し値受信関数の対応についての規則を次に示します。

- 受渡し値送信関数と受渡し値受信関数との間には，値を受け渡しできるものと受け渡しできないものがあります。受渡し値送信関数と受渡し値受信関数との対応関係については，各種プラグインマニュアルを参照してください。
- 受渡し値送信関数と受渡し値受信関数の第 1 引数は同じで，実表の列指定，SQL パラメタ，又は SQL 変数である必要があります。なお，コンポネント指定は指定できません。
- 受渡し値送信関数と受渡し値受信関数は，一つの問合せ指定で閉じるようにしてください。ただし，リスト作成時に，受渡し値送信関数を指定して受渡し値をリストに格納しておいて，リストを介した検索時に受渡し値受信関数を指定してリストから受渡し値を取得する場合，受渡し値送信関数と受渡し値受信関数は，複数の問合せにわたって指定できます（「リスト間の集合演算実行方法の種類」を参照してください）。

受渡し値受信関数と受渡し値送信関数の組み合わせと，HiRDB の動作を次の表に示します。

表 5–2　受渡し値受信関数と受渡し値送信関数との組み合わせ

受渡し値受信関数の指定	受渡し値送信関数の指定	HiRDB の動作
なし	なし	実行できます。
	あり	
あり	なし	実行できません。※
	あり(一つ)	実行できます。
	あり(二つ以上)	実行できません。

注※

リストから受渡し値を取得する場合，受渡し値送信関数と受渡し値受信関数は，複数の問合せにわたって指定できます。

(3)　各プラグイン提供関数の制限

- **受渡し値なし関数**
  関数を指定できる箇所であれば，すべて指定できます。
- **受渡し値受信関数**
  - SELECT 文の選択式，問合せ指定がある INSERT 文の選択式，及び UPDATE 文の SET 句の更新値にだけ指定できます。
  - CASE 式中，及びスカラ関数 VALUE 中には指定できません。
  - GROUP BY 句，HAVING 句，又は集合関数を指定した場合，第 1 引数が SQL 変数又は SQL パラメタの受渡し値受信関数は，集合関数の引数中以外では指定できません。

- **受渡し値送信関数**

  (i) 受渡し値受信関数がない場合

  関数を指定できる箇所であれば，すべて指定できます。

  (ii) 受渡し値受信関数がある場合

  - WHERE 句，又は ON 探索条件にだけ指定できます。
  - 外結合を指定した結合表の ON 探索条件に，受渡し値送信関数を指定する場合，第 1 引数に外表の列は指定できません。
  - 受渡し値送信関数を OR のオペランドの探索条件中に指定する場合，次の条件をすべて満たす必要があります。

    ・受渡し値送信関数の第 1 引数にプラグインインデクスを定義している。

    ・受渡し値送信関数の第 1 引数が，外への参照列を除く実表の列指定である。

    ・受渡し値送信関数の第 1 引数を除く引数に，外への参照列を除く列指定，及び列に対するコンポネント指定の値式を含む引数を指定していない。

    ・受渡し値送信関数に対して，IS FALSE，IS UNKNOWN，及び NOT が含まれる述語を指定していない。

    ・受渡し値送信関数を CAST 指定中に指定していない。

    ・FROM 句に 2 表以上指定した場合，受渡し値送信関数の第 1 引数の列と異なる表の列を，OR のオペランドの探索条件中に指定していない（WHERE 句，及び ON 探索条件に論理演算子 NOT を含む場合，ド・モルガンの定理によって論理演算子 NOT を排除した結果が条件を満たすときも同様です）。
  - CASE 式中，及びスカラ関数 VALUE 中には指定できません。
  - GROUP BY 句，HAVING 句，又は集合関数を指定して定義した名前付き導出表を FROM 句に指定していて，かつこの名前付き導出表が内部導出表を作成しない場合，名前付き導出表を指定した問合せ指定の探索条件には，第 1 引数が SQL 変数，又は SQL パラメタとなる受渡し値送信関数は指定できません。
- **受渡し値送受信関数**

  SQL 中には指定できません。

5.2.2　プラグイン提供関数の実行方法に関する制限

(1)　プラグイン提供関数の実行方法

プラグイン提供関数を実行する方法には，次の二つがあります。

- インデクス型プラグインを使用して，プラグイン提供関数を実行する方法
- インデクス型プラグインを使用しないで，プラグイン提供関数を実行する方法

プラグイン提供関数には，インデクス型プラグインを使用しないと実行できない関数（インデクス型プラグイン専用関数）があります。

HiRDB がインデクス型プラグイン専用関数を実行する場合に，インデクス型プラグインを使用できないと判断したときはエラーとなります。エラーとなる組み合わせを次の表に示します。プラグイン提供関数が，インデクス型プラグイン専用関数であるかどうかについては，各種プラグインマニュアルを参照してください。

表 5–3　プラグイン提供関数実行時にエラーとなる組み合わせ

インデクス型プラグインを使用して関数を実行する方法	インデクス型プラグインを使用しないで関数を実行する方法	HiRDB が選択した検索方式	
		インデクス型プラグインを使用する検索	インデクス型プラグインを使用しない検索
提供されている	提供されている	○	○
提供されている※	提供されていない※	○	×
提供されていない	提供されている	–	○

（凡例）

○：実行できます。

×：実行時にエラーとなります。

–：該当しません。

注※

インデクス型プラグイン専用関数です。例えば，HiRDB Text Search Plug-in の場合は，contains 及び contains_with_score を指します。

(2)　インデクス型プラグイン専用関数の実行方法に関する制限

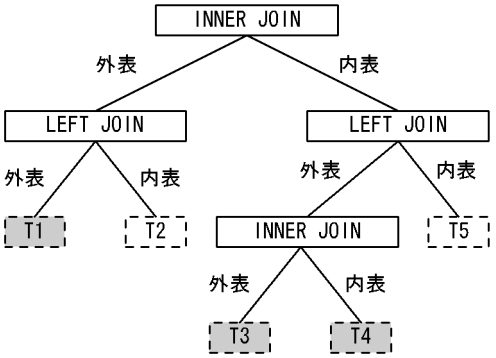
インデクス型プラグイン専用関数を使用する場合には，次の制限があります。

1. 第 1 引数には，外への参照列を除く実表の列指定だけ指定できます。
2. 第 1 引数を除く引数に，次の値式を含む引数を指定できません。
   - 外への参照列を除く列指定
   - 列に対するコンポネント指定
3. インデクス型プラグイン専用関数は，WHERE 句，又は ON 探索条件に指定できます。
4. 外結合を指定した問合せ指定の WHERE 句に，インデクス型プラグイン専用関数を指定する場合，第 1 引数には外結合の内表になる列は指定できません。例を次に示します。

SQL文

```
SELECT * FROM
  (T1 LEFT JOIN T2 ON 探索条件1)
  INNER JOIN
   ((T3 INNER JOIN T4 ON 探索条件)
     LEFT JOIN T5 ON 探索条件3)
  ON 探索条件4
  WHERE 探索条件5
```

・・・ 探索条件5にインデクス型プラグイン専用関数を指定する場合，第1引数には表T1，T3，又はT4の列を指定できます。



5. 外結合を指定した結合表の ON 探索条件にインデクス型プラグイン専用関数を指定する場合，第 1 引数に次の列は指定できません。
   - 外表の列
   - 内表が外結合を含む結合表の場合，内表に含まれる外結合の内表の列
6. FROM 句に 2 表以上の指定がある場合，インデクス型プラグイン専用関数の第 1 引数の列と異なる表の列を，OR のオペランドの探索条件中には指定できません。ただし，WHERE 句，及び ON 探索条件中に論理演算子 NOT を含む場合は，ド・モルガンの定理※によって論理演算子 NOT を排除した結果，上記の条件を満たさないときは指定できます。また，UNION を使用すると，上記の条件を満たしていても実行できる場合があります。例を次に示します。

   (例)

```
SELECT T1.C1,T2.C2 FROM T1,T2
  WHERE T1.C1=10 AND ( (CONTAINS(T2.ADT,'ABC') IS TRUE)
                    OR (CONTAINS(T2.ADT,'DEF') IS TRUE) )
```

   この SQL を，UNION を使用して表すと次のようになります。

```
( SELECT T1.C1,T2.C2 FROM T1,T2
  WHERE T1.C1=10 AND (CONTAINS(T1.ADT,'ABC') IS TRUE)
UNION ALL
  SELECT T1.C1,T2.C2 FROM T1,T2
      WHERE T1.C1=10 AND (CONTAINS(T2.ADT,'DEF') IS TRUE))
EXCEPT ALL
  SELECT T1.C1,T2.C2 FROM T1,T2
      WHERE T1.C1=10 AND (CONTAINS(T2.ADT,'DEF') IS TRUE)
            AND (CONTAINS(T1.ADT,'ABC') IS TRUE)
```

   注※

   次のような SQL 文があるとします。

```
SELECT T1.C1,T2.C2 FROM T1,T2
  WHERE NOT(CONTAINS(T1.ADT, ... ) IS NOT TRUE AND T1.C1=10)
  AND T1.C1=T2.C1
```

これを，ド・モルガンの定理で論理演算子 NOT を排除すると，次のようになります。

```
SELECT T1.C1,T2.C2 FROM T1,T2
  WHERE (CONTAINS(T1.ADT, ... ) IS TRUE OR T1.C1<>10)
  AND T1.C1=T2.C1
```

7. CASE 式中，及び CAST 指定中には指定できません。
8. インデクス型プラグイン専用関数に対して，IS FALSE，IS UNKNOWN，及び否定（NOT）が含まれる述語は指定できません。

これらの制限に関する例を次に示します。

（例 1）

WHERE 句にインデクス型プラグイン専用の受渡し値送信関数の指定がある場合，一つの問合せ指定中にその受渡し値送信関数と同一表の列を第 1 引数に持つインデクス型プラグイン専用関数を指定できない例を次に示します。

```
SELECT C1,C2,score(SENTENCES) FROM T1
   WHERE contains(SENTENCES, … ) IS TRUE
       AND contains_with_score(SENTENCES, … ) IS TRUE
```

（例 2）

表 T1，T2 を外結合させて，WHERE 句にインデクス型プラグイン専用関数を指定して検索する例を次に示します。

```
SELECT T1.C1,T2.C2 FROM T1 LEFT OUTER JOIN T2
   ON T1.C1=T2.C1 WHERE contains(T1.C3, … ) IS TRUE
```

5.2.3　受渡し値を格納したリストについての注意事項

(1)　リストへの受渡し値の格納

段階的に対象レコードを絞り込む場合（絞込み検索をする場合），リストへ受渡し値を格納しておくことで，受渡し値受信関数の結果を高速に取得できます。

ASSIGN LIST 文を使用して実表からリストを作成する場合の探索条件に，リストに受渡し値を格納できる受渡し値送信関数を指定することで，リストに受渡し値を格納できます（ただし，ASSIGN LIST 文には，リストに受渡し値を格納できる受渡し値送信関数は二つ以上指定できません）。

プラグイン提供関数が，リストに受渡し値を格納できるかどうかについては，各種プラグインマニュアルを参照してください。

また，ASSIGN LIST 文を使用して，受渡し値を格納したリストから，新たなリストに受渡し値を格納することもできます。

(2)　リストからの受渡し値の取得

リストを介した検索のカーソル指定の選択式に，リストから受渡し値を取得できる受渡し値受信関数（リスト用受渡し値受信関数）を指定することで，受渡し値送信関数を指定しないで，リストに格納された受渡し値を取得できます。

プラグイン提供関数が，リストから受渡し値を取得できるかどうかについては，各種プラグインマニュアルを参照してください。

リストを介した検索のカーソル指定の選択式に，リストから受渡し値を取得できる受渡し値受信関数を指定した場合，HiRDB はリストに受渡し値を格納した受渡し値送信関数の種別を意識しないで受渡し値を取得します。そのため，必ずリストを作成したときに指定した，受渡し値送信関数に対応する受渡し値受信関数を指定してください。

(3)　リスト間の集合演算実行方法の種類

リスト間の集合演算をする場合，リスト作成時に探索条件に指定した受渡し値送信関数によって，集合演算実行方法が変わります。

「リスト名 1 ｛AND ｜ OR ｜ AND NOT ｜ ANDNOT｝ リスト名 2」の集合演算結果の受渡し値を次の表に示します。

表 5–4　集合演算結果の受渡し値

<table>
<tr><th colspan="2" rowspan="3">リスト名 1 作成時の受渡し値送信関数※1</th><th colspan="3">リスト名 2 作成時の受渡し値送信関数※1</th></tr>
<tr><th colspan="2">リストに受渡し値を格納できる場合</th><th rowspan="2">そのほかの場合</th></tr>
<tr><th>「絞込み対象受渡し値使用」指定あり</th><th>集合演算方法の指定なし</th></tr>
<tr><td rowspan="2">リストに受渡し値を格納できる場合</td><td>「絞込み対象受渡し値使用」指定あり</td><td>×</td><td>×</td><td>リスト名 1 の受渡し値※2</td></tr>
<tr><td>集合演算方法の指定なし</td><td>×</td><td>×</td><td>×</td></tr>
<tr><td colspan="2">そのほかの場合</td><td>×</td><td>×</td><td>なし</td></tr>
</table>

（凡例）

×：実行できません。

注※1

集合演算方法の指定のある受渡し値送信関数については，各種プラグインマニュアルを参照してください。

注※2

OR 演算結果で受渡し値がない場合は，ナル値となります。

6 クライアントの環境設定

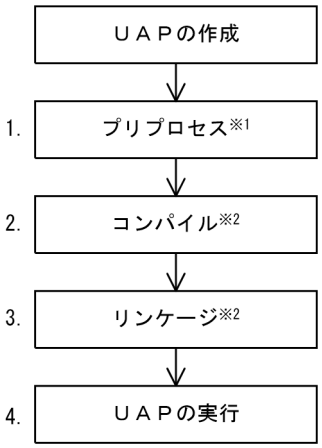
この章では，インストールや UAP の作成と実行に必要な環境定義の方法などについて説明します。

6.1 HiRDB クライアントの種類

HiRDB クライアントには次の 2 種類のプログラムプロダクトがあります。この二つのプログラムプロダクトを，HiRDB クライアントといいます。

- HiRDB/Developer's Kit
- HiRDB/Run Time

HiRDB クライアントの種類によって，UAP の作成から実行までの作業のうち，実行できる作業が異なります。UAP の作成から実行までの作業の流れを次に示します。



注※1

詳細は，「プリプロセス」を参照してください。

注※2

詳細は，「コンパイルとリンケージ」を参照してください。

HiRDB クライアントの種類によって実行できる作業は次のとおりです。

- HiRDB/Developer's Kit

  1～4 の作業ができます。

- HiRDB/Run Time

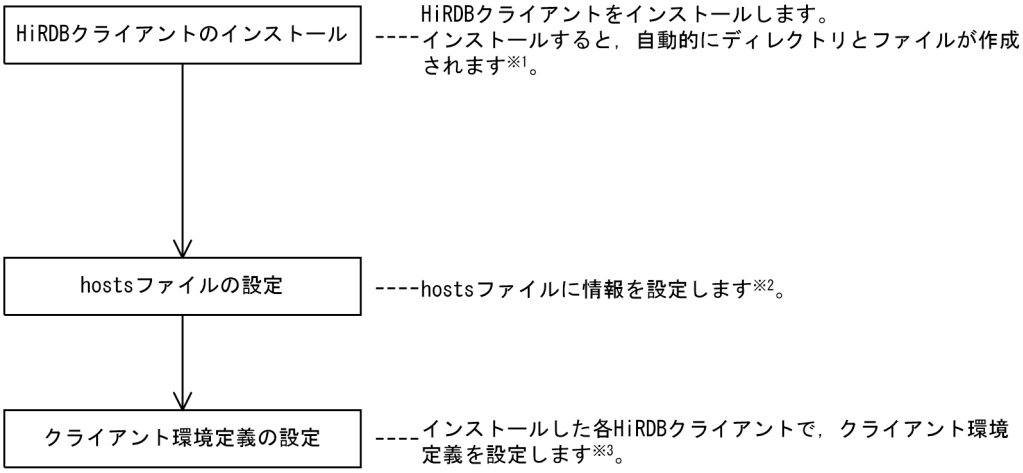
  4 の作業ができます。HiRDB クライアントの提供物は HiRDB サーバにも含まれています。したがって，1～3 は HiRDB サーバの機能で実行してください。

注意事項：

UAP を開発する HiRDB/Developer's Kit と，UAP を実行する HiRDB/Developer's Kit のプラットフォームは，同じにしてください。

6.2　HiRDB クライアントの環境設定手順

クライアントの環境設定手順を次に示します。



注※1

HiRDB クライアントのインストールについては「HiRDB クライアントのインストール及びアンインストール」を，自動的に作成されるディレクトリとファイルについては「HiRDB クライアントのディレクトリ及びファイル構成」を参照してください。

注※2

詳細は，「hosts ファイルの設定」を参照してください。

注※3

詳細は，「クライアント環境定義（環境変数の設定）」を参照してください。

6.3　HiRDB クライアントのインストール及びアンインストール

インストール方法は，HiRDB/Developer's Kit，HiRDB/Run Time とともに同じです。

なお，HiRDB クライアントの提供物は HiRDB サーバに含まれています。このため，HiRDB サーバと同一のサーバマシンでクライアントを運用する場合，サーバマシンには，HiRDB クライアントをインストールする必要はありません。HiRDB サーバと同一のサーバマシンでクライアントを運用する運用形態については，図「HiRDB サーバと同一のサーバマシンでクライアントを実行する運用形態」を参照してください。

同一のクライアントマシン又は同一のサーバマシンでは，次に示す製品のうち，一つの製品だけインストールできます。複数製品を共存させることはできません。

- 32 ビット版 HiRDB/Run Time
- 64 ビット版 HiRDB/Run Time
- 32 ビット版 HiRDB/Developer's Kit
- 64 ビット版 HiRDB/Developer's Kit

6.3.1　UNIX クライアントでのインストール

日立 PP インストーラを起動してインストールします。

6.3.2　Windows クライアントでの GUI によるインストール

Windows クライアントでインストールする場合，必ず Windows 環境で実行してください。

インストール手順：

インストールの手順を次に示します。なお，環境定義ファイル（HIRDB.INI）は，システムディレクトリに格納されます。

1. インストーラの起動
   統合 CD-ROM 中の hcd_inst.exe を実行して，日立総合インストーラを起動してください。
   「日立総合インストーラ」画面で次のどちらかを選択して，［インストール実行］ボタンをクリックしてください。
   - HiRDB/Run Time の場合は［HiRDB/Run Time］
   - HiRDB/Developer's Kit の場合は［HiRDB/Developer's Kit］

   選択したプログラムプロダクトのセットアッププログラムが起動します。
2. ユーザ情報の登録
   「ユーザ情報」画面が表示されます。

ユーザ名，及び会社名を入力して，[次へ]ボタンをクリックしてください。

3. インストールの開始

「インストール先の選択」画面が表示されます。

「インストール先のフォルダ」には HiRDB クライアントのインストール先を指定してください。省略した場合は，Windows がインストールされているドライブが仮定されます。インストール先を指定したら，[次へ]ボタンをクリックしてください。

4. セットアップタイプの選択

「セットアップタイプ」画面が表示されます。

セットアップタイプによって，インストールする機能を変更できます。

すべて

すべての機能をインストールします。

カスタム

選択した機能をインストールします。

「すべて」を選択した場合にインストールする機能を次の表に示します。

表 6–1　インストールする機能

項番	機能	製品種別
1	クライアントライブラリ	RT，DK
2	SQL プリプロセサ	DK
3	ODBC ドライバ	RT，DK
4	HiRDB OLE DB プロバイダ	RT，DK
5	HiRDB データプロバイダ for .NET Framework	RT，DK
6	JDBC ドライバ	RT，DK
7	SQLJ	RT，DK
8	XML 変換ライブラリ	RT，DK
9	コマンド・ユティリティ	RT，DK
10	サンプル UAP	DK

（凡例）

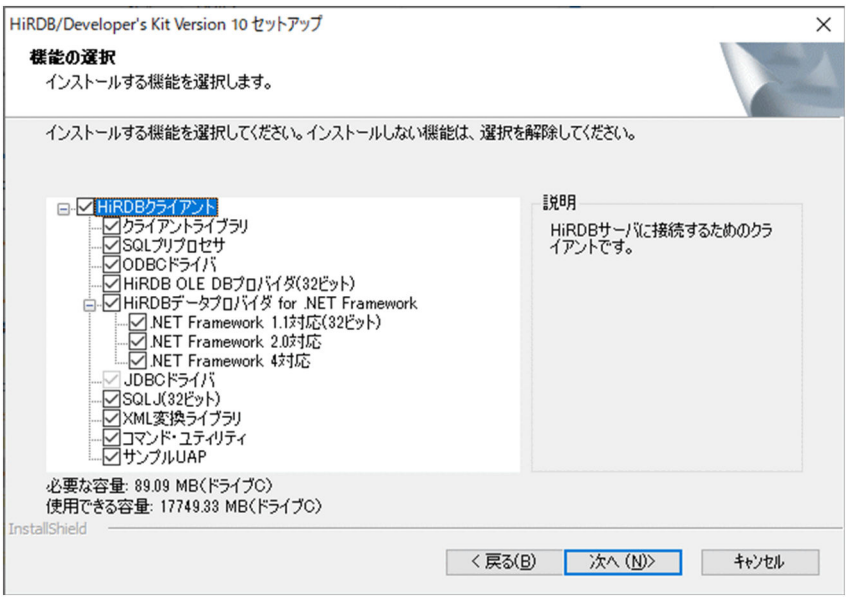
RT：HiRDB/Run Time

DK：HiRDB/Developer's Kit

セットアップタイプを選択したら，[次へ]ボタンをクリックしてください。

5. 機能の選択

「カスタム」を選択した場合，「機能の選択」画面が表示されます。



インストールする機能を指定して，[次へ]ボタンをクリックしてください。

6. ファイルコピーの開始

   「ファイルコピーの開始」画面が表示されます。

   現在の設定を確認して，[次へ]ボタンをクリックしてください。

7. インストールの実行状況の確認

   インストールの実行状況が表示されます。

   HiRDB クライアントのインストール先に必要な容量分の空きがないと，インストールの実行状況の表示中に，「データ転送プロセスでエラーが発生しました」というメッセージが表示され，インストールを中止します。この場合，インストール先の空き容量を確保し，再度インストールを実行してください。

   上書きインストール時に，インストール先のファイルが使用中のため更新できなかった場合，「使用中のため更新できないファイルを検出しました。更新を完了するには，コンピュータを再起動する必要があります。」というメッセージが表示されます。この場合，インストールを継続し，次回 Windows 起動時にファイルが更新されます。

8. インストールの完了

   インストールが完了すると，「セットアップの完了」画面が表示されます。

注意事項

- 環境定義ファイルの 1 行の最大長は，512 バイトです。512 バイトを超えて定義すると，その定義は無効になります。
- Administrators 権限又は Power Users 権限があるユーザがインストールしてください。
  Administrators 権限又は Power Users 権限がないユーザがインストールすると，システム環境変数が更新されません。

6.3.3 UNIX クライアントでのアンインストール

日立 PP インストーラを起動してアンインストールします。

HiRDB クライアントをアンインストールする場合は，コマンド，アプリケーションをあらかじめ停止しておいてください。これらを停止しないと，実行形式ファイルや共用ライブラリの削除に失敗し，アンインストール後にファイルが残ることがあります。この場合，残ったファイルを削除してください。

6.3.4 Windows クライアントでの GUI によるアンインストール

アンインストールの手順を次に示します。

1. ［スタート］－［HiRDB Client］－［HiRDB Client のアンインストール］を選択します。［HiRDB Client のアンインストール］が登録されていない場合は，［コントロールパネル］－［プログラムと機能］の［プログラムのアンインストールまたは変更］を表示し，一覧の中から「HiRDB」を選択します。

2. 「選択したアプリケーション，およびすべての機能を完全に削除しますか？」というメッセージが表示されます。「はい」を選択すると，アンインストールを開始します。

3. アンインストールが完了すると，アンインストール完了を示すメッセージが表示されます。［完了］ボタンをクリックしてください。

注意事項：

- HiRDB クライアントをアンインストールする場合は，コマンド，アプリケーションをあらかじめ停止してください。これらを停止しないと，実行形式ファイルや共用ライブラリの削除に失敗し，アンインストール後にファイルが残ることがあります。この場合，残ったファイルを削除してください。
- HiRDB クライアントのバージョンダウン時は，新バージョンをアンインストールしてから，旧バージョンをインストールしてください。バージョンダウン時には，上書きインストールは実施しないでください。
- HiRDB クライアントのバージョンアップ，及び修正版 HiRDB クライアントのインストーラによる入れ替えには，次のどちらかの方法を適用できます。
  ・旧バージョンをアンインストールしないで，新バージョンを上書きインストールする。
  ・旧バージョンをアンインストールしてから，新バージョンをインストールする。

6.3.5 Windows クライアントでのサイレントインストール及びアンインストール

HiRDB クライアントのサイレントインストール，サイレントアンインストール，および修正パッチのサイレントインストールについて説明します。サイレントインストール，およびサイレントアンインストールでは，実行時にダイアログが表示されません。

(1)　サイレントインストール

サイレントインストール用のコマンドを起動し，インストールします。サイレントインストールでは，すべてのクライアント機能をインストールします。

クライアント機能の一覧については，表「Windows 系 HiRDB 製品でのクライアント機能同梱状況」を参照してください。

(a)　格納場所

この機能で使用するコマンドは，統合 CD-ROM 中の次のフォルダに格納されています。

```
_PPDIR¥形名フォルダ¥DISK1
```

形名フォルダは，形名を短縮した名称となります。例えば，形名 P-2662-11A4 の場合，「_PPDIR¥P266211A4¥DISK1」となります。

(b)　実行者

Administrators グループの管理者権限で実行してください。

(c)　コマンドラインの形式

```
pdslinst.exe 〔/AUTO;オプションファイルのパス〕
```

/AUTO;オプションファイルのパス

サイレントインストールの設定情報を記述したオプションファイルを，259 文字（バイト）以内の絶対パスで指定します。

(d)　オプションファイルの形式

オプションファイルには，次の形式でオプションを記載します。

```
オプション名=指定値
```

最後に CR+LF 形式の「改行」を入れてください。また，オプションファイルの文字コードは MS932 です。

オプションファイルの記述例

```
instdir=C:¥Program Files¥HITACHI¥HiRDB
```

指定できるオプションを次の表に示します。

表 6–2　オプションファイルに指定できるオプション

オプション	指定値	説明
instdir	インストール先フォルダ	インストール先フォルダを，200 文字（バイト）以内の絶対パスで指定してください。 指定値とインストール先の対応を次に示します。 指定値とインストール先 【新規インストール】 • 省略時はデフォルトのインストール先※へインストールします。 【上書きインストール】 • 省略時はインストール済みのパスへインストールします。 • 指定を省略しないで，インストール済みのパス以外を指定した場合にはエラーが発生します（新規インストール時と異なるパス文字列表現をした場合にもエラーが発生します）。

注※

デフォルトのインストール先です。デフォルトのインストール先を次の表に示します。

表 6–3　デフォルトのインストール先

OS	HiRDB クライアント	
	Windows(x86)	Windows(x64)
Windows(x86)	C:¥Program Files¥HITACHI¥HiRDB	インストール不可
Windows(x64)	C:¥Program Files (x86)¥HITACHI¥HiRDB	C:¥Program Files¥HITACHI¥HiRDB

(e)　戻り値

0：正常終了

2：正常終了（Windows の再起動要）

-1：インストール失敗

インストールが失敗した場合のエラー情報の詳細は，インストール実行結果情報ファイルを参照してください。インストール実行結果情報ファイルが出力されていない場合は，環境変数 ALLUSERSPROFILE 及び環境変数 TEMP に書き込みできるフォルダを設定した上で再インストールしてください。インストール実行結果情報ファイルが参照できない場合は保守員に連絡してください。

(f)　インストール実行結果情報ファイル

インストールコマンドを実行すると，%ALLUSERSPROFILE%¥HITACHI¥HiRDB¥pdslinst.log にインストール実行結果情報ファイルを出力します。出力できない場合は%TEMP%¥pdslinst.log に出力します。

インストール実行結果情報ファイルには，次の表に示す情報を出力します。

表 6-4　インストール実行結果情報ファイルの内容

出力行	情報		内容
1 行目	コマンドライン		実行されたコマンドライン （コマンドは絶対パスで出力）
2 行目	終了ステータス		インストール実行結果を示す 2 桁の 16 進数
3 行目	終了ステータスに応じて次のどれかを出力		
	終了ステータス=00,02	アンインストールコマンド	サイレントアンインストールを実行する場合に使用するコマンドライン
	終了ステータス=B6	詳細コード	InstallShield の戻り値
	終了ステータス=BA	詳細情報	Windows API：エラーコード
	上記以外	なし	なし

出力例を次に示します。

なお，出力例の↓は 1 行の末尾を示します。実際に出力するファイルでは表示されません。

インストール実行結果情報ファイルの出力例

- 正常終了の場合

```
C:¥tmp¥DISK1¥pdslinst.exe /AUTO;C:¥tmp¥test.ini↓
00↓
"C:¥Program Files (x86)¥InstallShield Installation Information¥{1CD2D8D0-BFC9-4A6D-B5A0-E
CA663C9C9E5}¥setup.exe" -runfromtemp -l0x0411 -uninst /a /s /f1"C:¥Program Files (x86)¥In
stallShield Installation Information¥{1CD2D8D0-BFC9-4A6D-B5A0-ECA663C9C9E5}¥setup.iss"↓
```

- InstallShield でエラーが発生した場合

```
C:¥tmp¥DISK1¥pdslinst.exe /AUTO;C:¥tmp¥test.ini↓
B6↓
-5001↓
```

- オプション不正の場合

```
C:¥tmp¥DISK1¥pdslinst.exe /AUTO;C:¥tmp¥test.ini↓
88↓
```

- Windows API でエラーが発生した場合

```
C:¥tmp¥DISK1¥pdslinst.exe "/AUTO;C:¥tmp¥test.ini"  ↓
BA↓
CreateProcess : 5↓
```

インストール実行結果情報ファイルの終了ステータスの意味を次の表に示します。対処方法でインストールできない場合は，保守員に連絡してください。

表 6–5　インストール実行結果情報ファイルの終了ステータスの意味

終了ステータス	区分	内容	対処方法
00	正常終了	インストールを正常に終了しました。	なし。
02		インストールを正常に終了しましたが，使用中のファイルがあるため，Windows の再起動が必要です。	なし。
83	異常終了	レジストリアクセス中にエラーが発生しました。	権限不足のため，エラーが発生しています。Administrators 権限を持っているユーザで再インストールしてください。
85		ディスクアクセス中にエラーが発生しました，又はインストール先ディスクの空き容量が不足しています。	ディスクアクセス時のメモリ不足を回避するため，起動中のプログラムやサービスを停止し，次のフォルダがあるどれかのドライブの不要ファイルを削除してから，Administrators 権限を持っているユーザで再インストールしてください。 • インストール先フォルダ • 環境変数 TEMP のフォルダ • 環境変数 WINDIR のフォルダ
88		コマンドライン又はオプションファイルの内容に誤りがあります。	コマンドライン又はオプションファイルの内容を修正し，再インストールしてください。
92		Windows(x64)版の HiRDB クライアントを，Windows(x86)の OS にインストールしました。	Windows(x86)版の HiRDB クライアントをインストールするか，又は Windows(x64)の OS にインストールしてください。
B1		HiRDB ODBC ドライバのインストールでエラーが発生しました。	保守員に連絡してください。
B4		Windows Installer でエラーが発生しました。	コントロールパネルから Windows Installer のサービスが開始できない場合は，Windows Installer を修復（Windows Installer を再インストールするなど）して再インストールしてください。
B5		上書きインストールで，既存のインストール先と異なる絶対パスがインストール先に指定されました。	コマンドライン引数を削除して再インストールしてください。
B6		InstallShield でエラーが発生しました。	インストール実行結果情報ファイルの詳細コードを確認し，保守員に連絡してください。

終了ステータス	区分	内容	対処方法
B7		オプションファイルの読み込み時にエラーが発生しました。	オプションファイルへの参照権限があるユーザで，/AUTO オプションにオプションファイルの絶対パスを正しく指定して，再インストールしてください。
B8		システム環境変数への登録時にエラーが発生しました。	システム環境変数に次のサイズのパスが追加できるよう，不要なパスを削除して，再インストールしてください。 <PATH> インストール先の絶対パス＋5 <CLASSPATH> • HiRDB/Developer's Kit の場合 インストール先の絶対パス×3＋49 • HiRDB/Run Time の場合 インストール先の絶対パス×4＋72
B9		メモリ不足が発生しました。	起動中のプログラムやサービスを停止し，Administrators 権限を持っているユーザで再インストールしてください。
BA		Windows API でエラーが発生しました。	保守員に連絡してください。
BB		.NET Framework4 以降が未インストールです。	.NET Framework4 以降をインストールして，再インストールしてください。
80		上記以外のエラーが発生しました。	保守員に連絡してください。

(g) InstallShield のログファイル

インストールコマンドを実行すると，%ALLUSERSPROFILE%¥HITACHI¥HiRDB¥pdslinst_setup.log に InstallShield のログファイルを出力します。出力できない場合は%TEMP%¥pdslinst_setup.log に出力します。

(2) サイレントアンインストール

サイレントアンインストール用のコマンドを起動し，アンインストールします。

(a) 実行者

Administrators グループの管理者権限で実行してください。

(b) コマンドラインの形式

インストール実行結果情報ファイルで出力されているアンインストールコマンド

インストール結果情報ファイルがない場合には，スタートメニューに登録されたショートカット「HiRDB Client のアンインストール」のコマンドのプロパティを参照し，最後に次のオプションを追加してください。インストール結果情報ファイルで出力されるアンインストールコマンドと同等の形式になります。

```
/a /s /f1”「HiRDB Clientのアンインストール」のsetup.exeコマンドのあるフォルダ¥setup.iss”
```

サイレントアンインストールをするためのコマンドラインの例を，次に示します。

「HiRDB Client のアンインストール」のコマンドのプロパティ

```
”C:¥Program Files (x86)¥InstallShield Installation Information¥{1CD2D8D0-BFC9-4A6D-B5A0-ECA6
63C9C9E5}¥setup.exe” -runfromtemp -l0x0411  -uninst
```

サイレントアンインストールをするためのコマンドライン

```
”C:¥Program Files (x86)¥InstallShield Installation Information¥{1CD2D8D0-BFC9-4A6D-B5A0-ECA6
63C9C9E5}¥setup.exe” -runfromtemp -l0x0411  -uninst /a /s /f1” C:¥Program Files (x86)¥Instal
lShield Installation Information¥{1CD2D8D0-BFC9-4A6D-B5A0-ECA663C9C9E5}¥setup.iss”
```

(c)　アンインストールエラー情報ファイル

アンインストール時にエラーが発生した場合，%ALLUSERSPROFILE%¥HITACHI¥HiRDB¥unsetup.err にアンインストールエラー情報ファイルを出力します。出力できない場合は%TEMP%¥unsetup.err に出力します。

アンインストールエラー情報ファイルには，発生したエラーを示すエラーコードを出力します。出力例を次に示します。

アンインストールエラー情報ファイルの出力例

```
80
```

エラーコードの意味について，次の表に示します。

対処方法でアンインストールできない場合は，保守員に連絡してください。

表 6–6　エラーコードの意味

エラーコード	内容	対処方法
80	Windows Installer でエラーが発生しました。	コントロールパネルから Windows Installer のサービスが開始できることを確認し，開始できない場合は，Windows Installer を修復（Windows Installer を再インストールするなど）し，再アンインストールしてください。
81	メモリ不足が発生しました。	起動中のプログラムやサービスを停止し，再アンインストールしてください。

エラーコード	内容	対処方法
82	レジストリアクセス中にエラーが発生しました。	Administrator グループの管理者権限でアンインストールしていない場合，Administrator グループの管理者権限で再アンインストールしてください。

(3)　修正パッチのサイレントインストール

修正パッチ適用時のサイレントインストールについて説明します。

なお，修正パッチだけをサイレントアンインストールすることはできません。

(a)　実行者

Administrators グループの管理者権限で実行してください。

(b)　コマンドラインの形式

```
cmd /c ”setup.exe /s”
```

修正パッチを任意のフォルダへ展開し，展開したフォルダに移動した後，コマンドを実行してください。

(c)　戻り値

戻り値について次の表に示します。

対処方法でアップデートできない場合は，保守員に連絡してください。

表 6–7　修正パッチコマンドの戻り値

戻り値	内容	対処方法
0	正常に終了しました。	なし。
1	正常に終了しましたが，使用中のファイルがあるため，Windows の再起動が必要です。	なし。
132	メモリ不足が発生しました。	起動中のプログラムやサービスを停止し，再アップデートしてください。
134	ディスクの空き容量が不足しています。	次のフォルダがあるどれかのドライブの不要ファイルを削除してから，Administrators 権限を持っているユーザで再アップデートしてください。 • インストール先フォルダ • 環境変数 WINDIR のフォルダ
135	ログファイルのオープン時にエラーが発生しました。	インストール先フォルダに書き込み権限を与えた上で，再アップデートしてください。

戻り値	内容	対処方法
144	ファイル及びディレクトリ操作で，エラーが発生しました。	ディスクアクセス時のメモリ不足を回避するため，起動中のプログラムやサービスを停止し，Administrators 権限を持っているユーザで再アップデートしてください。
159	HiRDB クライアントが未インストールであるか，又は修正パッチ適用対象のバージョンが不正です。	HiRDB クライアントがインストール済みであることを確認してください。又は，適用する修正パッチのバージョン番号，リビジョン番号，修正版コード（VV-RR-SS の SS）が正しいことを確認し，再アップデートしてください。
上記以外	その他エラーが発生しました。	保守員に連絡してください。

(d)　バージョン情報の確認

パッチ適用に成功した場合は，修正パッチのコマンド（setup.exe）を格納したディレクトリ内の RELEASE.TXT を参照し，「◆パッチユーティリティ」の記載内容に従ってバージョン情報を確認してください。

6.4　HiRDB クライアントのディレクトリ及びファイル構成

HiRDB クライアントをインストールすると，ディレクトリ，及びファイルが自動的に作成されます。ここでは，それらのディレクトリ，及びファイル構成を説明します。

6.4.1　UNIX クライアントのディレクトリ及びファイル構成

インストール時に自動的に作成されるファイルとディレクトリを次の表に示します。

- 表「HiRDB/Developer's Kit の場合のファイルとディレクトリ（UNIX クライアント）」
- 表「HiRDB/Run Time の場合のファイルとディレクトリ（UNIX クライアント）」
- 表「HiRDB/Developer's Kit の場合のファイルとディレクトリ（Linux(EM64T)）」
- 表「HiRDB/Run Time の場合のファイルとディレクトリ（Linux(EM64T)）」

表 6–8　HiRDB/Developer's Kit の場合のファイルとディレクトリ（UNIX クライアント）

名称	ディレクトリ※1	ファイル名	プラットフォーム		
			AIX (32)	AIX (64)	Linux (32)
ヘッダファイル	/HiRDB/include	SQLCA.CBL	○	○	○
		SQLCA64.CBL	×	○	×
		SQLDA.CBL	○	○	○
		SQLDA64.CBL	×	○	×
		SQLIOA.CBL	○	○	○
		SQLIOA64.CBL	×	○	×
		pdbtypes.h	○	○	○
		pdberrno.h	○	○	○
		pdbmisc.h	○	○	○
		pdbmiscm.h	○	○	○
		pdbsqlda.h	○	○	○
		pdbsqlcsna.h	○	○	○
		pddbhash.h	○	○	○
		pdauxcnv.h	○	○	○
		SQLCAM.cbl	○	○	○
		SQLDAM.cbl	○	○	○

名称	ディレクトリ※1	ファイル名	プラットフォーム		
			AIX (32)	AIX (64)	Linux (32)
		SQLIOAM.cbl	○	○	○
		SQLIOAMTH.CBL	○	○	○
		SQLIOAMTH64.CBL	×	○	×
		SQLCAMTH.CBL	○	○	○
		SQLCAMTH64.CBL	×	○	×
		SQLCSNA.CBL	○	○	○
		SQLCSNA64.CBL	×	○	×
		SQLIOA32.CBL	○	○	○
		SQLIOALP.CBL	×	○	×
		SQLCNSTS.CBL	○	○	○
		SQLCNSTD.CBL	○	○	○
		SQLMCNCT.CBL	○	○	○
アーカイブファイル	/HiRDB/client/lib	libclt.a	○	○	○
		libclt64.a	×	○	×
		libcltxa.a	○	○	○
		libcltya.a	○	○	○
		libcltk.a	○	○	○
		libcltk64.a	×	○	×
		libclts.a	○	○	○
共用ライブラリ※2	/HiRDB/client/lib	libzclt.sl	○	○	○
		libzclt64.sl	×	○	×
		libzcltx.sl	○	○	○
		libzclty.sl	○	○	○
		libzcltk.sl	○	○	○
		libzcltkt.sl	×	×	○
		libzcltk64.sl	×	○	×
		libsqlauxf.sl	○	○	○
		libsqlauxf64.sl	×	○	×
		libzcltxk.sl	○	○	△

名称	ディレクトリ※1	ファイル名	プラットフォーム		
			AIX (32)	AIX (64)	Linux (32)
		libzcltyk.sl	○	○	△
		libzclts.sl	○	○	○
		libzcltxs.sl	○	○	○
		libzcltys.sl	○	○	○
		libzclty64.sl	×	○	×
		libzcltys64.sl	×	○	×
		libzclt6k.sl	○	○	×
		libzclt6k64.sl	×	○	×
		libzclt6yk.sl	○	○	×
		libzclt6yk64.sl	×	○	×
		libzclt6ys.sl	○	○	×
		libzclt6ys64.sl	×	○	×
		libzsocrypto.sl	○	○	○
		libzsocrypto_r.sl	○	○	○
		libzsocrypto64.sl	×	○	×
		libzsocrypto_r64.sl	×	○	×
JDBC ドライバ	/HiRDB/client/lib	libjjdbc.sl	○	○	○
		pdjdbc.jar	○	○	○
		pdjdbc2.jar	○	○	○
		pdjdbc4.jar	○	○	○
		pdjdbc43.jar	○	○	○
ODBC ドライバ	/HiRDB/client/lib	libodbcdrv.sl	×	×	○
		libodbcdrv64.sl	×	×	×
コマンド・ユティリティ	/HiRDB/client/utl	pdcpp	○	○	○
		pdocc	○	○	○
		pdcbl	○	○	○
		pdocb	○	○	○
		pdprep	○	○	○
		pdtrcmgr	○	○	○

名称	ディレクトリ※1	ファイル名	プラットフォーム		
			AIX (32)	AIX (64)	Linux (32)
	/HiRDB/bin	pddef	○	○	○
SQLJ	/HiRDB/client/lib	pdruntime.jar	○	×	○
		pdnativert.jar	○	×	○
		pdsqlj.jar	○	×	○
		libpdparse.sl	○	×	○
		libpdsqljn.sl	○	×	○
	/HiRDB/client/utl	pdjava	○	×	○
メッセージオブジェクトファイル	/HiRDB/lib	msgtxt	○	○	○
構文解析ライブラリ※2	/HiRDB/lib/sjis	libasqap.sl	○	○	○
	/HiRDB/lib/chinese		○	○	○
	/HiRDB/lib/lang-c		○	○	○
	/HiRDB/lib/ujis		○	○	○
	/HiRDB/lib/utf-8		○	○	○
	/HiRDB/lib/utf-8_ivs		○	○	○
	/HiRDB/lib/chinese-gb18030		○	○	○
サンプルソース	/HiRDB/client/sampleap/uap	CREATE.ec	○	○	○
		SAMPLE1.ec	○	○	○
		SAMPLE2.ec	○	○	○
		SAMPLE3.ec	○	○	○
		sample1.ecb	○	○	○
		sample.mk	○	○	○
		inputf1	○	○	○
		inputf2	○	○	○
XML 変換コマンド※3	/HiRDB/client/utl	phdxmlcnv	○	○	○
XML 変換ライブラリ※3	/HiRDB/client/lib	XMLConverter.jar	○	○	○

（凡例）

AIX(32)：32 ビットモードの AIX

AIX(64)：64 ビットモードの AIX

Linux(32)：32 ビットモードの Linux

○：ファイルは作成されます。

△：ファイルは作成されますが，そのファイルを使用した機能は動作しません。

×：ファイルは作成されません。

注※1

下線で示す部分は，HiRDB のインストールディレクトリを示します。

注※2

共用ライブラリ，構文解析ライブラリのサフィックスは，プラットフォームによって異なります。Linux の場合は「.so」，AIX の場合は「.a」となります。

注※3

HiRDB XML Extension に同梱されるため，HiRDB サーバ製品中の HiRDB クライアントには同梱されません。

表 6–9　HiRDB/Run Time の場合のファイルとディレクトリ（UNIX クライアント）

名称	ディレクトリ※1	ファイル名	プラットフォーム		
			AIX (32)	AIX (64)	Linux (32)
アーカイブファイル	/HiRDB/client/lib	libclt.a	○	○	○
		libclt64.a	×	○	×
		libcltxa.a	○	○	○
		libcltya.a	○	○	○
		libcltk.a	○	○	○
		libcltk64.a	×	○	×
		libclts.a	○	○	○
共用ライブラリ※2	/HiRDB/client/lib	libzclt.sl	○	○	○
		libzclt64.sl	×	○	×
		libzcltx.sl	○	○	○
		libzclty.sl	○	○	○
		libzcltk.sl	○	○	○
		libzcltkt.sl	×	×	○
		libzcltk64.sl	×	○	×

名称	ディレクトリ※1	ファイル名	プラットフォーム		
			AIX (32)	AIX (64)	Linux (32)
		libsqlauxf.sl	○	○	○
		libsqlauxf64.sl	×	○	×
		libzcltxk.sl	○	○	△
		libzcltyk.sl	○	○	△
		libzclts.sl	○	○	○
		libzcltxs.sl	○	○	○
		libzcltys.sl	○	○	○
		libzclty64.sl	×	○	×
		libzcltys64.sl	×	○	×
		libzclt6k.sl	○	○	×
		libzclt6k64.sl	×	○	×
		libzclt6yk.sl	○	○	×
		libzclt6yk64.sl	×	○	×
		libzclt6ys.sl	○	○	×
		libzclt6ys64.sl	×	○	×
		libzsocrypto.sl	○	○	○
		libzsocrypto_r.sl	○	○	○
		libzsocrypto64.sl	×	○	×
		libzsocrypto_r64.sl	×	○	×
JDBC ドライバ	/HiRDB/client/lib	libjjdbc.sl	○	○	○
		pdjdbc.jar	○	○	○
		pdjdbc2.jar	○	○	○
		pdjdbc4.jar	○	○	○
		pdjdbc43.jar	○	○	○
ODBC ドライバ	/HiRDB/client/lib	libodbcdrv.sl	×	×	○
		libodbcdrv64.sl	×	×	×
SQLJ ランタイム	/HiRDB/client/lib	pdruntime.jar	○	×	○
		pdnativert.jar	○	×	○
		libpdsqljn.sl	○	×	○

名称	ディレクトリ※1	ファイル名	プラットフォーム		
			AIX (32)	AIX (64)	Linux (32)
コマンド・ユティリティ	/HiRDB/client/utl	pdtrcmgr	○	○	○
XML 変換コマンド※3	/HiRDB/client/utl	phdxmlcnv	○	○	○
XML 変換ライブラリ※3	/HiRDB/client/lib	XMLConverter.jar	○	○	○

（凡例）

AIX(32)：32 ビットモードの AIX

AIX(64)：64 ビットモードの AIX

Linux(32)：32 ビットモードの Linux

○：ファイルは作成されます。

△：ファイルは作成されますが，そのファイルを使用した機能は動作しません。

×：ファイルは作成されません。

注※1

下線で示す部分は，HiRDB のインストールディレクトリを示します。

注※2

共用ライブラリのサフィックスは，プラットフォームによって異なります。Linux の場合は「.so」，AIX の場合は「.a」となります。

注※3

HiRDB XML Extension に同梱されるため，HiRDB サーバ製品中の HiRDB クライアントには同梱されません。

表 6–10　HiRDB/Developer's Kit の場合のファイルとディレクトリ（Linux(EM64T)）

名称	ディレクトリ	ファイル名
ヘッダファイル	/HiRDB/include	SQLCA.CBL
		SQLCA64.CBL
		SQLDA.CBL
		SQLDA64.CBL
		SQLIOA.CBL
		SQLIOA64.CBL
		pdbtypes.h
		pdberrno.h

名称	ディレクトリ	ファイル名
		pdbmisc.h
		pdbmiscm.h
		pdbsqlda.h
		pdbsqlcsna.h
		pddbhash.h
		pdauxcnv.h
		SQLCAM.cbl
		SQLDAM.cbl
		SQLIOAM.cbl
		SQLIOAMTH.CBL
		SQLIOAMTH64.CBL
		SQLCAMTH.CBL
		SQLCAMTH64.CBL
		SQLCSNA.CBL
		SQLCSNA64.CBL
		SQLIOA32.CBL
		SQLIOALP.CBL
		SQLCNSTS.CBL
		SQLCNSTD.CBL
		SQLMCNCT.CBL
共用ライブラリ	/HiRDB/client/lib	libzclt.so
		libzclt64.so
		libzcltx.so
		libzclty.so
		libzclty64.so
		libzcltys64.so
		libzcltk.so
		libzcltkt.so
		libzcltk64.so
		libsqlauxf.so
		libsqlauxf64.so

名称	ディレクトリ	ファイル名
		libzcltxk.so
		libzcltyk.so
		libzcltyk64.so
		libzclts.so
		libzcltxs.so
		libzcltys.so
		libzsocrypto.so
		libzsocrypto_r.so
		libzsocrypto64.so
		libzsocrypto_r64.so
JDBC ドライバ	/HiRDB/client/utl	libjjdbc.so
		pdjdbc.jar
		pdjdbc2.jar
		pdjdbc4.jar
		pdjdbc43.jar
ODBC ドライバ	/HiRDB/client/lib	libodbcdrv.so
		libodbcdrv64.so
コマンド・ユティリティ	/HiRDB/client/utl	pdcpp
		pdocc
		pdcbl
		pdocb
		pdprep
		pdtrcmgr
	/HiRDB/bin	pddef
		pddivinfgt
SQLJ	/HiRDB/client/lib	pdsqlj.jar
		pdruntime.jar
		pdnativert.jar
		libpdparse.so
		libpdsqljn.so
	/HiRDB/client/utl	pdjava

名称	ディレクトリ	ファイル名
メッセージオブジェクトファイル	/HiRDB/lib	msgtxt
構文解析ライブラリ	/HiRDB/lib/sjis	libasqap.so
	/HiRDB/lib/chinese	
	/HiRDB/lib/lang-c	
	/HiRDB/lib/ujis	
	/HiRDB/lib/utf-8	
	/HiRDB/lib/utf-8_ivs	
	/HiRDB/lib/chinese-gb18030	
サンプルソース	/HiRDB/client/sampleap/uap	CREATE.ec
		SAMPLE1.ec
		SAMPLE2.ec
		SAMPLE3.ec
		Sample1.ecb
		Sample.mk
		inputf1
		inputf2
XML 変換コマンド※	/HiRDB/client/utl	phdxmlcnv
XML 変換ライブラリ※	/HiRDB/client/lib	XMLConverter.jar

注

下線で示す部分は，HiRDB のインストールディレクトリを示します。

注※

HiRDB XML Extension に同梱されるため，HiRDB サーバ製品中の HiRDB クライアントには同梱されません。

表 6–11　HiRDB/Run Time の場合のファイルとディレクトリ（Linux(EM64T)）

名称	ディレクトリ	ファイル名
共用ライブラリ	/HiRDB/client/lib	libzclt.so
		libzclt64.so
		libzcltx.so
		libzclty.so
		libzclty64.so

名称	ディレクトリ	ファイル名
		libzcltys64.so
		libzcltk.so
		libzcltkt.so
		libzcltk64.so
		libsqlauxf.so
		libsqlauxf64.so
		libzcltxk.so
		libzcltyk.so
		libzcltyk64.so
		libzclts.so
		libzcltxs.so
		libzcltys.so
		libzsocrypto.so
		libzsocrypto_r.so
		libzsocrypto64.so
		libzsocrypto_r64.so
JDBC ドライバ	/HiRDB/client/utl	libjjdbc.so
		pdjdbc.jar
		pdjdbc2.jar
		pdjdbc4.jar
		pdjdbc43.jar
ODBC ドライバ	/HiRDB/client/lib	libodbcdrv.so
		libodbcdrv64.so
SQLJ ランタイム	/HiRDB/client/lib	libpdsqljn.so
		pdruntime.jar
		pdnativert.jar
コマンド・ユティリティ	/HiRDB/client/utl	pdtrcmgr
	/HiRDB/bin	pddivinfgt
XML 変換コマンド※	/HiRDB/client/utl	phdxmlcnv
XML 変換ライブラリ※	/HiRDB/client/lib	XMLConverter.jar

注

下線で示す部分は，HiRDB のインストールディレクトリを示します。

注※

HiRDB XML Extension に同梱されるため，HiRDB サーバ製品中の HiRDB クライアントには同梱されません。

●アーカイブファイル，共用ライブラリの用途別の使用ファイル

アーカイブファイルの用途別の使用ファイルを次の表に示します。

表 6–12　アーカイブファイルの用途別の使用ファイル（UNIX クライアント）

用途		使用するファイル
通常の UAP		libclt.a
XA インタフェース接続	動的登録（シングルスレッド用）	libcltxa.a
	静的登録又は動的登録（シングルスレッド用）※	libcltya.a
複数接続機能	DCE スレッド	libcltm.a
	カーネルスレッド	libcltk.a
	シングルスレッド	libclts.a

注※

TM に登録するスイッチ名称で，静的登録又は動的登録のどちらかに切り替えられます。

共用ライブラリの用途別の使用ファイルを次の表に示します。

表 6–13　共用ライブラリの用途別の使用ファイル（UNIX クライアント）

用途			使用するファイル
通常の UAP			libzclt.sl
XA インタフェース接続	動的登録	シングルスレッド用	libzcltx.sl libzcltx64.sl libzcltxs.sl（複数接続機能を使用する場合） libzcltxs64.sl（複数接続機能を使用する場合）
		マルチスレッド用	libzcltxk.sl
	静的登録又は動的登録※	シングルスレッド用	libzclt6ys.sl（複数接続機能を使用する場合） libzclt6ys64.sl（複数接続機能を使用する場合） libzclty.sl libzclty64.sl libzcltys.sl（複数接続機能を使用，又は TUXEDO 対応の場合）

用途			使用するファイル
			libzcltys64.sl（複数接続機能を使用する場合）
		マルチスレッド用	libzclt6yk.sl libzclt6yk64.sl libzcltyk.sl libzcltyk64.sl
複数接続機能	DCE スレッド		libzcltm.sl
	カーネルスレッド		libzcltk.sl
	シングルスレッド		libzclts.sl
ODBC 接続			libzpdodbc.sl
SQL 補助関数用			libsqlauxf.sl

注

共用ライブラリのサフィックスは，プラットフォームによって異なります。Linux の場合は「.so」，AIX の場合は「.a」となります。

注※

TM に登録するスイッチ名称で，静的登録又は動的登録のどちらかに切り替えられます。

●各トランザクションマネジャが使用するライブラリ

各トランザクションマネジャが使用するライブラリ一覧を次の表に示します。

表 6–14　各トランザクションマネジャが使用するライブラリ一覧（UNIX クライアント）

トランザクションマネジャ	ライブラリ名	バックエンドサーバ接続保持機能
OpenTP1	libzclt6ys.sl	○
	libzclt6ys64.sl	○
	libzcltx.sl	○
	libzclty.sl	○
	libzcltxs.sl	○
	libzcltys.sl	○
	libzcltx64.sl	○
	libzclty64.sl	○
	libzcltxs64.sl	○
	libzcltys64.sl	○
TPBroker for C++	libzcltxk.sl	×
	libzcltyk.sl	×

トランザクションマネジャ	ライブラリ名	バックエンドサーバ接続保持機能
TUXEDO	libzcltys.sl	○
TP1/EE	libzclt6yk.sl	○
	libzclt6yk64.sl	○
	libzcltyk.sl	○
	libzcltyk64.sl	○

（凡例）

○：バックエンドサーバ接続保持機能を使用できます。

×：バックエンドサーバ接続保持機能を使用できません。

注

共用ライブラリのサフィックスは，プラットフォームによって異なります。Linux の場合は「.so」，AIX の場合は「.a」となります。

6.4.2 Windows クライアントのディレクトリ及びファイル構成

インストール時に自動的に作成されるファイルとディレクトリを次の表に示します。

- 表「HiRDB/Developer's Kit の場合のファイルとディレクトリ（Windows (x86)版のクライアント）」
- 表「HiRDB/Run Time の場合のファイルとディレクトリ（Windows (x86)版のクライアント）」
- 表「HiRDB/Developer's Kit の場合のファイルとディレクトリ（Windows (x64)版のクライアント）」
- 表「HiRDB/Run Time の場合のファイルとディレクトリ（Windows (x64)版のクライアント）」
- 表「ODBC ドライバの場合のファイルとディレクトリ（Windows クライアント）」

表 6–15 HiRDB/Developer's Kit の場合のファイルとディレクトリ（Windows (x86)版のクライアント）

名　　称	ディレクトリ	ファイル名
ヘッダファイル	xxxx¥INCLUDE	PDBTYPES.H
		PDBERRNO.H
		PDBMISC.H
		SQLCA.CBL
		SQLIOA.CBL
		PDBMISCM.H
		SQLDA.CBL
		PDBSQLDA.H

名　　称	ディレクトリ	ファイル名
		PDBSQLCSNA.H
		SQLIOAD.CBL
		SQLCAD.CBL
		PDDBHASH.H
		PDAUXCNV.H
		SQLIOAMTH.CBL
		SQLCAMTH.CBL
		SQLCSNA.CBL
		SQLIOA32.CBL
		SQLCNSTS.CBL
		SQLCNSTD.CBL
		SQLMCNCT.CBL
メッセージオブジェクトファイル	xxxx¥LIB	msgtxt
リンケージ用ライブラリ	xxxx¥LIB	CLTDLL.LIB
		PDCLTM32.LIB
		PDCLTM50.LIB
		PDCLTX32.LIB
		PDCLTXM.LIB
		PDSQLAUXF.LIB
		PDCLTXS.LIB
		PDCLTXM5.LIB
		PDCLTM71.LIB
		PDCLTM80S.LIB
		PDCLTM90.LIB
		PDCLTM100.LIB
		PDCLTM140.LIB
		PDCLTM140T.LIB
コマンド・ユティリティ	xxxx¥UTL	PDCPP.EXE
		PDOCC.EXE
		PDCBL.EXE
		PDOCB.EXE

名　　称	ディレクトリ	ファイル名
		PDPREP.EXE
		PDPREP7.EXE
		PDPREP8.EXE
		PDPREPA.EXE
		PDPREPC.EXE
		PDPREPG.EXE
		PDTRCMGR.EXE
		PDCLTADM.EXE
DLL ファイル	xxxx¥UTL	CLTDLL.DLL
		PDCLTM32.DLL
		PDCLTM50.DLL
		PDCLTX32.DLL
		PDCLTXM.DLL
		PDSQLAUXF.DLL
		PDSQLAUXF71.DLL
		PDOLEDB.DLL
		PDCLTXS.DLL
		PDCLTXM5.DLL
		PDCLTM71.DLL
		PDCLTM80S.DLL
		PDCLTM90.DLL
		PDCLTM100.DLL
		PDCLTM140.DLL
		PDCLTM140T.DLL
		PDCLTSOCRYPTO.DLL
		PDCLTSOCRYPTO_R.DLL
JDBC ドライバ	xxxx¥UTL	JJDBC.DLL
		PDJDBC.JAR
		PDJDBC2.JAR
		PDJDBC4.JAR
		PDJDBC43.JAR

名　　称	ディレクトリ	ファイル名
SQLJ	xxxx¥UTL	PDSQLJ.JAR
		PDRUNTIME.JAR
		PDNATIVERT.JAR
		PDPARSE.DLL
		PDJAVA.EXE
		PDSQLJN.DLL
HiRDB データプロバイダ for .NET Framework	xxxx¥UTL 及び¥Windows¥assembly	PDDNDP.DLL
		PDDNDPCORE.DLL
		PDDNDP20.DLL
		PDDNDPCORE20.DLL
	xxxx¥UTL 及び ¥Windows¥Microsoft.NET¥assembly	PDDNDP40.DLL
		PDDNDPCORE40.DLL
HiRDB データプロバイダ for .NET Framework 用発行者ポリシー	¥Windows¥assembly	policy.vv.rr.pddndp.dll※1
		policy.vv.rr.pddndpcore.dll※1
		policy.vv.rr.pddndp20.dll※1
		policy.vv.rr.pddndpcore20.dll※1
	¥Windows¥Microsoft.NET¥assembly	policy.vv.rr.pddndp40.dll※1
		policy.vv.rr.pddndpcore40.dll※1
XML 変換コマンド※2	xxxx¥UTL	phdxmlcnv.bat
XML 変換ライブラリ※2	xxxx¥UTL	XMLConverter.jar
ODBC ドライバ	¥Windows¥system32	pdodbcdrv3.dll
		pdodbstp3.dll
		pdclto32.dll
インタフェース定義ファイル	xxxx¥LIB	HIRDB.PKG
サンプル	xxxx¥SAMPLEAP	CREATE.EC
		SAMPLE1.EC
		SAMPLE2.EC
		SAMPLE3.EC
		SAMPLE1.ECB
		INPUTF1

名　　称	ディレクトリ	ファイル名
		INPUTF2
README ファイル	xxxx	README.TXT
環境定義ファイル	¥Windows	HIRDB.INI

注 1

xxxx は HiRDB のインストールディレクトリ名を示します。ディレクトリ名はインストール時に指定できます。また，¥Windows はシステムディレクトリを示します。

注 2

再配布ファイル，及びインストーラ用の管理ファイルは含まれていません。

注※1

HiRDB データプロバイダ for .NET Framework 用発行者ポリシーは，バージョン 09-01 以降のリビジョンごとのファイルを累積します。

vv はバージョン番号，rr はリビジョン番号を示します。vv，rr ともにゼロサプレスします。

注※2

HiRDB XML Extension に同梱されるため，HiRDB サーバ製品中の HiRDB クライアントには同梱されません。

表 6–16　HiRDB/Run Time の場合のファイルとディレクトリ（Windows (x86)版のクライアント）

名　　称	ディレクトリ	ファイル名
リンケージ用ライブラリ	xxxx¥LIB	PDCLTX32.LIB
		PDCLTXM.LIB
		PDCLTXS.LIB
		PDCLTXM5.LIB
コマンド・ユティリティ	xxxx¥UTL	PDTRCMGR.EXE
		PDCLTADM.EXE
DLL ファイル	xxxx¥UTL	CLTDLL.DLL
		PDCLTM32.DLL
		PDCLTM50.DLL
		PDCLTP32.DLL
		PDCLTX32.DLL
		PDSQLAUXF.DLL
		PDSQLAUXF71.DLL
		PDCLTXM.DLL

名　　称	ディレクトリ	ファイル名
		PDOLEDB.DLL
		PDCLTXS.DLL
		PDCLTXM5.DLL
		PDCLTM71.DLL
		PDCLTM80S.DLL
		PDCLTM90.DLL
		PDCLTM100.DLL
		PDCLTM140.DLL
		PDCLTM140T.DLL
		PDCLTSOCRYPTO.DLL
		PDCLTSOCRYPTO_R.DLL
JDBC ドライバ	xxxx¥UTL	JJDBC.DLL
		PDJDBC.JAR
		PDJDBC2.JAR
		PDJDBC4.JAR
		PDJDBC43.JAR
SQLJ	xxxx¥UTL	PDRUNTIME.JAR
		PDNATIVERT.JAR
		PDSQLJN.DLL
HiRDB データプロバイダ for .NET Framework	xxxx¥UTL 及び¥Windows¥assembly	PDDNDP.DLL
		PDDNDPCORE.DLL
		PDDNDP20.DLL
		PDDNDPCORE20.DLL
	xxxx¥UTL 及び ¥Windows¥Microsoft.NET¥assembly	PDDNDP40.DLL
		PDDNDPCORE40.DLL
HiRDB データプロバイダ for .NET Framework 用発行者ポリシー	¥Windows¥assembly	policy.vv.rr.pddndp.dll※1
		policy.vv.rr.pddndpcore.dll※1
		policy.vv.rr.pddndp20.dll※1
		policy.vv.rr.pddndpcore20.dll※1
	¥Windows¥Microsoft.NET¥assembly	policy.vv.rr.pddndp40.dll※1

名　　称	ディレクトリ	ファイル名
		policy.vv.rr.pddndpcore40.dll※1
XML 変換コマンド※2	xxxx¥UTL	phdxmlcnv.bat
XML 変換ライブラリ※2	xxxx¥UTL	XMLConverter.jar
ODBC ドライバ	¥Windows¥system32	pdodbcdrv3.dll
		pdodbstp3.dll
		pdclto32.dll
README ファイル	xxxx	README.TXT
環境定義ファイル	¥WINDOWS	HIRDB.INI

注 1

xxxx は HiRDB のインストールディレクトリ名を示します。ディレクトリ名はインストール時に指定できます。また，¥ Windows はシステムディレクトリを示します。

注 2

再配布ファイル，及びインストーラ用の管理ファイルは含まれていません。

注※1

HiRDB データプロバイダ for .NET Framework 用発行者ポリシーは，バージョン 09-01 以降のリビジョンごとのファイルを累積します。

vv はバージョン番号，rr はリビジョン番号を示します。vv，rr ともにゼロサプレスします。

注※2

HiRDB XML Extension に同梱されるため，HiRDB サーバ製品中の HiRDB クライアントには同梱されません。

表 6–17　HiRDB/Developer's Kit の場合のファイルとディレクトリ（Windows (x64)版のクライアント）

名　　称	ディレクトリ	ファイル名
ヘッダファイル	xxxx¥INCLUDE	PDBTYPES.H
		PDBERRNO.H
		PDBMISC.H
		PDBMISCM.H
		SQLDA.CBL
		SQLDA64.CBL
		PDBSQLDA.H
		PDBSQLCSNA.H

名　　称	ディレクトリ	ファイル名
		SQLIOA.CBL
		SQLIOA64.CBL
		SQLCA.CBL
		SQLCA64.CBL
		SQLIOAD.CBL
		SQLIOAD64.CBL
		SQLCAD.CBL
		SQLCAD64.CBL
		PDDBHASH.H
		PDAUXCNV.H
		SQLIOAMTH.CBL
		SQLIOAMTH64.CBL
		SQLCAMTH.CBL
		SQLCAMTH64.CBL
		SQLCSNA.CBL
		SQLCSNA64.CBL
		SQLIOA32.CBL
		SQLIOALL.CBL
		SQLCNSTS.CBL
		SQLCNSTD.CBL
		SQLMCNCT.CBL
リンケージ用ライブラリ	xxxx¥LIB	CLTDLL.LIB
		PDCLTM32.LIB
		PDCLTM50.LIB
		PDCLTM64.LIB
		PDCLTM90X.LIB
		PDCLTM100X.LIB
		PDCLTM140X.LIB
		PDCLTX32.LIB
		PDCLTX64.LIB
		PDCLTXM.LIB

名　　称	ディレクトリ	ファイル名
		PDSQLAUXF.LIB
		PDSQLAUXF64.LIB
		PDCLTXS.LIB
		PDCLTXS64.LIB
		PDCLTXM5.LIB
		PDCLTXM64.LIB
		PDCLTM71.LIB
		PDCLTM80S.LIB
		PDCLTM90.LIB
		PDCLTM100.LIB
		PDCLTM140.LIB
		PDCLTM140T.LIB
コマンド・ユティリティ	xxxx¥UTL	PDCPP.EXE
		PDOCC.EXE
		PDCBL.EXE
		PDOCB.EXE
		PDPREP.EXE
		PDPREP7.EXE
		PDPREP8.EXE
		PDPREPA.EXE
		PDPREPC.EXE
		PDPREPG.EXE
		PDJAVA.EXE
		PDTRCMGR.EXE
		PDTRCMGR32.EXE
		PDCLTADM.EXE
		PDCLTADM32.EXE
DLL ファイル	xxxx¥UTL	CLTDLL.DLL
		PDCLTM32.DLL
		PDCLTM50.DLL
		PDCLTM64.DLL

名　　称	ディレクトリ	ファイル名
		PDCLTM90X.DLL
		PDCLTM100X.DLL
		PDCLTM140X.DLL
		PDCLTM71.DLL
		PDCLTM80S.DLL
		PDCLTM90.DLL
		PDCLTM100.DLL
		PDCLTM140.DLL
		PDCLTM140T.DLL
		PDCLTX32.DLL
		PDCLTX64.DLL
		PDCLTXM.DLL
		PDCLTXM64.DLL
		PDOLEDB.DLL
		PDSQLAUXF.DLL
		PDSQLAUXF64.DLL
		PDPARSE.DLL
		PDCLTXS.DLL
		PDCLTXS64.DLL
		PDCLTSOCRYPTO.DLL
		PDCLTSOCRYPTO_R.DLL
		PDCLTSOCRYPTOX.DLL
		PDCLTSOCRYPTO_RX.DLL
JDBC ドライバ	xxxx¥UTL	JJDBC.DLL
		PDJDBC.JAR
		PDJDBC2.JAR
		PDJDBC4.JAR
		PDJDBC43.JAR
SQLJ	xxxx¥UTL	PDSQLJ.JAR
		PDRUNTIME.JAR
		PDNATIVERT.JAR

名　　称	ディレクトリ	ファイル名
		PDSQLJN.DLL
HiRDB データプロバイダ for .NET Framework	xxxx¥UTL 及び¥Windows¥assembly	PDDNDP.DLL
		PDDNDPCORE.DLL
		PDDNDP20.DLL
		PDDNDPCORE20.DLL
		PDDNDP20x.DLL
		PDDNDPCORE20x.DLL
	xxxx¥UTL 及び ¥Windows¥Microsoft.NET¥assembly	PDDNDP40.DLL
		PDDNDPCORE40.DLL
		PDDNDP40x.DLL
		PDDNDPCORE40x.DLL
HiRDB データプロバイダ for .NET Framework 用発行者ポリシー	¥Windows¥assembly	policy.vv.rr.pddndp.dll※1
		policy.vv.rr.pddndpcore.dll※1
		policy.vv.rr.pddndp20.dll※1
		policy.vv.rr.pddndpcore20.dll※1
		policy.vv.rr.pddndp20x.dll※1
		policy.vv.rr.pddndpcore20x.dll※1
	¥Windows¥Microsoft.NET¥assembly	policy.vv.rr.pddndp40.dll※1
		policy.vv.rr.pddndpcore40.dll※1
		policy.vv.rr.pddndp40x.dll※1
		policy.vv.rr.pddndpcore40x.dll※1
XML 変換コマンド※2	xxxx¥UTL	phdxmlcnv.bat
XML 変換ライブラリ※2	xxxx¥UTL	XMLConverter.jar
ODBC3.5 ドライバ	¥Windows¥SysWOW64	pdodbcdrv3.dll
		pdodbstp3.dll
		pdclto32.dll
	¥Windows¥system32	pdodbcdrv3x.dll
		pdodbstp3x.dll
		pdclto64.dll
インタフェース定義ファイル	xxxx¥BIN	HIRDB.PKG

名　　称	ディレクトリ	ファイル名
サンプル	xxxx¥SAMPLEAP	CREATE.EC
		SAMPLE1.EC
		SAMPLE2.EC
		SAMPLE3.EC
		SAMPLE1.ECB
		INPUTF1
		INPUTF2
README ファイル	xxxx	README.TXT
環境定義ファイル	¥Windows	HIRDB.INI
メッセージオブジェクトファイル	xxxx¥LIB	msgtxt

注 1

xxxx は HiRDB のインストールディレクトリ名を示します。ディレクトリ名はインストール時に指定できます。また，¥Windows はシステムディレクトリを示します。

注 2

再配布ファイル，及びインストーラ用の管理ファイルは含まれていません。

注※1

HiRDB データプロバイダ for .NET Framework 用発行者ポリシーは，バージョン 09-01 以降のリビジョンごとのファイルを累積します。

vv はバージョン番号，rr はリビジョン番号を示します。vv，rr ともにゼロサプレスします。

注※2

HiRDB XML Extension に同梱されるため，HiRDB サーバ製品中の HiRDB クライアントには同梱されません。

表 6–18　HiRDB/Run Time の場合のファイルとディレクトリ（Windows (x64)版のクライアント）

名　　称	ディレクトリ	ファイル名
リンケージ用ライブラリ	xxxx¥LIB	PDCLTX32.LIB
		PDCLTX64.LIB
		PDCLTXM.LIB
		PDCLTXS.LIB
		PDCLTXS64.LIB
		PDCLTXM5.LIB
		PDCLTXM64.LIB

名　　称	ディレクトリ	ファイル名
コマンド・ユティリティ	xxxx¥UTL	PDTRCMGR.EXE
		PDTRCMGR32.EXE
		PDCLTADM.EXE
		PDCLTADM32.EXE
DLL ファイル	xxxx¥UTL	CLTDLL.DLL
		PDCLTM32.DLL
		PDCLTM50.DLL
		PDCLTM64.DLL
		PDCLTM90X.DLL
		PDCLTM100X.DLL
		PDCLTM140X.DLL
		PDCLTX32.DLL
		PDCLTX64.DLL
		PDCLTM71.DLL
		PDCLTM80S.DLL
		PDCLTM90.DLL
		PDCLTM100.DLL
		PDCLTM140.DLL
		PDCLTM140T.DLL
		PDCLTXM.DLL
		PDCLTXM64.DLL
		PDOLEDB.DLL
		PDSQLAUXF.DLL
		PDSQLAUXF64.DLL
		PDPARSE.DLL
		PDCLTXS.DLL
		PDCLTXS64.DLL
		PDCLTSOCRYPTO.DLL
		PDCLTSOCRYPTO_R.DLL
		PDCLTSOCRYPTOX.DLL
		PDCLTSOCRYPTO_RX.DLL

名　　称	ディレクトリ	ファイル名
JDBC ドライバ	xxxx¥UTL	JJDBC.DLL
		PDJDBC.JAR
		PDJDBC2.JAR
		PDJDBC4.JAR
		PDJDBC43.JAR
SQLJ ランタイム	xxxx¥UTL	PDRUNTIME.JAR
		PDNATIVERT.JAR
		PDSQLJN.DLL
HiRDB データプロバイダ for .NET Framework	xxxx¥UTL 及び¥Windows¥assembly	PDDNDP.DLL
		PDDNDPCORE.DLL
		PDDNDP20.DLL
		PDDNDPCORE20.DLL
		PDDNDP20x.DLL
		PDDNDPCORE20x.DLL
	xxxx¥UTL 及び ¥Windows¥Microsoft.NET¥assembly	PDDNDP40.DLL
		PDDNDPCORE40.DLL
		PDDNDP40x.DLL
		PDDNDPCORE40x.DLL
HiRDB データプロバイダ for .NET Framework 用発行者ポリシー	¥Windows¥assembly	policy.vv.rr.pddndp.dll※1
		policy.vv.rr.pddndpcore.dll※1
		policy.vv.rr.pddndp20.dll※1
		policy.vv.rr.pddndpcore20.dll※1
		policy.vv.rr.pddndp20x.dll※1
		policy.vv.rr.pddndpcore20x.dll※1
	¥Windows¥Microsoft.NET¥assembly	policy.vv.rr.pddndp40.dll※1
		policy.vv.rr.pddndpcore40.dll※1
		policy.vv.rr.pddndp40x.dll※1
		policy.vv.rr.pddndpcore40x.dll※1
XML 変換コマンド※2	xxxx¥UTL	phdxmlcnv.bat
XML 変換ライブラリ※2	xxxx¥UTL	XMLConverter.jar

名　　称	ディレクトリ	ファイル名
ODBC3.5 ドライバ	¥Windows¥SysWOW64	pdodbcdrv3.dll
		pdodbstp3.dll
		pdclto32.dll
	¥Windows¥system32	pdodbcdrv3x.dll
		pdodbstp3x.dll
		pdclto64.dll
README ファイル	xxxx	README.TXT
環境定義ファイル	¥Windows	HIRDB.INI

注 1

xxxx は HiRDB のインストールディレクトリ名を示します。ディレクトリ名はインストール時に指定できます。また，¥Windows はシステムディレクトリを示します。

注 2

再配布ファイル，及びインストーラ用の管理ファイルは含まれていません。

注※1

HiRDB データプロバイダ for .NET Framework 用発行者ポリシーは，バージョン 09-01 以降のリビジョンごとのファイルを累積します。

vv はバージョン番号，rr はリビジョン番号を示します。vv，rr ともにゼロサプレスします。

注※2

HiRDB XML Extension に同梱されるため，HiRDB サーバ製品中の HiRDB クライアントには同梱されません。

表 6–19　ODBC ドライバの場合のファイルとディレクトリ（Windows クライアント）

名　　称	ディレクトリ	ファイル名
セットアップファイル	¥Windows	DRVSETUP.EXE※
		DRVSTP32.EXE
セットアップ用 DLL		HIRDBSTP.DLL※
		HRDSTP32.DLL
ドライバ本体		PDODBDRV.DLL※
		PDODBD32.DLL
HiRDB/ClientDLL		PDCLTLIB.DLL※
		PDCLTL32.DLL

注

¥Windows は，システムディレクトリを示します。

注※

EM64T マシンの Windows クライアントでは作成されません。

●リンケージ用ライブラリの用途別の使用ファイル

リンケージ用ライブラリの用途別の使用ファイルを次の表に示します。

表 6–20　リンケージ用ライブラリの用途別の使用ファイル（Windows クライアント）

用途		使用するファイル	HiRDB クライアントでのプラットフォームごとのライブラリ提供状況		アドレッシングモード
			Windows (x86)	Windows (x64)	
通常の UAP		cltdll.dll	○	○	32 ビット
XA インタフェース接続（静的登録又は動的登録）※	シングルスレッド用	pdcltx32.dll	○	○	32 ビット
		pdcltxs.dll	○	○	32 ビット
		pdcltx64.dll	−	○	64 ビット
		pdcltxs64.dll	−	○	64 ビット
	マルチスレッド用	pdcltxm.dll	○	○	32 ビット
		pdcltxm64.dll	−	○	64 ビット
複数接続機能（マルチスレッド用）		pdcltm32.dll	○	○	32 ビット
		pdcltm50.dll	○	○	32 ビット
		pdcltm71.dll	○	○	32 ビット
		pdcltm80s.dll	○	○	32 ビット
		pdcltm90.dll	○	○	32 ビット
		pdcltm100.dll	○	○	32 ビット
		pdcltm140.dll	○	○	32 ビット
		pdcltm64.dll	−	○	64 ビット
		pdcltm90x.dll	−	○	64 ビット
		pdcltm100x.dll	−	○	64 ビット
		pdcltm140x.dll	−	○	64 ビット
SQL 補助関数用		pdsqlauxf.dll	○	○	32 ビット
		pdsqlauxf71.dll	○	○	32 ビット
		pdsqlauxf64.dll	−	○	64 ビット

（凡例）

○：サポートしています。

×：サポートしていません。

−：該当しません。

注※

TM に登録するスイッチ名称で，静的登録又は動的登録のどちらかに切り替えられます。

●各トランザクションマネジャが使用するライブラリ

各トランザクションマネジャが使用するライブラリ一覧を次の表に示します。

表 6–21　各トランザクションマネジャが使用するライブラリ一覧（Windows クライアント）

トランザクションマネジャ	ライブラリ名	バックエンドサーバ接続保持機能	HiRDB クライアントでのプラットフォームごとのライブラリ提供状況		アドレッシングモード
			Windows (x86)	Windows (x64)	
OpenTP1	pdcltx32.dll	可	○	○	32 ビット
	pdcltxs.dll	可	○	○	32 ビット
	pdcltx64.dll	可	×	○	64 ビット
	pdcltxs64.dll	可	×	○	64 ビット
TPBroker for C++	pdcltxm.dll	否	○	○	32 ビット
TUXEDO	pdcltxs.dll	可	○	○	32 ビット

（凡例）

可：バックエンドサーバ接続保持機能を使用できます。

否：バックエンドサーバ接続保持機能を使用できません。

○：サポートしています。

×：サポートしていません。

●ライブラリと作成コンパイラの一覧

ライブラリと作成コンパイラの一覧を次の表に示します。

表 6–22　ライブラリと作成コンパイラの一覧（Windows クライアント）

プラットフォーム	ライブラリ名	作成コンパイラのバージョン	使用する VisualC ランタイム
Windows (x86)	cltdll.dll	VisualC++2.0	マルチスレッドスタティック
	pdsqlauxf.dll		
	pdcltm32.dll	VisualC++4.2	マルチスレッド DLL
	pdcltx32.dll		
	pdcltxm.dll		
	pdcltxs.dll		
	pdcltm50.dll	VisualC++5.0	
	pdcltxm5.dll		

プラットフォーム	ライブラリ名	作成コンパイラのバージョン	使用する VisualC ランタイム
	pdcltm71.dll	Visual Studio .NET 2003	マルチスレッドスタティック
	pdsqlauxf71.dll		
	jjdbcinter.dll	Visual Studio 2003	
	pdcltm80s.dll	Visual Studio 2005	
	pdcltm90.dll	Visual Studio 2008	マルチスレッド DLL
	pdcltm100.dll	Visual Studio 2010	
	pdcltm140.dll	Visual Studio 2017	
Windows (x64)	pdcltm64.dll	Visual Studio 2005	マルチスレッド DLL
	pdcltx64.dll		
	pdcltxs64.dll		
	pdcltxm64.dll		
	pdsqlauxf64.dll		マルチスレッドスタティック
	pdcltm90x.dll	Visual Studio 2008	マルチスレッド DLL
	pdcltm100x.dll	Visual Studio 2010	
	pdcltm140x.dll	Visual Studio 2017	

6.5 hosts ファイルの設定

6.5.1 hosts ファイルの設定方法

クライアントマシンとサーバマシンが異なる場合，クライアントマシンの hosts ファイルに，次に示す情報を指定します。

- IP アドレス
- ホスト名

Linux の場合，hosts ファイルの 1 行の長さは 1500 文字以内としてください。

DNS を利用する場合は，hosts ファイルを設定する必要はありません。

また，hosts ファイルに標準ホストを指定しない場合は，クライアント環境定義の PDCLTRCVADDR の指定が必要となります。

(1) HiRDB/シングルサーバの場合

- IP アドレス
  HiRDB/シングルサーバの IP アドレスを指定します。
- ホスト名
  HiRDB/シングルサーバのホスト名を指定します。
- **IP アドレスを引き継がない系切り替えをする場合**
  実行系及び待機系の両方の IP アドレスとホスト名を指定します。

(2) HiRDB/パラレルサーバの場合

- IP アドレス
  システムマネジャ，及びフロントエンドサーバを定義したサーバマシンの IP アドレスを指定します。
- ホスト名
  システムマネジャ，及びフロントエンドサーバを定義したサーバマシンのホスト名を指定します。
- **IP アドレスを引き継がない系切り替えをする場合**
  実行系及び待機系の両方の IP アドレスとホスト名を指定します。

6.6　クライアント環境定義（環境変数の設定）

6.6.1　クライアント環境定義の設定形式

UAP を実行するためには，クライアントごとにクライアント環境定義を設定しておく必要があります。

(1)　UNIX 環境の場合

コマンド，及びユティリティを実行するために，環境変数 PATH に次のディレクトリを追加してください。

クライアントのサーバマシンで実行する場合：

/opt/HiRDB/client/utl/

HiRDB サーバにリモートログインして実行する場合：

$PDDIR/client/utl/

- **クライアント環境定義の検索順序**

  クライアント環境定義を複数の箇所に設定している場合，次の順序でクライアント環境定義ごとに検索し，指定値がないクライアント環境定義については，デフォルト値を適用します。

  1. 環境変数グループ※
  2. ユーザ環境変数

  注※

  複数接続機能使用時に ALLOCATE CONNECTION HANDLE でファイル名を指定します。また，OLTP 下の UAP をクライアントとする場合，オープン文字列にファイル名を指定します。オープン文字列については，マニュアル「HiRDB システム導入・設計ガイド」を参照してください。

  使用するクライアント機能によっては，上記以外の方法でもクライアント環境定義を設定できます。検索順序の詳細は，次の箇所を参照してください。

  ＜ HiRDB ODBC ドライバの場合＞

  「接続情報の優先順位」

  ＜ Type4 JDBC ドライバの場合＞

  「接続情報の優先順位」

(a)　sh（ボーンシェル）の場合

.profile ファイルに次の環境変数を格納してください。ファイルに格納すると，起動時に環境変数が自動的に実行されます。

```
$ PDHOST=HiRDBサーバのホスト名[,予備系HiRDBサーバのホスト名]
$ PDNAMEPORT=HiRDBサーバのポート番号
$ PDFESHOST=フロントエンドサーバのホスト名
          [:フロントエンドサーバがあるユニットのポート番号]
```

```
                [,予備系フロントエンドサーバのホスト名
                [:予備系フロントエンドサーバがあるユニットのポート番号]]
$ PDSERVICEGRP=サーバ名
$ PDSRVTYPE={WS｜PC}
$ PDSERVICEPORT=高速接続用のポート番号
                      [,予備系の高速接続用のポート番号]
$ PDFESGRP=FESグループ[,切替FESグループ[,切替FESグループ]…]
$ PDCLTRCVPORT=クライアントの受信ポート番号
$ PDCLTRCVADDR={クライアントのIPアドレス｜クライアントのホスト名}
$ PDTMID=OLTP識別子
$ PDXAMODE={0｜1}
$ PDTXACANUM=1UAP当たりのトランザクション最大同時実行数
$ PDXARCVWTIME=トランザクションが回復できない場合の待ち合わせ時間
$ PDXATRCFILEMODE={LUMP｜SEPARATE}
$ PDXAAUTORECONNECT={YES｜NO}
$ HiRDB_PDHOST=HiRDBサーバのホスト名[,予備系HiRDBサーバのホスト名]
$ HiRDB_PDNAMEPORT=HiRDBサーバのポート番号
$ HiRDB_PDTMID=OLTP識別子
$ HiRDB_PDXAMODE={0｜1}
$ PDUSER=[認可識別子/パスワード]
$ PDCLTAPNAME=実行するUAPの識別名称
$ PDCLTLANG={SJIS｜CHINESE｜UJIS｜C｜UTF-8｜CHINESE-GB18030}
$ PDLANG= {UTF-8｜SJIS｜CHINESE｜CHINESE-GB18030｜ANY}
$ PDDBLOG={ALL｜NO}
$ PDEXWARN={YES｜NO}
$ PDSUBSTRLEN={3｜4｜5｜6｜7｜8｜9｜10}
$ PDCLTCNVMODE={AUTO｜NOUSE｜UJIS｜UJIS2｜UTF8｜UTF8MS
                 ｜UTF8_TXT｜UTF8_EX｜UTF8_EX2｜UTF8MS_TXT
                 ｜UCS2_UJIS｜UCS2_UJIS2｜UCS2_UTF8}
$ PDCLTGAIJIDLL=ユーザ定義外字変換DLLファイル名
$ PDCLTGAIJIFUNC=ユーザ定義外字変換関数名
$ PDCLTGRP=クライアントグループ名
$ PDAUTORECONNECT={YES｜NO}
$ PDRCCOUNT=自動再接続機能でのCONNECTのリトライ回数
$ PDRCINTERVAL=自動再接続機能でのCONNECTのリトライ間隔
$ PDRCTIMING={ALL｜再接続契機[,再接続契機[,再接続契機]]}
$ PDUAPENVFILE=UAP環境定義のファイル名
$ PDDBBUFLRU={YES｜NO}
$ PDHATRNQUEUING=NO
$ PDEXTDECCHECK={YES｜NO}
$ PDASTHOST=HiRDB Control Manager - Agentのホスト名
           [,予備系HiRDB Control Manager - Agentのホスト名]
$ PDASTPORT=HiRDB Control Manager - Agentのポート番号
$ PDSYSTEMID=HiRDB Control Manager - Agentが管理するHiRDBサーバのHiRDB識別子
$ PDASTUSER=OSのユーザ名/パスワード
$ PDCMDWAITTIME=コマンド実行時のクライアントの最大待ち時間
$ PDCMDTRACE=コマンドトレースファイルのサイズ
$ PDIPC={MEMORY｜DEFAULT}
$ PDSENDMEMSIZE=クライアント側のデータ送信用メモリサイズ
$ PDRECVMEMSIZE=クライアント側のデータ受信用メモリサイズ
$ PDCWAITTIME=クライアントの最大待ち時間
$ PDSWAITTIME=トランザクション処理中のサーバの最大待ち時間
$ PDSWATCHTIME=トランザクション処理以外のサーバの最大待ち時間
$ PDCWAITTIMEWRNPNT=SQL実行時間警告出力の契機
$ PDKALVL={0｜1｜2｜3}
$ PDKATIME=パケットの送信間隔
$ PDTIMEDOUTRETRY=リトライ回数
$ PDNBLOCKWAITTIME=ノンブロックモードでのコネクション確立監視時間
```

```
$ PDCONNECTWAITTIME=サーバ接続時のHiRDBクライアントの最大待ち時間
$ PDCLTPATH=トレースファイル格納ディレクトリ
$ PDSQLTRACE=SQLトレースファイルのサイズ
$ PDUAPERLOG=クライアントエラーログファイルのサイズ
$ PDERRSKIPCODE=SQLCODE[,SQLCODE]…
$ PDPRMTRC= {YES｜NO｜IN｜OUT｜INOUT}
$ PDPRMTRCSIZE=SQLトレースに出力するパラメタ情報の最大データ長
$ PDTRCMODE={ERR｜NONE}
$ PDUAPREPLVL= {[s][u[o][t]][p][r]｜[a[o][t]]}
$ PDREPPATH=UAP統計レポートファイルの格納ディレクトリ
$ PDTRCPATH=動的SQLトレースファイルの格納先ディレクトリ
$ PDSQLTRCOPENMODE={CNCT｜SQL}
$ PDSQLTEXTSIZE=SQL文のサイズ
$ PDSQLEXECTIME={YES｜NO}
$ PDRCTRACE=再接続トレースファイルのサイズ
$ PDWRTLNPATH=WRITE LINE文の値式の値を出力するファイルの
              格納先ディレクトリ
$ PDWRTLNFILSZ=WRITE LINE文の値式の値を出力するファイルの
                    最大サイズ
$ PDWRTLNCOMSZ=WRITE LINE文の値式の値の合計サイズ
$ PDUAPEXERLOGUSE={YES｜NO}
$ PDUAPEXERLOGPRMSZ=パラメタ情報の最大データ長
$ PDSQLTRCFMT={1|2}
$ PDVWOPTMODE={0｜1｜2}
$ PDTAAPINFPATH=アクセスパス情報ファイル出力ディレクトリ名
$ PDTAAPINFMODE={0｜1}
$ PDTAAPINFSIZE=アクセスパス情報ファイルサイズ
$ PDSTJTRNOUT={YES｜NO}
$ PDLOCKLIMIT=ユーザ当たりの最大排他資源要求数
$ PDDLKPRIO={96｜64｜32}
$ PDLOCKSKIP={YES｜NO}
$ PDFORUPDATEEXLOCK={YES｜NO}
$ PDISLLVL=データ保証レベル
$ PDSQLOPTLVL=SQL最適化オプション[,SQL最適化オプション]…
$ PDADDITIONALOPTLVL=SQL拡張最適化オプション
                        [,SQL拡張最適化オプション]…
$ PDHASHTBLSIZE=ハッシュジョイン，副問合せのハッシュ実行適用時の
                ハッシュ表サイズ
$ PDDFLNVAL={USE｜NOUSE}
$ PDAGGR=グループ分けのときに発生するグループ数
$ PDCMMTBFDDL={YES｜NO}
$ PDPRPCRCLS={YES｜NO}
$ PDAUTOCONNECT={ON｜OFF}
$ PDDDLDEAPRPEXE={YES｜NO}
$ PDDDLDEAPRP={YES｜NO}
$ PDLCKWAITTIME=排他待ち限界経過時間
$ PDCURSORLVL={0｜1｜2}
$ PDDELRSVWDFILE=SQL予約語削除ファイル名
$ PDCALCMDWAITTIME=CALL COMMAND文の最大待ち時間
$ PDSTANDARDSQLSTATE={YES｜NO}
$ PDBLKF=ブロック転送の行数
$ PDBINARYBLKF={YES｜NO}
$ PDBLKBUFFSIZE=通信バッファサイズ
$ PDBINDRETRYCOUNT=bindシステムコールのリトライ回数
$ PDBINDRETRYINTERVAL=bindシステムコールのリトライ間隔
$ PDDBACCS=アクセスするRDエリアの世代番号
$ PDDBORGUAP={YES｜NO}
$ PDSPACELVL={0｜1｜3}
```

```
$ PDCLTRDNODE=XDM/RD E2のデータベース識別子
$ PDTP1SERVICE={YES｜NO}
$ PDCNSTRNTNAME={LEADING｜TRAILING}
$ PDTMPTBLRDAREA=RDエリア名 [, RDエリア名…]
$ PDBESCONHOLD={YES｜NO}
$ PDBESCONHTI=バックエンドサーバ接続保持期間
$ PDODBSTATCAHE={0｜1}
$ PDODBESCAPE={0｜1}
$ PDGDATAOPT={YES｜NO}
$ PDODBLOCATOR={YES｜NO}
$ PDODBSPLITSIZE=分割取得サイズ
$ PDODBCWRNSKIP={YES｜NO}
$ PDJETCOMPATIBLE={YES｜NO}
$ PDODBGINFOSUPPRESS={YES｜NO}
$ PDODBSTANDARDGTYPEINFO={YES｜NO}
$ PDODBERRTRC={YES｜NO}
$ PDODBERRTRCSIZE=ODBCエラーログの最大ファイルのサイズ
$ PDODBSYSUNSIGNED={YES｜NO}
$ PDODBSYSKEYS{YES｜NO}
$ PDPLGIXMK={YES｜NO}
$ PDPLGPFSZ=遅延一括作成用のインデクス情報ファイルの初期容量
$ PDPLGPFSZEXP=遅延一括作成用のインデクス情報ファイルの増分値

$ export PDHOST PDNAMEPORT PDFESHOST PDSERVICEGRP PDSRVTYPE
         PDSERVICEPORT PDFESGRP PDCLTRCVPORT PDCLTRCVADDR PDTMID PDXAMODE
         PDTXACANUM PDXARCVWTIME PDXATRCFILEMODE PDXAAUTORECONNECT PDUSER
         PDCLTAPNAME PDCLTLANG PDLANG PDDBLOG PDEXWARN PDSUBSTRLEN
         PDCLTCNVMODE PDCLTGAIJIDLL PDCLTGAIJIFUNC PDCLTGRP
         PDAUTORECONNECT PDRCCOUNT PDRCINTERVAL PDRCTIMING
         PDUAPENVFILE PDDBBUFLRU
         PDHATRNQUEUING PDEXTDECCHECK PDASTHOST PDASTPORT PDSYSTEMID
         PDASTUSER PDCMDWAITTIME PDCMDTRACE PDIPC PDSENDMEMSIZE
         PDRECVMEMSIZE PDCWAITTIME PDSWAITTIME PDSWATCHTIME
         PDCWAITTIMEWRNPNT PDKALVL PDKATIME PDTIMEDOUTRETRY
         PDNBLOCKWAITTIME PDCONNECTWAITTIME PDCLTPATH PDSQLTRACE
         PDUAPERLOG PDERRSKIPCODE PDPRMTRC PDPRMTRCSIZE PDTRCMODE
         PDUAPREPLVL PDREPPATH PDTRCPATH PDSQLTRCOPENMODE PDSQLTEXTSIZE
         PDSQLEXECTIME PDRCTRACE PDWRTLNPATH PDWRTLNFILSZ PDWRTLNCOMSZ
         PDUAPEXERLOGUSE PDUAPEXERLOGPRMSZ PDSQLTRCFMT PDVWOPTMODE
         PDTAAPINFPATH PDTAAPINFMODE PDTAAPINFSIZE PDSTJTRNOUT
         PDLOCKLIMIT PDDLKPRIO PDLOCKSKIP PDFORUPDATEEXLOCK PDISLLVL
         PDSQLOPTLVL PDADDITIONALOPTLVL PDHASHTBLSIZE PDDFLNVAL PDAGGR
         PDCMMTBFDDL PDPRPCRCLS PDAUTOCONNECT PDDDLDEAPRPEXE
         PDDDLDEAPRP PDLCKWAITTIME PDCURSORLVL
         PDDELRSVWDFILE PDCALCMDWAITTIME
         PDSTANDARDSQLSTATE PDBLKF PDBINARYBLKF PDBLKBUFFSIZE
         PDBINDRETRYCOUNT
         PDBINDRETRYINTERVAL PDDBACCS
         PDDBORGUAPPDSPACELVL PDCLTRDNODE
         PDTP1SERVICE PDCNSTRNTNAME PDTMPTBLRDAREA PDBESCONHOLD
         PDBESCONHTI
         PDODBSTATCAHE PDODBESCAPE PDGDATAOPT PDODBLOCATOR PDODBSPLITSIZE
         PDODBCWRNSKIP PDJETCOMPATIBLE PDODBGINFOSUPPRESS
         PDODBSTANDARDGTYPEINFO PDODBERRTRC PDODBERRTRCSIZE
         PDODBSYSUNSIGNED PDODBSYSKEYS PDPLGIXMK PDPLGPFSZ PDPLGPFSZEXP
```

(b) csh（C シェル）の場合

.login ファイル，又は.cshrc ファイルに，次の環境変数を格納してください。ファイルに格納すると，起動時に環境変数が自動的に実行されます。

```
% setenv PDHOST HiRDBサーバのホスト名
                [,予備系HiRDBサーバのホスト名]
% setenv PDNAMEPORT HiRDBサーバのポート番号
% setenv PDFESHOST フロントエンドサーバのホスト名
          [:フロントエンドサーバがあるユニットのポート番号]
            [,予備系フロントエンドサーバのホスト名
             [:予備系フロントエンドサーバがあるユニットのポート番号]]
% setenv PDSERVICEGRP サーバ名
% setenv PDSRVTYPE {WS|PC}
% setenv PDSERVICEPORT 高速接続用のポート番号
                [,予備系の高速接続用のポート番号]
% setenv PDFESGRP FESグループ[,切替FESグループ[,切替FESグループ]…]
% setenv PDCLTRCVPORT クライアントの受信ポート番号
% setenv PDCLTRCVADDR {クライアントのIPアドレス
                      |クライアントのホスト名}
% setenv PDTMID OLTP識別子
% setenv PDXAMODE {0|1}
% setenv PDTXACANUM 1UAP当たりのトランザクション最大同時実行数
% setenv PDXARCVWTIME トランザクションが回復できない場合の
                待ち合わせ時間
% setenv PDXATRCFILEMODE {LUMP|SEPARATE}
% setenv PDXAAUTORECONNECT {YES|NO}
% setenv HiRDB_PDHOST HiRDBサーバのホスト名
                [,予備系HiRDBサーバのホスト名]
% setenv HiRDB_PDNAMEPORT HiRDBサーバのポート番号
% setenv HiRDB_PDTMID OLTP識別子
% setenv HiRDB_PDXAMODE {0|1}
% setenv PDUSER 認可識別子/パスワード
% setenv PDCLTAPNAME 実行するUAPの識別名称
% setenv PDCLTLANG {SJIS|CHINESE|UJIS|C|UTF-8|CHINESE-GB18030}
% setenv PDLANG {UTF-8|SJIS|CHINESE|CHINESE-GB18030|ANY}
% setenv PDDBLOG {ALL|NO}
% setenv PDEXWARN {YES|NO}
% setenv PDSUBSTRLEN {3|4|5|6|7|8|9|10}
% setenv PDCLTCNVMODE {AUTO|NOUSE|UJIS|UJIS2|UTF8|UTF8MS
                |UTF8_TXT|UTF8_EX|UTF8_EX2|UTF8MS_TXT
                |UCS2_UJIS|UCS2_UJIS2|UCS2_UTF8}
% setenv PDCLTGAIJIDLL ユーザ定義外字変換DLLファイル名
% setenv PDCLTGAIJIFUNC ユーザ定義外字変換関数名
% setenv PDCLTGRP クライアントグループ名
% setenv PDAUTORECONNECT {YES|NO}
% setenv PDRCCOUNT 自動再接続機能でのCONNECTのリトライ回数
% setenv PDRCINTERVAL 自動再接続機能でのCONNECTのリトライ間隔
% setenv PDRCTIMING {ALL|再接続契機[,再接続契機[,再接続契機]]}
% setenv PDUAPENVFILE UAP環境定義のファイル名
% setenv PDDBBUFLRU {YES|NO}
% setenv PDHATRNQUEUING NO
% setenv PDEXTDECCHECK {YES|NO}
% setenv PDASTHOST HiRDB Control Manager - Agentのホスト名
           [,予備系HiRDB Control Manager - Agentのホスト名]
% setenv PDASTPORT HiRDB Control Manager - Agentのポート番号
% setenv PDSYSTEMID HiRDB Control Manager - Agentが管理するHiRDBサーバの
```

```
                        HiRDB識別子
% setenv PDASTUSER [OSのユーザ名/パスワード]
% setenv PDCMDWAITTIME コマンド実行時のクライアントの最大待ち時間
% setenv PDCMDTRACE コマンドトレースファイルのサイズ
% setenv PDIPC {MEMORY｜DEFAULT}
% setenv PDSENDMEMSIZE クライアント側のデータ送信用メモリサイズ
% setenv PDRECVMEMSIZE クライアント側のデータ受信用メモリサイズ
% setenv PDCWAITTIME クライアントの最大待ち時間
% setenv PDSWAITTIME トランザクション処理中のサーバの最大待ち時間
% setenv PDSWATCHTIME トランザクション処理以外のサーバの最大待ち時間
% setenv PDCWAITTIMEWRNPNT SQL実行時間警告出力の契機
% setenv PDKALVL {0｜1｜2｜3}
% setenv PDKATIME パケットの送信間隔
% setenv PDTIMEDOUTRETRY リトライ回数
% setenv PDNBLOCKWAITTIME ノンブロックモードでのコネクション確立監視時間
% setenv PDCONNECTWAITTIME サーバ接続時のHiRDBクライアントの最大待ち時間
% setenv PDCLTPATH トレースファイル格納ディレクトリ
% setenv PDSQLTRACE SQLトレースファイルのサイズ
% setenv PDUAPERLOG クライアントエラーログファイルのサイズ
% setenv PDERRSKIPCODE SQLCODE[,SQLCODE]…
% setenv PDPRMTRC  {YES｜NO｜IN｜OUT｜INOUT}
% setenv PDPRMTRCSIZE SQLトレースに出力するパラメタ情報の最大データ長
% setenv PDTRCMODE {ERR｜NONE}
% setenv PDUAPREPLVL  {[s][u[o][t]][p][r]｜[a[o][t]]}
% setenv PDREPPATH UAP統計レポートファイルの格納ディレクトリ
% setenv PDTRCPATH 動的SQLトレースファイルの格納先ディレクトリ
% setenv PDSQLTRCOPENMODE {CNCT｜SQL}
% setenv PDSQLTEXTSIZE SQL文のサイズ
% setenv PDSQLEXECTIME {YES｜NO}
% setenv PDRCTRACE 再接続トレースファイルのサイズ
% setenv PDWRTLNPATH WRITE LINE文の値式の値を出力するファイルの
                      格納先ディレクトリ
% setenv PDWRTLNFILSZ WRITE LINE文の値式の値を出力するファイルの
                           最大サイズ
% setenv PDWRTLNCOMSZ WRITE LINE文の値式の値の合計サイズ
% setenv PDUAPEXERLOGUSE {YES｜NO}
% setenv PDUAPEXERLOGPRMSZ パラメタ情報の最大データ長
% setenv PDSQLTRCFMT {1|2}
% setenv PDVWOPTMODE {0｜1｜2}
% setenv PDTAAPINFPATH アクセスパス情報ファイル出力ディレクトリ名
% setenv PDTAAPINFMODE {0｜1}
% setenv PDTAAPINFSIZE アクセスパス情報ファイルサイズ
% setenv PDSTJTRNOUT {YES｜NO}
% setenv PDLOCKLIMIT ユーザ当たりの最大排他資源要求数
% setenv PDDLKPRIO {96｜64｜32}
% setenv PDLOCKSKIP {YES｜NO}
% setenv PDFORUPDATEEXLOCK {YES｜NO}
% setenv PDISLLVL データ保証レベル
% setenv PDSQLOPTLVL SQL最適化オプション[,SQL最適化オプション]…
% setenv PDADDITIONALOPTLVL SQL拡張最適化オプション
                              [,SQL拡張最適化オプション]…
% setenv PDHASHTBLSIZE ハッシュジョイン，副問合せのハッシュ実行適用時の
                        ハッシュ表サイズ
% setenv PDDFLNVAL {USE｜NOUSE}
% setenv PDAGGR グループ分けのときに発生するグループ数
% setenv PDCMMTBFDDL {YES｜NO}
% setenv PDPRPCRCLS {YES｜NO}
% setenv PDAUTOCONNECT {ON｜OFF}
```

```
% setenv PDDDLDEAPRPEXE {YES | NO}
% setenv PDDDLDEAPRP {YES | NO}
% setenv PDLCKWAITTIME 排他待ち限界経過時間
% setenv PDCURSORLVL {0 | 1 | 2}
% setenv PDDELRSVWDFILE SQL予約語削除ファイル名
% setenv PDCALCMDWAITTIME CALL COMMAND文の最大待ち時間
% setenv PDSTANDARDSQLSTATE {YES | NO}
% setenv PDBLKF ブロック転送の行数
% setenv PDBINARYBLKF {YES | NO}
% setenv PDBLKBUFFSIZE 通信バッファサイズ
% setenv PDBINDRETRYCOUNT bindシステムコールのリトライ回数
% setenv PDBINDRETRYINTERVAL bindシステムコールのリトライ間隔
% setenv PDDBACCS アクセスするRDエリアの世代番号
% setenv PDDBORGUAP {YES | NO}
% setenv PDSPACELVL {0 | 1 | 3}
% setenv PDCLTRDNODE XDM/RD E2のデータベース識別子
% setenv PDTP1SERVICE {YES | NO}
% setenv PDCNSTRNTNAME {LEADING | TRAILING}
% setenv PDTMPTBLRDAREA RDエリア名 [, RDエリア名…]
% setenv PDBESCONHOLD {YES | NO}
% setenv PDBESCONHTI バックエンドサーバ接続保持期間
% setenv PDODBSTATCAHE {0 | 1}
% setenv PDODBESCAPE {0 | 1}
% setenv PDGDATAOPT {YES | NO}
% setenv PDODBLOCATOR {YES | NO}
% setenv PDODBSPLITSIZE 分割取得サイズ
% setenv PDODBCWRNSKIP {YES | NO}
% setenv PDJETCOMPATIBLE {YES | NO}
% setenv PDODBGINFOSUPPRESS {YES | NO}
% setenv PDODBSTANDARDGTYPEINFO {YES | NO}
% setenv PDODBERRTRC {YES | NO}
% setenv PDODBERRTRCSIZE ODBCエラーログの最大ファイルのサイズ
% setenv PDODBSYSUNSIGNED {YES | NO}
% setenv PDODBSYSKEYS {YES | NO}
% setenv PDPLGIXMK {YES | NO}
% setenv PDPLGPFSZ 遅延一括作成用のインデクス情報ファイルの初期容量
% setenv PDPLGPFSZEXP 遅延一括作成用のインデクス情報ファイルの増分値
```

注意事項：

- プリプロセスするときには，環境変数を設定する必要があります。プリプロセスについては，「プリプロセス」を参照してください。
- Type4 JDBC ドライバを使用する場合，この方法で設定したクライアント環境定義は有効になりません。
- PDJDB で始まるクライアント環境定義は，この方法で設定しても有効になりません。
- 指定値に引用符（"）で囲む値を含む場合の注意事項については，「クライアント環境定義の一覧」の「注※5」を参照してください。

(2) Windows 環境の場合

Windows 環境では，インストール時に環境変数を設定する選択をした場合，環境変数 PATH にディレクトリが設定されます。ただし，パス名が長い場合，PATH への書き込み権限がない場合など，自動的に設定

されないことがあります。したがって，PATH にディレクトリが設定されているかどうかを確認し，設定されていない場合は PATH に次のディレクトリを追加する必要があります。xxxx は HiRDB クライアントのインストールディレクトリ名を示します。

```
xxxx¥UTL
```

環境変数は，システム環境変数，若しくはユーザ環境変数に設定するか，又は Windows ディレクトリ下の HiRDB.INI ファイルに設定してください。なお，UAP 中で関数を使用して環境変数を設定する場合は，putenv 関数は使用しないで，SetEnvironmentVariable 関数を使用してください。

- **クライアント環境定義の検索順序**

  クライアント環境定義を複数の箇所に設定している場合，次の順序でクライアント環境定義ごとに検索し，指定値がないクライアント環境定義については，デフォルト値を適用します。

  1. 環境変数グループ※
  2. ユーザ環境変数
  3. システム環境変数
  4. HIRDB.ini

  注※

  複数接続機能使用時に ALLOCATE CONNECTION HANDLE でグループ名，又はファイル名を指定します。また，OLTP 下の UAP をクライアントとする場合，オープン文字列にグループ名，又はファイル名を指定します。オープン文字列については，マニュアル「HiRDB システム導入・設計ガイド」を参照してください。

  使用するクライアント機能によっては，上記以外の方法でもクライアント環境定義を設定できます。検索順序の詳細は，次の箇所を参照してください。

  ＜ HiRDB ODBC ドライバの場合＞

  「接続情報の優先順位」

  ＜ HiRDB データプロバイダ for .NET Framework の場合＞

  「接続情報の優先順位」

  ＜ Type4 JDBC ドライバの場合＞

  「接続情報の優先順位」

HiRDB.INI ファイルの設定例を次に示します。

```
[HIRDB]
PDHOST=HiRDBサーバのホスト名[,予備系HiRDBサーバのホスト名]
PDNAMEPORT=HiRDBサーバのポート番号
PDFESHOST=フロントエンドサーバのホスト名
          [:フロントエンドサーバがあるユニットのポート番号]
            [,予備系フロントエンドサーバのホスト名
              [:予備系フロントエンドサーバがあるユニットのポート番号]]
PDSERVICEGRP=サーバ名
PDSRVTYPE={WS | PC}
```

```
PDSERVICEPORT=高速接続用のポート番号
                [,予備系の高速接続用のポート番号]
PDFESGRP=FESグループ[,切替FESグループ[,切替FESグループ]…]
PDCLTRCVPORT=クライアントの受信ポート番号
PDCLTRCVADDR={クライアントのIPアドレス｜クライアントのホスト名}
PDXATRCFILEMODE={LUMP｜SEPARATE}
PDUSER=[認可識別子/パスワード]
PDCLTAPNAME=実行するUAPの識別名称
PDCLTLANG={SJIS｜CHINESE｜UJIS｜C｜UTF-8｜CHINESE-GB18030}
PDLANG={UTF-8｜SJIS｜CHINESE｜CHINESE-GB18030｜ANY}
PDDBLOG={ALL｜NO}
PDEXWARN={YES｜NO}
PDSUBSTRLEN={3｜4｜5｜6｜7｜8｜9｜10}
PDCLTCNVMODE={AUTO｜NOUSE｜UJIS｜UJIS2｜UTF8｜UTF8MS
              ｜UTF8_TXT｜UTF8_EX｜UTF8_EX2｜UTF8MS_TXT
              ｜UCS2_UJIS｜UCS2_UJIS2｜UCS2_UTF8}
PDCLTGAIJIDLL=ユーザ定義外字変換DLLファイル名
PDCLTGAIJIFUNC=ユーザ定義外字変換関数名
PDCLTGRP=クライアントグループ名
PDAUTORECONNECT={YES｜NO}
PDRCCOUNT=自動再接続機能でのCONNECTのリトライ回数
PDRCINTERVAL=自動再接続機能でのCONNECTのリトライ間隔
PDRCTIMING={ALL｜再接続契機[,再接続契機[,再接続契機]]}
PDUAPENVFILE=UAP環境定義のファイル名
PDDBBUFLRU={YES｜NO}
PDHATRNQUEUING=NO
PDCLTBINDLOOPBACKADDR={YES｜NO}
PDEXTDECCHECK={YES｜NO}
PDASTHOST=HiRDB Control Manager - Agentのホスト名
          [,予備系HiRDB Control Manager - Agentのホスト名]
PDASTPORT=HiRDB Control Manager - Agentのポート番号
PDSYSTEMID=HiRDB Control Manager - Agentが管理するHiRDBサーバのHiRDB識別子
PDASTUSER=OSのユーザ名/パスワード
PDCMDWAITTIME=コマンド実行時のクライアントの最大待ち時間
PDCMDTRACE=コマンドトレースファイルのサイズ
PDIPC={MEMORY｜DEFAULT}
PDSENDMEMSIZE=クライアント側のデータ送信用メモリサイズ
PDRECVMEMSIZE=クライアント側のデータ受信用メモリサイズ
PDCWAITTIME=クライアントの最大待ち時間
PDSWAITTIME=トランザクション処理中のサーバの最大待ち時間
PDSWATCHTIME=トランザクション処理以外のサーバの最大待ち時間
PDCWAITTIMEWRNPNT=SQL実行時間警告出力の契機
PDKALVL={0｜1｜2｜3}
PDKATIME=パケットの送信間隔
PDTIMEDOUTRETRY=リトライ回数
PDNBLOCKWAITTIME=ノンブロックモードでのコネクション確立監視時間
PDCONNECTWAITTIME=サーバ接続時のHiRDBクライアントの最大待ち時間
PDCLTPATH=トレースファイル格納ディレクトリ
PDSQLTRACE=SQLトレースファイルのサイズ
PDUAPERLOG=クライアントエラーログファイルのサイズ
PDERRSKIPCODE=SQLCODE[,SQLCODE]…
PDPRMTRC= {YES｜NO｜IN｜OUT｜INOUT}
PDPRMTRCSIZE=SQLトレースに出力するパラメタ情報の最大データ長
PDTRCMODE={ERR｜NONE}
PDUAPREPLVL= {[s][u[o][t]][p][r]｜[a[o][t]]}
PDREPPATH=UAP統計レポートファイルの格納ディレクトリ
PDTRCPATH=動的SQLトレースファイルの格納先ディレクトリ
PDSQLTRCOPENMODE={CNCT｜SQL}
```

```
PDSQLTEXTSIZE=SQL文のサイズ
PDSQLEXECTIME={YES｜NO}
PDRCTRACE=再接続トレースファイルのサイズ
PDWRTLNPATH=WRITE LINE文の値式の値を出力するファイルの
            格納先ディレクトリ
PDWRTLNFILSZ=WRITE LINE文の値式の値を出力するファイルの
                  最大サイズ
PDWRTLNCOMSZ=WRITE LINE文の値式の値の合計サイズ
PDUAPEXERLOGUSE={YES｜NO}
PDUAPEXERLOGPRMSZ=パラメタ情報の最大データ長
PDSQLTRCFMT={1|2}
PDDNDPTRACE=メソッドトレースのファイルサイズ
PDDNDPCOMPATIBLE={NO|ALL}
PDVWOPTMODE={0｜1｜2}
PDTAAPINFPATH=アクセスパス情報ファイル出力ディレクトリ名
PDTAAPINFMODE={0｜1}
PDTAAPINFSIZE=アクセスパス情報ファイルサイズ
PDSTJTRNOUT={YES｜NO}
PDLOCKLIMIT=ユーザ当たりの最大排他資源要求数
PDDLKPRIO={96｜64｜32}
PDLOCKSKIP={YES｜NO}
PDFORUPDATEEXLOCK={YES｜NO}
PDISLLVL=データ保証レベル
PDSQLOPTLVL=SQL最適化オプション[,SQL最適化オプション]…
PDADDITIONALOPTLVL=SQL拡張最適化オプション
                   [,SQL拡張最適化オプション]…
PDHASHTBLSIZE=ハッシュジョイン，副問合せのハッシュ実行適用時の
              ハッシュ表サイズ
PDDFLNVAL={USE｜NOUSE}
PDAGGR=グループ分けのときに発生するグループ数
PDCMMTBFDDL={YES｜NO}
PDPRPCRCLS={YES｜NO}
PDAUTOCONNECT={ON｜OFF}
PDDDLDEAPRPEXE={YES｜NO}
PDDDLDEAPRP={YES｜NO}
PDLCKWAITTIME=排他待ち限界経過時間
PDCURSORLVL={0｜1｜2}
PDDELRSVWDFILE=SQL予約語削除ファイル名
PDCALCMDWAITTIME=CALL COMMAND文の最大待ち時間
PDSTANDARDSQLSTATE={YES｜NO}
PDBLKF=ブロック転送の行数
PDBINARYBLKF={YES｜NO}
PDBLKBUFFSIZE=通信バッファサイズ
PDBINDRETRYCOUNT=bindシステムコールのリトライ回数
PDBINDRETRYINTERVAL=bindシステムコールのリトライ間隔
PDDBACCS=アクセスするRDエリアの世代番号
PDDBORGUAP={YES｜NO}
PDSPACELVL={0｜1｜3}
PDCLTRDNODE=XDM/RD E2のデータベース識別子
PDTP1SERVICE={YES｜NO}
PDRDCLTCODE={SJIS｜UTF-8}
PDCNSTRNTNAME={LEADING｜TRAILING}
PDTMPTBLRDAREA=RDエリア名 [, RDエリア名…]
PDBESCONHOLD={YES｜NO}
PDBESCONHTI=バックエンドサーバ接続保持期間
PDODBSTATCAHE={0｜1}
PDODBESCAPE={0｜1}
PDGDATAOPT={YES｜NO}
```

```
PDODBLOCATOR={YES | NO}
PDODBSPLITSIZE=分割取得サイズ
PDODBCWRNSKIP={YES | NO}
PDJETCOMPATIBLE={YES | NO}
PDODBGINFOSUPPRESS={YES | NO}
PDODBSTANDARDGTYPEINFO={YES | NO}
PDODBERRTRC={YES | NO}
PDODBERRTRCSIZE=ODBCエラーログの最大ファイルのサイズ
PDODBSYSUNSIGNED={YES | NO}
PDODBSYSKEYS{YES | NO}
PDPLGIXMK={YES | NO}
PDPLGPFSZ=遅延一括作成用のインデクス情報ファイルの初期容量
PDPLGPFSZEXP=遅延一括作成用のインデクス情報ファイルの増分値
```

注意事項：

- プリプロセスするときには，環境変数を設定する必要があります。プリプロセスについては，「プリプロセス」を参照してください。
- Type4 JDBC ドライバを使用する場合，この方法で設定したクライアント環境定義は有効になりません。
- PDJDB で始まるクライアント環境定義は，この方法で設定しても有効になりません。
- 指定値に引用符（"）で囲む値を含む場合の注意事項については，「クライアント環境定義の一覧」の「注※5」を参照してください。

6.6.2 OLTP 下の X/Open に従った API を使用した UAP をクライアントとする場合の指定方法

(1) OpenTP1 下の UAP をクライアントとする場合

OpenTP1 下の UAP をクライアントとする運用形態の場合，クライアント環境定義は OpenTP1 のシステムサービス定義に指定してください。環境変数は，次に示す OpenTP1 の定義で指定します。

- **全環境に共通の指定をする場合**
  システム環境定義
- **トランザクション障害時の回復制御に関する指定をする場合**
  トランザクションサービス定義
- **全 UAP に共通の指定をする場合**
  ユーザサービスデフォルト定義
- **各 UAP に個別の指定をする場合**
  各ユーザサービス定義

環境変数を指定する OpenTP1 の定義を次の表に示します。なお，指定する場合は，putenv 形式又は dcputenv 形式で指定してください。

表 6–23　環境変数を指定する OpenTP1 の定義

環境変数	システム 環境定義	トランザクション サービス定義	ユーザサービス デフォルト定義	ユーザサービス 定義
HiRDB_PDHOST※9	○※1	×	×	×
HiRDB_PDNAMEPORT※9	○※2	×	×	×
HiRDB_PDTMID※9	△※3※4	×	×	×
HiRDB_PDXAMODE※9	△※5	×	×	×
PDHOST	×	○※1※6	○※1※6	△※1※6※7
PDNAMEPORT	×	○※2※6	○※2※6	△※2※6※7
PDTMID※9	×	△※3※4※6	△※3※4※6	△※3※4※6※7
PDXAMODE※9※10	×	△※5※6	△※5※6	△※5※6※7
PDTXACANUM※9	×	△	△	△
PDCLTPATH	×	△	△	△
PDUSER	×	×	○	○
PDCWAITTIME	×	△	△	△
PDSWAITTIME	×	○	○	○
PDSQLTRACE	×	△	△	△
PDUAPERLOG	×	△	△	△
PDCLTAPNAME	×	△	△※8	△※8
PDSWATCHTIME	×	×	○	○
PDTRCMODE	×	△	△	△
PDUAPREPLVL	×	△	△	△
PDREPPATH	×	△	△	△
PDTRCPATH	×	△	△	△
PDSQLTRCOPENMODE	×	△	△	△
PDAUTOCONNECT	×	×	×	×
PDXARCVWTIME※9	×	△	×	×
PDCWAITTIMEWRNPNT	×	△	△	△
PDAUTORECONNECT	×	×	×	×
PDRCCOUNT	×	×	×	×
PDRCINTERVAL	×	×	×	×

環境変数	システム環境定義	トランザクションサービス定義	ユーザサービスデフォルト定義	ユーザサービス定義
PDRCTIMING	×	×	×	×
PDKALVL	×	△※11	△※11	△※11
PDKATIME	×	△※11	△※11	△※11
PDSQLTEXTSIZE	×	△	△	△
PDSQLEXECTIME	×	△	△	△
PDRCTRACE	×	×	×	×
PDSQLTRCFMT	×	△	△	△
上記以外の環境変数	×	×	△	△

（凡例）

○：必要です。

△：任意です。必要に応じて指定してください。

×：不要です。

注

OpenTP1 のシステムサービス定義については，マニュアル「OpenTP1 システム定義」を参照してください。

注※1

HiRDB_PDHOST を指定する場合は，HiRDB_PDHOST に指定する値を PDHOST にも設定するため，PDHOST の指定は不要です。HiRDB_PDHOST を指定しない場合は，PDHOST を必ず指定してください。PDHOST と HiRDB_PDHOST の両方を指定した場合は，HiRDB_PDHOST の指定が優先されます。

なお，環境変数グループに PDHOST を指定した場合は，環境変数グループの PDHOST の指定が有効になります。

PDHOST に指定する値の目安については，「PDHOST に PDFESHOST のホスト名を指定することによる通信先サーバの固定化（HiRDB/パラレルサーバ限定）」を参照してください。

注※2

HiRDB_PDNAMEPORT を指定する場合は，HiRDB_PDNAMEPORT に指定する値を PDNAMEPORT にも設定するため，PDNAMEPORT の指定は不要です。HiRDB_PDNAMEPORT を指定しない場合は，PDNAMEPORT を必ず指定してください。PDNAMEPORT と HiRDB_PDNAMEPORT の両方を指定した場合は，HiRDB_PDNAMEPORT の指定が優先されます。

なお，環境変数グループに PDNAMEPORT を指定した場合は，環境変数グループの PDNAMEPORT の指定が有効になります。

注※3

複数の OLTP から X/Open に従った API を使用して一つの HiRDB サーバにアクセスする場合，必ず指定してください。

注※4

HiRDB_PDTMID を指定する場合は，HiRDB_PDTMID に指定する値を PDTMID にも設定するため，PDTMID の指定は不要です。HiRDB_PDTMID を指定しない場合は，PDTMID を必ず指定してください。PDTMID と HiRDB_PDTMID の両方を指定した場合は，HiRDB_PDTMID の指定が優先されます。

注※5

HiRDB_PDXAMODE を指定する場合は，HiRDB_PDXAMODE に指定する値を PDXAMODE にも設定するため，PDXAMODE の指定は不要です。PDXAMODE と HiRDB_PDXAMODE の両方を指定した場合は，HiRDB_PDXAMODE の指定が優先されます。

注※6

指定する場合は，トランザクションサービス定義と HiRDB にアクセスするすべてのユーザサーバで同じ内容の指定が必要です。

トランザクションサービス定義と，ユーザサービスデフォルト定義又は HiRDB にアクセスするすべてのユーザサーバのユーザサービス定義で，必ず同じ内容を指定してください。

注※7

HiRDB にアクセスするすべてのユーザのサーバに対し，同じ内容の指定が必要なため，各ユーザサービス定義には指定しないで，ユーザサービスデフォルト定義で指定することをお勧めします。

注※8

各ユーザのサーバを識別できるようにユーザサービスデフォルト定義には指定しないで，各ユーザサービス定義で指定することをお勧めします。

注※9

複数接続機能を使用する場合，これらの環境変数を接続先ごとに登録した環境変数グループに設定しても，環境変数の指定値は無効になります。また，Windows 環境の場合，HiRDB.ini ファイルに指定しても無効になります。これらの環境変数は，OpenTP1 のシステムサービス定義に指定した内容が有効になります。

注※10

trnstring のオプションと，PDXAMODE の設定が合っていない場合，xa 関数が-6 エラーになるため，注意してください。

注※11

OpenTP1 では指定できません（TP1/EE でだけ指定できます）。

(2)　TP1/LiNK 下の UAP をクライアントとする場合

UAP を実行するためには，TP1/LiNK の定義にクライアント環境定義を指定する必要があります。指定方法を次に示します。

- **トランザクション障害時の回復制御に関する指定をする場合**

  ［リソースマネジャ］ウィンドウの［オプション（P）...］ボタンをクリックして，［オプション］ダイアログボックスを開き，［トランザクションサービスの環境設定］欄に指定してください。

- **全 UAP に共通の指定をする場合**

［SPP（又は SUP）環境設定］ダイアログボックスを開き，［ユーザサーバの環境変数］欄の［グローバル］欄に指定してください。

- **各 UAP に個別の指定をする場合**

  ［SPP（又は SUP）環境設定］ダイアログボックスを開き，［ユーザサーバの環境変数］欄の［ローカル］欄に指定してください。

環境変数を指定する TP1/LiNK の定義を次の表に示します。

表 6–24　環境変数を指定する TP1/LiNK の定義

環境変数	［トランザクションサービスの環境変数］欄	［ユーザサーバの環境変数］欄	
		［グローバル］欄	［ローカル］欄
HiRDB_PDHOST	×	×	×
HiRDB_PDNAMEPORT	×	×	×
HiRDB_PDTMID	×	×	×
HiRDB_PDXAMODE	×	×	×
PDHOST	○※2	○※2	△※2※3
PDNAMEPORT	○※2	○※2	△※2※3
PDTMID※5	△※1※2	△※1※2	△※1※2※3
PDXAMODE※5	△※2	△※2	△※2※3
PDTXACANUM※5	△	△	△
PDCLTPATH	△	△	△
PDUSER	×	○	○
PDCWAITTIME	△	△	△
PDSWAITTIME	○	○	○
PDSQLTRACE	△	△	△
PDUAPERLOG	△	△	△
PDCLTAPNAME	△	△※4	△※4
PDSWATCHTIME	○	○	○
PDTRCMODE	△	△	△
PDUAPREPLVL	△	△	△
PDREPPATH	△	△	△
PDTRCPATH	△	△	△
PDSQLTRCOPENMODE	△	△	△

環境変数	［トランザクションサービスの環境変数］欄	［ユーザサーバの環境変数］欄	
		［グローバル］欄	［ローカル］欄
PDAUTOCONNECT	×	×	×
PDXARCVWTIME※5	△	×	×
PDCWAITTIMEWRNPNT	△	△	△
PDSQLTRCFMT	△	△	△
上記以外の環境変数	×	△	△

（凡例）

○：必要です。

△：任意です。必要に応じて指定してください。

×：不要です。

注

TP1/LiNK の定義については，マニュアル「TP1/LiNK 使用の手引」を参照してください。

注※1

複数 OLTP から X/Open に従った API を使用して，一つの HiRDB サーバにアクセスする場合は必ず指定してください。

注※2

指定する場合は，それぞれの定義で必ず同じ内容を指定してください。

注※3

HiRDB にアクセスするすべてのユーザのサーバに対し，同じ内容の指定が必要なため，［ユーザサーバの環境変数］欄の［ローカル］欄に指定しないで，［グローバル］欄に指定することをお勧めします。

注※4

各ユーザのサーバを識別できるように［ユーザサーバの環境変数］欄の［グローバル］欄に指定しないで，［ローカル］欄に指定することをお勧めします。

注※5

複数接続機能を使用する場合，これらの環境変数を接続先ごとに登録した環境変数グループに設定しても，環境変数の指定値は無効になります。また，Windows 環境の場合，HIRDB.ini ファイルに指定しても無効になります。これらの環境変数は，TP1/LiNK の定義に指定した内容が有効になります。

(3)　TPBroker for C++下の UAP をクライアントとする場合

TPBroker for C++下の UAP をクライアントとする運用形態の場合，クライアント環境定義は TPBroker for C++のシステム定義に指定してください。TPBroker for C++のシステム定義については，マニュアル「TPBroker ユーザーズガイド」を参照してください。

クライアント環境定義を指定する場合，それぞれ次の形式で指定してください。

- **トランザクション決着プロセスに指定をする場合**

  トランザクション定義にクライアント環境定義を指定します。この場合，TPBroker for C++の tsdefvalue コマンドで指定してください。定義キーは/OTS，定義パラメタは completion_process_env です。

```
tsdefvalue /OTS completion_process_env
    -a '環境変数名=指定値', ['環境変数名=指定値',…]
```

- **トランザクション障害時のトランザクション回復プロセスに指定をする場合**

  トランザクション定義にクライアント環境定義を指定します。この場合，TPBroker for C++の tsdefvalue コマンドで指定してください。定義キーは/OTS，定義パラメタは recovery_process_env です。

```
tsdefvalue /OTS recovery_process_env
    -a '環境変数名=指定値', ['環境変数名=指定値',…]
```

- **各 UAP に個別の指定をする場合**

  各 UAP の動作環境でクライアント環境定義を指定します。SET 形式，SETENV 形式など，動作環境の環境変数の設定方式に従ってください。

- **監視対象の各 UAP に個別の指定をする場合**

  TPBroker for C++の各プロセス監視定義の定義ファイルに，クライアント環境定義を記述してください。

環境変数を指定する TPBroker for C++の定義を次の表に示します。

表 6–25　環境変数を指定する TPBroker for C++の定義

環境変数	トランザクション決着プロセス	トランザクション回復プロセス	各 UAP
HiRDB_PDHOST※8	△※1※4	△※1※4	△※1※4
HiRDB_PDNAMEPORT※8	△※1※5	△※1※5	△※1※5
HiRDB_PDTMID※8	△※1※3※6	△※1※3※6	△※1※3※6
HiRDB_PDXAMODE※8	△※1※7	△※1※7	△※1※7
PDHOST	△※1※4	△※1※4	△※1※4
PDNAMEPORT	△※1※5	△※1※5	△※1※5
PDTMID※8	△※1※3※6	△※1※3※6	△※1※3※6
PDXAMODE※8	△※1※7	△※1※7	△※1※7
PDTXACANUM※8	△	△	△
PDCLTPATH	△	△	△
PDUSER	○	×	○
PDCWAITTIME	△	△	△
PDSWAITTIME	○	○	○
PDSQLTRACE	△	△	△

環境変数	トランザクション決着プロセス	トランザクション回復プロセス	各 UAP
PDUAPERLOG	△	△	△
PDCLTAPNAME	△	△	△※2
PDSWATCHTIME	×	×	×
PDTRCMODE	△	△	△
PDUAPREPLVL	△	△	△
PDREPPATH	△	△	△
PDTRCPATH	△	△	△
PDSQLTRCOPENMODE	△	△	△
PDAUTOCONNECT	×	×	×
PDCWAITTIMEWRNPNT	△	△	△
PDAUTORECONNECT	×	×	×
PDRCCOUNT	×	×	×
PDRCINTERVAL	×	×	×
PDRCTIMING	×	×	×
PDKALVL	△	△	△
PDKATIME	△	△	△
PDSQLTEXTSIZE	△	△	△
PDSQLEXECTIME	△	△	△
PDRCTRACE	×	×	×
PDSQLTRCFMT	△	△	△
上記以外の環境変数	△	×	△

（凡例）

○：必要です。

△：任意です。必要に応じて指定してください。

×：不要です。

注※1

「トランザクション決着プロセス」，「トランザクション回復プロセス」，及び「各 UAP」のそれぞれのクライアント環境定義は必ず同じ内容を指定してください。

注※2

各プロセスを識別できるように，プロセスごとに指定することをお勧めします。

注※3

複数の OLTP から X/Open に従った API を使用して一つの HiRDB システムをアクセスする場合，必ず指定してください。

注※4

HiRDB_PDHOST を指定する場合は，HiRDB_PDHOST に指定した値が PDHOST に設定されるので，PDHOST の指定は不要です。HiRDB_PDHOST を指定しない場合は，PDHOST を必ず指定してください。PDHOST と HiRDB_PDHOST の両方を指定した場合は，HiRDB_PDHOST の指定を優先します。

なお，環境変数グループに PDHOST を指定した場合は，環境変数グループの PDHOST の指定が有効になります。

PDHOST に指定する値の目安については，「PDHOST に PDFESHOST のホスト名を指定することによる通信先サーバの固定化（HiRDB/パラレルサーバ限定）」を参照してください。

注※5

HiRDB_PDNAMEPORT を指定する場合は，HiRDB_PDNAMEPORT に指定した値が PDNAMEPORT に設定されるので，PDNAMEPORT の指定は不要です。HiRDB_PDNAMEPORT を指定しない場合は，PDNAMEPORT を必ず指定してください。PDNAMEPORT と HiRDB_PDNAMEPORT の両方を指定した場合は，HiRDB_PDNAMEPORT の指定を優先します。

なお，環境変数グループに PDNAMEPORT を指定した場合は，環境変数グループの PDNAMEPORT の指定が有効になります。

注※6

HiRDB_PDTMID を指定する場合は，HiRDB_PDTMID に指定した値が PDTMID に設定されるので，PDTMID の指定は不要です。HiRDB_PDTMID を指定しない場合は，PDTMID を必ず指定してください。PDTMID と HiRDB_PDTMID の両方を指定した場合は，HiRDB_PDTMID の指定を優先します。

注※7

HiRDB_PDXAMODE を指定する場合は，HiRDB_PDXAMODE に指定した値が PDXAMODE に設定されるので，PDXAMODE の指定は不要です。PDXAMODE と HiRDB_PDXAMODE の両方を指定した場合は，HiRDB_PDXAMODE の指定を優先します。

注※8

複数接続機能を使用する場合，これらの環境変数を接続先ごとに登録した環境変数グループに設定しても，環境変数の指定値は無効になります。また，Windows 環境の場合，HiRDB.ini ファイルに指定しても無効になります。これらの環境変数は，TPBroker for C++のシステム定義に指定した内容が有効になります。

(4) TUXEDO 下の UAP をクライアントとする場合

TUXEDO 下の UAP をクライアントとする運用形態の場合，TUXEDO コンフィギュレーションファイル（UBBCONFIG ファイル）の ENVFILE パラメタで指定したファイルに，クライアント環境定義を指定してください。

環境変数の指定可否を次の表に示します。

表 6–26　環境変数の指定可否（TUXEDO 下の UAP の場合）

環境変数	指定可否
HiRDB_PDHOST	×
HiRDB_PDNAMEPORT	×
HiRDB_PDTMID	×
HiRDB_PDXAMODE	×
PDHOST	○※1
PDNAMEPORT	○※1
PDTMID※4	△※1※3
PDXAMODE※4	○※1
PDTXACANUM	×
PDUSER	○
PDSWAITTIME	○
PDCLTAPNAME	△※2
PDSWATCHTIME	×
PDAUTORECONNECT	×
PDRCCOUNT	×
PDRCINTERVAL	×
PDRCTIMING	×
PDKALVL	×
PDKATIME	×
PDRCTRACE	×
上記以外の環境変数	△

（凡例）

○：必要です。

△：任意です。必要に応じて指定してください。

×：不要です。

注※1

「トランザクションマネジャサーバ」，「TUXEDO システムのサーバ」，及び「各 UAP」のそれぞれの環境変数は同じ内容にしてください。

PDHOST に指定する値の目安については，「PDHOST に PDFESHOST のホスト名を指定することによる通信先サーバの固定化（HiRDB/パラレルサーバ限定）」を参照してください。

注※2

各プロセスを識別できるように，プロセスごとに指定することをお勧めします。

注※3

複数の OLTP から X/Open に従った API を使用して一つの HiRDB システムをアクセスする場合，必ず指定してください。

注※4

Windows 環境の場合，HIRDB.ini ファイルに指定しても無効になります。これらの環境変数は，TUXEDO コンフィギュレーションファイルの ENVFILE パラメタで指定したファイルに指定した内容が有効になります。

(5) TP1/EE 下の UAP をクライアントとする場合（UNIX 版限定）

TP1/EE 下の UAP をクライアントとする運用形態の場合，クライアント環境定義は TP1/EE を実行する環境の OpenTP1 のシステムサービス定義に指定してください。詳細は「OpenTP1 下の UAP をクライアントとする場合」を参照してください。

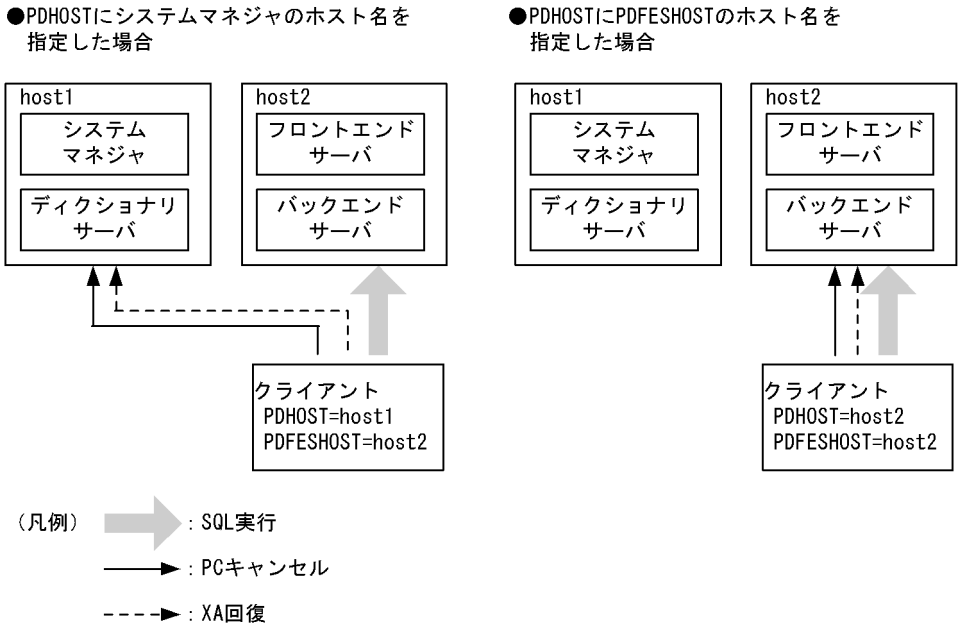
なお，PDXAMODE は必ず指定してください。TP1/EE を実行する OpenTP1 と PDXAMODE の指定値が異なる場合は，TP1/EE を実行する OpenTP1 のユーザサービス定義に PDXAMODE を指定してください。

(6) PDHOST に PDFESHOST のホスト名を指定することによる通信先サーバの固定化（HiRDB/パラレルサーバ限定）

PDHOST に，PDFESHOST のホスト名を指定すると，システムマネジャユニットに障害が発生しても，HiRDB サーバに接続できます。また，SQL 実行先，PC キャンセル先，及び XA 回復要求先の通信先サーバを一つに固定化できます。PC キャンセルとは，PDCWAITTIME オーバー時のサーバ決着指示のことをいいます。また，XA 回復とは，OLTP 下の UAP 使用時のトランザクション決着指示のことをいいます。なお，クライアントとサーバのバージョンの組み合わせによって，PDHOST に指定できるホスト名は，システムマネジャのホスト名に限定されます。

通信先サーバを固定化する場合としない場合の違いを次の図に示します。

図 6–1　通信先サーバを固定化する場合としない場合の違い



適用基準：

通信先サーバを固定化する場合，次の条件を満たしている必要があります。

- HiRDB/パラレルサーバである。
- FES ホストダイレクト接続，又は高速接続である。
- 次の表の，UAP の実行環境の推奨指定可否が○である。

<table>
<tr><th colspan="5">UAP の実行環境</th><th>推奨指定可否</th></tr>
<tr><td>非 OLTP 下</td><td colspan="4">1UAP が 1 接続，又は 1UAP が複数の接続をする場合に，同一の PDFESHOST を指定しているとき</td><td>○</td></tr>
<tr><td></td><td colspan="4">1UAP が複数の接続をする場合に，それぞれ異なる PDFESHOST を指定しているとき</td><td>×</td></tr>
<tr><td>OLTP 下</td><td>単一プロセス（マルチスレッド）で動作する OLTP（TP1/EE）</td><td colspan="3">OLTP 下の全接続先が，同一の PDFESHOST を指定している場合（接続先が同一の場合）※1</td><td>○</td></tr>
<tr><td></td><td></td><td colspan="3">OLTP 下の全スレッドが，それぞれ異なる PDFESHOST を指定している場合※1</td><td>×</td></tr>
<tr><td></td><td rowspan="3">複数プロセスで動作する OLTP（OpenTP1，TUXEDO，TPBroker for C++，及び TP1/LiNK）</td><td colspan="3">全 UAP が，同一の PDFESHOST を指定している場合※1</td><td>○</td></tr>
<tr><td rowspan="2">各 UAP が，別々の PDFESHOST を指定している場合※1</td><td rowspan="2">PDFESHOST を指定した UAP のクライアント環境定義※2</td><td>PDFESHOST を指定した UAP の全接続先が，同一の PDFESHOST を指定している場合</td><td>○</td></tr>
<tr><td>PDFESHOST を指定した UAP の全接続先が，それぞれ異なる PDFESHOST を指定している場合</td><td>×</td></tr>
</table>

UAP の実行環境				推奨指定可否
			トランザクションマネジャ用のクライアント環境定義※3	×

（凡例）

○：PDHOST に PDFESHOST のホスト名を指定することをお勧めします。

×：PDHOST にはシステムマネジャのホスト名を指定してください。

注※1

次の箇所に指定します。

- OpenTP1 又は TP1/EE の場合
  ユーザサービス定義，ユーザデフォルト定義，又はシステム環境定義に指定する環境変数
- TUXEDO の場合
  トランザクションマネジャサーバ，TUXEDO システムのサーバ，及び各 UAP のクライアント環境定義
- TPBroker for C++の場合
  トランザクション定義（決着プロセス用と回復プロセス用），及び各 UAP のクライアント環境定義
- TP1/LiNK の場合
  ［ユーザサーバの環境変数］欄の［グローバル］欄，及び［ローカル］欄
- 複数接続機能を使用している場合
  環境変数設定ファイル

注※2

次の箇所に指定します。

- OpenTP1 の場合
  ユーザサービス定義，又はユーザデフォルト定義に指定する環境変数
- TUXEDO の場合
  各 UAP のクライアント環境定義
- TPBroker for C++の場合
  トランザクション定義（決着プロセス用），及び UAP のクライアント環境定義
- TP1/LiNK の場合
  ［ユーザサーバの環境変数］欄の［グローバル］欄，及び［ローカル］欄
- 複数接続機能を使用する場合
  環境変数設定ファイル

注※3

次の箇所に指定します。

- OpenTP1 の場合
  トランザクションサービス定義に指定する環境変数
- TUXEDO の場合
  トランザクションマネジャサーバ，及び TUXEDO システムのサーバのクライアント環境定義
- TPBroker for C++の場合
  トランザクション定義（回復プロセス）
- TP1/LiNK の場合
  ［トランザクションサービスの環境変数］欄
- 複数接続機能を使用する場合
  環境変数設定ファイル

注意事項：

PDFESHOST にポート番号を指定している場合，PDNAMEPORT には接続先のポート番号を接続してください。

6.6.3　クライアント環境定義の一覧

クライアント環境定義の一覧を次の表に示します。なお，各環境変数の詳細は「クライアント環境定義の設定内容」を参照してください。

●必ず指定する環境変数

太字表示されている環境変数は，HiRDB システムの環境に関係なく必ず指定してください。太字表示以外の環境変数については，それぞれの HiRDB システムの環境に合わせて指定してください。

表 6–27　クライアント環境定義の一覧

環境変数名	機能	環境変数の分類
PDHOST	接続する HiRDB サーバのホスト名を指定します。	システム構成※3
PDNAMEPORT	HiRDB サーバのポート番号を指定します。	
PDFESHOST	フロントエンドサーバのホスト名を指定します。	
PDSERVICEGRP	シングルサーバ又はフロントエンドサーバのサーバ名を指定します。	
PDSRVTYPE	HiRDB サーバの種別を指定します。	
PDSERVICEPORT	高速接続用のポート番号を指定します。	
PDFESGRP	高速接続をする場合，接続する FES グループを指定します。	
PDCLTRCVPORT	クライアントの受信ポート番号を指定します。	
PDCLTRCVADDR	クライアントの IP アドレス又はホスト名を指定します。	

環境変数名	機能	環境変数の分類
PDCONTYPE	HiRDB 接続時の接続方式を指定します。	
PDTMID	複数の OLTP から一つの HiRDB サーバをアクセスする場合，それぞれの OLTP にユニークな識別子を指定します。	OLTP 下の X/Open に従った API を使用するクライアント※1
PDXAMODE	OLTP システムと連携する場合に，トランザクションの移行機能を使用するかしないかを指定します。	
PDTXACANUM	X/Open に従った API を使用した UAP から同時実行する，最大トランザクション数を指定します。	
PDXARCVWTIME	トランザクションが回復できない場合の待ち合わせ時間を指定します。	
PDXATRCFILEMODE	X/Open に従った API を使用した接続形態での，各種トレースファイル名の形式を指定します。	
PDXAAUTORECONNECT	TP1/EE との連携の場合に，自動再接続をするかどうかを指定します。	
HiRDB_PDHOST	接続する HiRDB サーバのホスト名を指定します。	
HiRDB_PDNAMEPORT	HiRDB サーバのポート番号を指定します。	
HiRDB_PDTMID	複数の OLTP から一つの HiRDB サーバをアクセスする場合，それぞれの OLTP にユニークな識別子を指定します。	
HiRDB_PDXAMODE	OLTP システムと連携する場合に，トランザクションの移行機能を使用するかしないかを指定します。	
PDUSER※4※5	認可識別子，及びパスワードを指定します。UNIX 環境の場合は，この環境変数を省略できます。	ユーザ実行環境
PDCLTAPNAME	HiRDB サーバに対してアクセスする，UAP の識別情報（UAP 識別子）を指定します。	
PDCLTLANG	プリプロセサが処理する，UAP の記述に使われている文字コード種別を指定します。	
PDLANG	HiRDB サーバの文字コード種別をチェックする場合に，HiRDB サーバの文字コード種別を指定します。	
PDDBLOG	UAP を実行するときに，データベースの更新ログを取得するかしないかを指定します。	
PDEXWARN	サーバから警告付きのリターンコードを受け取るかどうかを指定します。	
PDSUBSTRLEN	1 文字を表現する最大バイト数を指定します。	
PDCLTCNVMODE	HiRDB サーバと HiRDB クライアントの文字コード種別が異なる場合，文字コードを変換するかどうかを指定します。	
PDCLTGAIJIDLL	ユーザ定義外字変換 DLL ファイルの名称を指定します。	

環境変数名	機能	環境変数の分類
PDCLTGAIJIFUNC	ユーザ定義外字変換関数の名称を指定します。	
PDCLTCNVUOCLIB	UOC による文字コード変換をする場合，UOC のライブラリファイルの名称を指定します。	
PDCLTCNVUOCFUNC	UOC による文字コード変換をする場合，UOC の文字コード変換関数の名称を指定します。	
PDCLTCNVBYTERATIO	UOC による文字コード変換をする場合，１文字当たりのバイト数の比率を指定します。	
PDCLTGRP	クライアントグループの接続枠保証機能を使用する場合，クライアントグループ名を指定します。	
PDAUTORECONNECT	自動再接続機能を使用するかどうかを指定します。	
PDRCCOUNT	自動再接続機能での CONNECT のリトライ回数を指定します。	
PDRCINTERVAL	自動再接続機能での CONNECT のリトライ間隔を指定します。	
PDRCTIMING	自動再接続機能での再接続契機を指定します。	
PDAUTHTYPE	HiRDB クライアントがサーバへ接続するときの認証方式を指定します。	
PDUAPENVFILE	UAP を個別の環境で実行する場合，実行する環境を定義した UAP 環境定義ファイルを指定します。	
PDDBBUFLRU	UAP がアクセスしたページをグローバルバッファにキャッシュするときの処理に，LRU 方式を適用するかどうかを指定します。	
PDHATRNQUEUING	クライアントでトランザクションキューイング機能を使用しない場合に指定します。	
PDCLTBINDLOOPBACKADDR	HiRDB サーバとの通信で使用する受信ポートの生成時，ループバックアドレスで bind()するかどうかを指定します。	
PDEXTDECCHECK	外部 10 進項目の入力データのチェック有無を指定します。	
PDDEFAULTOPTION	クライアント環境定義及びプリプロセスオプションについて，省略時の動作を指定します。	
PDIPCFILEDIR	接続する HiRDB サーバの通信情報ファイルディレクトリを指定します。	
PDSOCKETCRYPTO	HiRDB クライアントと HiRDB サーバ間の通信の暗号化を行うかどうかを指定します。	
PDASTHOST	UAP 実行時に接続する，HiRDB Control Manager - Agent のホスト名を指定します。	UAP からのコマンド実行

環境変数名	機能	環境変数の分類
PDASTPORT	UAP 実行時に接続する，HiRDB Control Manager - Agent のポート番号を指定します。	
PDSYSTEMID	UAP 実行時に接続する，HiRDB Control Manager - Agent が管理する HiRDB サーバの HiRDB 識別子を指定します。	
PDASTUSER※5	コマンドを実行する OS のユーザ名，及びパスワードを指定します。	
PDCMDWAITTIME	クライアントが HiRDB Control Manager - Agent へ要求をしてから応答が返るまでの，クライアントの最大待ち時間を指定します。	
PDCMDTRACE	UAP 実行時にコマンドトレースを出力する場合，そのファイルの大きさを指定します。	
PDIPC	プロセス間の通信方法を指定します。	プロセス間メモリ通信機能
PDSENDMEMSIZE	プロセス間メモリ通信機能を使用する場合，クライアントからサーバへデータを送るときのデータ格納領域サイズを指定します。	
PDRECVMEMSIZE	プロセス間メモリ通信機能を使用する場合，クライアントがサーバからデータを受け取るときのデータ格納領域サイズを指定します。	
PDCWAITTIME※4	HiRDB クライアントから HiRDB サーバへ要求をしてから，応答が戻ってくるまでの HiRDB クライアントの最大待ち時間を指定します。	システム監視
PDSWAITTIME※4	HiRDB サーバが HiRDB クライアントからの要求に対する応答を返してから，次に HiRDB クライアントから要求が来るまでの HiRDB サーバの最大待ち時間を指定します。 この時間監視は，トランザクション処理中の時間を対象とします。	
PDSWATCHTIME	HiRDB サーバが HiRDB クライアントからの要求に対する応答を返してから，次に HiRDB クライアントから要求が来るまでの HiRDB サーバの最大待ち時間を指定します。 この時間監視は，トランザクション処理以外の時間を対象とします。	
PDCWAITTIMEWRNPNT	SQL 実行時間警告出力機能使用時に，SQL 実行時間警告情報ファイルを出力する契機を，HiRDB クライアントの最大待ち時間に対する比率，又は時間で指定します。	
PDKALVL	HiRDB クライアントから HiRDB サーバに対して，定期的にパケットを送信する機能を使用するかどうかを指定します。	

環境変数名	機能	環境変数の分類
PDKATIME	HiRDB クライアントから HiRDB サーバに対して，定期的にパケットを送信する間隔を指定します。	
PDTIMEDOUTRETRY	HiRDB クライアントが HiRDB サーバと接続する場合に実行する，connect()システムコールで ETIMEDOUT エラーが発生したときに，connect()システムコールをリトライする回数を指定します。	
PDCONREFRCOUNT	HiRDB クライアントが HiRDB サーバと接続する場合に実行する，connect()システムコールで ECONNREFUSED エラーが発生したときに，connect()システムコールをリトライする回数を指定します。	
PDCONREFRINTERVAL	HiRDB クライアントが HiRDB サーバと接続する場合に実行する，connect()システムコールで ECONNREFUSED エラーが発生したときに，connect()システムコールをリトライする間隔を指定します。	
PDNBLOCKWAITTIME	HiRDB サーバ，HiRDB クライアント間のコネクション接続完了を監視する場合，ノンブロックモード時のコネクション確立監視時間を指定します。	
PDCONNECTWAITTIME	HiRDB サーバとの接続時，HiRDB サーバから応答が戻ってくるまでの HiRDB クライアントの最大待ち時間を指定します。	
PDCLTPATH	HiRDB クライアントが作成する SQL トレースファイル及びクライアントエラーログファイルの格納先ディレクトリを指定します。	トラブルシュート
PDSQLTRACE	UAP の SQL トレースを出力する SQL トレースファイルのサイズを，バイト単位で指定します。	
PDUAPERLOG	UAP のエラーログを出力するクライアントエラーログファイルのサイズを，バイト単位で指定します。	
PDERRSKIPCODE	特定のクライアントエラーログを出力しない場合に指定します。	
PDPRMTRC	SQL トレースにパラメタ情報及び検索データを出力するかどうかを指定します。	
PDPRMTRCSIZE	SQL トレースに出力するパラメタ情報及び検索データの最大データ長を指定します。	
PDTRCMODE	SQL トレース以外のトラブルシュート情報を出力するかどうかを指定します。	
PDUAPREPLVL	UAP 統計レポートの出力情報を指定します。	
PDREPPATH	PDCLTPATH で指定したディレクトリとは別のディレクトリに，UAP 統計レポートを出力するかどうかを指定します。	

環境変数名	機能	環境変数の分類
PDTRCPATH	動的 SQL トレースファイルの格納先ディレクトリを指定します。	
PDSQLTRCOPENMODE	PDREPPATH を指定している場合，SQL トレースファイルのオープンモードを指定します。	
PDSQLTEXTSIZE	SQL トレースに出力する SQL 文のサイズを指定します。	
PDSQLEXECTIME	SQL トレースに SQL 実行時間を出力するかどうかを指定します。	
PDRCTRACE	UAP の再接続トレースを出力するファイルのサイズを指定します。	
PDWRTLNPATH	WRITE LINE 文の値式の値を出力する，ファイルの格納先ディレクトリを指定します。	
PDWRTLNFILSZ	WRITE LINE 文の値式の値を出力する，ファイルの最大サイズを指定します。	
PDWRTLNCOMSZ	WRITE LINE 文の値式の値の合計サイズを指定します。	
PDUAPEXERLOGUSE	拡張 SQL エラー情報出力機能を使用するかどうかを指定します。	
PDUAPEXERLOGPRMSZ	拡張 SQL エラー情報出力機能を使用する場合，クライアントエラーログファイル及び SQL エラーレポートファイルに出力するパラメタ情報の最大データ長を指定します。	
PDDNDPTRACE	ADO.NET 2.0 に対応した HiRDB データプロバイダ for .NET Framework で出力するメソッドトレースのファイルサイズを指定します。	
PDSQLTRCFMT	SQL トレースの出力形式を指定します。	
PDSQLTRCTYPE	SQL トレース情報の出力タイミングを指定します。	
PDJDBSQLTRCMEMNUM	Type4 JDBC ドライバでメモリ上に退避する SQL トレース情報数を指定します。	
PDSQLTRCBUFFSIZE	SQL トレース情報の退避領域のサイズを指定します。	
PDVWOPTMODE	アクセスパス情報ファイルを取得するかどうかを指定します。	アクセスパス表示ユティリティ用アクセスパス情報ファイル
PDTAAPINFPATH	アクセスパス情報ファイルを HiRDB クライアント側に出力する場合に，出力先ディレクトリを指定します。この指定がない場合は出力しません。	HiRDB SQL Tuning Advisor 用アクセスパス情報ファイル
PDTAAPINFMODE	アクセスパス情報ファイルを HiRDB クライアント側に出力する場合に，アクセスパス情報ファイルのファイル名の形式を指定します。	

環境変数名	機能	環境変数の分類
PDTAAPINFSIZE	アクセスパス情報ファイルを HiRDB クライアント側に出力する場合に，アクセスパス情報ファイルのファイルサイズを指定します。	
PDSTJTRNOUT	UAP に関する統計情報を，トランザクションごとに統計ログファイルに出力するかどうかを指定します。	UAP に関する統計情報の出力単位
PDLOCKLIMIT	一つのサーバに対して UAP から発行する排他要求の上限値を指定します。	排他制御
PDDLKPRIO	UAP のデッドロックプライオリティ値を指定します。	
PDLOCKSKIP	無排他条件判定をするかどうかを指定します。	
PDFORUPDATEEXLOCK	FOR UPDATE 句を指定した（又は仮定された）SQL の排他オプションに，WITH EXCLUSIVE LOCK を適用するかどうかを指定します。	
PDISLLVL	SQL 文のデータ保証レベルを指定します。	SQL 関連
PDSQLOPTLVL※5	データベースの状態を考慮して，最も効率的なアクセスパスを決定するための最適化の方法（SQL 最適化オプション）を指定します。	
PDADDITIONALOPTLVL※5	データベースの状態を考慮して，最も効率的なアクセスパスを決定するための最適化の方法（SQL 拡張最適化オプション）を指定します。	
PDHASHTBLSIZE	SQL の最適化で，ハッシュジョイン，副問合せのハッシュ実行を適用する場合，ハッシュ表サイズを指定します。	
PDDFLNVAL	表中のデータを埋込み変数に取り出す場合，取り出した値がナル値のときに埋込み変数に既定値を設定するかどうかを指定します。	
PDAGGR	GROUP BY 処理に使用するメモリ量を決定するため，サーバごとに発生するグループ数の最大値を指定します。	
PDCMMTBFDDL	操作系 SQL を実行していたトランザクションで定義系 SQL を実行する場合，自動的にコミットしてから定義系 SQL を実行するかどうかを指定します。	
PDPRPCRCLS	開いているカーソルで使用している SQL 識別子を再度 PREPARE 文で使用する場合，開いているカーソルを自動的にクローズするかどうかを指定します。	
PDAUTOCONNECT	HiRDB と接続していない状態で SQL 文を実行した場合，自動的に接続するかどうかを指定します。	
PDDDLDEAPRPEXE	先行トランザクションの前処理結果を無効にし，定義系トランザクションの実行を優先します。	
PDDDLDEAPRP	閉じているホールダブルカーソルで使用している表の定義情報を，トランザクション間に他 UAP からの変更を許可するかどうかを指定します。	

環境変数名	機能	環境変数の分類
PDLCKWAITTIME	排他要求が待ち状態になってから，解除されるまでを監視する最大時間を指定します。	
PDCURSORLVL	カーソルを使用した検索をする場合に，HiRDB サーバに対してのカーソルオープン・クローズの要求を，自動で行うかどうかを指定します。	
PDDELRSVWDFILE	SQL 予約語削除機能を使用する場合に，SQL 予約語削除ファイル名を指定します。	
PDCALCMDWAITTIME	CALL COMMAND 文によってコマンド，又はユティリティを開始してから終了するまでの，HiRDB クライアントの最大待ち時間を指定します。	
PDSTANDARDSQLSTATE	SQLSTATE の値を詳細に出力するかどうかを指定します。	
PDBLKF	HiRDB サーバから HiRDB クライアントに検索結果を転送するときの，一回の転送処理で送られる行数を指定します。	ブロック転送機能
PDBINARYBLKF	定義長が 32,001 バイト以上の BINARY 型の選択式がある表を検索する場合，ブロック転送機能を適用するかどうかを指定します。	
PDBLKBUFFSIZE	ブロック転送機能で使用する，サーバ，クライアント間の通信バッファのサイズを指定します。	
PDBINDRETRYCOUNT	UNIX ドメインでの bind システムコールで EADDRINUSE が返却された場合のリトライ回数を指定します。	HiRDB の通信処理
PDBINDRETRYINTERVAL	UNIX ドメインでの bind システムコールで EADDRINUSE が返却された場合のリトライ間隔を指定します。	
PDDBACCS	インナレプリカ機能を使用している場合，カレント RD エリアではない RD エリアをアクセスしたいときに，その RD エリアの世代番号を指定します。	インナレプリカ機能
PDDBORGUAP	オンライン再編成閉塞のオリジナル RD エリアに対して UAP を実行する場合に指定します。	更新可能なオンライン再編成
PDSPACELVL	データの格納，比較，及び検索時の，空白変換レベルを指定します。	データの空白変換
PDCLTRDNODE	XDM/RD E2 接続機能使用時に，接続する XDM/RD E2 のデータベース識別子を指定します。	XDM/RD E2 接続機能
PDTP1SERVICE	XDM/RD E2 接続機能使用時に，XDM/RD E2 に OpenTP1 のサービス名を通知するかどうかを指定します。	
PDRDCLTCODE	XDM/RD E2 接続機能使用時に，クライアントで使用する文字コード種別を指定します。	

環境変数名	機能	環境変数の分類
PDCNSTRNTNAME	参照制約，及び検査制約を定義する場合，制約名定義の位置を指定します。	参照制約及び検査制約
PDTMPTBLRDAREA※5	一時表及び一時インデクスを格納する RD エリアの候補を指定します。	一時表
PDBESCONHOLD	バックエンドサーバ接続保持機能を使用するかどうかを指定します。	バックエンドサーバ接続保持機能
PDBESCONHTI	バックエンドサーバ接続保持機能を使用する場合，バックエンドサーバ接続保持期間を指定します。	
PDODBSTATCACHE	ODBC 関数の SQLColumns()関数，SQLStatistics()関数で，1 度発行して取得したカラム情報，又はインデクス情報をキャッシュするかどうかを指定します。	ODBC 関数
PDODBESCAPE	カタログ系の ODBC 関数の検索で，パターン文字列に対して ESCAPE 文字（'＆'）を指定するかどうかを指定します。	
PDGDATAOPT	ODBC 関数の SQLGetData 関数で，データを取得済みの列に対して，繰り返しデータを取得する場合に指定します。	
PDODBLOCATOR	DB アクセス部品を使用して，BLOB 型，又は BINARY 型の列を検索する場合に，位置付け子機能を使用してデータを分割取得するかどうかを指定します。DB アクセス部品とは，ODBC ドライバ，OLE DB プロバイダ，及び HiRDB データプロバイダ for .NET Framework を示します。	
PDODBSPLITSIZE	PDODBLOCATOR=YES を指定している場合，分割取得のサイズを指定します。	
PDODBCWRNSKIP	ODBC，OLE DB 接続時のワーニングを回避するかどうかを指定します。	
PDJETCOMPATIBLE	ODBC3.5 ドライバを，ODBC3.5 の規格ではなく Microsoft Access 互換モードで動作させるかどうかを指定します。	
PDODBGINFOSUPPRESS	ODBC 関数の SQLGetInfoW()関数使用時に発生するエラーを回避するために指定します。	
PDODBSTANDARDARGSIZE	64 ビットモードで動作する ODBC3.5 ドライバについて，一部の ODBC 関数の引数に指定されたポインタが指すバッファを 4 バイト整数として扱うか，8 バイト整数として扱うかを指定します。	
PDODBSTANDARDSQLSTATE	ODBC3.5 ドライバから ODBC 規格に準拠した SQLSTATE を返却するかどうかを指定します。	
PDODBSTANDARDDESCCOL	ODBC3.5 ドライバでの SQLDescribeCol 関数の実行時に，ODBC 規格に準拠した値を返却するかどうかを指定します。	

環境変数名	機能	環境変数の分類
PDODBSTANDARDGTYPEINFO	ODBC3.5 ドライバの SQLGetTypeInfo 関数で，ODBC 規格に準拠した値を返却するかどうかを指定します。	
PDODBERRTRC	ODBC3.5 ドライバでのエラーログの出力有無を指定します。	
PDODBERRTRCSIZE	ODBC3.5 ドライバでのエラーログの最大ファイルサイズをバイト単位で指定します。	
PDODBSYSUNSIGNED	ODBC3.5 ドライバが提供するシステム情報取得関数の結果セット列に対する SQLColAttribute（SQL_DESC_UNSIGNED）関数の返却値を変更する場合に指定します。	
PDODBSYSKEYS	ODBC3.5 ドライバが提供する SQLPrimaryKeys 関数及び SQLForeignKeys 関数の，ODBC 規格で規定された結果セットを返却する場合に指定します。	
PDPLGIXMK	プラグインインデクスの遅延一括作成を使用するかどうかを指定します。	プラグイン
PDPLUGINNSUB※2	詳細については，各プラグインマニュアルを参照してください。	
PDPLGPFSZ	プラグインの遅延一括作成用のインデクス情報ファイルの初期容量を指定します。	
PDPLGPFSZEXP	プラグインの遅延一括作成用のインデクス情報ファイルの増分値を指定します。	
PDHSICOPTIONS	HSIC を使用する運用の場合に，HSIC で集積する情報を指定します。	HSIC 限定
PDJDBFILEDIR	Type4 JDBC ドライバでの Exception トレースログ及び不正電文トレースのファイル出力先を指定します。	JDBC ドライバ
PDJDBFILEOUTNUM	Type4 JDBC ドライバでの Exception トレースログのファイルへの出力数を指定します。	
PDJDBONMEMNUM	Type4 JDBC ドライバでの Exception トレースログのメモリ内取得情報数を指定します。	
PDJDBTRACELEVEL	Type4 JDBC ドライバでの Exception トレースログのトレース取得レベルを指定します。	
PDJDBFILESIZE	Type4 JDBC ドライバでの Exception トレースログの最大ファイルサイズを指定します。	
PDDNDPCOMPATIBLE	ADO.NET2.0 に対応した HiRDB データプロバイダ for .NET Framework のメソッドが，.NET Framework の規格に準拠した仕様とするか，又は HiRDB 独自の仕様とするかを指定します。	ADO.NET

注※1

OLTP 下の X/Open に従った API を使用して，HiRDB サーバをアクセスするクライアントの場合だけ指定します。そのほかの場合は，指定しても無効になります。

各環境変数が必要かどうかについては，「OLTP 下の X/Open に従った API を使用した UAP をクライアントとする場合の指定方法」を参照してください。

なお，複数接続機能を使用する場合，接続先ごとに環境変数のグループ登録をしても，環境変数の指定値は無効になります。

注※2

プラグイン用に設定する環境変数です。この環境変数の設定内容については，クライアントライブラリではチェックされません。また，SQL トレースにも情報は出力されません。

注※3

システム構成に関する環境変数には，HiRDB サーバと接続するときに必要な情報を指定します。HiRDB サーバとの接続形態によっては，環境変数が指定できたりできなかったりします。HiRDB サーバとの接続形態については，「HiRDB サーバと接続するための環境変数と接続形態との関係」を参照してください。

注※4

CALL COMMAND 文の ENVIRONMENT オペランドに指定できます。

注※5

指定値に引用符（ " ）で囲む値を含む場合の注意事項を次に示します。

- UNIX のシェルで OS 環境変数にこのクライアント環境変数を指定する場合は，指定値全体をアポストロフィ（ ' ）で囲む必要があります。

  sh（ボーンシェル）での指定例

  export PDTMPTBLRDAREA=’”rdtmp01”,”rdtmp02”’

- HiRDB への接続時に使用する HiRDB.ini，又は環境変数グループ（ファイル登録）にこのクライアント環境変数を指定する場合は，指定値全体をアポストロフィ（ ' ）で囲む必要があります。ただし，Type4 JDBC ドライバを使用する場合，その必要はありません。

  なお，環境変数グループ（レジストリ登録）にこのクライアント環境変数を指定する場合は，指定値をアポストロフィ（ ' ）で囲むとエラーになります。

  HiRDB.ini での指定例

  PDTMPTBLRDAREA=’”rdtmp01”,”rdtmp02”’

- UNIX のシェルで Java の-D オプションにこのクライアント環境変数に対応するシステムプロパティを指定する場合は，指定値全体をアポストロフィ（ ' ）で囲む必要があります。

  Java コマンドでの指定例

  java -DHiRDB_for_Java_PDTMPTBLRDAREA=
  ’”rdtmp01”,”rdtmp02”’ sample

- Windows のコマンドプロンプトで Java の-D オプションにこのクライアント環境変数に対応するシステムプロパティを指定する場合は，RD エリア名を¥"で囲んで指定値全体を引用符（ " ）で囲む必要があります。

  Java コマンドでの指定例

  java -DHiRDB_for_Java_PDTMPTBLRDAREA=

  ”¥”rdtmp01¥”,¥”rdtmp02¥”” sample

6.6.4　クライアント環境定義の設定内容

(1)　PDHOST=HiRDB サーバのホスト名［，予備系 HiRDB サーバのホスト名］

〜<識別子>((最大 511 バイト))

接続する HiRDB サーバのホスト名を指定します。

HiRDB/シングルサーバの場合はシングルサーバのサーバマシンのホスト名を，HiRDB/パラレルサーバの場合はシステムマネジャのサーバマシンのホスト名を指定します。また，PDFESHOST を指定している場合は，PDFESHOST のホスト名を指定できます。PDFESHOST のホスト名を指定した場合，システムマネジャユニットに障害が発生しても，HiRDB サーバに接続できます。

ホスト名以外にも，FQDN，及び IP アドレスでも指定できます。指定方法を次に示します。

ホスト名：

システム共通定義の pdunit -x オペランドで指定したホスト名を指定します。

（指定例）

```
PDHOST=host1
```

FQDN：

HiRDB サーバのホスト名とドメイン名とを，ピリオドで結んで指定します。

（指定例）

```
PDHOST=host1.soft.hitachi.co.jp
```

IP アドレス：

IP アドレスは，バイトごとにピリオドで区切られた 10 進数で指定します。

（指定例）

```
PDHOST=172.18.131.34
```

《IP アドレスを引き継ぐ系切り替えをする場合》

- UNIX 版の場合

  現用系のホスト名を指定してください。

- Windows 版の場合

MSFC のネットワーク名に登録した仮想ネットワーク名を指定してください。仮想ネットワークについては，マニュアル「HiRDB システム運用ガイド」を参照してください。

《IP アドレスを引き継がない系切り替えをする場合》

- 現用系及び予備系の二つのホスト名を指定してください。現用系のホスト名だけを指定すると，系が切り替わった後に，この環境変数の指定を新しく実行系になったホスト名に変更する必要があります。
- 現用系及び予備系の二つのホスト名を指定した場合，実行系が現用系から予備系に切り替わると，UAP は一度現用系（待機系）に接続を試みて失敗した後に予備系（実行系）に接続するため，UAP の接続処理時間がその分だけ長くなります。この問題を解決するには，次のクライアント環境定義を指定して，接続先にするホストの切り替え時間を短くしてください。

  ブロックモードで接続する場合

  ・PDTIMEDOUTRETRY

  ・PDCONREFRCOUNT

  ・PDCONREFRINTERVAL

  ノンブロックモードで接続する場合

  ・PDNBLOCKWAITTIME

  ・PDCONREFRCOUNT

  ・PDCONREFRINTERVAL

《OLTP 下で X/Open に従ったアプリケーションプログラムをクライアントとし，システム環境定義で HiRDB_PDHOST を指定している場合》

HiRDB_PDHOST の指定を優先します。PDHOST の設定値は HiRDB_PDHOST で指定した値に置き換わります。

《FQDN を指定する場合の規則》

バージョン 05-03 より前の HiRDB サーバと接続する場合は，FQDN を指定しないでください。指定した場合，クライアントの最大待ち時間（PDCWAITTIME）経過後，HiRDB サーバへのキャンセル処理ができないでサーバプロセスが残ることがあります。

《システム共通定義の pdunit -x オペランドにループバックアドレスを指定している場合》

システム共通定義の pdunit -x オペランドにループバックアドレスを指定している場合は，この環境変数にも同じループバックアドレスを，IP アドレス形式で指定してください。

(2) PDNAMEPORT=HiRDB サーバのポート番号

～<符号なし整数>((5001～65535))《20000》

接続する HiRDB サーバのポート番号を指定します。PDHOST に指定したホストのサーバマシンの，アクセスする HiRDB サーバのポート番号を指定してください。

マルチ HiRDB の場合，それぞれの HiRDB サーバでポート番号が異なります。したがって，接続する HiRDB サーバのポート番号を指定してください。

《OLTP 下で X/Open に従ったアプリケーションプログラムをクライアントとし，システム環境定義で HiRDB_PDNAMEPORT を指定している場合》

HiRDB_PDNAMEPORT の指定を優先します。PDNAMEPORT の設定値は HiRDB_PDNAMEPORT で指定した値に置き換わります。

(3) PDFESHOST=フロントエンドサーバのホスト名［：フロントエンドサーバがあるユニットのポート番号］［，予備系フロントエンドサーバのホスト名［：予備系フロントエンドサーバがあるユニットのポート番号]]

～<識別子>((最大 523 バイト))

この環境変数は，HiRDB/パラレルサーバに関するものです。

マルチフロントエンドサーバの場合に，接続する HiRDB サーバのフロントエンドサーバのホスト名を指定します。また，システム定義の pdunit で-p ポート番号を指定しているホストへ接続する場合（系切り替え機能を使用している場合）は，そのポート番号を指定する必要があります。

省略した場合，接続するフロントエンドサーバはシステムマネジャが決定します。システムマネジャが決定するフロントエンドサーバには，回復不要 FES も含まれます。

ホスト名以外にも，FQDN，及び IP アドレスでも指定できます。指定方法を次に示します。

ホスト名：

システム共通定義の pdunit -x オペランドで指定したホスト名を指定します。

（指定例）

```
PDFESHOST=host1
```

FQDN：

HiRDB サーバのホスト名とドメイン名とを，ピリオドで結んで指定します。

（指定例）

```
PDFESHOST=host1.soft.hitachi.co.jp
```

IP アドレス：

IP アドレスは，バイトごとにピリオドで区切られた 10 進数で指定します。

（指定例）

```
PDFESHOST=172.18.131.34
```

《IP アドレスを引き継ぐ系切り替えをする場合》

- UNIX 版の場合
  現用系のホスト名を指定してください。
- Windows 版の場合
  MSFC のネットワーク名に登録した仮想ネットワーク名を指定してください。仮想ネットワークについては，マニュアル「HiRDB システム運用ガイド」を参照してください。

《IP アドレスを引き継がない系切り替えをする場合》

現用系，及び予備系の二つのホスト名を指定してください。現用系のホスト名だけを指定すると，系が切り替わった後に，この環境変数の指定を新しく実行系になったホスト名に変更する必要があります。

《FQDN を指定する場合の規則》

バージョン 05-03 より前の HiRDB サーバと接続する場合は，FQDN を指定しないでください。指定した場合，クライアントの最大待ち時間（PDCWAITTIME）経過後，HiRDB サーバへのキャンセル処理ができないでサーバプロセスが残ることがあります。

《ポート番号を省略する場合の規則》

ポート番号を省略すると，PDNAMEPORT で指定したポート番号が仮定されます。予備系フロントエンドサーバがあるホストのポート番号を省略した場合も，PDNAMEPORT で指定したポート番号が仮定されます。

《ほかの環境変数との関係》

1. マルチフロントエンドサーバの場合にコネクトするフロントエンドサーバをクライアントユーザで決定するとき，又は PDSERVICEPORT を指定するときは，この環境変数を必ず指定します。
2. この環境変数を指定するときは，PDSERVICEGRP も指定してください。

《留意事項》

1. X/Open XA インタフェースを使用するプログラムから回復不要 FES に接続した場合，そのプログラムからはデータベースの参照，更新ができません。この場合，PDFESHOST 及び PDSERVICEGRP を指定して，必ず回復不要 FES でないフロントエンドサーバに接続してください。
2. マルチフロントエンドサーバの場合，接続先のフロントエンドサーバに負荷が集中しないようにするために，PDFESHOST に指定するホスト名は均等になるようにしてください。
3. PDFESHOST に指定したホスト名を，PDHOST にも指定できます。この場合，システムマネジャユニットに障害が発生しても，HiRDB サーバに接続できます。
4. 反映側 Datareplicator の同期点処理方式に二相コミット方式を利用（反映システム定義の commitment_method オペランドに fxa_sqle を指定）した反映処理を実行する場合，反映側 HiRDB の回復不要 FES に接続すると，反映処理が失敗します。この場合，反映側 Datareplicator の環境変数に PDFESHOST 及び PDSERVICEGRP を指定して，必ず回復不要 FES でないフロントエンドサーバに接続してください。

(4) PDSERVICEGRP=サーバ名

～<文字列>((最大 30 バイト))

接続する HiRDB サーバの，シングルサーバ名又はフロントエンドサーバ名を指定します。

HiRDB/パラレルサーバでマルチフロントエンドサーバの場合，接続するフロントエンドサーバのサーバ名を指定してください。

《ほかの環境変数との関係》

1. 同時に PDSERVICEPORT を指定することで，HiRDB サーバへの接続時間を短縮できます（高速接続機能）。
2. HiRDB/パラレルサーバの場合，PDFESHOST も指定してください。

《留意事項》

1. X/Open XA インタフェースを使用するプログラムから回復不要 FES に接続した場合，そのプログラムからはデータベースの参照，更新ができません。この場合，PDSERVICEGRP 及び PDFESHOST を指定して，必ず回復不要 FES でないフロントエンドサーバに接続してください。
2. 反映側 Datareplicator の同期点処理方式に二相コミット方式を利用（反映システム定義の commitment_method オペランドに fxa_sqle を指定）した反映処理を実行する場合，反映側 HiRDB の回復不要 FES に接続すると，反映処理が失敗します。この場合，反映側 Datareplicator の環境変数に PDSERVICEGRP 及び PDFESHOST を指定して，必ず回復不要 FES でないフロントエンドサーバに接続してください。

(5) PDSRVTYPE={WS ｜ PC}

接続する HiRDB サーバのサーバ種別を指定します。

WS：

HiRDB サーバが AIX 版の場合に指定します。

PC：

HiRDB サーバが Linux 版，又は Windows 版の場合に指定します。

(6) PDSERVICEPORT=高速接続用のポート番号［，予備系の高速接続用ポート番号］

～<符号なし整数>((5001～65535))

高速接続機能を使用する場合の高速接続用のポート番号を指定します。高速接続用のポート番号とは，システム定義に指定するスケジューラプロセスのポート番号及びスケジューラプロセスの暗号化通信接続用のポート番号のことです。スケジューラプロセスのポート番号を指定するシステム定義のオペランドを次に示します。

- pd_service_port オペランド
- pd_scd_port オペランド
- pdunit オペランドの-s オプション

スケジューラプロセスの暗号化通信接続用のポート番号は，システム定義の pd_crypto_service_port オペランドで指定します。

各オペランドについては，マニュアル「HiRDB システム定義」を参照してください。

HiRDB サーバ側にファイアウォールや NAT が設置されている場合は，このオペランドを指定してください。ファイアウォールや NAT が設置されている場合の設定については，マニュアル「HiRDB システム導入・設計ガイド」を参照してください。

また，マルチフロントエンドサーバの場合，接続するフロントエンドサーバの高速接続用のポート番号を指定してください。

なお，接続するフロントエンドサーバが固定されるため，サーバ負荷が分散されるように接続先を決定してください。

《利点》

この環境変数を指定することで，HiRDB サーバへの接続時間を短縮できます。

《ほかの環境変数との関係》

この環境変数を指定する場合，次に示す環境変数も必ず指定してください。また，HiRDB サーバが Windows，又は Linux の場合，環境変数 PDSRVTYPE に PC を指定してください。

＜ HiRDB/シングルサーバの場合＞

- PDHOST
- PDNAMEPORT
- PDSERVICEGRP

＜ HiRDB/パラレルサーバの場合＞

- PDHOST
- PDNAMEPORT
- PDFESHOST
- PDSERVICEGRP

《留意事項》

1. 相互系切り替えの系切り替え機能を使用していて，システム定義の pd_service_port オペランドでそれぞれ異なるポート番号を指定している場合，予備系の高速接続用ポート番号も併せて指定してください。
2. 広域 LAN を経由して HiRDB サーバに接続する環境で通信障害が発生した場合，システムマネジャプロセスが影響を受けて，HiRDB への接続要求を多数同時に受け付けられないおそれがあります。そのため，広域 LAN を経由して接続するシステムの場合は，この環境変数を指定することをお勧めします。

(7) PDFESGRP=FES グループ［，切替 FES グループ［，切替 FES グループ］...］

～<文字列>((最大 1024 バイト))

この環境変数は，HiRDB/パラレルサーバで，高速接続をする場合に指定します。また，システム定義に次のオペランドを指定している必要があります。

- pd_service_port オペランド
- pd_scd_port オペランド
- pdunit オペランドの-s オプション
- pd_crypto_service_port オペランド

高速接続をする場合，接続する FES グループを指定します。また，マルチフロントエンドサーバ構成の場合，接続先の FES グループと，その FES グループの障害時に接続を切り替える切替 FES グループを指定します。

FES グループと切替 FES グループに指定する内容を次に示します。

FES グループ：

高速接続の接続先（PDFESHOST，PDSERVICEGRP，及び PDSERVICEPORT）をまとめて記述する指定方法のことをいいます。それぞれの接続先はコロン（：）で区切って指定します。指定例を次に示します。

```
host1:fes1:20001
```

なお，PDFESHOST には二つ（現用系と予備系）のホスト名を指定できますが，一つの FES グループには一つのホスト名しか指定できません。同様に，PDSERVICEPORT には二つのポート番号を指定できますが，一つの FES グループには一つのポート番号しか指定できません。

切替 FES グループ：

マルチフロントエンドサーバ構成の場合，接続先の FES グループのフロントエンドサーバに障害が発生したとき，接続を切り替える FES グループのことをいいます。切替 FES グループを指定した場合に障害が発生すると，切替 FES グループに接続を切り替えます。切替 FES グループを複数指定した場合は，指定した順番で接続を切り替えます。

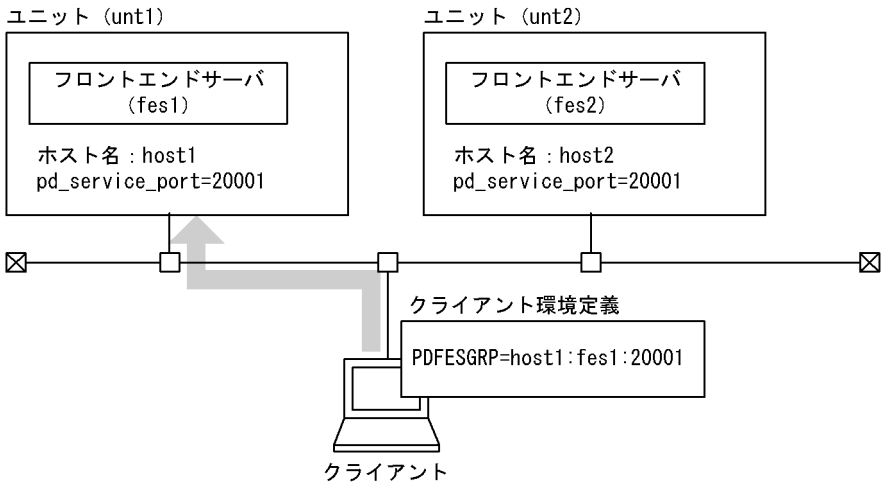
切替 FES グループの指定方法は，FES グループと同じです。

《留意事項》

1. この環境変数を指定した場合，PDFESHOST，PDSERVICEGRP，及び PDSERVICEPORT の指定は無効になります。
2. 切替 FES グループを指定する場合，サーバ障害，及び接続ユーザ数オーバーで接続先を切り替えるため，一時的に切替 FES グループ側の接続数が増えることがあります。そのため，切替 FES グループ側の pd_max_users オペランドの値を見直す必要があります。
3. 切替 FES グループを指定する場合，指定したすべての切替 FES グループで障害が発生しているとき，又は接続ユーザ数オーバーが発生しているときは，UAP にエラーを返却するのに時間が掛かることがあります。
4. 切替 FES グループを指定した場合で，複数の FES への接続がすべてエラーとなったときは，最後に接続を試みた FES のエラー情報が UAP に返却されます。

《使用例》

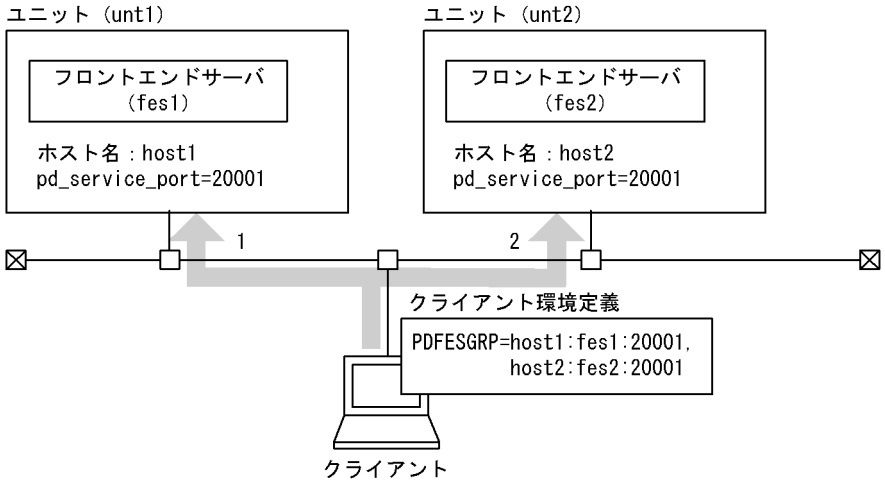
FES グループを一つだけ指定する場合：



[説明]

FES グループを一つだけ指定した場合，ホスト host1 のフロントエンドサーバ fes1 にだけ接続します。

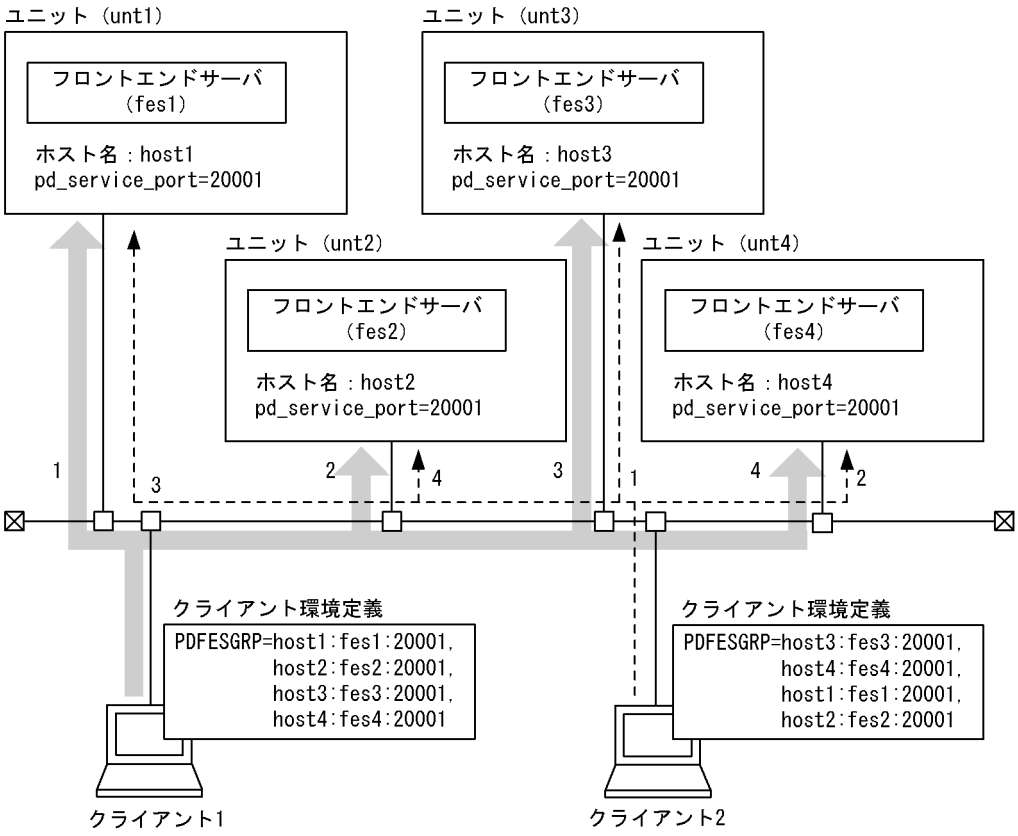
FES グループを一つ，切替 FES グループを一つ指定する場合：



[説明]

1 の接続でエラーとなった場合，2 で接続をします。2 もエラーとなった場合は，UAP にエラーを返却します。

FES グループを一つ，切替 FES グループを複数指定する場合：



[説明]

1 の接続でエラーとなった場合，2 で接続をします。以降 2，3，4 と接続をして，すべての接続がエラーとなった場合は，UAP にエラーを返却します。

(8)　PDCLTRCVPORT=クライアントの受信ポート番号

～<符号なし整数>((0，5001～65535，5001～65535-5001～65535))《0》

HiRDB クライアントが HiRDB サーバと通信する場合の，受信ポート番号又は受信ポート番号の範囲を指定します。

この環境変数を省略した場合，自動的に OS によって空いているポート番号が割り当てられます。したがって，通常時はこの環境変数を指定する必要はありません。

《指定方法》

受信ポート番号の指定例を次に示します。

- ポート番号を一つ指定する場合

  10000-10000，又は 10000

- ポート番号の範囲を指定する場合

  10000-10500

なお，0 を指定した場合は，この環境変数を指定しないことになります。

《利点》

HiRDB サーバと HiRDB クライアントの間にファイアウォールが設定されていて，ファイアウォールを通過できる受信ポート番号が限られている場合，この環境変数を指定することでファイアウォールを通過するようにできます。

《留意事項》

1. この環境変数に受信ポート番号の範囲を指定した場合，HiRDB クライアントが指定した範囲で空いているポート番号を自動的に割り当てます。指定した範囲で空いているポート番号がない場合はエラーとなります。
2. HiRDB クライアントは，HiRDB サーバへの一回の接続で一つのポート番号を使用します。したがって，次のような場合には，一つの UAP で複数のポート番号を使用することになります。
   ・ODBC で複数の接続を使用する場合
   ・複数接続機能を使用している場合
3. 同時に複数の UAP を実行する場合，一つのポート番号は一つの UAP でしか使用できません。したがって，同時に実行する複数の UAP に対して，同じポート番号を含む範囲を指定すると，ポート番号の割り当てが競合する可能性があります。この場合，ポート番号が不足しないように，使用するポート番号の個数の最大値より大きな個数のポート番号を含む範囲を指定してください。
4. 指定する受信ポート番号は，OS が自動的に割り当てるポート番号の範囲と重複しないようにしてください。OS が自動的に割り当てるポート番号の範囲は，OS ごとに異なります。
5. ODBC で Microsoft Access などを経由している場合は，暗黙的に HiRDB サーバと複数接続します。
6. 多数（10 個程度以上）のポート番号を含む範囲を指定する場合は，その範囲で実際に使用されるポート番号の数に対して 20%程度の余裕を持たせてください。余裕がないと，空いているポート番号を探す処理で効率が低下します。
7. HiRDB クライアント以外のプログラムが使用しているポート番号は，HiRDB クライアントでは使用できません。
8. HiRDB クライアントが使用しているポート番号は，HiRDB クライアント以外のプログラムでは使用できません。この環境変数に指定した範囲内のポート番号を固定的に使用するサービスがある場合，そのサービスを起動できなくなる可能性があります。
9. HiRDB クライアント用にファイアウォールを通過できるように設定されたポート番号が，HiRDB クライアント以外のプログラムから不正に使用されないように，ファイアウォールの内側のプログラムを管理してください。

(9) PDCLTRCVADDR=｛クライアントの IP アドレス｜クライアントのホスト名｝

～<符号なし整数>又は<識別子>((最大 255 バイト))

HiRDB クライアントのホストに複数の通信経路が設定されている場合，HiRDB サーバと通信をするための通信経路を特定したいときに，その通信経路に対応する IP アドレス，FQDN，又はホスト名を指定します。指定方法を次に示します。

IP アドレス：

バイトごとにピリオドで区切られた 10 進数で指定します。

（指定例）

```
PDCLTRCVADDR=172.18.131.34
```

FQDN：

HiRDB クライアントのホスト名とドメイン名とを，ピリオドで結んで指定します。

（指定例）

```
PDCLTRCVADDR=host1.soft.hitachi.co.jp
```

《留意事項》

1. この環境変数を省略した場合は，クライアントマシンの標準ホスト名に対応する IP アドレスが仮定されます。クライアントマシンの標準ホスト名は，hosts ファイル又は DNS などに 255 バイト以内で登録してください。標準ホスト名が hosts ファイル又は DNS などに登録されていない場合はエラーになります。ただし，HiRDB/シングルサーバで，HiRDB クライアントと HiRDB サーバを同一マシンで構成している場合に，標準ホスト名が hosts ファイル又は DNS などに登録されていないときは，HiRDB サーバのホスト名の IP アドレスが仮定されます。
2. この環境変数に不正な IP アドレス又はホスト名を指定した場合，HiRDB サーバへの CONNECT 時に HiRDB サーバからの応答が受け付けられないため，5 分間のタイマ監視後にエラー（SQLCODE　-732）となります。
3. 次の場合は，PDCLTRCVADDR の指定は無効になります。
   ・システム定義の pd_change_clt_ipaddr オペランドに 1 を指定している場合
   ・環境変数 PDIPC に MEMORY を指定している場合
4. PDCLTRCVADDR と PDHOST の指定値の組み合わせと，使用されるクライアントの IP アドレスを次に示します。

<table>
<tr><th colspan="3" rowspan="2">PDCLTRCVADDR の指定値</th><th colspan="3">PDHOST</th></tr>
<tr><th>ループバックアドレス</th><th>ローカル IP アドレス</th><th>別マシンの IP アドレス</th></tr>
<tr><td colspan="3">ループバックアドレス</td><td colspan="3" rowspan="2">PDCLTRCVADDR の指定値</td></tr>
<tr><td colspan="3">ローカル IP アドレス</td></tr>
<tr><td rowspan="3">指定なし</td><td colspan="2">hosts ファイルへの標準ホストの登録あり</td><td colspan="3">hosts ファイルの標準ホスト</td></tr>
<tr><td rowspan="2">hosts ファイルへの標準ホストの登録なし</td><td>Type4 JDBC ドライバ</td><td>PDHOST の指定値</td><td>KFPZ02444-E エラー</td><td>KFPZ02444-E エラー</td></tr>
<tr><td>上記以外</td><td>PDHOST の指定値</td><td>PDHOST の指定値</td><td>KFPZ02444-E エラー</td></tr>
</table>

(10)　PDCONTYPE=｛ACTIVE ｜ PASSIVE｝

HiRDB 接続時の接続方式を指定します。

ACTIVE：

クライアントプロセスからサーバプロセスへ接続する方式で，HiRDB サーバと接続します。

サーバプロセスからクライアントプロセスには接続しません。

この指定は，HiRDB サーバのバージョンが，UNIX の場合は 10-07 以降，Windows の場合は 10-08 以降である場合に有効になります。

ACTIVE 指定をサポートするクライアント種別及びプラットフォームを次に示します。

クライアント			サーバプラットフォーム	
クライアント種別		プラットフォーム	UNIX	Windows
埋込み型 UAP ODBC ドライバ※ Type2 JDBC ドライバ SQLJ		UNIX Windows	○	○
OLE DB プロバイダ HiRDB データプロバイダ for .NET Framework		Windows	○	○
Type4 JDBC ドライバ	JDBC2.0 版	UNIX Windows	×	×
	JDBC4.0 版	UNIX Windows	○	○
	JDBC4.3 版	UNIX Windows	○	○

（凡例）

○：サポート

×：未サポート

注※

UNIX では Linux だけでサポートしています。

ACTIVE 指定で未サポートのサーバに接続した場合，エラーとなり KFPA11723-E メッセージが出力されます。

ACTIVE 指定の場合，クライアントプロセスとサーバプロセスが同一ホストであっても，TCP/IP で通信を行います。

ACTIVE 指定の場合，次のクライアント環境定義の指定は無効になります。

・PDIPC

・PDBINDRETRYCOUNT

・PDBINDRETRYINTERVAL

・PDCLTRCVPORT

・PDCLTRCVADDR

・PDCLTBINDLOOPBACKADDR

・PDIPCFILEDIR

ACTIVE 指定の場合，1 つのユニットに対して一斉に接続する UAP の数が 450 以上になると，接続エラーが発生する可能性があります。そのため，UAP の起動タイミングや接続するタイミングをずらすなど，一斉に接続する接続要求数を 450 未満にしてください。また，ACTIVE 指定で接続エラーが発生した場合，即座にエラーとならずにクライアント環境定義 PDCONNECTWAITTIME に指定した時間が経過するまで待ち合わせることがあります。クライアント環境定義 PDCONNECTWAITTIME の指定値の目安を参考に，適切な値を設定してください。

PASSIVE：

サーバプロセスからクライアントプロセスへ接続する方式で，HiRDB サーバと接続します（クライアントプロセスからサーバプロセスにも，接続処理の過程で一時的に接続します）。

クライアントプロセス上に作成された受信ポートに対しサーバプロセスから接続します。そのため，クライアントプロセス上に受信ポートを作成するほか，クライアントとサーバ間にファイアウォールがある場合には，サーバプロセスからクライアントプロセスへの接続についても透過させる必要があります。ファイアウォールの設定については，マニュアル「HiRDB システム導入・設計ガイド」を参照してください。

クライアント環境定義 PDCONTYPE の指定値と，クライアントプロセス上の受信ポートの作成有無及び接続方向を次に示します。

クライアント環境定義 PDCONTYPE の指定値	クライアントプロセス上の受信ポート	接続方向
ACTIVE	作成しない	クライアントプロセスからサーバプロセスへの接続だけ
PASSIVE	作成する	クライアントプロセスからサーバプロセスへの接続，及びサーバプロセスからクライアントプロセスへの接続

JDBC4.0 版 Type4 JDBC ドライバ及び JDBC4.3 版 Type4 JDBC ドライバを使用した場合に，クライアント環境定義 PDCONTYPE の指定によってファイル名称が変わるトレースファイルを次に示します。

トレース種別	トレースファイル名称	
	クライアント環境定義 PDCONTYPE に ACTIVE を指定した場合	クライアント環境定義 PDCONTYPE に PASSIVE を指定した場合
SQL トレース（クライアント環境定義 PDSQLTRCFMT に 1 を指定した場合）	pdjsqlHHMMSSfff_XXXXXXXX_YYYYYYYYYY_1.trc pdjsqlHHMMSSfff_XXXXXXXX_YYYYYYYYYY_2.trc	pdjsqlXXXXXXXX_ppppp_1.trc pdjsqlXXXXXXXX_ppppp_2.trc
再接続トレース	pdrcnct_HHMMSSfff_XXXXXXXX_YYYYYYYYYY_1.trc pdrcnct_HHMMSSfff_XXXXXXXX_YYYYYYYYYY_2.trc	pdrcnct_ppppp_1.trc pdrcnct_ppppp_2.trc
WRITE LINE 文の値式の値を出力するファイル	pdwrtlnHHMMSSfff_XXXXXXXX_YYYYYYYYYY_1.trc pdwrtlnHHMMSSfff_XXXXXXXX_YYYYYYYYYY_2.trc	pdwrtlnXXXXXXXX_ppppp_1.trc pdwrtlnXXXXXXXX_ppppp_2.trc

（凡例）

HHMMSSfff：コネクト要求時間（HH：時，MM：分，SS：秒，fff：ミリ秒）

XXXXXXXX：接続サーバ名

YYYYYYYYYY：コネクト通番

ppppp：クライアント受信ポート番号

(11) PDTMID=OLTP 識別子

～<識別子>((4 文字))

複数の OLTP から X/Open に従った API を使用して一つの HiRDB サーバをアクセスする場合，それぞれの OLTP にユニークな識別子を指定します。

なお，この環境変数の指定で次に示す条件のどれかに該当する場合，どの OLTP からのトランザクションであるかが識別されないため，OLTP 内でシステムダウンやトランザクション異常が発生すると，トランザクション決着の同期が合わなくなります。

- 複数の OLTP からアクセスする運用形態で，この環境変数を省略した場合
- 複数の OLTP からアクセスする運用形態で，OLTP ごとに指定する識別子がユニークでない場合
- 一つの OLTP 内で OLTP 識別子が同一でない場合

《OLTP 下で X/Open に従ったアプリケーションプログラムをクライアントとし，システム環境定義で HiRDB_PDTMID を指定している場合》

HiRDB_PDTMID の指定を優先します。PDTMID の設定値は HiRDB_PDTMID で指定した値に置き換わります。

(12) PDXAMODE={0 | 1}

この環境変数は，OLTP 下の X/Open に従った API を使用した UAP と連携する場合に，トランザクションの移行機能を使用するかしないかを指定します。

0：トランザクションの移行機能を使用しません。

1：トランザクションの移行機能を使用します。

なお，この環境変数の指定値については，HiRDB 管理者の指示に従ってください。トランザクションの移行機能については，マニュアル「HiRDB システム導入・設計ガイド」を参照してください。

《OLTP 下で X/Open に従ったアプリケーションプログラムをクライアントとし，システム環境定義で HiRDB_PDXAMODE を指定している場合》

HiRDB_PDXAMODE の指定を優先します。PDXAMODE の設定値は HiRDB_PDXAMODE で指定した値に置き換わります。

《OpenTP1 と連携している場合》

OpenTP1 の trnstring オペランドと PDXAMODE の指定を合わせる必要があります。

《TPBroker for C++と連携している場合》

TPBroker for C++のトランザクションの決着は，UAP とは異なるトランザクション決着プロセスを使用します。このため，PDXAMODE には 1 を指定する必要があります。0 を指定した場合，UAP のトランザクションは決着できません。

《TUXEDO と連携している場合》

TUXEDO のグローバルトランザクションの決着は，UAP とは異なるトランザクションマネジャサーバ（TMS）を使用します。このため，PDXAMODE には 1 を指定してください。0 を指定した場合，UAP のトランザクションは決着できません。

《TP1/EE と連携している場合（UNIX 版限定）》

PDXAMODE には 0 を指定してください。省略した場合，及び 1 を指定した場合，トランザクションを決着できなくなることがあります。

(13) PDTXACANUM=1 プロセス当たりのトランザクション最大同時実行数

～＜符号なし整数＞((1～2147483647))《20》

マルチスレッド対応の X/Open に従った API を使用した UAP，又は X/Open に従った API の複数接続機能を使用した UAP から HiRDB をアクセスする場合，1 プロセス当たりに同時実行できる最大トランザクション数を指定します。

《見積もり方法》

指定値は，次の計算式から見積もってください。

```
指定値＝（該当するプロセスで発生する可能性があるトランザクション数）
　　×（該当するプロセスでアクセスする可能性があるHiRDB数）
```

TP1/EE と連携している場合：

TP1/EE の回復スレッド及び監視スレッドのスレッド数も見積もりに加えてください。

```
指定値＝（該当するプロセスで発生する可能性があるトランザクション数
　　＋回復スレッド及び監視スレッドのスレッド数）
　　×（該当するプロセスでアクセスする可能性があるHiRDB数）
```

(14) PDXARCVWTIME=トランザクションが回復できない場合の待ち合わせ時間

～<符号なし整数>((0～270))《2》（単位：秒）

X/Open に従った API で HiRDB にアクセスする OpenTP1 で，OpenTP1 のトランザクション回復プロセス，リソースマネジャ監視プロセスで HiRDB に接続できない場合，又は HiRDB がトランザクションを回復できない場合，次に HiRDB への接続要求をするまでの時間を指定します。

0 を指定した場合は，HiRDB のトランザクション回復指示ごとに接続要求をします。

《見積もり方法》

指定値は，次の計算式から見積もってください。

```
指定値＝a×b÷（c-d×e）

a：270
b：該当するHiRDBに接続するOpenTP1のトランザクション回復プロセスの総数
c：HiRDBのシングルサーバ又はシステムマネジャがあるサーバマシンの自動割り当てポート番号の総数
d：HiRDBのシングルサーバ又はシステムマネジャがあるサーバマシンの，ピーク時のポート番号使用数
e：系切り替え機能を使用している場合は2，使用していない場合は1
```

《留意事項》

1. OpenTP1 で多数のトランザクションが停止した場合，この環境変数に指定した時間が小さいと，HiRDB のシングルサーバ又はシステムマネジャがあるサーバマシンでポート番号不足になることがあります。このため，見積もり方法で算出した時間が省略時仮定値より大きい場合には，見積もり方法で算出した時間を指定することをお勧めします。
2. OpenTP1 のトランザクション回復プロセスが，この環境変数で指定した時間の待ち合わせに入った直後に，HiRDB のシングルサーバ又はシステムマネジャのユニットの開始が完了した場合，HiRDB に接続しているトランザクションの回復完了までの時間が長くなることがあります。

(15) PDXATRCFILEMODE=｛LUMP ｜ SEPARATE｝

X/Open に従った API を使用した接続形態での，各種トレースファイル名の形式を指定します。X/Open に従った API を使用した接続形態以外の場合，PDXATRCFILEMODE の指定は無効になります。

LUMP：

各種トレースファイル名に実行プロセス ID を付けないで出力します。

UAP が非常駐で何回も実行されて，プロセス ID がその都度変わる場合には，LUMP を指定することをお勧めします。LUMP を指定した場合，非常駐の UAP を実行するたびに各種トレースファイルが増えて，OS や他プログラムの動作が不安定になることを防げます。

なお，LUMP を指定した場合，トレース情報の出力先が限定されるため，トレース出力サイズを大きくする必要があります。また，トレース出力時に別プロセスの出力と競合するため，処理時間が長くなることがあります。

SEPARATE：

各種トレースファイル名に実行プロセス ID を付けて出力します。

UAP が常駐している場合は，SEPARATE を指定することをお勧めします。

(16) PDXAAUTORECONNECT=｛YES ｜ NO｝

TP1/EE との連携で，トランザクション開始時に HiRDB サーバとの接続状態チェックを行い，接続が切断されていた場合，自動的に再接続するかどうかを指定します。TP1/EE と HiRDB との接続が切断された要

因が，マシン障害，又はネットワーク障害ではない場合は，この環境変数を指定していなくても自動再接続を行います。

なお，次に示す場合，この環境変数の指定は無効になります。

- TP1/EE 連携ではない場合
- トランザクションマネジャに動的登録をした場合
- クライアント環境変数 PDCWAITTIME に 0 を指定している場合

YES：

トランザクション開始時に自動再接続を行います。ただし，各トランザクションの開始時に HiRDB サーバと通信を行うため，トランザクション性能に影響を与えます。

NO：

トランザクション開始時に自動再接続を行いません。マシン障害，又はネットワーク障害が要因で接続が切断された場合は，再接続されないため，SQL エラーが返却されることがあります。

《留意事項》

- HiRDB サーバとの接続状態チェックに掛かる時間は，最大でクライアント環境変数 PDCWAITTIME に指定した時間となります。
- 自動再接続に失敗した場合は，失敗の要因となったエラーを UAP に返却します。

(17) HiRDB_PDHOST=HiRDB サーバのホスト名［，予備系 HiRDB サーバのホスト名］

～<識別子>

接続する HiRDB サーバのホスト名を指定します。この環境変数に指定した値が，PDHOST の設定値に置き換わります。

HiRDB/シングルサーバの場合はシングルサーバのサーバマシンのホスト名を，HiRDB/パラレルサーバの場合はシステムマネジャのサーバマシンのホスト名を指定します。

ホスト名以外に，FQDN，及び IP アドレスでも指定できます。指定方法を次に示します。

ホスト名：

システム共通定義の pdunit -x オペランドで指定したホスト名を指定します。

（指定例）

```
PDHOST=host1
```

FQDN：

HiRDB サーバのホスト名とドメイン名とを，ピリオドで結んで指定します。

（指定例）

```
PDHOST=host1.soft.hitachi.co.jp
```

IP アドレス：

IP アドレスは，バイトごとにピリオドで区切られた 10 進数で指定します。

（指定例）

```
PDHOST=172.18.131.34
```

《IP アドレスを引き継ぐ系切り替えをする場合》

- UNIX 版の場合
  IP アドレスを引き継ぐ場合は，現用系のホスト名を指定します。
- Windows 版の場合
  IP アドレスを引き継ぐ場合は，MSFC のネットワーク名に登録した仮想ネットワーク名を指定します。仮想ネットワークについては，マニュアル「HiRDB システム運用ガイド」を参照してください。

《IP アドレスを引き継がない系切り替えをする場合》

IP アドレスを引き継がない系切り替えをする場合，現用系及び予備系の二つのホスト名を指定してください。現用系のホスト名だけを指定すると，系が切り替わった後に，この環境変数の指定を新しく実行系になったホスト名に変更する必要があります。

(18) HiRDB_PDNAMEPORT=HiRDB サーバのポート番号

～<符号なし整数>((5001～65535))

HiRDB サーバのポート番号を指定します。接続する HiRDB サーバのシステム定義の pd_name_port で指定した値を指定してください。この環境変数に指定した値が，PDNAMEPORT の設定値に置き換わります。

マルチ HiRDB の場合，それぞれの HiRDB サーバでポート番号が異なります。したがって，接続する HiRDB サーバのポート番号を指定してください。

pd_name_port については，マニュアル「HiRDB システム定義」を参照してください。

(19) HiRDB_PDTMID=OLTP 識別子

～<識別子>((4 文字))

複数の OLTP から X/Open に従った API を使用して一つの HiRDB サーバをアクセスする場合，それぞれの OLTP にユニークな識別子を指定してください。この環境変数に指定した値が，PDTMID の設定値に置き換わります。

なお，この環境変数の指定で次に示す条件のどれかに該当する場合，どの OLTP からのトランザクションであるかが識別されないため，OLTP 内でシステムダウンやトランザクション異常が発生すると，トランザクション決着の同期が合わなくなります。

- 複数の OLTP からアクセスする運用形態で，この環境変数を省略し，PDTMID の指定も省略した場合
- 複数の OLTP からアクセスする運用形態で，OLTP ごとに指定する識別子がユニークでない場合

(20) HiRDB_PDXAMODE={0 | 1}

OLTP 下の X/Open に従った API を使用する UAP をクライアントとする場合に，トランザクションの移行機能を使用するかしないかを指定します。この環境変数に指定した値が，PDXAMODE の設定値に置き換わります。

0：トランザクションの移行機能を使用しません。

1：トランザクションの移行機能を使用します。

なお，この環境変数の指定値については，HiRDB 管理者の指示に従ってください。トランザクションの移行機能については，マニュアル「HiRDB システム導入・設計ガイド」を参照してください。

(21) PDUSER=認可識別子［/パスワード］

- **Type4 JDBC ドライバ使用時**
  ～＜文字列＞
- **クライアントライブラリ使用時**
  - UNIX 環境の場合
    ～＜文字列＞《パスワードなしのカレントユーザ名》
  - Windows 環境の場合
    ～＜文字列＞

認可識別子，及びパスワードを指定します。「認可識別子/パスワード」の形式で指定します。パスワードの指定が必要ない（パスワードのないユーザに対して設定する）場合は，パスワードを省略できます。

認可識別子，パスワードはそれぞれ大文字，小文字の指定に関係なく大文字として扱われます。ただし，小文字を引用符で囲んだ場合は，小文字として扱われます。

《注意事項》

- OpenTP1 を使用する場合は，システム環境変数に PDUSER を登録しないでください。システム環境変数に PDUSER を登録すると，OpenTP1 起動時にアボートコード psti0rf を出力して HiRDB が終了します。

《留意事項》

1. OpenTP1 下の UAP をクライアントとする運用形態の場合，「'認可識別子/パスワード'」の形式で指定します。また，認可識別子，パスワードに英小文字を使用する場合は，「'"認可識別子"/"パスワード"'」の形式で指定してください。
2. パスワードを省略する場合，「認可識別子」だけの形式で指定すると，ユティリティによってはパスワードの入力を要求するものがあります。このような場合は，「認可識別子/パスワード」の形式で，パスワードに任意の文字列を指定してください。また，UAP からコマンドを実行する（COMMAND EXECUTE を実行する）場合，パスワードは省略できません。

3. OS ログインユーザの簡易認証機能を使用して HiRDB に接続する場合は，ユーザ名及びパスワードの両方に半角ハイフン（ - ）を指定してください。その際，半角ハイフンを引用符（ " ）で囲むこともできます。

(22) PDCLTAPNAME=実行する UAP の識別名称

～<文字列>((30 文字))《Unknown》

HiRDB サーバに対してアクセスする UAP の識別情報（UAP 識別子）を指定します。これは，どの UAP を実行しているのかを認識するための名称です。

ここで指定した名称は，次の各情報の UAP 名称として表示されます。

- pdls コマンドの表示結果
- SQL トレースファイル
- 接続ユーザ情報ファイル（%PDDIR%¥spool¥cnctusrinf）
- UAP に関する統計情報

《留意事項》

1. UAP の識別名称に英数字以外の文字を使用した場合は，pdcancel コマンドを実行できないことがありますので，英数字だけで構成される名称にすることをお勧めします。
2. 次の文字列は，UAP の識別名称に使用しないでください。
   ・「pd」で始まる文字列※
   ・「hds」で始まる文字列
   ・「0」で始まる文字列
3. Type4 JDBC ドライバ使用時の省略値については「接続情報の優先順位」を参照してください。

注※

pd で始まる文字列を UAP の識別名称に使用した場合，その UAP はシンクポイントダンプ有効化のスキップ回数監視機能の監視対象から除外されることがあります。

(23) PDCLTLANG={SJIS ｜ CHINESE ｜ UJIS ｜ C ｜ UTF-8 ｜ CHINESE-GB18030}

埋込み型 UAP のソースファイルの記述に使われている文字コード種別を指定します。

この環境変数は，SQL プリプロセサが使用するので，SQL プリプロセサの実行時に設定します。この環境変数を UAP の実行時に設定する必要はありません。

SJIS：

シフト JIS 漢字コードを使う場合に指定します。Linux 版の場合にシフト JIS 漢字コードを使うときは，この環境変数で設定してください。

CHINESE：

EUC 中国語漢字コードを使う場合に指定します。

UJIS：

EUC 日本語漢字コードを使う場合に指定します。ただし，Windows 版の場合は UJIS を指定できません。

C：

単一バイト文字コードを使う場合に指定します。

UTF-8：

Unicode（UTF-8 又は IVS 対応 UTF-8）を使う場合に指定します。

CHINESE-GB18030：

中国語漢字コード（GB18030）を使う場合に指定します。

《留意事項》

1. PDCLTLANG に上記以外の値を設定するとエラーになります。
2. Windows 版の場合に，UJIS を設定するとエラーになります。
3. Windows 版の場合に，この環境変数を省略すると，SJIS が仮定されます。
4. UNIX 版の場合に，この環境変数を省略すると，環境変数 LANG で指定した文字コード種別が仮定されます。SQL プリプロセサの実行時に設定する環境変数 LANG については，「UNIX 環境でのプリプロセス」 を参照してください。

(24) PDLANG ＝ {UTF-8 ｜ UTF-8_IVS ｜ SJIS ｜ CHINESE ｜ CHINESE-GB18030 ｜ ANY}

《UNIX 版の HiRDB クライアントの場合》

HiRDB サーバに接続する際に，HiRDB サーバの文字コード種別（サーバが UNIX 版の場合は pdsetup コマンド，Windows 版の場合は pdntenv コマンドで指定した文字コード種別）と，クライアント側で指定した文字コード種別が一致するかチェックする場合，HiRDB サーバの文字コード種別を指定します。文字コード種別が一致しない場合はエラーとなります。

HiRDB サーバの文字コード種別をチェックしない場合は ANY を指定します。

通常は ANY を指定してください。ANY を指定すると，HiRDB サーバの文字コード種別に関係なくサーバに接続できます。

HiRDB サーバの文字コード種別をチェックする場合の詳細は，「文字コード種別についての注意事項」を参照してください。

《Windows 版の HiRDB クライアントの場合》

この環境変数は無効です。指定しても無視されます。HiRDB サーバに接続する際に，HiRDB サーバの文字コード種別をチェックしません。

(25) PDDBLOG={ALL | NO}

UAP を実行するときに，データベースの更新ログを取得するかしないかを指定します。

ALL：

ログ取得モードで UAP を実行します。

ALL を指定すると，障害対策のための運用が簡単になりますが，大量のデータを更新する場合，処理に時間が掛かります。

NO：

ログレスモードで UAP を実行します。

UAP の実行途中で異常終了した場合，このトランザクションで更新したデータベースは回復されません。NO を指定する場合，データベースの更新ログを取得しない分，処理時間を短縮できますが，UAP の実行前後にバックアップを取得する必要があるので，必ず HiRDB 管理者の許可を受けてください。

ログレスモードで UAP を実行するときの方法については，マニュアル「HiRDB システム運用ガイド」を参照してください。

次に示すログは，この環境変数の指定に関係なく取得されます。

- マスタディレクトリ，データディレクトリ，及びデータディクショナリ用 RD エリアへの更新に関するログ
- ユーザ用 RD エリアの定義情報への更新に関するログ

(26) PDEXWARN= {YES | NO}

サーバから警告付きのリターンコードを受け取るかどうかを指定します。

YES：警告付きのリターンコードを受け取ります。

NO：警告付きのリターンコードを受け取りません。

この環境変数に YES を指定した場合，SQLCODE が 0，又は＋ 100 以外をすべてエラーとして処理している UAP（ストアドプロシジャを含む）は，エラーの判定方法を変更する必要があります。エラーの判定方法については，「SQL のエラーの判定と処置」を参照してください。

(27) PDSUBSTRLEN={3 | 4 | 5 | 6 | 7 | 8 | 9 | 10}

1 文字を表現する最大バイト数を指定します。

セットアップした文字コード種別によって，推奨値と指定できる値の範囲が異なります。セットアップした文字コード種別と，対応する推奨値，指定できる値を次に示します。

HiRDB サーバをセットアップした文字コード種別	推奨値	HiRDB サーバで有効な値
utf-8	4	3～6

HiRDB サーバをセットアップした文字コード種別	推奨値	HiRDB サーバで有効な値
utf-8_ivs	8	3～10
上記以外（この環境変数は無効）	－	－

（凡例）

－：この環境変数は無効となるため，推奨値，有効な値はありません。

この環境変数は，文字コード種別を utf-8，又は utf-8_ivs でセットアップした場合にだけ有効になり，スカラ関数 SUBSTR の結果の長さに影響します。SUBSTR の詳細については，マニュアル「HiRDB SQL リファレンス」を参照してください。

《システム定義との関係》

この環境変数を省略すると，システム共通定義の pd_substr_length オペランドの指定値が仮定されます。

《注意事項》

この環境変数を指定する場合の注意事項は，マニュアル「HiRDB システム定義」の pd_substr_length オペランドを参照してください。

(28) PDCLTCNVMODE ＝ {AUTO ｜ NOUSE ｜ UJIS ｜ UJIS2 ｜ UTF8 ｜ UTF8MS ｜ UTF8_TXT ｜ UTF8_EX ｜ UTF8_EX2 ｜ UTF8MS_TXT ｜ UCS2_UJIS ｜ UCS2_UJIS2 ｜ UCS2_UTF8 ｜ UOC}

HiRDB サーバと HiRDB クライアントの文字コード種別が異なる場合，文字コードを変換するかどうかを指定します。HiRDB クライアントがシフト JIS 漢字コード及び UCS-2，HiRDB サーバが EUC 日本語漢字コード及び Unicode のときだけ文字コード変換ができます。

指定値とその内容を次に示します。

指定値	内容
AUTO	HiRDB クライアントが自動的に HiRDB サーバの文字コード種別を調査して，文字コード変換をします。HiRDB クライアントがシフト JIS 漢字コードで，HiRDB サーバが EUC 日本語漢字コード又は Unicode の場合に，文字コード変換ができます。AUTO を指定した場合，指定値としては NOUSE，UJIS，又は UTF8 が設定されます。
NOUSE	文字コード変換をしません。文字コード変換をしないで，データはそのまま受け渡しします。
UJIS	HiRDB サーバの文字コードを調査しないで，HiRDB クライアントはシフト JIS 漢字コードで，HiRDB サーバは EUC 日本語漢字コードとして，文字コード変換をします。また，可変長文字列型（VARCHAR，MVARCHAR，及び NVARCHAR）のデータを受け取る場合，SQL 記述領域が指す SQLDATA 領域は SQLLEN 分の空白でクリアします。
UJIS2	UJIS と同じですが，可変長文字列型（VARCHAR，MVARCHAR，及び NVARCHAR）のデータを受け取る場合，SQL 記述領域が指す SQLDATA 領域の空白でクリアしません。

指定値	内容
UTF8	HiRDB クライアントはシフト JIS 漢字コードで，HiRDB サーバは Unicode（UTF-8）として，文字コード変換をします。ただし，可変長文字列型（VARCHAR 及び MVARCHAR）のデータを受け取る場合，SQL 記述領域が指す SQLDATA 領域は SQLLEN 分の空白でクリアします。
UTF8MS	UTF8 と同じですが，HiRDB サーバのコード系が MS-Unicode，HiRDB クライアントのコード系が Windows 符号化文字集合として変換をします。
UTF8_TXT	UTF8 と同じですが，固定長文字列型（CHAR 及び MCHAR），可変長文字列型（VARCHAR 及び MVARCHAR）のデータは文字コード変換しません。
UTF8_EX	UTF8 と同じですが，HiRDB サーバからバックスラッシュ（0x5C）を受け取った場合，HiRDB クライアントは文字コードを変換しないで，SJIS の¥記号（0x5C）として扱います。Unicode（UTF-8）での¥記号（0xC2A5）を受け取った場合は，UTF8 指定時と同様に SJIS の¥記号（0x5C）に変換します。 HiRDB クライアントで SJIS の¥記号（0x5C）を入力した場合は，文字コードを変換しないで 0x5C を HiRDB サーバへ受け渡します。
UTF8_EX2	UTF8_EX と同じですが，HiRDB クライアントで SJIS の¥記号（0x5C）を入力した場合は，UTF8 指定時と同様に Unicode（UTF-8）での¥記号（0xC2A5）に文字コードを変換して HiRDB サーバへ受け渡します。
UTF8MS_TXT	UTF8MS と同じですが，固定長文字列型（CHAR 及び MCHAR），可変長文字列型（VARCHAR 及び MVARCHAR）のデータは文字コード変換しません。
UCS2_UJIS	HiRDB クライアントは UCS-2 で，HiRDB サーバは EUC 日本語漢字コードとして，文字コードを変換します※。HiRDB サーバの文字コードが EUC 日本語漢字コード以外の場合，HiRDB サーバ接続時にエラーとなります。UCS2_UJIS は，Unicode 対応の ODBC3.0 ドライバ，ODBC3.5 ドライバ，HiRDB データプロバイダ for .NET Framework，又はバージョン 02-06 以降の HiRDB SQL Executer からアクセスする場合にだけ指定できます。
UCS2_UJIS2	UCS2_UJIS と同じですが，図「UCS2_UJIS と UCS2_UJIS2 の変換差異」に示す差異があります。
UCS2_UTF8	HiRDB クライアントは UCS-2 で，HiRDB サーバは Unicode（UTF-8）として，文字コードを変換します※。HiRDB サーバの文字コードが Unicode（UTF-8）以外の場合，HiRDB サーバ接続時にエラーとなります。UCS2_UTF8 は，Unicode 対応の ODBC3.0 ドライバ，ODBC3.5 ドライバ，HiRDB データプロバイダ for .NET Framework，又はバージョン 02-06 以降の HiRDB SQL Executer からアクセスする場合にだけ指定できます。
UOC	HiRDB クライアントは文字コード変換を行わないで，すべての文字について UOC をコールして文字コードを変換します。 このオプションを指定する場合は PDCLTCNVUOCLIB，PDCLTCNVUOCFUNC 及び PDCLTCNVBYTERATIO を必ず指定してください。指定値がない場合，サーバ接続時にクライアント環境定義指定不正（KFPA11724-E）で SQL エラーになります。 また，このオプションを指定して各国文字データ型でデータを設定又は取得する場合，クライアントの文字コードに UTF-8 は使用できません。クライアントの文字コードが UTF-8 の場合は，SQL 実行時にパラメタのデータ型不正（KFPA11311-E）で SQL エラーになります。 HiRDB ODBC3.5 ドライバ及び，HiRDB データプロバイダ for .NET Framework からアクセスする場合，このオプションは表「HiRDB ODBC3.5 ドライバ及び HiRDB データプロバイダ for .NET Framework 使用時の指定可否」で指定可能としているときだけ指定できます。 HiRDB が提供するクライアントライブラリには，共用ライブラリとアーカイブライブラリがあります。このオプションは共用ライブラリを使用しているときだけ指定できます。アーカイブライブラリを使用したときは，指定値を無視します。

注※

変換する文字コードの範囲は，UCS-4 の範囲となります。

表 6–28　HiRDB ODBC3.5 ドライバ及び HiRDB データプロバイダ for .NET Framework 使用時の指定可否

名称	プラットフォーム		指定可否
HiRDB ODBC3.5 ドライバ	Windows		△
	UNIX	UNICODE 対応 ODBC 関数使用時	△
		非 UNICODE 対応 ODBC 関数使用時	○
HiRDB データプロバイダ for .NET Framework	Windows		△

（凡例）

○：指定できます

△：クライアントの文字コードが SJIS の場合だけ指定できます

AUTO は，HiRDB サーバの文字コード種別が特定できない場合に指定します。UJIS は，HiRDB サーバの文字コード種別が EUC 日本語漢字コードと特定できる場合に指定します。

変換対象となる文字列を次に示します。

- SQL 文中の文字列
- SQL 記述領域に設定されるデータコードが，CHAR,VARCHAR,NCHAR，NVARCHAR，MCHAR，又は MVARCHAR の文字列
- 列名記述領域に格納される列名
- SQL 連絡領域に格納されるエラーメッセージ
- 型名記述領域に格納されるデータ型名

HiRDB クライアントと HiRDB サーバ間の文字コードの組み合わせによる，PDCLTCNVMODE の指定値を次に示します。

UOC を指定した場合は，クライアント側の文字コードを限定しないため，すべての組み合わせで文字コード変換ができます。

HiRDB クライアントのアプリケーションで使用する文字コード	HiRDB サーバ側の文字コード				
	SJIS	Unicode（UTF-8），Unicode（IVS 対応 UTF-8）	UJIS	C	GB18030
SJIS	−	UTF8，UTF8MS，UTF8_TXT，UTF8_EX，	UJIS，UJIS2	NOUSE	×

HiRDB クライアントのアプリケーションで使用する文字コード	HiRDB サーバ側の文字コード				
	SJIS	Unicode（UTF-8），Unicode（IVS 対応 UTF-8）	UJIS	C	GB18030
		UTF8_EX2，UTF8MS_TXT			
Unicode（UTF-8）※	×	−	×	NOUSE	×
UJIS	×	×	−	NOUSE	×
UCS-2	×	UCS2_UTF8	UCS2_UJIS，UCS2_UJIS2	×	×
C	NOUSE	NOUSE	NOUSE	NOUSE	×
GB18030	×	×	×	×	−

（凡例）

×：変換できないため，指定できません。

−：コード変換が不要のため，指定する必要がありません。

注※

PDCLTCNVMODE に UOC を指定している場合，各国文字データ型のデータ設定及び取得はできません。各国文字データ型のデータ設定及び取得を行おうとした場合，SQL 実行時にパラメタのデータ型不正（KFPA11311-E）で SQL エラーになります。

UTF8，UTF8_EX，及び UTF8_EX2 指定時の文字コード変換の差異（HiRDB サーバから受け取った文字の場合）を次の表に示します。

表 6–29　UTF8，UTF8_EX，及び UTF8_EX2 指定時の文字コード変換の差異（HiRDB サーバから受け取った文字の場合）

HiRDB サーバから受け取った文字（Unicode（UTF-8））	PDCLTCNVMODE の指定値	HiRDB クライアント変換後の文字コード（SJIS）
0x5C（バックスラッシュ）	UTF8	0x815F（全角バックスラッシュ）
	UTF8_EX	0x5C（¥記号）※
	UTF8_EX2	
0xC2A5（¥記号）	UTF8	0x5C（¥記号）
	UTF8_EX	
	UTF8_EX2	

注※

文字コードを変換しません。

UTF8，UTF8_EX，及び UTF8_EX2 指定時の文字コード変換の差異（HiRDB クライアントから入力した文字の場合）を次の表に示します。

表 6–30　UTF8，UTF8_EX，及び UTF8_EX2 指定時の文字コード変換の差異（HiRDB クライアントから入力した文字の場合）

HiRDB クライアントから入力した文字（SJIS）	PDCLTCNVMODE の指定値	HiRDB クライアント変換後の文字コード（Unicode（UTF-8））
0x5C（¥記号）	UTF8	0xC2A5（¥記号）
	UTF8_EX	0x5C（バックスラッシュ）※
	UTF8_EX2	0xC2A5（¥記号）

注※

文字コードを変換しません。

《留意事項》

- 文字列中に 2 バイト外字が含まれている場合には，"＃"の全角文字と置き換えます。ただし，クライアント環境定義 PDCLTGAIJIDLL 及び PDCLTGAIJIFUNC を指定している場合を除きます。また，EUC の 3 バイト外字は使用できません。
- 半角片仮名文字はシフト JIS 漢字コードでは 1 バイト，EUC 日本語漢字コードでは 2 バイトのコードであるため，半角片仮名文字を含む文字列は変換の前後で長さが変わります。サーバから受け取った文字列に半角片仮名が含まれていると，変換後の文字列は短くなります。サーバへ送る文字列に半角片仮名が含まれていると，変換後の文字列は長くなります。

  また，Unicode の場合は，ASCII（0x0～0x7f）以外の文字は 2 バイトから 4 バイトの文字として表現されるため，変換の前後で長さが変わります。サーバから受け取った文字列に ASCII 以外の文字列が含まれていると，変換後の文字列は短くなります。サーバへ送る文字列に ASCII 以外の文字列が含まれていると，変換後の文字列は長くなります。

  長さが変わる場合，次のようになります。

1. SQL 記述領域に設定されるデータコードが CHAR，MCHAR の文字列の場合

   文字列長が短くなった場合（HiRDB サーバから受け取る UJIS の文字列に半角片仮名文字が含まれるとき，又は Unicode の文字列に ASCII 以外の文字が含まれるとき）は，元の長さまで半角空白（0x20）を埋めます。

   文字列長が長くなった場合（HiRDB サーバへ渡す UJIS 変換文字列に半角片仮名文字が含まれるとき，又は Unicode 変換文字列に ASCII 以外の文字列が含まれるとき）は，文字列を切り捨てないで変換後の文字列をそのまま HiRDB サーバへ渡します。

   したがって，文字列を格納する列の長さを，十分に確保する必要があります。また，すべての文字が半角片仮名文字と特定できる場合は，文字数の 2 倍のバイト数の領域（Unicode の場合は文字数の 3 倍のバイト数の領域）が必要です。

2. SQL 記述領域に設定されるデータコードが VARCHAR，MVARCHAR の文字列，SQL 文中の文字列，列名記述領域に格納される列名，SQL 連絡領域に格納されるエラーメッセージ，及び型名記述領域に格納されるデータ型名の場合

   文字列が短くなった場合，文字列長を変換後の文字列長に変更します。

   文字列が長くなった場合，文字列長を変換後の文字列長に変更します。

   すべての文字が半角片仮名文字と特定できる場合は，文字数の 2 倍のバイト数の領域（Unicode の場合は文字数の 3 倍のバイト数の領域）が必要です。

3. SQL 記述領域からポイントされる NCHAR，NVARCHAR の文字列（データコードが NCHAR，NVARCHAR）の場合

   半角片仮名文字を使用できないため，変換しても長さは変わりません。

- バイナリデータを格納するために CHAR，VARCHAR の列を使用している場合，アクセスのときに文字コード変換によって予期しない変換をすることがあります。この場合，文字コード変換をしないようにしてください（PDCLTCNVMODE に NOUSE を指定してください）。
- BLOB 型のデータは文字コード変換ができません。例えば，テキストデータを格納するために BLOB 型の列を使用している場合など，文字コード変換は UAP で実行してください。
- シフト JIS 漢字コードと Unicode での文字のマッピングには，次の 2 種類があります。

  JIS 方式：

  JIS X 0221 で規定されたマッピングに従います。PDCLTCNVMODE に UTF8MS 及び UTF8MS_TXT 以外を指定した場合，JIS 方式となります。

  変換対象：シフト JIS～JIS X0221

  漢字範囲：JIS 第 1 水準，JIS 第 2 水準

  MS 方式：

  Microsoft 社が規定したマッピングに従います。PDCLTCNVMODE に UTF8MS 又は UTF8MS_TXT を指定した場合，MS 方式となります。

  変換対象：Windows 符号化文字集合～MS-UNICODE

  漢字範囲：JIS 第 1 水準，JIS 第 2 水準，IBM 拡張漢字，NEC 選定 IBM 拡張漢字，NEC 特殊文字
- シフト JIS 漢字コードと Windows 符号化文字集合とでは，外字コードの範囲が異なるので注意してください。
- 取り扱える漢字が多い MS 方式は，PDCLTCNVMODE で UTF8MS 又は UTF8MS_TXT を指定するとマッピングできます。MS 方式を使用する場合は，マッピングの差異によって発生する可能性がある問題点を，十分に把握してから使用してください。
- UJIS 用のユーザ定義外字変換 DLL を，Unicode の外字変換にはそのまま使用できません。Unicode の外字変換をする場合は，Unicode の外字変換機能を追加したユーザ定義外字変換 DLL を使用する必要があります。

《クライアントの文字コードが UCS-2 の場合の留意事項》

- 文字コード変換によって，SQL 連絡領域に設定されるエラーメッセージ（SQLERRMC）が 254 バイトを超える場合があります。その場合，列名及び型名は 254 バイトまで設定され，それ以降は切り捨てられます。
- CHAR，MCHAR，VARCHAR，又は MVARCHAR のデータを受け取る場合，SQL 記述領域のデータの長さ（SQLLEN）は，最大定義長の 2 倍必要となります。
- 文字コード変換後のデータは，次に示す値を超えて受け取ることはできません。

  ・CHAR，NCHAR，及び MCHAR の場合は 30,000 バイト

  ・VARCHAR，NVARCHAR，及び MVARCHAR の場合は 32,000 バイト

  したがって，サーバに格納されているデータは，固定文字列型の場合は 15,000 バイト，可変長文字列型の場合は 16,000 バイトを超えると，検索できないことがあります。
- 入力パラメタで指定するデータ長は，次に示す値を超えてはいけません。

  ・CHAR，NCHAR，及び MCHAR の場合は 30,000 バイト

  ・VARCHAR，NVARCHAR，MVARCHAR の場合は 32,000 バイト
- SQL 記述領域に設定されるデータコードが CHAR，MCHAR，VARCHAR，又は MVARCHAR で，その文字列をサーバに送る場合，文字列長は変換後の長さに変更されます（固定長の場合は，SQLLEN が変換後の文字列長に変更されます）。
- UCS-2 の文字列の先頭に BOM は付けないでください。BOM が付いた文字列の場合，正しく変換されません。UCS-2 のバイトオーダは，プログラムを実行するホストのバイトオーダとして処理されます。
- UCS2_UJIS と UCS2_UJIS2 の変換差異を次の図に示します。

図 6–2　UCS2_UJIS と UCS2_UJIS2 の変換差異

入力文字		変換後の文字コード（EUC）	
字形	UCS-2	PDCLTCNVMODE=UCS2_UJIS	PDCLTCNVMODE=UCS2_UJIS2
№	0x2116	0xA14F(#)	0xADE2
℡	0x2121	0xA14F(#)	0xADE4
Ⅰ	0x2160	0xA14F(#)	0xADB5
Ⅱ	0x2161	0xA14F(#)	0xADB6
Ⅲ	0x2162	0xA14F(#)	0xADB7
Ⅳ	0x2163	0xA14F(#)	0xADB8
Ⅴ	0x2164	0xA14F(#)	0xADB9
Ⅵ	0x2165	0xA14F(#)	0xADBA
Ⅶ	0x2166	0xA14F(#)	0xADBB
Ⅷ	0x2167	0xA14F(#)	0xADBC
Ⅸ	0x2168	0xA14F(#)	0xADBD
Ⅹ	0x2169	0xA14F(#)	0xADBE
∑	0x2211	0xA14F(#)	0xADF4
∟	0x221F	0xA14F(#)	0xADF8
∮	0x222E	0xA14F(#)	0xADF3
⊿	0x22BF	0xA14F(#)	0xADF9
①	0x2460	0xA14F(#)	0xADA1
②	0x2461	0xA14F(#)	0xADA2
③	0x2462	0xA14F(#)	0xADA3
④	0x2463	0xA14F(#)	0xADA4
⑤	0x2464	0xA14F(#)	0xADA5
⑥	0x2465	0xA14F(#)	0xADA6
⑦	0x2466	0xA14F(#)	0xADA7
⑧	0x2467	0xA14F(#)	0xADA8
⑨	0x2468	0xA14F(#)	0xADA9
⑩	0x2469	0xA14F(#)	0xADAA
⑪	0x246A	0xA14F(#)	0xADAB
⑫	0x246B	0xA14F(#)	0xADAC
⑬	0x246C	0xA14F(#)	0xADAD
⑭	0x246D	0xA14F(#)	0xADAE
⑮	0x246E	0xA14F(#)	0xADAF

入力文字		変換後の文字コード（EUC）	
字形	UCS-2	PDCLTCNVMODE=UCS2_UJIS	PDCLTCNVMODE=UCS2_UJIS2
㉯	0x246F	0xA14F(#)	0xADB0
⑰	0x2470	0xA14F(#)	0xADB1
⑱	0x2471	0xA14F(#)	0xADB2
⑲	0x2472	0xA14F(#)	0xADB3
⑳	0x2473	0xA14F(#)	0xADB4
〝	0x301D	0xA14F(#)	0xADE0
〟	0x301F	0xA14F(#)	0xADE1
㈱	0x3231	0xA14F(#)	0xADEA
㈲	0x3232	0xA14F(#)	0xADEB
㈹	0x3239	0xA14F(#)	0xADEC
㉤	0x32A4	0xA14F(#)	0xADE5
㉥	0x32A5	0xA14F(#)	0xADE6
㉦	0x32A6	0xA14F(#)	0xADE7
㉧	0x32A7	0xA14F(#)	0xADE8
㉨	0x32A8	0xA14F(#)	0xADE9
㌀	0x3303	0xA14F(#)	0xADC6
㌍	0x330D	0xA14F(#)	0xADCA
㌔	0x3314	0xA14F(#)	0xADC1
㌘	0x3318	0xA14F(#)	0xADC4
㌢	0x3322	0xA14F(#)	0xADC2
㌣	0x3323	0xA14F(#)	0xADCC
㌦	0x3326	0xA14F(#)	0xADCB
㌧	0x3327	0xA14F(#)	0xADC5
㌫	0x332B	0xA14F(#)	0xADCD
㌶	0x3336	0xA14F(#)	0xADC7
㍊	0x333B	0xA14F(#)	0xADCF
㍉	0x3349	0xA14F(#)	0xADC0
㍋	0x334A	0xA14F(#)	0xADCE
㍍	0x334D	0xA14F(#)	0xADC3
㍑	0x3351	0xA14F(#)	0xADC8
㍗	0x3357	0xA14F(#)	0xADC9

入力文字		変換後の文字コード（EUC）	
字形	UCS-2	PDCLTCNVMODE=UCS2_UJIS	PDCLTCNVMODE=UCS2_UJIS2
弧	0x337B	0xA14F(#)	0xADDF
㮈	0x337C	0xA14F(#)	0xADEF
㮾	0x337D	0xA14F(#)	0xADEE
舩	0x337E	0xA14F(#)	0xADED
㎎	0x338E	0xA14F(#)	0xADD3
㎏	0x338F	0xA14F(#)	0xADD4
㎜	0x339C	0xA14F(#)	0xADD0
㎝	0x339D	0xA14F(#)	0xADD1
㎞	0x339E	0xA14F(#)	0xADD2
㎡	0x33A1	0xA14F(#)	0xADD6
㏄	0x33C4	0xA14F(#)	0xADD5
㏍	0x33CD	0xA14F(#)	0xADE3
倶	0x4FF1	0xA14F(#)	ユーザ定義外字変換DLLによる変換
剝	0x525D	0xA14F(#)	ユーザ定義外字変換DLLによる変換
呑	0x541E	0xA14F(#)	ユーザ定義外字変換DLLによる変換
噓	0x5653	0xA14F(#)	ユーザ定義外字変換DLLによる変換
姸	0x59F8	0xA14F(#)	ユーザ定義外字変換DLLによる変換
屛	0x5C5B	0xA14F(#)	ユーザ定義外字変換DLLによる変換
幷	0x5E77	0xA14F(#)	ユーザ定義外字変換DLLによる変換
痩	0x7626	0xA14F(#)	ユーザ定義外字変換DLLによる変換
繋	0x7E6B	0xA14F(#)	ユーザ定義外字変換DLLによる変換
髙	0x9B1C	0xA14F(#)	ユーザ定義外字変換DLLによる変換
鬮	0x9B2D	0xA14F(#)	ユーザ定義外字変換DLLによる変換
﴾	0xFF5F	0xA1B2(_)	ユーザ定義外字変換DLLによる変換
﴿	0xFF60	0xA14F(#)	ユーザ定義外字変換DLLによる変換

(29)　PDCLTGAIJIDLL ＝ユーザ定義外字変換 DLL ファイル名

～<文字列>

この環境変数は Windows 版の場合にだけ有効になります。

ユーザ定義外字変換 DLL ファイルの名称を指定します。文字コード変換時，この環境変数は PDCLTCNVMODE に次の値を指定している場合，無効になります。

- NOUSE
- UCS2_UTF8
- UOC

また，この環境変数を省略した場合，2 バイト外字を全角文字"＃"に変換します。

(30) PDCLTGAIJIFUNC ＝ユーザ定義外字変換関数名

～<文字列>

この環境変数は Windows 版の場合にだけ有効になります。

ユーザ定義外字変換関数の名称を指定します。文字コード変換時，この環境変数は PDCLTGAIJIDLL を指定しているときだけ有効になります。

《ユーザ定義外字関数の記述形式》

ユーザ定義外字変換関数の記述形式を次に示します。

```
void __stdcall ユーザ定義外字変換関数名 (
   long        direct,
   unsigned char far *instr,
   unsigned char far *outstr) ;
```

DLL は Microsoft Visual C++で作成してください。

DLL の作成及び関数のエクスポートは，[方法 1]または[方法 2]のどちらかで行ってください。

[方法 1]

関数宣言の前に「_declspec(dllexport)」を追加し，MAP ファイル又は Microsoft Visual C++の dumpbin コマンドを使用してエクスポート関数名を調べる。

(例)DLL 名を MyDll，ユーザ定義外字変換関数名を Sample1 とした場合

1.関数の記述形式

```
_declspec(dllexport) void __stdcall Sample1(
        long                direct,
        unsigned char far * instr,
        unsigned char far * outstr );
```

2.エクスポート関数名の調査方法

次のどちらかの方法で調査してください。

- DLL 作成時に，プロジェクトの設定で MAP ファイルを出力するように指定し，MAP ファイルからエクスポート関数名を確認してください。
- Microsoft Visual C++の dumpbin コマンド（dumpbin /exports DLL 名）を使用してエクスポート関数名を確認してください。

次に dumpbin コマンドの出力例を示します。

```
File Type: DLL

  Section contains the following exports for MyDll.dll

    00000000 characteristics
    50BF0381 time date stamp Wed Dec 05 17:19:13 2012
        0.00 version
           1 ordinal base
           3 number of functions
           3 number of names
```

```
    ordinal hint RVA      name
          1    0 00001160 Sample1@32 = Sample1@32 …… 下線部がエクスポート関数名にな
ります
```

3.PDCLTGAIJIFUNC の指定値

```
Sample1@32
```

[方法 2]

DLL 作成時にモジュール定義(.def)ファイルを同時に作成し，モジュール定義ファイルにエクスポート関数名を指定する。

(例) DLL 名を MyDll，ユーザ定義外字変換関数名を Sample1 とした場合

1.関数の記述形式

```
void __stdcall Sample1(
long                direct,
        unsigned char far * instr,
        unsigned char far * outstr );
```

2.エクスポート関数名の指定方法

モジュール定義ファイルに次の内容を記載してください。

```
LIBRARY    MyDll
EXPORTS
           Sample1
```

3.PDCLTGAIJIFUNC の指定値

```
Sample1
```

[入力]

direct：

変換方向を表します。1～6 のどれかが設定されます。

1：HiRDB クライアントから HiRDB サーバへのデータ変換

2：HiRDB サーバから HiRDB クライアントへのデータ変換

3：HiRDB クライアントから HiRDB サーバへのデータ変換（Unicode の場合）※

4：HiRDB サーバから HiRDB クライアントへのデータ変換（Unicode の場合）※

5：HiRDB クライアント UCS-2 から HiRDB サーバ UJIS へのデータ変換

6：HiRDB サーバ UJIS から HiRDB クライアント UCS-2 へのデータ変換※

7：HIRDB クライアント UTF-16（サロゲートペア）から HiRDB サーバ UJIS へのデータ変換※

注※

外字変換 DLL が Unicode を受け渡す場合，UCS-2 形式（UTF-16 形式）の 2 バイト又は 4 バイトとなります。UTF-8 形式への変換はライブラリが行います。

instr：

変換対象の外字格納領域へのポインタを表します。領域サイズは，direct に 7 以外が設定された場合は 2 バイト，7 が設定された場合は 4 バイトになります。

instr[0]=変換対象の外字の 1 バイト目

instr[1]=変換対象の外字の 2 バイト目

instr[2]=変換対象の外字の 3 バイト目

instr[3]=変換対象の外字の 4 バイト目

UCS-2（Unicode）の外字コードの場合，バイト列はビッグエンディアンのバイト列です。例えば，「東」の場合は，1 バイト目に 0x67，2 バイト目に 0x71 が設定されます。

outstr：

変換後の外字格納領域へのポインタを表します。領域サイズは，direct に 6 以外が設定された場合は 2 バイト，6 が設定された場合は 4 バイトになります。

outstr[0]=変換後の文字コード（外字）の 1 バイト目

outstr[1]=変換後の文字コード（外字）の 2 バイト目

outstr[2]=変換後の文字コード（外字）の 3 バイト目

outstr[3]=変換後の文字コード（外字）の 4 バイト目

コード変換できなかった場合でも，変換値として妥当な値を設定してください（渡された値を無条件で使用します）。

UCS-2（Unicode）の外字コードの場合，バイト列はビッグエンディアンのバイト列で返す必要があります。例えば，「東」の場合は，1 バイト目に 0x67，2 バイト目に 0x71 を設定してください。

[出力]

***outstr**：

変換後の文字列を格納します。

注意事項

*instr と*outstr へ設定する文字コードの組み合わせを次の表に示します。

表 6–31　*instr と*outstr へ設定する文字コードの組み合わせ

direct	instr	outstr	PDCLTCNVMODE
1	シフト JIS 漢字コードの外字コード	EUC 日本語漢字コードの外字コード	UJIS 又は UJIS2
2	EUC 日本語漢字コードの外字コード	シフト JIS 漢字コードの外字コード	UJIS 又は UJIS2
3	シフト JIS 漢字コードの外字コード	Unicode の外字コード	UTF8 又は UTF8_TXT
	Windows 符号化文字集合の外字コード	MS-Unicode の外字コード	UTF8MS 又は UTF8MS_TXT
4	Unicode の BMP（基本多言語面）の外字コード	シフト JIS 漢字コードの外字コード	UTF8 又は UTF8_TXT
	MS-Unicode の外字コード	Windows 符号化文字集合の外字コード	UTF8MS 又は UTF8MS_TXT

direct	instr	outstr	PDCLTCNVMODE
5	Unicode の BMP（基本多言語面）の外字コード，及び Unicode の BMP（基本多言語面）に配置された JIS の第 3 水準，第 4 水準の漢字コード	EUC 日本語漢字コードの外字コード	UCS2_UJIS 又は UCS2_UJIS2
6	EUC 日本語漢字コードの外字コード	任意の UTF-16 形式のコード	UCS2_UJIS 又は UCS2_UJIS2
7	Unicode のサロゲートペア	EUC 日本語漢字コードの外字コード	UCS2_UJIS 又は UCS2_UJIS2

各文字コードの外字コードの範囲を次の表に示します。

表 6–32　各文字コードの外字コードの範囲

文字コード	1 バイト目	2 バイト目
シフト JIS 漢字コード	0xf0～0xfc	0x40～0x7e
		0x80～0xfc
Windows 符号化文字集合	0xf0～0xfa	0x40～0x7e
		0x80～0xfc
EUC 日本語漢字コード	0xf5～0xfe	0xa1～0xfe
Unicode 及び MS-Unicode※	0xe0～0xf8	0x00～0xff

注※　0xe000～0xe757，0xf8f0～0xf8ff には，Microsoft 社が独自に文字を割り当てているため，これらの外字コードではユーザ定義外字変換 DLL を呼び出しません。

サロゲートペアの範囲を次の表に示します。

表 6–33　サロゲートペアの範囲

エンコード	上位サロゲート		下位サロゲート	
	1 バイト目	2 バイト目	1 バイト目	2 バイト目
UTF-16	0xd8～0xdb	0x00～0xff	0xdc～0xdf	0x00～0xff

(31)　PDCLTCNVUOCLIB=UOC のライブラリファイル名

～<文字列>((最大 256 バイト))

UOC による文字コード変換をする場合に，UOC を格納したライブラリファイルの名称を絶対パスで指定します。

この環境変数は PDCLTCNVMODE に UOC を指定しているときだけ有効になります。UOC 以外の値を指定している場合は，指定値を無視します。

このクライアント環境定義に指定する絶対パスに，空白や括弧が含まれるときは指定値の前後を二重引用符(")で囲んでください。

(32)　PDCLTCNVUOCFUNC=UOC の文字コード変換関数名

～<文字列>((最大 30 バイト))

UOC による文字コード変換をする場合に，文字コード変換関数の名称を指定します。

この環境変数は，PDCLTCNVUOCLIB を指定しているときだけ有効になります。

PDCLTCNVUOCLIB に指定したライブラリの読み込みと，PDCLTCNVUOCFUNC に指定した関数アドレスの解決は，UAP からの CONNECT 要求時のサーバ接続確立前に行います。ライブラリの読み込み失敗又は関数アドレス解決失敗時は，文字コード変換処理失敗（KFPA19803-E）で SQL エラーを返却します。

《UOC の文字コード変換関数の記述形式》

文字コード変換関数の記述形式を次に示します。

UNIX 版の場合

```
void_文字コード変換関数名（
long                          direct ,
unsigned int        inlen ,
unsigned char     * insetr ,
unsigned int        outlen ,
unsigned char     * outstr ,
unsigned int      * cnvlen ,
int                           * errinf ,
unsigned char     * errmsg ) ;
```

Windows 版の場合

```
void __stdcall 文字コード変換関数名（
long                           direct ,
unsigned int    inlen ,
unsigned char * insetr ,
unsigned int    outlen ,
unsigned char * outstr ,
unsigned int  * cnvlen ,
int                            * errinf ,
unsigned char    * errmsg ) ;
```

DLL の作成方法及び関数のエクスポート方法は，「PDCLTGAIJIFUNC」の「ユーザ定義外字変換関数の記述形式」を参照してください。

[入力]

direct：

変換方向を表します。次のどちらかが設定されます。

1：HiRDB クライアントから HiRDB サーバへのデータ変換

2：HiRDB サーバから HiRDB クライアントへのデータ変換

inlen：

変換対象の文字列の長さ（バイト数）を表します。

***instr：**

変換対象の文字列格納領域へのポインタを表します。

outlen：

変換後の文字列格納領域の長さ（バイト数）を表します。

[出力]

***outstr：**

変換後の文字列格納領域へのポインタを表します。

文字コード変換後の文字列を設定してください。終端の NULL ターミネイトは不要です。

文字コード変換後の文字列長が outlen の値より大きくなる場合は，文字コード変換後の文字列を outlen 分まで設定してください。

***cnvlen：**

変換後の文字列の長さ（バイト数）を格納する領域のポインタを表します。

文字コード変換後の文字列長（バイト数）を設定してください。

***errinf：**

文字コード変換時に発生したエラー情報を格納する領域のポインタを表します。

この引数に設定する値及び設定値による HiRDB クライアントの処理を次の表に示します。

表 6–34　errinf の設定値と HiRDB クライアントの処理

設定値	内容	HiRDB クライアントの処理
0	正常終了（エラーなし）	文字コード変換成功とし，その後の処理を続行します。
-1	領域不足 文字コード変換後の文字列の長さ(バイト数)が outlen より大きくなる場合に，この値を設定してください。	領域不足が発生しているため，変換対象のデータによって，表「領域不足(errinf=-1)時の HiRDB クライアントの処理」に示す動作をします。
-2	文字コード変換エラー 文字コード変換処理でエラーが発生した場合に，この値を設定してください。	文字コード変換失敗のため，変換対象のデータによって，次の処理をします。 • エラーメッセージの場合 文字コード変換前に発生していたエラー情報を UAP に返却します。また，クライアントエラーログに，文字コード変換前に発生していたエラー情報及び，KFPZ03000-I を設定します。 • 上記以外の場合 UAP に KFPA19803-E を返却します。

表 6–35　領域不足(errinf=-1)時の HiRDB クライアントの処理

変換対象のデータ	HiRDB クライアントの処理	
	クライアントからサーバへの変換(direct=1)の場合	サーバからクライアントへの変換(direct=2)の場合
SQL 文中の文字列	文字コード変換失敗として，UAP に KFPA19803-E を返却します。	−
SQL 記述領域中に設定されるデータコードが CHAR，VARCHAR，NCHAR，NVARCHAR，MCHAR，MVARCHAR の文字列	−	SQL 連絡領域の SQLWARN1 を設定します。 標識変数の指定がある場合，標識変数には切り捨て後の長さを設定します。 文字コード変換成功として，その後の処理を続行します。
次の SQL 文で取得する，列名及びユーザ定義型名 ・DESCRIBE 文 ・DESCRIBE TYPE 文 ・PREPARE 文	−	文字コード変換成功として，その後の処理を続行します。※

（凡例）

−：存在しない組み合わせ

注※

HiRDB クライアントでは列名，ユーザ定義型名を取得するために確保する領域の長さが決まっています。そのため，文字コード変換によってデータ長が大きくなる場合は，次に示す最大長を超えないよう注意してください。

・列名：30 バイト

・ユーザ定義型名の認可識別子：8 バイト

・ユーザ定義型名のデータ型識別子：30 バイト

errmsg：

文字コード変換でエラーが発生した場合（errinf=-2），エラーの詳細を示すメッセージテキストを格納する領域のポインタを表します。メッセージテキストの最大長は 128 バイト（終端 NULL 含む）です。テキスト終端は NULL で終了してください。

この引数に設定したメッセージテキストは，KFPA19803-E の詳細情報として，SQLCA 及びクライアントエラーログに出力されます。

UOC は処理結果に応じて，次の表で示す情報を出力用引数に設定してください。

表 6–36　UOC 処理結果による設定値

処理結果	UOC での設定値			
	outstr	cnvlen	errinf	errmsg
正常終了	文字コード変換後の文字列	outstr に設定した文字列長（バイト数）	0	設定不要
領域不足	文字コード変換後の文字列を outlen の長さで設定する	outstr に設定した文字列長（バイト数）	-1	設定不要

処理結果	UOC での設定値			
	outstr	cnvlen	errinf	errmsg
変換エラー	設定不要	設定不要	-2	変換エラーの詳細情報文字列

(33) PDCLTCNVBYTERATIO=文字コード変換時の 1 文字当たりのバイト数の比率

～<符号なし整数>((1～10))

UOC による文字コード変換機能使用時に，クライアントの文字コードでの 1 文字のバイト数に対して，サーバの文字コードでの 1 文字のバイト数が最大で何倍になるかを指定します。増加率が 1 未満の場合や小数になる場合は，小数点以下を切り上げて指定してください。文字コード変換時，この環境変数は PDCLTCNVMODE に UOC を指定しているときだけ有効になります。UOC 以外の値を指定している場合は，指定値を無視します。

(例)クライアントの文字コードが SJIS，サーバの文字コード：UTF-8（IVS 対応）の場合：

SJIS の全角文字 2 バイトに対して，UTF-8 は IVS 使用時に最大 8 バイトとなり，1 文字を表現するバイト数が最大で 4 倍となるため，このクライアント環境定義には 4 を指定します。PDCLTCNVBYTERATIO に 4 を指定した場合のデータ取得と，DCLTCNVBYTERATIO がない場合のデータ取得を次の図に示します。

図 6–3　PDCLTCNVBYTERATIO に 4 を指定した場合のデータ取得

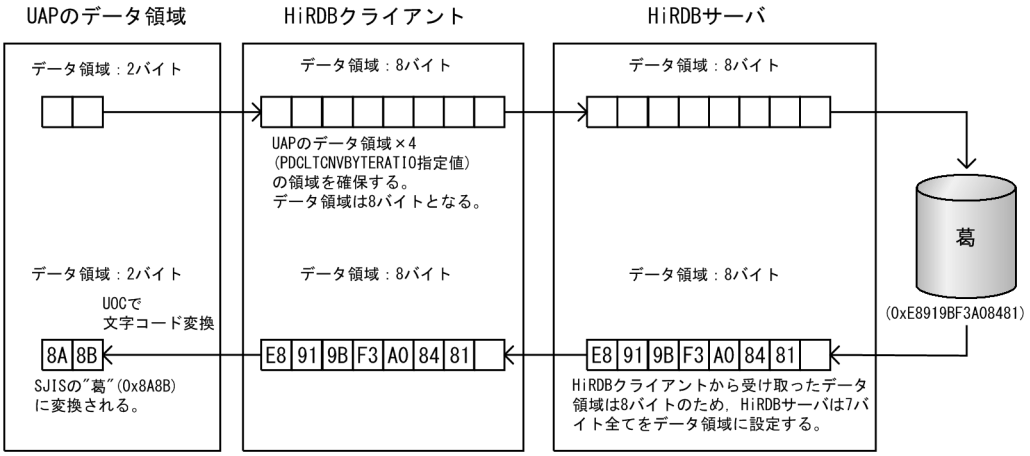
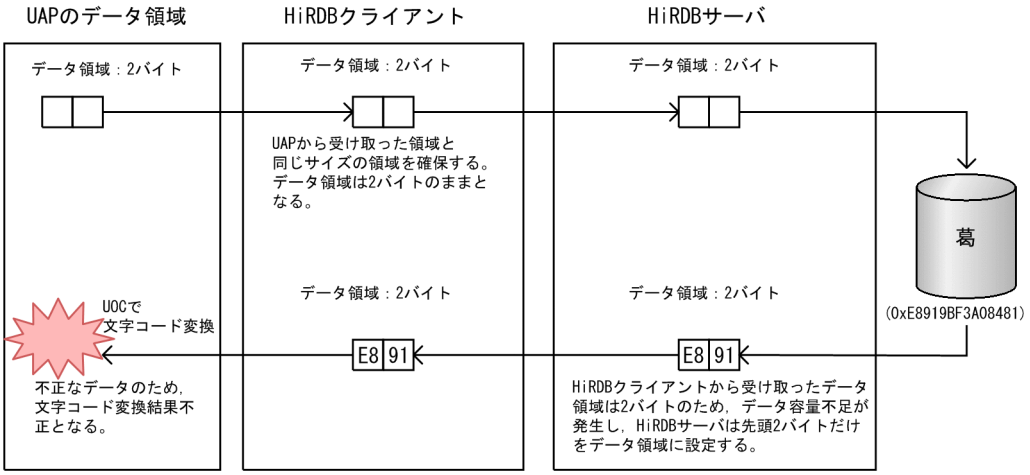


図 6–4　DCLTCNVBYTERATIO がない場合のデータ取得



(34)　PDCLTGRP ＝クライアントグループ名

～<英字>((1 文字))

クライアントグループの接続枠保証機能を使用する場合，クライアントグループ名を指定します。システム定義の pdcltgrp オペランドで指定したクライアントグループ名を，英大文字 1 文字で指定します。英小文字を指定した場合は，英大文字を指定したとみなします。

システム定義の pdcltgrp オペランドが指定されていない場合，又は pdcltgrp オペランドで指定されていないクライアントグループ名を指定した場合，この環境変数の指定は無効になります。

クライアントグループの接続枠保証機能については，マニュアル「HiRDB システム運用ガイド」を参照してください。

(35)　PDAUTORECONNECT= {YES ｜ NO}

自動再接続機能を使用するかどうかを指定します。

自動再接続機能については，「自動再接続機能」を参照してください。

YES：

自動再接続機能を使用します。

自動再接続機能を使用すると，サーバプロセスダウン，系切り替え，ネットワーク障害などの要因で HiRDB サーバとの接続が切断された場合に，自動的に再接続します。

NO：

自動再接続機能を使用しません。

《適用基準》

HiRDB サーバで，システム構成変更コマンド（pdchgconf）を実行している場合，及び修正版 HiRDB の入れ替え（pdprgcopy，pdprgrenew）をしている場合に適用してください。この場合に自動再接続

機能を使用すると，HiRDB サーバとの接続が切断しても UAP にエラーを返却しないで処理を続行できます。

《留意事項》

1. 再接続するときの動作は，次のクライアント環境定義で設定できます。
   ・CONNECT のリトライ回数：PDRCCOUNT
   ・CONNECT のリトライ間隔：PDRCINTERVAL
   ・再接続する契機：PDRCTIMING
2. CONNECT 文以外の SQL で自動再接続機能が動作している間は，PDCWAITTIME の時間で監視されます。PDCWAITTIME の時間を超えた場合は，自動再接続の処理は打ち切られます。
3. 自動再接続が失敗した場合は，要因となったエラーを UAP に返却します。
4. X/Open に従った API を使用してアクセスしているアプリケーションの場合は，この環境変数を指定しても無効になり，常に NO が仮定されます。
5. 次のどれかに該当する場合は，CONNECT 文を実行したときだけ自動再接続機能が有効になります。
   ・HiRDB サーバのバージョンが 07-00 より前である。
   ・XDM/RD E2 接続機能を使用している。
   ・XDM/RD E2 のバージョンが 10-02 以前である。

(36)　PDRCCOUNT=自動再接続機能での CONNECT のリトライ回数

～＜符号なし整数＞((1～200))《5》

自動再接続機能で再接続するときの，CONNECT のリトライ回数を指定します。この環境変数は，PDAUTORECONNECT=YES を指定した場合に有効になります。

(37)　PDRCINTERVAL=自動再接続機能での CONNECT のリトライ間隔

～＜符号なし整数＞((1～600))《5》（単位：秒）

自動再接続機能で再接続するときの CONNECT のリトライ間隔を指定します。この環境変数は，PDAUTORECONNECT=YES を指定した場合に有効になります。

(38)　PDRCTIMING=｛ALL｜再接続契機［，再接続契機［，再接続契機］］｝

自動再接続機能で再接続する場合の再接続契機を指定します。

SQL 実行シーケンスでの自動再接続契機と PDRCTIMING 指定値の対応については，「自動再接続機能」を参照してください。

この環境定義は PDAUTORECONNECT=YES を指定した場合に有効になります。

ALL：

すべての契機で自動再接続を適用します。

再接続契機：

次のどれか一つ，又は複数を組み合わせて指定してください。

CNCT：

CONNECT 文を実行した場合で接続障害を検知したときに，自動再接続を適用します。

NOTRN：

トランザクション未発生で実行した SQL が接続障害を検知した場合に，自動再接続を適用します。

TRN：

トランザクション発生中に実行した SQL が接続障害でエラーとなった後，CONNECT 文を実行しないで SQL を実行した場合に，自動再接続を適用します。

《規則》

1. 再接続契機は順不同で指定できます。
2. 再接続契機を複数指定する場合は，再接続契機ごとにコンマ（,）で区切って指定してください。Type4 JDBC ドライバ以外を使用している場合，コンマの前後には空白を含めないでください。空白を含んだ場合はエラーとなります。
3. 同じ再接続契機を複数回指定した場合はエラーとなります。

《指定例》

（例 1）すべての契機で自動再接続を適用したい場合

PDRCTIMING = ALL（または，PDRCTIMING = CNCT,NOTRN,TRN）

（例 2）CONNECT 文を実行したとき，およびトランザクション未発生で SQL 文を実行したときに，自動再接続を適用したい場合

PDRCTIMING = CNCT,NOTRN

(39) PDAUTHTYPE= {PA ｜ CHA [/SHA-256] [/SHA-512]}

HiRDB クライアントがサーバへ接続するときの認証方式を指定します。

PA：

PA 方式で接続します。

CHA：

CHA 方式で接続します。

次のオプションを指定することで，CHA 方式で使用するハッシュ関数を選択できます。

- /SHA-256：CHA 方式のハッシュ関数として SHA-256 を使用します。
- /SHA-512：CHA 方式のハッシュ関数として SHA-512 を使用します。

/SHA-256 と/SHA-512 の両方指定した場合，及び両方を省略した場合は，サーバと接続でき，かつセキュリティがより高い方式が自動的に選択されます。

《注意事項》

システム共通定義 pd_connect_auth_type オペランドと一致しない認証方式を指定すると，HiRDB サーバ接続時に認証エラーとなります。詳細はマニュアル「HiRDB システム運用ガイド」の「HiRDB 接続時のパスワード秘匿化機能の設定方法」を参照してください。

(40) PDUAPENVFILE=UAP 環境定義のファイル名

～＜識別子＞((最大 8 文字))

UAP を個別の環境で実行する場合，その環境を定義した UAP 環境定義のファイル名を指定します。この環境変数を指定すると，UAP ごとに実行環境を切り替える運用ができます。

UAP 環境定義については，マニュアル「HiRDB システム定義」を参照してください。

UAP 環境定義の定義内容に誤りがある場合は，CONNECT 時に定義エラーとなります。また，UAP 環境定義のファイルに定義がない場合は，PDUAPENVFILE の指定は無効になります。

Windows 版の HiRDB の場合，UAP 環境定義のファイル名は，大文字，小文字が区別されません。したがって，大文字，小文字が異なるだけで名前が同じファイルは，同一ファイルとして扱われるため注意してください。

(41) PDDBBUFLRU= {YES | NO}

OLTP 環境の UAP の場合，UAP ごとにグローバルバッファの LRU 方式を変更するかどうかを指定します。

YES：

LRU 方式を適用します。

NO：

LRU 方式を適用しません。この場合，バッファヒットしなかったページは，グローバルバッファ不足発生時に，アクセス頻度に関係なくグローバルバッファからの追い出し対象となります。そのため，新たにグローバルバッファ上にキャッシュするページ数を最小限にできます。

《適用基準》

通常はこの環境変数を省略してください（LRU 方式を適用してください）。OLTP 環境で，グローバルバッファを使用した大量検索，又は大量更新の UAP を実行する場合，グローバルバッファにキャッシュされた最新の内容がグローバルバッファから追い出されるため，性能が一時的に低下することがあります。これを回避する場合，OLTP 環境で大量検索，又は大量更新をする UAP に対して，PDDBBUFLRU=NO を指定します。

《留意事項》

1. LRU 方式を適用しない UAP がアクセスしたページは，グローバルバッファ不足発生時に，アクセス頻度に関係なくグローバルバッファからの追い出し対象となります。そのため，LRU 方式を適用しない UAP は，バッファヒット率の低下に伴う入出力回数の増加によって，レスポンス性能が低下することがあります。

2. UAP の SQL 処理では，1〜4 面のグローバルバッファが同時に確保されます。したがって，LRU 方式を適用しない場合でも，UAP ごとにグローバルバッファにキャッシュされているページのうち，1〜4 面はグローバルバッファから追い出される可能性があります。
3. 更新する UAP に対して LRU 方式を適用しない場合，DB への書き込み頻度が高くなります。そのため，ログ出力契機が頻繁に発生し，LRU 方式適用時に比べると，出力されるログ量が多くなります。この場合，システムログファイルの容量不足が発生する可能性があるため，次のどちらかの対処をしてください。

   ・システムログファイルの容量を再度見積もる。

   ・クライアント環境定義の PDDBLOG オペランドに NO を指定する。

   LRU 方式を適用しない場合のログ量の見積もり計算式を次に示します。なお，システム定義の pd_log_rec_leng オペランドの指定値を 1,024 にすると，LRU 方式を適用しない場合の出力ログ量を最小に抑えられます。

   更新 GET 数※×pd_log_rec_leng オペランドの指定値

   注※

   更新 GET 数は，UAP 統計レポートの DIDUC の値，又は UAP に関する統計情報の DIDUC の値で確認できます。

(42) PDHATRNQUEUING=NO

システム定義の pd_ha_transaction オペランドに queuing を指定している場合，トランザクションキューイング機能の適用を，各クライアントで変更したいときに指定します。トランザクションキューイング機能を適用しないクライアントの場合に，NO を指定してください。

NO：

クライアントからの接続時にトランザクションキューイング機能を適用しません。

トランザクションキューイング機能については，マニュアル「HiRDB システム運用ガイド」を参照してください。

(43) PDCLTBINDLOOPBACKADDR={YES | NO}

HiRDB サーバとの通信で使用する受信ポートの生成時，ループバックアドレスで bind()するかどうかを指定します。

YES：

ループバックアドレスで bind()します。

NO：

ループバックアドレスで bind()しません。

システム共通定義の pdunit -x オペランドにループバックアドレスを指定している場合は，この環境変数に YES を指定してください。

この環境変数は，HiRDB サーバが Windows 版の場合に指定できます。この環境変数を指定する場合は，マニュアル「HiRDB システム導入・設計ガイド」の，Windows ファイアウォールの例外リストへの登録についての説明を参照してください。

(44) PDEXTDECCHECK= {YES | NO}

外部 10 進項目の DISPLAY SIGN LEADING SEPARATE 形式，及び DISPLAY SIGN TRAILING 形式の入力データが，正又は負の符号と数字で構成されているかをチェックします。

YES：

入力データが正又は負の符号と数字で構成されているかをクライアント側でチェックします。

各形式の入力データの内容が，次のすべての条件を満たしているときは，クライアント側でエラーになりません。

- DISPLAY SIGN LEADING SEPARATE 形式の場合
  - 先頭 1 バイトが符号（'+'(0x2B)又は'-'(0x2D)）
  - 2 バイトから最終バイトまで数字（0x30～0x39）
- DISPLAY SIGN TRAILING 形式の場合
  - 先頭バイトから最終バイトの一つ前まで数字（0x30～0x39）
  - 最終バイトが 0x30～0x39 又は 0x70～0x79

上記以外の場合は，エラーとなり KFPA11326-E メッセージが出力されます。

NO：

入力データが正又は負の符号と数字で構成されているかをチェックしません。この場合，入力データは無条件に DECIMAL 形式に変換してサーバに送信します。

《適用基準》

HiRDB では，外部 10 進項目の入力データを数字として扱います。外部 10 進項目の DISPLAY SIGN LEADING SEPARATE 形式，及び DISPLAY SIGN TRAILING 形式の入力データに，数字以外のデータが含まれているかどうかをチェックしたい場合に，PDEXTDECCHECK=YES を指定してください。

(45) PDDEFAULTOPTION = {RECOM | V0904}

クライアント環境定義及びプリプロセスのオプションについて，省略時の動作を指定します。通常は，この環境変数を指定する必要はありません。互換モードを適用したい場合に，指定してください。

バージョンによって省略値が異なるクライアント環境定義及びプリプロセスのオプションについては，「バージョン，リビジョンによるクライアント環境定義及びプリプロセスオプションの変更点」を参照してください。

RECOM：

推奨モードを適用します。

V0904：

0904 互換モードを適用します。

(46)　PDIPCFILEDIR=通信情報ファイルディレクトリ

～<パス名>((最大 80 バイト))《/dev/HiRDB/pth》

HiRDB クライアントと HiRDB サーバが同一マシンにある場合の通信情報ファイルディレクトリを指定します。このクライアント環境定義の有効範囲を次に示します。

プラットフォーム	クライアント環境定義 PDIPC の値	有効/無効
AIX	MEMORY	○
	DEFAULT	○
Linux	MEMORY	○
	DEFAULT	×
Windows	MEMORY	×
	DEFAULT	×

（凡例）

○：有効

×：無効

このクライアント環境定義には，同一マシンの HiRDB サーバのシステム共通定義 pd_ipc_file_dir と同じ値を設定する必要があります。値が不一致であった場合の動作を次に示します。

プラットフォーム	クライアント環境定義 PDIPC の値	HiRDB サーバへの接続可否	動作
AIX	MEMORY	×	HiRDB クライアント側で，KFPA11723-E（HiRDB NOT UP 又は NETWORK）となります。
	DEFAULT		
Linux	MEMORY	○	プロセス間メモリ通信機能を使用しないで，デフォルトの通信方式（TCP/IP）を使用して，HiRDB サーバに接続します。

（凡例）

○：接続できる

×：接続できない

プロセス間メモリ通信機能が有効になっているかどうかは，SQL トレースの「システムが使用する情報」で確認できます。詳細は「SQL トレース機能」を参照してください。

このクライアント環境定義が有効で，かつ/dev/HiRDB/pth 以外の値を指定した場合，HiRDB 10-04 より前の HiRDB サーバへの接続がエラーとなります。また，HiRDB グループが異なるユーザでの HiRDB サーバへの接続がエラーとなります。HiRDB サーバへの接続可否を次に示します。

プラットフォーム	クライアント環境定義 PDIPC の値	クライアント環境定義 PDIPCFILEDIR の値	10-04 より前の HiRDB サーバへの接続	HiRDB グループが異なるユーザでの HiRDB サーバへの接続
AIX	MEMORY	指定あり	×	
		指定なし	○	
	DEFAULT	指定あり	×	
		指定なし	○	
Linux	MEMORY	指定あり	×	
		指定なし	○	

（凡例）

○：接続できる

×：接続できない（/dev/HiRDB/pth 以外の通信情報ファイルディレクトリを指定した場合）

(47) PDSOCKETCRYPTO= {YES | NO}

HiRDB クライアントと HiRDB サーバ間の通信を暗号化するかどうかを指定します。

YES：

通信を暗号化します。

NO：

通信を暗号化しません。

YES を指定する場合，次に示すすべての条件を満たしている必要があります。

YES を指定する条件	条件を満たさない場合の動作
クライアント環境定義 PDCONTYPE に ACTIVE を指定している	接続時にエラーとなり KFPA11724-E メッセージが出力されます。
高速接続機能を使用している	
クライアント環境定義 PDSERVICEPORT 又は PDFESGRP のポート番号に，システム定義 pd_crypto_service_port の指定値を指定している	接続時にエラーとなり KFPA11723-E 又は KFPA11732-E メッセージが出力されます。

YES を指定する場合，HSIC(Hitachi System Information Capture)は使用できません。

HiRDB が提供するクライアントライブラリには共用ライブラリとアーカイブライブラリがありますが，埋込み型 UAP で YES を指定する場合は，共用ライブラリを使用する必要があります。アーカイブライブラリを使用する場合は，常に NO が仮定されます。

YES を指定する（通信の暗号化を行う）場合，HiRDB クライアントと HiRDB サーバ間の通信を暗号化するため，トランザクション性能に影響を与えます。

(48) PDASTHOST=HiRDB Control Manager - Agent のホスト名［，予備系 HiRDB Control Manager - Agent のホスト名］

～＜識別子＞《PDHOST の指定値》

UAP からコマンドを実行する場合，接続する HiRDB Control Manager - Agent のホスト名を指定します。なお，UAP からコマンドを実行するには，SQL の COMMAND EXECUTE を使用します。

UAP からコマンドを実行すると，実際には HiRDB Control Manager - Agent がそのコマンドを実行します。

HiRDB/パラレルサーバの場合は，システムマネジャがあるサーバマシンのホスト名を指定してください。

ホスト名以外にも，FQDN，及び IP アドレスでも指定できます。指定方法を次に示します。

ホスト名：

システム共通定義の pdunit -x オペランドで指定したホスト名を指定します。

（指定例）

```
PDASTHOST=host1
```

FQDN：

HiRDB サーバのホスト名とドメイン名とを，ピリオドで結んで指定します。

（指定例）

```
PDASTHOST=host1.soft.hitachi.co.jp
```

IP アドレス：

IP アドレスは，バイトごとにピリオドで区切られた 10 進数で指定します。

（指定例）

```
PDASTHOST=172.18.131.34
```

《IP アドレスを引き継がない系切り替えをする場合》

IP アドレスを引き継がない系切り替えをする場合，現用系及び予備系の二つのホスト名を指定してください。現用系のホスト名だけを指定すると，系が切り替わった後に，この環境変数の指定を新しく実行系になったホスト名に変更する必要があります。

(49) PDASTPORT=HiRDB Control Manager - Agent のポート番号

～＜符号なし整数＞((5001～49999))

UAP からコマンドを実行する場合，接続する HiRDB Control Manager - Agent のポート番号を指定します。

ポート番号は，services ファイル（UNIX 版の場合は/etc/services，Windows 版の場合は%windir%¥system32¥drivers¥etc¥services）に登録したものを指定してください。

(50) PDSYSTEMID=HiRDB Control Manager - Agent が管理する HiRDB サーバの HiRDB 識別子

～＜識別子＞((4 文字))

UAP からコマンドを実行する場合，接続する HiRDB Control Manager - Agent が管理している HiRDB サーバの HiRDB 識別子を指定します。HiRDB 識別子は，システム定義の pd_system_id オペランドでの指定値を指定してください。

(51) PDASTUSER=OS のユーザ名/パスワード

～《PDUSER の指定値》

UAP からコマンドを実行する場合，そのコマンドを実行する OS のユーザ名，及びパスワードを指定します。そのコマンドの実行権限を持つ OS のユーザ名，及びパスワードでなければなりません。「ユーザ名/パスワード」の形式で指定してください。

パスワードの指定が必要ない（パスワードのないユーザに対して設定する）場合は，パスワードを省略できます。

OS のユーザ名，パスワードはそれぞれ大文字，小文字の指定に関係なく大文字として扱われます。ただし，小文字を引用符で囲んだ場合は，小文字として扱われます。

なお，簡易認証ユーザで HiRDB に接続している場合は，この環境変数を省略しないで，コマンドを実行するユーザ名，及びパスワードを指定してください。省略した場合は，エラーメッセージ（KFPA11724-E）が表示され，サーバへの接続に失敗します。

(52) PDCMDWAITTIME=コマンド実行時のクライアントの最大待ち時間

～＜符号なし整数＞((0，6～43200))《0》（単位：分）

UAP からコマンドを実行する場合，クライアントが HiRDB Control Manager - Agent へ要求をしてから応答が返るまでの，クライアントの最大待ち時間を指定します。

0 を指定した場合，クライアントは HiRDB Control Manager - Agent から応答が返るまで待ち続けます。

指定した最大待ち時間を過ぎても HiRDB Control Manager - Agent から応答が返らない場合，クライアント（UAP）にエラーリターンします。このとき，UAP 中のコマンドがまだ処理中の場合は，HiRDB Control Manager - Agent，又は実行中のコマンドをキャンセルする必要があります。

(53) PDCMDTRACE=コマンドトレースファイルのサイズ

～＜符号なし整数＞((0，4096～2000000000))（単位：バイト）

UAP からコマンドを実行する場合，コマンドトレースを出力するファイルのサイズを指定します。

0 を指定した場合はファイルの最大サイズとなり，最大サイズを超えるとコマンドトレースは出力されません。4,096～2,000,000,000 を指定した場合は指定値のファイルサイズとなり，指定値のファイルサイズを超えると出力先が切り替わります。この環境変数を省略した場合，コマンドトレースは取得されません。

コマンドトレースについては，「コマンドトレース機能」を参照してください。

《ほかの環境変数との関係》

コマンドトレースを出力するファイルは，PDCLTPATH に指定したディレクトリに作成されます。PDCLTPATH を省略している場合，UAP を実行したときのカレントディレクトリ（OpenTP1 から実行する UAP の場合は OpenTP1 のインストールディレクトリ¥tmp¥home¥サーバ名 XX）下に作成されます。

(54) PDIPC=｛MEMORY｜DEFAULT｝

サーバとクライアントが同一ホストにある場合，プロセス間の通信方法を指定します。クライアント環境定義 PDCONTYPE に ACTIVE を指定している場合は，クライアント環境定義 PDIPC の指定は無効になり，プロセス間の通信に TCP/IP を使用します。

MEMORY：

プロセス間の通信にメモリを使用します。これを，プロセス間メモリ通信機能といいます。

DEFAULT：

プロセス間の通信に，各プラットフォームでのデフォルトの通信方式（TCP/IP 又は PIPE）を使用します。

《留意事項》

1. クライアントとサーバが同一ホストでない場合，PDIPC の指定は無効になります（DEFAULT が仮定されます）。このとき，接続処理が遅くなることがあります。
2. マルチスレッド用 XA インタフェースライブラリ（Windows 版クライアントの場合は pdcltxm.dll，UNIX 版クライアントの場合は libzcltxk.sl(so)，又は libzcltyk.sl(so)）を使用して XA インタフェースで HiRDB にアクセスし，かつ TPBroker for C++下の UAP をクライアントとする場合，この環境変数の指定は無効になり，DEFAULT が仮定されます。
3. UNIX 版クライアントで PDIPC=MEMORY を指定した場合，クライアントの接続ごとに PDSENDMEMSIZE 及び PDRECVMEMSIZE の指定値分の共用メモリが確保されます。したがって，複数のクライアントを同時に実行した場合，共用メモリが不足することがあるため，使用できる共用メモリのサイズを考慮して，PDSENDMEMSIZE 及び PDRECVMEMSIZE を指定する必要があります。
4. PDIPC=MEMORY を指定した場合，PDCLTRCVADDR の指定は無効になります。
5. PDIPC=MEMORY を指定して，同時に PDUAPREPLVL に p，r，若しくは a を指定している場合，又は PDWRTLNFILSZ を指定している場合，PDIPC の指定は無効になります。

(55) PDSENDMEMSIZE=クライアント側のデータ送信用メモリサイズ

～＜符号なし整数＞((4～2097152))《16》（単位：キロバイト）

プロセス間メモリ通信機能を使用する場合，クライアントからサーバへデータを送るときの，データ格納領域サイズを 4 の倍数で指定します。この環境変数は，PDIPC=MEMORY の場合に有効になります。

指定値が 4 の倍数でない場合，4 の倍数の値に切り上げられます。

ここで指定したサイズ以上のデータを送ると，プロセス間メモリ通信機能は使用できなくなります（PDIPC ＝ DEFAULT の通信方法となります）。

《見積もり方法》

指定値は，次の計算式から見積もってください。

```
指定値（単位：バイト）＝
  ↑（400＋16×検索列数＋16×？パラメタ数＋SQL文長）÷4096↑×4
```

なお，この計算式は，実際の通信で送信するデータ量とは異なります。

(56)　PDRECVMEMSIZE=クライアント側のデータ受信用メモリサイズ

～＜符号なし整数＞((4～2097152))《32》（単位：キロバイト）

プロセス間メモリ通信機能を使用する場合，クライアントがサーバからデータを受け取るときの，データ格納領域サイズを 4 の倍数で指定します。この環境変数は，PDIPC=MEMORY の場合に有効になります。

指定値が 4 の倍数でない場合，4 の倍数の値に切り上げられます。

ここで指定したサイズ以上のデータを受け取ると，プロセス間メモリ通信機能は使用できなくなります（PDIPC ＝ DEFAULT の通信方法となります）。

《見積もり方法》

指定値は，次の計算式から見積もってください。

```
指定値（単位：バイト）＝
  ↑（600＋25×検索列数＋Σ 列のデータ長）÷4096↑×4
```

列のデータ長は，VARCHAR の場合は構造体長にしてください。また，配列 FETCH，又は繰返し列を受け取る場合は，「列のデータ長×配列数」，又は「列のデータ長×繰り返し数の要素」にしてください。

PDBLKF を指定している場合は，次の計算式から値を求めてください。

```
指定値（単位：バイト）＝
  ↑（600＋19×検索列数＋（7×検索列数＋Σ 列のデータ長）
      ×PDBLKFの値）÷4096↑×4
```

なお，この計算式は，実際の通信で送信するデータ量とは異なります。

(57)　PDCWAITTIME=クライアントの最大待ち時間

～<符号なし整数>((0～65535))《0》（単位：秒）

HiRDB クライアントから HiRDB サーバへ要求をしてから，応答が戻ってくるまでの HiRDB クライアントの最大待ち時間を指定します。長時間 SQL の時間監視をする場合など指定してください。

《見積もり方法》

指定値は次の条件を満たすようにしてください。SQL 処理に時間が掛かったときの原因を特定しやすくなります。

OpenTP1 システム定義の watch_time 又は trn_expiration_time ＞ PDCWAITTIME ＞ pd_lck_wait_timeout

なお，OpenTP1 を使用していない場合，OpenTP1 システム定義は無視してください。

《留意事項》

1. 0 を指定した場合，HiRDB クライアントは HiRDB サーバからの応答があるまで待ち続けます。
2. 最大待ち時間を経過しても HiRDB サーバから応答がない場合には，UAP にエラーリターンします。このとき，トランザクション処理中の場合は，HiRDB サーバ側のプロセスをキャンセルします。
3. 0 を指定した場合に，次のような障害が発生すると，HiRDB クライアントが無応答状態になることがあります。

   ・通信障害（HiRDB クライアントと HiRDB サーバ，及び HiRDB サーバと HiRDB サーバの通信障害（一時的な障害も含む））

   ・ディスク障害などでのプロセスの沈み込み

   したがって，設定値には，0 以外で，かつ各 SQL 実行時間の最大値より大きな値を指定することをお勧めします。設定値は次に示す点を考慮してください。

   ・排他待ち限界経過時間を迎えても待ち状態が解除されない場合に，排他待ちのタイムアウト情報を出力したい場合は，次の条件を満たす値となるよう設定してください。

   PDCWAITTIME ＞ pd_lck_wait_timeout（PDLCKWAITTIME 指定時は PDLCKWAITTIME）

   タイムアウト情報については，マニュアル「HiRDB システム運用ガイド」の「タイムアウト情報の出力内容」，pd_lck_wait_timeout オペランドについては，マニュアル「HiRDB システム定義」の「システム共通定義」を参照してください。

   ・トランザクションキューイング機能によってトランザクションをリトライ，又はキューイングしている間に，PDCWAITTIME に指定した時間を超過してトランザクションをエラーとしたくない場合は，次の条件を満たす値となるよう設定してください。

   PDCWAITTIME ＞ pd_ha_trn_queuing_wait_time ＋ pd_ha_trn_restart_retry_time

   トランザクションキューイング機能については，マニュアル「HiRDB システム運用ガイド」の「トランザクションキューイング機能」，pd_ha_trn_queuing_wait_time 及び pd_ha_trn_restart_retry_time オペランドについては，マニュアル「HiRDB システム定義」の「システム共通定義」を参照してください。

(58) PDSWAITTIME=トランザクション処理中のサーバの最大待ち時間

～<符号なし整数>((0～65535))《600》（単位：秒）

HiRDB サーバが HiRDB クライアントからの要求に対する応答を返してから，次に HiRDB クライアントから要求が来るまでの HiRDB サーバの最大待ち時間を指定します。この時間監視は，トランザクション処理中（SQL 実行開始からコミット，ロールバックまでの区間）の時間を対象とします。監視時間は，次の時点でリセットされます。

1. HiRDB クライアントからの要求が HiRDB サーバへ到着した時点
2. HiRDB クライアントが定期的に送信するパケットを HiRDB サーバが受信した時点

ただし，上記 2 の時点でリセットするのは，次のどれかの条件に該当する場合です。

- 環境変数 PDKALVL に 2 を指定
- 環境変数 PDKALVL を省略し，環境変数 PDDEFAULTOPTION に RECOM を指定
- 環境変数 PDKALVL，及び環境変数 PDDEFAULTOPTION を省略

指定した時間内に HiRDB クライアントから次の要求が来なかった場合，HiRDB クライアントに異常が発生したものとみなし，実行中のトランザクションをロールバックします。また，HiRDB クライアントとの接続を HiRDB クライアントへ通知しないで切断します。

0 を指定した場合，HiRDB サーバは HiRDB クライアントからの要求があるまで待ち続けます。また，クライアントマシンやネットワークがダウンしたとき，ダウンを検知できないことがあります。

大き過ぎる値を指定した場合，クライアントマシンやネットワークがダウンしたとき，ダウンの検知までに時間が掛かります。

この環境変数は，プロセスの残存を回避する場合など指定してください。

《留意事項》

1. ブロック転送機能（PDBLKF）を使用していると，HiRDB サーバからブロック転送されてきた行がなくなるまで HiRDB クライアント内で FETCH 文処理をします。このため，FETCH 文処理が終了するまで HiRDB クライアントから HiRDB サーバへ要求をしません。したがって，ブロック転送機能を使用する場合，この環境変数には，ブロック転送数分の FETCH 文処理の時間を含めた値を設定してください。
2. OLTP 下の UAP をクライアントとする運用形態の場合，必ずこの環境変数を指定してください。指定しない場合は，600 秒が仮定され，不当に接続が切れる場合があります。
3. このオペランドの値を指定する場合，PDKATIME と同じ値は設定しないでください。

(59)　PDSWATCHTIME=トランザクション処理以外のサーバの最大待ち時間

〜<符号なし整数>((0〜65535))（単位：秒）

HiRDB サーバが HiRDB クライアントからの要求に対する応答を返してから，次に HiRDB クライアントから要求が来るまでの HiRDB サーバの最大待ち時間を指定します。この監視時間は，トランザクション処理以外（SQL 実行開始からコミット，又はロールバックするまでの区間外）の時間を対象とします。監視時間は，次の時点でリセットされます。

1. HiRDB クライアントからの要求が HiRDB サーバへ到着した時点
2. HiRDB クライアントが定期的に送信するパケットを HiRDB サーバが受信した時点

ただし，上記 2 の時点でリセットするのは，次のどれかの条件に該当する場合です。

- 環境変数 PDKALVL に 2 又は 3 を指定
- 環境変数 PDKALVL を省略し，環境変数 PDDEFAULTOPTION に RECOM を指定
- 環境変数 PDKALVL，及び環境変数 PDDEFAULTOPTION を省略

指定した時間内に HiRDB クライアントからの要求がなかった場合，HiRDB クライアントに異常が発生したものとみなし，HiRDB サーバが HiRDB クライアントとの接続を切断します。この場合，HiRDB クライアントへは切断の通知をしません。

0 を指定した場合，HiRDB サーバは HiRDB クライアントからの要求があるまで待ち続けます。

なお，0 を指定した場合，クライアントマシンやネットワークがダウンしたとき，ダウンを検知できないことがあります。

大き過ぎる値を指定した場合，クライアントマシンやネットワークがダウンしたとき，ダウンの検知までに時間が掛かります。

この環境変数は，プロセスの残存を回避する場合など指定してください。

《システム定義との関係》

この環境変数を省略すると，次の値を使用してトランザクション開始までの時間を監視します。

- UNIX 版の場合：0
- Windows 版の場合：システム定義の pd_watch_pc_client_time オペランドで指定した値

pd_watch_pc_client_time オペランドについては，マニュアル「HiRDB システム定義」を参照してください。

《留意事項》

1. OLTP 下の UAP をクライアントとする運用形態の場合，及び Java EE アプリケーションサーバのコネクションプーリングを使用する場合など，HiRDB システムと常に接続している UAP では，次に示すどれかの設定をしてください。

   1.PDSWATCHTIME に 0 を指定する。

   2.PDSWATCHTIME に 0 以外を指定し，PDKALVL には 3 を指定する。PDKATIME には，PDSWATCHTIME より短い値を指定する。

   3.（Windows 環境の場合だけ）PDSWATCHTIME を未指定とし，PDKALVL には 3 を指定する。PDKATIME には，システム定義の pd_watch_pc_client_time オペランドで指定した値より短い値を指定する。

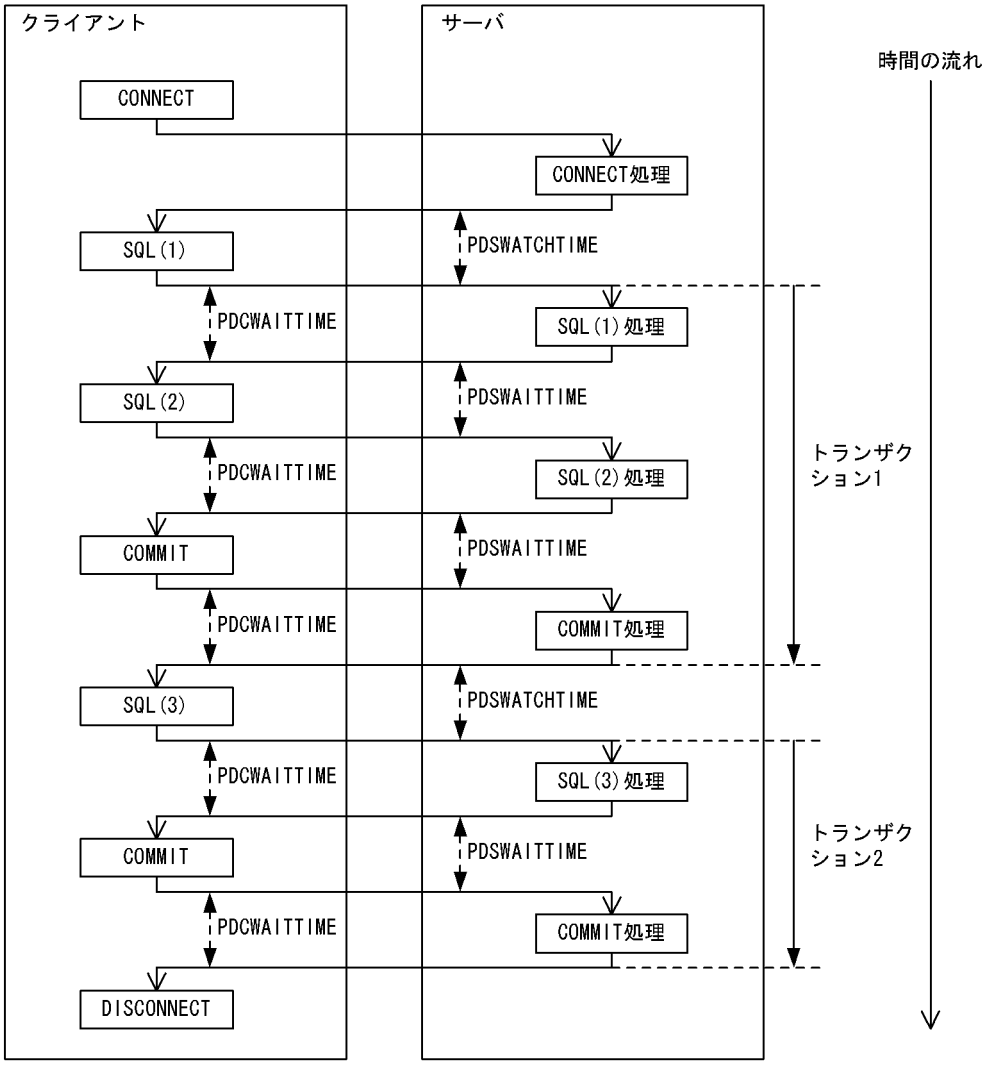
   上記 1 の場合，通信障害が発生したときにサーバ側プロセスが残存するおそれがあるため，2,3 を推奨します。ただし，シングルスレッド版のクライアントライブラリを使用している UAP の場合には，PDKALVL が無効になるため，1 を設定してください。

2. HiRDB サーバが HiRDB クライアントとの接続を切断しても，HiRDB サーバは HiRDB クライアントに切断したことを通知しないので，注意が必要です。

《ほかの環境変数との関係》

PDCWAITTIME，PDSWAITTIME，及び PDSWATCHTIME の関係を次の図に示します。

図 6–5　PDCWAITTIME，PDSWAITTIME，及び PDSWATCHTIME の関係



［説明］

PDSWATCHTIME：サーバが，トランザクション処理以外（SQL実行開始からコミット，又はロールバックするまでの区間外）の時間を監視します。指定した時間内にクライアントから要求が来なかった場合，クライアントとの接続を切断します。

PDCWAITTIME：クライアントが，サーバへ要求をしてから応答が戻ってくるまでの時間を監視します。指定した時間内にサーバから応答が返ってこなかった場合，サーバとの接続を切断します。

PDSWAITTIME：サーバが，クライアントからの要求に対する応答をしてから，次にクライアントから要求が来るまでの時間を監視します。指定した時間内にクライアントから要求が来なかった場合，クライアントとの接続を切断します。

(60)　PDCWAITTIMEWRNPNT=SQL 実行時間警告出力の契機

SQL 実行時間警告出力機能使用時に，SQL 実行時間警告情報ファイルを出力する契機を指定します。

SQL 実行時間警告出力機能とは，SQL の実行時間がある一定時間を超えた場合，SQL 実行時間警告情報ファイルと警告メッセージ（KFPA20009-W）を出力する機能のことをいいます。SQL 実行時間警告出力機能については，マニュアル「HiRDB システム運用ガイド」を参照してください。

SQL 実行時間警告情報ファイルを出力する契機は，次のどれかの方法で指定します。

PDCWAITTIME の指定値に対する比率（小数点を指定しない場合）：

～<符号なし整数>((0～99))（単位：%）

PDCWAITTIME の指定値に対する比率を指定します。例えば，PDCWAITTIME オペランドに 100（秒）を指定し，PDCWAITTIMEWRNPNT に 90（%）を指定すると，SQL の実行後に HiRDB が SQL の実行時間を調べます。その結果，SQL の実行時間が 90 秒以上 100 秒未満の場合に警告情報が出力されます。

PDCWAITTIME の指定値に対する比率（小数点を指定する場合）：

～<符号なし 10 進数>((0～99.999999))（単位：%）

PDCWAITTIME の指定値に対する比率（小数点を含む比率）を指定します。

SQL 実行時間警告出力の契機となる時間（auto 指定なし）：

～<符号なし 10 進数>sec((0～PDCWAITTIME))（単位：秒）

SQL 実行時間警告出力の契機となる時間を指定します（例えば，60 秒の場合は PDCWAITTIMEWRNPNT=60sec と指定します）。このとき，時間には小数点を指定できます。なお，上限値は，PDCWAITTIME の指定値未満となります。

SQL 実行時間警告出力の契機となる時間（auto 指定あり）：

～<符号なし整数>sec,auto((0～PDCWAITTIME))（単位：秒）

SQL 実行時間警告出力の契機となる時間を指定します（例えば，60 秒の場合は PDCWAITTIMEWRNPNT=60sec,auto と指定します）。上限値は，PDCWAITTIME の指定値未満となります。

auto を使用した場合，PDCWAITTIME の指定有無に関係なく，SQL の実行時間が PDCWAITTIMEWRNPNT に指定した時間以上であったとき，その SQL に対して警告情報を出力します。

《システム定義との関係》

この環境変数を省略すると，システム定義の pd_cwaittime_wrn_pnt オペランドの省略値又は指定値が仮定されます。pd_cwaittime_wrn_pnt オペランドについては，マニュアル「HiRDB システム定義」を参照してください。

(61) PDKALVL= {0 | 1 | 2 | 3}

・0904 互換モードの場合：《0》

HiRDB クライアントから HiRDB サーバに対して，定期的にパケットを送信する機能を使用するかどうかを指定します。

この環境変数の有効範囲を次の表に示します。

表 6–37　PDKALVL の有効範囲

クライアント種別				有効/無効
埋込み型 UAP	シングルスレッド		X/Open 準拠の API 使用	×
			X/Open 準拠の API 未使用	×
	マルチスレッド		X/Open 準拠の API 使用	○
			X/Open 準拠の API 未使用	○
JDBC ドライバ	Type2		JTA 使用	×
			JTA 未使用	○
	Type4	JDBC2.0	JTA 使用	○※
			JTA 未使用	○※
		JDBC4.0	JTA 使用	○
			JTA 未使用	○
		JDBC4.3	JTA 使用	○
			JTA 未使用	○
ODBC ドライバ				○
OLE DB プロバイダ				○
HiRDB データプロバイダ for .NET Framework				○
SQLJ				○

（凡例）

○：有効

×：無効

注※

Type4 JDBC ドライバ（JDBC2.0 版）では，PDKALVL=3 を指定できません。

0 以外を指定した場合，HiRDB との接続ごとにパケット送信スレッドを一つ生成します。アプリケーションサーバを使用するなどして HiRDB との接続を多数生成する場合は，システムやプロセスでの作成スレッド数を制御する OS のパラメタを見直してください。パケットの送信間隔は，PDKATIME で指定できます。

0：

定期的にパケットを送信する機能を使用しません。

1：

定期的にパケットを送信する機能を使用します。パケット送信スレッドは，一定時間間隔で HiRDB サーバとの接続経路にパケットを送信します。

HiRDB サーバで時間監視している PDSWAITTIME，及び PDSWATCHTIME の監視時間をリセットしません。

HiRDB クライアントと HiRDB サーバが同一マシンの場合，1 を指定しないでください。

2：

定期的にパケットを送信する機能を使用します。パケット送信スレッドは，一定時間間隔で HiRDB サーバとの接続経路にパケットを送信し，HiRDB サーバからの返信パケットを受信します。

HiRDB サーバで時間監視している PDSWAITTIME，及び PDSWATCHTIME の監視時間をリセットします。

このパケット通信でエラーが発生した場合，SQL 実行スレッドが次の SQL を実行するときに，対応するエラーをアプリケーションに返却します。

パケット送信スレッドが HiRDB サーバからの応答の返却を待っている間に，SQL 実行スレッドにアプリケーションから SQL 要求があった場合，パケット送信スレッドが HiRDB サーバからの応答を受信するまで SQL 実行スレッドは待ち状態になります。このため，SQL 実行時間が遅くなることがあります。また，設定値に 1 を指定した場合よりも CPU 使用率が高くなります。HiRDB サーバで時間監視している PDSWAITTIME，及び PDSWATCHTIME の監視時間をリセットする必要のない場合は，設定値に 1 を指定することをお勧めします。

3：

定期的にパケットを送信する機能を使用します。パケット送信スレッドは，一定時間間隔で HiRDB サーバとの接続経路にパケットを送信し，HiRDB サーバからの返信パケットを受信します。

また，HiRDB サーバで時間監視している PDSWATCHTIME の監視時間をリセットします。PDSWAITTIME の監視時間はリセットしません。

コネクションプーリング使用時には，プーリング中の接続が PDSWATCHTIME の監視で切断されないようにするため，PDSWATCHTIME に PDKATIME よりも長い値を設定する必要があります。このような環境では，設定値に 3 を指定することをお勧めします。

このクライアント環境定義に PDKALVL=3 を指定し，10-10 より前の HiRDB サーバに接続した場合，エラーとなります。

《適用基準》

ルータやファイアウォールなどのネットワーク管理アプリケーションでは，一定時間パケットが流れないと接続を切断する，無通信時間監視機能を備えていることがあります。この環境変数に 0 以外を指定することで，HiRDB の接続を保持したまま，サービスの要求を待機する Web アプリケーションなどが，ネットワーク管理アプリケーションによって HiRDB の接続を不当に切断されることを防げます。

また，HiRDB サーバでの時間監視（PDSWAITTIME，及び PDSWATCHTIME）を無限にしておくと，HiRDB クライアント側のマシンダウンやネットワーク障害の場合，HiRDB サーバプロセスが残ることがあります。この環境変数に 2 を指定することで，HiRDB サーバの時間監視を無限に設定しないで，HiRDB サーバの時間監視での接続の切断も防げます。この時，PDKATIME は HiRDB サーバでの時間監視の指定値より小さい値を指定してください。HiRDB サーバでの時間監視の指定値より大きい値を指定すると，HiRDB サーバの時間監視で接続が切断されます。

コネクションプーリングを使用している場合，この環境変数に 3 を設定することで，HiRDB サーバでの時間監視時間（PDSWAITTIME，PDSWATCHTIME）で PDSWAITTIME の監視時間がリセットされなくなり，適切な値を設定できます。

《適用例》

1. 次の条件の場合には，PDKALVL に 1 を設定し，PDKATIME に Firewall の監視時間（例：1,200 秒）より少し短い時間（1,000 秒）を指定します。
   - Web アプリケーションでの DB サーバへの SQL 実行要求が不定期で，長時間 SQL が実行されないことがある。
   - Web サーバと DB サーバとの間に Firewall があり，一定時間パケットが流れないで接続が切られてしまうことがある。
2. 次の条件の場合には，PDKALVL に 3 を設定し，PDKATIME に PDSWATCHTIME の監視時間（例：3,600 秒）より少し短い時間（3,000 秒）を指定します。
   - コネクションをプーリングするアプリケーションから，HiRDB にアクセスする。
   - SQL 実行要求ごとに接続を再利用するが，長時間使用されない接続があり，PDSWATCHTIME の監視時間で接続が切られることがある。
3. DB サーバへの SQL 実行要求の間隔が長くなることがあり，PDSWAITTIME の監視時間で接続が切られることを避けたい場合には，PDKALVL に 2 を指定し，PDKATIME に PDSWAITTIME の監視時間（例：600 秒）より少し短い時間（500 秒）を指定します。
4. PDSWATCHTIME で HiRDB クライアント側のマシンダウンやネットワーク障害を監視し，PDSWAITTIME でクライアントアプリケーション内の処理遅延を監視したい場合には，次の指定をしてください。
   - 10-10 以降の HiRDB サーバに接続する場合

     PDKALVL に 3 を指定し，PDKATIME は次の条件を満たす値となるよう設定してください。

     PDKATIME ＜ PDSWATCHTIME

     例として，PDKATIME に PDSWATCHTIME の監視時間（例：800 秒）よりも短い時間（700 秒）を指定します。PDSWAITTIME には，PDKATIME の値よりも長い時間（例：900 秒）を指定できます。
   - 10-10 より前の HiRDB サーバに接続する場合

     PDKALVL に 2 を指定し，PDKATIME は次の条件を満たす値となるよう設定してください。

   PDSWAITTIME ＜ PDKATIME ＜ PDSWATCHTIME

   例として，PDKATIME に PDSWAITTIME の監視時間（例：600 秒）と PDSWATCHTIME の監視時間（例：800 秒）の間の時間（700 秒）を指定します。

(62)　PDKATIME=パケットの送信間隔

～＜符号なし整数＞((60～65535))（単位：秒）

・0904 互換モードの場合：《3000》

HiRDB クライアントから HiRDB サーバに対して，定期的にパケットを送信する間隔を指定します。リセットしたい監視時間よりも，短い時間を設定してください。

また，PDKATIME には，PDSWAITTIME と同じ値を指定しないでください。

この環境変数は，PDKALVL に 0 以外を指定した場合にだけ有効になります。

パケット送信時に，SQL 実行スレッドが SQL 実行中の場合は，パケット送信スレッドはパケット送信をしないで，次回の送信まで待ち状態になります。

PDKATIME のデフォルト値を次に示します。

- PDKALVL に 2（デフォルト）を指定し，PDSWAITTIME の指定がない場合：《540》
- 上記以外：《600》

(63) PDTIMEDOUTRETRY=ETIMEDOUT 発生時のリトライ回数

～<符号なし整数>((0～32767))《2》

HiRDB クライアントが HiRDB サーバと接続する場合に実行する，connect()システムコールで winsock の WSAETIMEDOUT エラー（UNIX 版の場合は ETIMEDOUT エラー）が発生したときに，connect()システムコールをリトライする回数を指定します。

《利点》

HiRDB サーバへの connect()が集中し，listen キューが一杯になった場合，connect()から WSAETIMEDOUT エラー又は ETIMEDOUT エラーが返されます。このとき，connect()システムコールをリトライすることで，接続エラーを回避できます。

《留意事項》

ネットワーク障害，及びサーバマシン電源ダウンが原因で発生する WSAETIMEDOUT エラー又は ETIMEDOUT エラーの場合，connect()システムコールからのリターンに時間が掛かることがあります。したがって，リトライ回数を多く設定した場合，UAP に接続エラーが返却されるまで時間が掛かることになります。特に，系切り替え時にクライアント接続用の IP アドレスを引き継がない場合，ネットワーク障害などが発生したときには，待機系に切り替わるまで時間が掛かります。このような環境では，リトライ回数を少なく設定することで待機系への切り替え時間を短くできます。

また，クライアント環境変数 PDNBLOCKWAITTIME に 0 以上を指定している場合，PDTIMEDOUTRETRY の指定は無効になります。

(64) PDCONREFRCOUNT=ECONNREFUSED 発生時のリトライ回数

～<符号なし整数>((0～32767))《9》

HiRDB クライアントが HiRDB サーバと接続する場合に実行する，connect()システムコールで winsock の WSAECONNREFUSED エラー（UNIX 版の場合は ECONNREFUSED エラー）が発生したときに，connect()システムコールをリトライする回数を指定します。

《利点》

このクライアント環境定義に 1 以上の値を指定すると，HiRDB クライアントはクライアント環境定義 PDCONREFRINTERVAL に指定した間隔で connect()システムコールをリトライします。これによって，一時的なネットワーク障害や HiRDB サーバが起動中の場合に発生する接続エラーを回避できます。

《留意事項》

connect()システムコールのリトライにかかる最大時間は，このクライアント環境定義とクライアント環境定義 PDCONREFRINTERVAL の指定値を乗算した値になります。このため，停止中の HiRDB サーバに接続しようとして WSAECONNREFUSED エラー又は ECONNREFUSED エラーが発生した場合，UAP に接続エラーが返却されるまでの時間が遅くなります。したがって，接続エラーを即時検知したい場合はこのクライアント環境定義に 0 を指定してください。

(65) PDCONREFRINTERVAL=ECONNREFUSED 発生時のリトライ間隔

～<符号なし整数>((0～32767))（単位：ミリ秒）

・UNIX 版の場合：《700》

・Windows 版の場合：《0》

HiRDB クライアントが HiRDB サーバと接続する場合に実行する，connect()システムコールで winsock の WSAECONNREFUSED エラー（UNIX 版の場合は ECONNREFUSED エラー）が発生したときに，connect()システムコールをリトライする間隔を指定します。

このクライアント環境定義は，クライアント環境定義 PDCONREFRCOUNT に 0 を指定した場合は無効になります。

(66) PDNBLOCKWAITTIME=ノンブロックモードでのコネクション確立監視時間

～<符号なし整数>((0～120))《0》（単位：秒）

HiRDB サーバ，HiRDB クライアント間でコネクション接続完了を監視する場合，ノンブロックモード時のコネクション確立監視時間を指定します。

この環境変数に 1 以上を指定すると，HiRDB サーバ，HiRDB クライアント間の通信をノンブロック通信にして，connect()システムコールの終了を監視します。これを**ノンブロックモード**といいます。0 を指定した場合は，OS のタイムアウト時間までコネクション接続完了を待ちます。これを**ブロックモード**といいます。

《適用基準》

LAN 障害時の，connect()システムコールが数十秒（OS に依存）待たされることを回避したいときに指定します（ノンブロックモードにします）。これを指定することで，LAN 障害を早く検知できます。なお，この環境変数に 0 以上を指定した場合，PDTIMEDOUTRETRY の指定の有無に関係なく，PDTIMEDOUTRETRY の指定は無効になります。

この環境変数の指定を省略した場合，又は 0 を指定した場合，HiRDB サーバから HiRDB クライアントへの通信時のコネクション確立方式は，システム共通定義 pd_ipc_clt_conn_nblock オペランドに指定した値によって決まります。

この環境変数と pd_ipc_clt_conn_nblock オペランドとの組み合わせで，コネクション確立をどちらのモードで行うかを次の表に示します。

クライアント環境定義 PDCONTYPE に ACTIVE を指定している場合，HiRDB サーバから HiRDB クライアントへのコネクション確立は行いません。

クライアント環境変数 PDNBLOCKWAITTIME の指定値	pd_ipc_clt_conn_nblock オペランドの指定値	
	Y	N
0	• **クライアントがサーバにコネクション確立する場合** ブロックモード。 • **サーバがクライアントにコネクション確立する場合** ノンブロックモード。 このとき，pd_ipc_clt_conn_nblock_time オペランドに指定した監視時間で監視します。	ブロックモード。
1 以上	ノンブロックモード。 このとき，クライアント環境変数の PDNBLOCKWAITTIME に指定した監視時間で監視します。	

《見積もり方法》

指定値が小さ過ぎる場合，ネットワークの状態によっては不当にエラーとなることがあります。指定値には，次の計算式以上の値を設定してください。

MAX（A ＋ 1，8）

A：

ping などの OS コマンドで，HiRDB サーバ，HiRDB クライアント間の到達時間を計測した値です。なお，ネットワークの負荷によっては，ping などの到達時間は変動します。最も負荷の高い状態を想定して測定してください。

《注意事項》

この環境変数には，コネクション確立をブロックモードで行ったときの OS 待ち時間よりも大きい値を指定しないでください。指定した場合は，OS の待ち時間で接続タイムアウトになります。

HiRDB サーバとの接続処理は，大きく分けて次の 2 つから成ります。

1. TCP/IP のコネクション確立処理(connect()システムコール)
2. HiRDB サーバとのユーザ認証などのネゴシエーション

この環境変数は 1.を監視対象としており，2.は監視対象に含みません。2.を監視する場合，クライアント環境定義 PDCONNECTWAITTIME を指定する必要があります。

(67) PDCONNECTWAITTIME=サーバ接続時の HiRDB クライアントの最大待ち時間

～<符号なし整数>((1～300))《300》（単位：秒）

HiRDB サーバとの接続時，HiRDB サーバから応答が戻ってくるまでの HiRDB クライアントの最大待ち時間を指定します。

HiRDB サーバが HiRDB クライアントから接続要求を受け取った状態で，系切り替えやシステムダウンが発生した場合，HiRDB クライアントは指定した時間だけ応答を待ちます。

《適用基準》

系切り替え機能を使用している場合，早期にアプリケーションに障害を検知させるときに指定します。この環境変数と同時に PDNBLOCKWAITTIME を指定すると，更に検知が早くなります。

《見積もり方法》

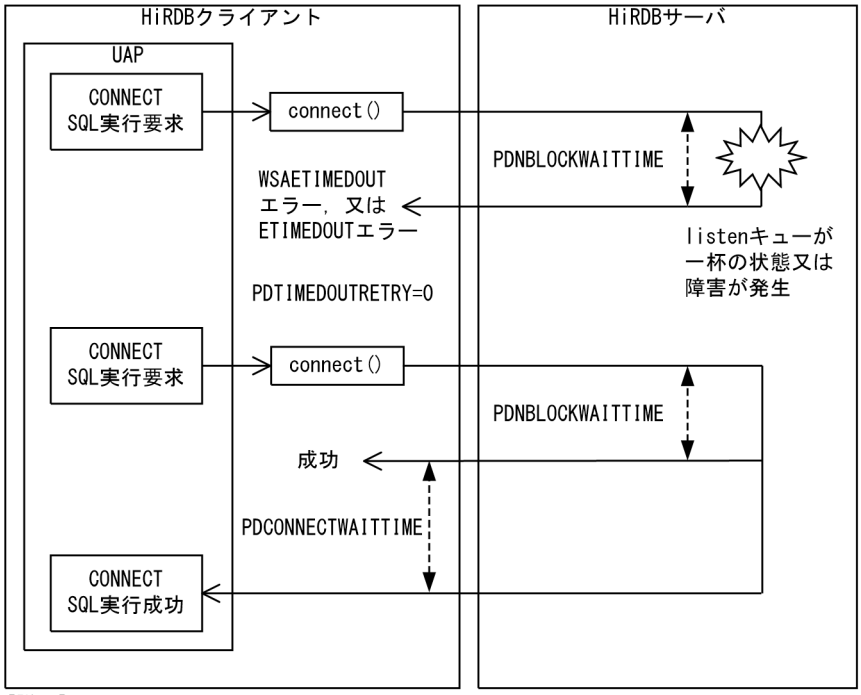
指定値が小さ過ぎる場合，ネットワークの状態や接続時のスケジュール待ちで時間が掛かって，正常な接続処理がエラーとなることがあります。指定値には，次の計算式以上の値を設定してください。

MIN（システム定義 pd_max_users オペランドの値×0.2，300）

《ほかの環境変数との関係》

PDTIMEDOUTRETRY，PDNBLOCKWAITTIME，及び PDCONNECTWAITTIME の関係を次の図に示します。

図 6–6　PDTIMEDOUTRETRY，PDNBLOCKWAITTIME，及び PDCONNECTWAITTIME の関係



[説明]
PDNBLOCKWAITTIME：connect()システムコールの終了までを監視します。
1以上を指定したノンブロックモードで，障害発生を早く検知できます。
PDTIMEDOUTRETRY：WSAETIMEDOUTエラー，又はETIMEDOUTエラーが発生した場合，要求をリトライする回数を指定します。PDNBLOCKWAITTIMEに0以上を指定している場合，指定の有無に関係なく0が仮定されます。
PDCONNECTWAITTIME：クライアントは指定した時間まで，サーバからの応答を待ちます。

(68)　PDCLTPATH=トレースファイル格納ディレクトリ

～<パス名>((最大 256 バイト))《カレントディレクトリのパス名》

HiRDB クライアントが作成する SQL トレースファイル及びクライアントエラーログファイルの格納先ディレクトリを指定します。

(69)　PDSQLTRACE=SQL トレースファイルのサイズ

～<符号なし整数>((0，4096～2000000000))（単位：バイト）

UAP の SQL トレースを出力する，SQL トレースファイルのサイズを指定します。

0 を指定した場合は，ファイルを切り替えないで，一つのファイルへ出力し続けます。サイズの制限はありません。4,096～2,000,000,000 を指定した場合は指定値のサイズとなり，指定値のサイズを超えると出力先が切り替わります。省略した場合は，SQL トレースを出力しません。

SQL トレースについては，「SQL トレース機能」を参照してください。

《ほかの環境変数との関係》

SQL トレースは，PDCLTPATH で指定したディレクトリに出力されます。PDCLTPATH の指定がない場合，UAP を起動したときのカレントディレクトリ（OpenTP1 から起動される UAP の場合，%DCDIR%¥tmp¥home¥サーバ名 xx のディレクトリ）の下に出力されます。

《見積もり方法》

SQL トレースファイルのサイズは，取得したい SQL 文の数から求めてください。取得したい SQL 文の，それぞれの「1 行（80 バイト）＋ SQL 文のサイズ」を求めて，すべてを合計した値を指定値の目安としてください。

(70)　PDUAPERLOG=クライアントエラーログファイルのサイズ

～<符号なし整数>((0，4096～2000000000))《65536》（単位：バイト）

UAP のエラーログを出力する，クライアントエラーログファイルのサイズを指定します。

0 を指定した場合はファイルの最大サイズとなり，最大サイズを超えるとクライアントエラーログは出力されません。4,096～2,000,000,000 を指定した場合は指定値のサイズとなり，指定値のサイズを超えると出力先が切り替わります。

クライアントエラーログについては，「クライアントエラーログ機能」を参照してください。

《ほかの環境変数との関係》

クライアントエラーログは，PDCLTPATH で指定したディレクトリに出力されます。PDCLTPATH の指定がない場合，UAP を起動したときのカレントディレクトリ（OpenTP1 から起動される UAP の場合，%DCDIR%¥tmp¥home¥サーバ名××のディレクトリ）の下に出力されます。

(71)　PDERRSKIPCODE=SQLCODE［，SQLCODE］...

クライアントエラーログへのメッセージ出力を抑止する SQLCODE を指定します。SQLCODE は最大 10 個指定できます。

例えば，SQLCODE−901 と−917 を抑止する場合は，次のように指定します。

```
PDERRSKIPCODE=-901,-917
```

《利点》

UAP の構造によっては，SQL の処理で必然的にエラーが発生することがあります。通常の業務でこのようなエラーが頻繁に発生すると，ファイルシステムを圧迫してしまうおそれがあります。特に，X/Open に従った API を使用する UAP の場合，プロセスごとにクライアントエラーログファイルを二つ作成してしまいます。この環境変数を指定すると，特定のエラーについてはメッセージの出力を抑止できるので，ファイルシステムへの負荷を軽減できます。

《適用基準》

次の条件をすべて満たす場合に適用してください。

- UAP の構造上エラーが頻繁に発生する。
- あらかじめ障害の原因が特定できて，その要因を調査する必要がない。

この環境変数を指定すると，不慮の障害が発生したときにその原因を調査できなくなります。適用する場合には注意してください。

(72) PDPRMTRC= {YES | NO | IN | OUT | INOUT}

SQL トレースにパラメタ情報及び検索データを出力するかどうかを指定します。出力内容については，「SQL トレース機能」を参照してください。

YES：

SQL トレースに入力用パラメタ情報を出力します。YES を指定した場合，検索データ情報と入力パラメタを出力します。

NO：

SQL トレースにパラメタ情報を出力しません。

IN：

SQL トレースに入力用パラメタ情報を出力します。CALL 文の IN パラメタと INOUT パラメタ※も該当します。

OUT：

SQL トレースに出力用パラメタ情報，及び検索データ情報を出力します。CALL 文の OUT パラメタと INOUT パラメタ※も該当します。

INOUT：

SQL トレースに入力用パラメタ情報，出力用パラメタ情報，及び検索データ情報を出力します。CALL 文の INOUT パラメタ※は 2 回出力します。

注※

CALL 文の INOUT パラメタの情報は，出力データだけとなります。

PDSQLTRCTYPE=ERROR 指定の場合，PDPRMTRC に YES は指定できません。

(73) PDPRMTRCSIZE=SQL トレースに出力するパラメタ情報の最大データ長

～<符号なし整数>((4～32008))《256》（単位：バイト）

SQL トレースに出力するパラメタ情報及び検索データの最大データ長を指定します。可変長文字列型，BLOB 型，及び BINARY 型の場合，文字列長の領域もデータ長に含まれます。

この環境変数は，PDPRMTRC=NO 以外の場合に有効になります。

この環境変数の指定値を大きくすると，出力する情報量が増えます。そのため，SQL トレースファイルのサイズ（PDSQLTRACE の指定値）も大きくする必要があります。

(74) PDTRCMODE={ERR | NONE}

SQL トレース以外のトラブルシュート情報（pderr*.trc の情報）を出力するかどうかを指定します。

ERR：pderr*.trc の情報を出力します。

NONE：pderr*.trc の情報を出力しません。

(75) PDUAPREPLVL={[s] [u[o][t]] [p] [r] | [a [o][t]]}

UAP 統計レポートの出力情報を指定します。UAP 統計レポートを出力するファイルを UAP 統計レポートファイルといいます。この環境変数は，PDSQLTRACE を指定しているときに有効になります。

省略した場合，SQL トレース情報だけ出力されます。

なお，UAP 統計レポートについては，「UAP 統計レポート機能」を参照してください。

s：SQL 単位の情報が出力されます。また，SQL トレース情報も出力されます。

u：UAP 単位の情報が出力されます。

p：アクセスパス情報が出力されます。

r：SQL 実行時の中間結果情報が出力されます。

o：UAP 単位の情報にスレッド間ロック待ち時間が出力されます。u 又は a が指定されていない場合，このオプションを指定しても無視されます。このオプションを指定すると，システム全体の性能に影響を与えるおそれがあります。通常の運用では指定しないでください。

t：UAP 単位の情報をトランザクション単位に集計して出力します。u 又は a が指定されていない場合，このオプションを指定しても無視されます。

a：supr を指定した場合と同じ情報が出力されます。

s，u，p，及び r を組み合わせて指定できます（su，sr，upr など）。u 又は a を指定した場合，o，t も指定できます。

a，o，及び t を組み合わせた指定方法と，同じ情報が出力される s，u，o，t，p，及び r を組み合わせた指定方法を次に示します。

a，o，及び t を組み合わせた指定方法	s，u，o，t，p，及び r を組み合わせた指定方法
a	supr
ao	suopr
aot	suotpr
at	sutpr

例えば，a を指定した場合と supr を指定した場合では同じ情報が出力されます。なお，s 又は a を指定しない場合，SQL トレース情報は出力されません。

PDSQLTRCTYPE=ERROR 指定の場合，PDUAPREPLVL は指定できません。

《留意事項》

1. アクセスパス情報，又は SQL 実行時の中間結果情報を取得する場合，SQL オブジェクトがバッファ中にあっても SQL オブジェクトを再作成するため，サーバの負荷が増えることがあります。
2. 次の場合で UAP 単位の情報を出力したい場合は，t を指定してトランザクション単位に集計して出力してください。t を指定しない場合は，情報は出力しません。
   ・OLTP 下の X/Open に従った API を使用するプログラムの場合
   ・DISCONNECT をしないで UAP が終了した場合
   ・接続プーリング機能を使用している場合
3. アクセスパス情報，及び SQL 実行時の中間結果情報は，1 ギガバイトを超えると出力されません。
4. 時間の表示（SQL の実行時間，排他待ち時間，CPU 時間など）は，OS のシステムコールで取得できない小さい値があると，0 が表示されます。
5. HiRDB/パラレルサーバの場合，CONNECT したディクショナリサーバでの権限チェック処理は，UAP 単位の情報には含まれません。
6. アクセスパス情報，又は SQL 実行時の中間結果情報の出力を指定して，プロセス間メモリ通信機能を指定した場合（クライアント環境定義 PDIPC=MEMORY を指定した場合），PDIPC の指定は無効になり PDIPC=DEFAULT となります。
7. t を指定している場合でも，手続き中で決着したトランザクションについては出力されません。手続き中で決着したトランザクションの情報については，後続のトランザクションの情報に含めて集計します。

(76)　PDREPPATH=UAP 統計レポートファイルの格納ディレクトリ

～＜パス名＞((最大 256 バイト))

UAP 統計レポートファイルを，PDCLTPATH で指定したディレクトリとは別のディレクトリに作成する場合に指定します。この環境変数は，PDUAPREPLVL を指定しているときだけ有効になります。

PDREPPATH を指定した UAP 統計レポートファイルの出力ファイル名については，「SQL トレース機能」を参照してください。

(77)　PDTRCPATH=動的 SQL トレースファイルの格納ディレクトリ

～＜パス名＞((最大 256 バイト))

HiRDB クライアントが作成する動的 SQL トレースファイルの格納先ディレクトリを指定します。トレース取得コマンド（pdtrcmgr）で動的 SQL トレースファイルを取得する場合には，この環境変数を必ず指定してください。

ここで指定したディレクトリを pdtrcmgr で指定すると，次回の CONNECT から，指定したディレクトリに SQL トレースファイルが作成されます。pdtrcmgr については，「SQL トレース動的取得機能」を参照してください。SQL トレースについては，「SQL トレース機能」を参照してください。

(78)　PDSQLTRCOPENMODE=｛CNCT ｜ SQL｝

SQL トレースファイルのオープンモードを指定します。

PDSQLTRCTYPE=ERROR 指定の場合，この指定は無視して，SQL エラー発生時に SQL トレースファイルをオープン，クローズします。

CNCT：

CONNECT，DISCONNECT 単位に SQL トレースファイルをオープン，クローズして，トレース情報を出力します。CNCT を指定した場合，PDSQLTRCOPENMODE に SQL を指定するよりオーバヘッドが削減されるため，SQL トレースを出力するための時間が短縮できます。

SQL：

オペレーション単位（SQL 単位）に SQL トレースファイルをオープン，クローズして，トレース情報を出力します。

《留意事項》

1. CNCT を指定した場合，SQL トレースファイルをオープンしたままで情報を書き込むため，正常に DISCONNECT できなかったときには，SQL トレース情報が欠落することがあります。
2. ほかのクライアント環境変数の指定値によっては CNCT を指定しても無効となります。この場合は SQL として動作します。ほかのクライアント環境変数の指定値による CNCT が有効となる組み合わせを次の表に示します。ほかのクライアント環境変数については，各クライアント環境変数の説明を参照してください。

表 6–38　ほかのクライアント環境定義指定値による CNCT が有効となる組み合わせ

クライアント環境変数 PDSQLTRCFMT	クライアント環境変数 PDREPPATH			
	指定なし			指定あり
	X/Open に従った API を使用した接続形態以外（非 XA 接続）	X/Open に従った API を使用した接続形態（XA 接続）		
		クライアント環境変数 PDXATRCFILEMODE		
		SEPARATE	LUMP	
1	×	×	×	○
2	○	○	×	○

（凡例）

○：CNCT は有効になります。

×：CNCT は有効になりません（SQL として動作します）。

(79)　PDSQLTEXTSIZE=SQL 文のサイズ

～＜符号なし整数＞((4096～2000000))《4096》

Type4 JDBC ドライバ使用時（単位：Unicode コード単位の数）

上記以外（単位：バイト）

SQL トレースに出力する SQL 文のサイズを指定します。

アクセスパス取得時に省略した場合，4096 ではなく，2000000 が仮定されます。

Type4 JDBC ドライバでは SQL 文のサイズを Unicode コード単位※で扱うため，サロゲートペアの場合は 1 文字を表現する Unicode コード単位の数が 2 以上となります。

（例）

「 𠮷 」は「0xD840 0xDC0B」のサロゲートペアです。文字数は 1 で Unicode コード単位の数は 2 となります。

このため，SQL 文中にサロゲートペアを含んでいる場合，SQL 文の文字数が指定値の範囲内であっても SQL トレースに出力する SQL 文が欠落することがあります。

注※

Unicode コード単位とは，UTF-16 エンコーディングのコード単位である 16 ビット char 値を示します。

(80)　PDSQLEXECTIME={YES ｜ NO}

・0904 互換モードの場合：《NO》

SQL トレースに SQL 実行時間を出力するかどうかを指定します。

推奨モードを適用している場合は，指定する必要はありません。0904 互換モードを適用している場合は，YES を指定することを検討してください。

YES：

SQL 実行時間を出力します。

出力される SQL 実行時間の単位はマイクロ秒となります。SQL トレースに出力される値は，実行時間が 24 時間以上のものは正常に出力されません。

NO：

SQL 実行時間を出力しません。

(81) PDRCTRACE=再接続トレースファイルのサイズ

～<符号なし整数>((0，4096～2000000000))（単位：バイト）

UAP の再接続トレースを出力するファイルのサイズを指定します。

0 を指定した場合はファイルの最大サイズとなり，最大サイズを超えると UAP の再接続トレースは出力されません。また，省略した場合も UAP の再接続トレースは出力されません。

4,096～2,000,000,000 を指定した場合は指定値のサイズとなり，指定値のサイズを超えると出力先が切り替わります。

再接続トレースは，PDCLTPATH で指定したディレクトリに出力されます。PDCLTPATH を指定していない場合は，UAP を実行したときのカレントディレクトリ（Cosminexus から実行される UAP の場合は J2EE サーバの実行時のカレントディレクトリ）下に出力されます。再接続トレースについては，「再接続トレース機能」を参照してください。

(82) PDWRTLNPATH=WRITE LINE 文の値式の値を出力するファイルの格納先ディレクトリ

～<パス名>((最大 256 バイト))

WRITE LINE 文の値式の値を出力する，ファイルの格納先ディレクトリを指定します。WRITE LINE 文については，マニュアル「HiRDB SQL リファレンス」を参照してください。

この環境変数を省略した場合，PDCLTPATH で指定したディレクトリが仮定されます。

(83) PDWRTLNFILSZ=WRITE LINE 文の値式の値を出力するファイルの最大サイズ

～<符号なし整数>((0，4096～2000000000))（単位：バイト）

WRITE LINE 文の値式の値を出力する，ファイルの最大サイズを指定します。

0 を指定した場合のファイルの最大サイズは OS で管理できる最大サイズとなり，最大サイズを超えると WRITE LINE 文の値式の値は出力されません。また，省略した場合も，WRITE LINE 文の値式の値は出力されません。

4,096～2,000,000,000 を指定した場合は指定値のサイズとなり，指定値のサイズを超えると出力先が切り替わります。

ファイルは二つ作成されます。作成されるファイル名を次に示します。

クライアント種別	ファイル名
埋込み型 UAP	TX_関数を使用していない場合： pdwrtln1.trc pdwrtln2.trc TX_関数を使用している場合： pdwrtlnxxxxx-1.trc pdwrtlnxxxxx-2.trc
ODBC ドライバ Type2 JDBC ドライバ SQLJ OLE DB プロバイダ HiRDB データプロバイダ for .NET Framework	pdwrtln1.trc pdwrtln2.trc
Type4 JDBC ドライバ	クライアント環境定義 PDCONTYPE に PASSIVE を指定した場合： pdwrtlnXXXXXXXX_ppppp_1.trc pdwrtlnXXXXXXXX_ppppp_2.trc クライアント環境定義 PDCONTYPE に ACTIVE を指定した場合： pdwrtlnHHMMSSfff_XXXXXXXX_YYYYYYYYYY_1.trc pdwrtlnHHMMSSfff_XXXXXXXX_YYYYYYYYYY_2.trc

（凡例）

xxxxx：クライアントプロセス ID
XXXXXXXX：接続サーバ名
ppppp：クライアント受信ポート番号
HHMMSSfff：コネクト要求時間（HH：時，MM：分，SS：秒，fff：ミリ秒）
YYYYYYYYYY：コネクト通番

《留意事項》

1. この環境変数を指定して，かつ PDIPC=MEMORY を指定した場合，PDIPC の指定が無効になります。
2. ファイルは，PDWRTLNPATH で指定したディレクトリに出力されます。
3. 値を出力しているときにファイルの容量が一杯になると，もう一方のファイルに値を出力します。このとき，切り替え先のファイルに格納されている情報は消去され，新しい情報が書き込まれます。このため，必要な情報がある場合は，切り替えが発生する前に退避するなどしておいてください。

なお，現在使用しているファイルを知りたい場合，次の方法で確認できます。このとき，最終更新日付の新しい方が現在使用しているファイルとなります。

・UNIX 版の場合は，OS の ls -l コマンドを実行します。

・Windows 版の場合は，コマンドプロンプトから DIR コマンドを実行，又はエクスプローラで参照します。

(84)　PDWRTLNCOMSZ=WRITE LINE 文の値式の値の合計サイズ

～<符号なし整数>((1024～131072))《1024》（単位：バイト）

WRITE LINE 文の値式の値の合計サイズを指定します。

WRITE LINE 文の値式の値の合計サイズが，この環境変数の指定値を超えた場合，超えた分の情報は無効になります。また，この場合，次の行に「**PDWRTLNCOMSZover**」が出力されます。

(85)　PDUAPEXERLOGUSE ＝{YES ｜ NO}

拡張 SQL エラー情報出力機能を使用するかどうかを指定します。

拡張 SQL エラー情報出力機能については，「拡張 SQL エラー情報出力機能」を参照してください。

YES：

拡張 SQL エラー情報出力機能を使用します。

NO：

拡張 SQL エラー情報出力機能を使用しません。

《システム定義との関係》

この環境変数を省略すると，システム定義の pd_uap_exerror_log_use オペランドの省略値又は指定値が仮定されます。

(86)　PDUAPEXERLOGPRMSZ ＝パラメタ情報の最大データ長

～<符号なし整数>((0～32008))（単位：バイト）

拡張 SQL エラー情報出力機能使用時に，クライアントエラーログファイル及び SQL エラーレポートファイルに出力するパラメタ情報の最大データ長を指定します。1 以上を指定した場合はパラメタ情報を出力しますが，0 を指定した場合はパラメタ情報を出力しません。

《システム定義との関係》

この環境変数を省略すると，システム定義の pd_uap_exerror_log_param_size オペランドの指定値が仮定されます。

《留意事項》

1. 可変長文字列型，BLOB 型，及び BINARY 型のデータの場合，データ長の領域も環境変数の指定値に含まれます。

2. 出力するパラメタ情報のデータ長が，この環境変数の指定値を超えた場合，超えた分の情報は切り捨てられます。

(87) PDDNDPTRACE ＝メソッドトレースのファイルサイズ

～＜符号なし整数＞((0，65536～2147483647))（単位：バイト）

ADO.NET 2.0 に対応した HiRDB データプロバイダ for .NET Framework で出力するメソッドトレースのファイルサイズを指定します。

指定値がある場合は，PDCLTPATH で指定されたディレクトリにメソッドトレースを出力します。

0 を指定した場合はマシンのディスクに空き領域がある限り出力します。65536～2147483647 を指定した場合は指定値が最大サイズとなり，最大サイズを超えるとトレース出力先が切り替わります。省略した場合は，メソッドトレースを出力しません。メソッドトレースについては，「HiRDB データプロバイダ for .NET Framework のトラブルシュート機能」を参照してください。

(88) PDSQLTRCFMT={1 ｜ 2}

・0904 互換モードの場合：《1》

SQL トレースの出力形式を指定します。出力形式により出力ファイル名称が異なります。出力ファイル名称及び指定値ごとの出力形式については，「SQL トレース機能」を参照してください。この環境変数は，PDSQLTRACE を指定しているときに有効になります。

出力形式 2 は，出力形式 1 よりも詳細情報を出力します。このため，通常は指定を省略するか 2 を指定してください。

PDSQLTRCTYPE=ERROR 指定の場合，PDSQLTRCFMT には 2 だけを指定できます。

1：

出力形式 1 で出力します。

2：

出力形式 2 で出力します。

《留意事項》

HiRDB SQL Tuning Advisor 08-03 以前を使用して SQL トレースの解析を行う場合，出力形式 2 には対応していないため 1 を指定してください。

(89) PDSQLTRCTYPE={SQL ｜ ERROR}

PDSQLTRACE 指定によって取得する SQL トレース情報の出力タイミングを指定します。PDSQLTRCTYPE は，PDSQLTRACE を指定した場合に有効になります。

SQL：

SQL 実行時に出力します。

ERROR：

エラー発生時に出力します。

SQL 実行時に SQL トレース情報をメモリ上に退避しておき，エラー発生時に退避している分を出力します。エラー発生時にだけ出力するため，SQL 指定に比べ，SQL トレース情報出力による性能劣化を軽減できます。

SQL トレース情報を退避できる上限については，次のクライアント環境定義で指定し，上限に達した場合は最も古い情報から順に上書きします。

- クライアントライブラリ使用時：PDSQLTRCBUFFSIZE
- Type4 JDBC ドライバ使用時：PDJDBSQLTRCMEMNUM

SQL トレースファイルは，SQL エラー発生時にオープンし，SQL トレース情報出力後クローズします。次のクライアント環境定義に影響があります。

クライアント環境定義	PDSQLTRCTYPE ＝ ERROR 指定時の影響
PDPRMTRC	YES は指定不可。指定すると接続時エラーになります。
PDUAPREPLVL	指定不可。指定すると接続時エラーになります。
PDSQLTRCOPENMODE	指定しても無効になります。 SQL トレースファイルは，SQL エラー発生時にオープンし，SQL トレース情報出力後クローズします。
PDSQLTRCFMT	2 以外指定不可。1 を指定すると接続時エラーになります。

(90)　PDJDBSQLTRCMEMNUM=メモリ上に退避する SQL トレース情報数

〜<符号なし整数>((100〜10000))《100》（単位：なし）

PDSQLTRCTYPE=ERROR を指定した場合に，メモリ上に退避する SQL トレース情報数を指定します。Type4 JDBC ドライバを使用し PDSQLTRCTYPE=ERROR 指定の場合に有効になります。

(91)　PDSQLTRCBUFFSIZE=SQL トレース情報の退避領域サイズ

〜<符号なし整数>((10〜2000000))《50》（単位：キロバイト）

PDSQLTRCTYPE=ERROR を指定した場合に使用する，SQL トレース情報の退避領域のサイズを指定します。クライアントライブラリを使用し PDSQLTRCTYPE=ERROR 指定の場合に有効になります。

SQL トレース情報の退避に使用するメモリは，次に示す計算式で求めてください。

SQL トレース情報数×376 ＋ a ＋ b ＋ c（単位：バイト）

a：SQL 文を退避する領域長

SQL トレースに SQL 文が出力される場合，出力ごとに 4 ＋↑SQL 文長÷4↑×4 を加算します。SQL 文が出力される契機については，「SQL トレース機能」の「SQL トレース情報の見方」を参照してください。

SQL 文長はクライアントの文字コードでのバイト数であり，最大長は以下のどちらかとなります。

- **クライアント環境定義 PDCLTCNVMODE が UCS2_UTF8，UCS2_UJIS，UCS2_UJIS2 のどれかである場合**

  クライアント環境定義 PDSQLTEXTSIZE× 2

- **上記以外の場合**

  クライアント環境定義 PDSQLTEXTSIZE

  また，退避領域サイズに対し SQL 文が大き過ぎる場合，クライアント環境定義 PDSQLTEXTSIZE のデフォルト値で，SQL 文を切り捨てます。

b：トランザクション識別子を退避する領域

SQL トレースにトランザクション識別子が出力される場合，出力ごとに 4 ＋↑トランザクション識別子長(最大 128)÷4↑×4 を加算します。トランザクション識別子が出力される契機については，「SQL トレース機能」の「SQL トレース情報の見方」を参照してください。

c：パラメタトレースを退避する領域

パラメタトレースを出力する場合，出力対象のパラメタごとに，44 ＋↑データの長さ（クライアント環境定義 PDPRMTRCSIZE の指定値で切り捨てます）÷4↑×4 を加算します。

データの長さの詳細については，表「SQL 記述領域に設定するデータコードとデータの長さ」を参照してください。パラメタが配列又は繰返し列の場合，各要素の長さの合計となります（要素ごとに切り捨てします)。

また，退避領域のサイズに対してパラメタトレース用の退避情報のサイズが大き過ぎる場合，該当するオペレーションのパラメタトレースは出力しません。

エラーが発生した SQL 処理の延長で SQL トレースへ出力するため，退避している情報の量が多い場合は，SQL 処理からの戻りが遅延します。

(92)　PDVWOPTMODE={0 ｜ 1 ｜ 2}

アクセスパス表示ユティリティ用のアクセスパス情報を取得するかどうかを指定します。

アクセスパス情報ファイルは，UAP が接続したシングルサーバ又はフロントエンドサーバがあるユニットの SQL 情報ディレクトリ（%PDDIR%¥spool¥pdsqldump）下に作成されます。

アクセスパス表示ユティリティについては，マニュアル「HiRDB コマンドリファレンス」を参照してください。

0：

アクセスパス情報を取得しません。

1：

アクセスパス情報を取得し，アクセスパス情報ファイルに出力します。このとき，SQL オブジェクトがバッファ中にある SQL については，情報を出力しません。

2：

アクセスパス情報を取得し，アクセスパス情報ファイルに出力します。このとき，SQL オブジェクトがバッファ中にある SQL についても SQL オブジェクトを再作成し，情報を出力します。

《留意事項》

1. HiRDB SQL Tuning Advisor 用にアクセスパス情報を取得する場合は，PDTAAPINFPATH を指定してください。HiRDB SQL Tuning Advisor 用アクセスパス情報ファイルについては，「HiRDB SQL Tuning Advisor 用アクセスパス情報ファイル」を参照してください。
2. 1 を指定した場合，SQL オブジェクトがバッファ中にある SQL については，情報が出力されないので注意してください。SQL オブジェクトがバッファ中にある SQL についても情報を出力したい場合は，2 を指定してください。
3. 2 を指定した場合，SQL オブジェクトがバッファ中にある SQL についても SQL オブジェクトを再作成するため，1 よりもサーバの負荷が増えることがあります。
4. Windows 版 HiRDB で，%PDDIR%のパス長＋認可識別子の長さ＋ UAP 名の長さの合計が 220 文字より大きい場合は，アクセスパス情報ファイルの作成に失敗することがあります。この場合，UAP 統計レポート機能を使用して，アクセスパス情報を取得します。UAP 統計レポート機能については，「UAP 統計レポート機能」を参照してください。
5. SQL 種別と PDVWOPTMODE の指定値の関係を次に示します。

SQL 種別	条件	PDVWOPTMODE の指定値		
		0	1	2
静的 SQL	SQL オブジェクトがバッファ中にない	×	○	○
	SQL オブジェクトがバッファ中にある	×	×	○
動的 SQL	SQL オブジェクトがバッファ中にない	×	○	○
	SQL オブジェクトがバッファ中にある	×	×	○
ルーチン定義	なし	×	○	○
CALL 文	インデクスの追加又は削除で，手続きの SQL オブジェクトのインデクス情報が無効になった場合	×	○	○
	上記以外の場合	×	×	×

（凡例）

○：アクセスパス情報を出力します。

×：アクセスパス情報を出力しません。

(93)　PDTAAPINFPATH=アクセスパス情報ファイル出力ディレクトリ名

～＜パス名＞((最大 256 バイト))

HiRDB SQL Tuning Advisor 用のアクセスパス情報ファイルを出力する場合に，出力先ディレクトリを指定します。この環境変数を指定しても，出力先ディレクトリがなかったり，書き込み権限がなかったりして，

出力処理でエラーが発生した場合，アクセスパス情報を出力しません。なお，出力処理でエラーが発生しても，実行中の SQL はエラーにはなりません。ただし，JDBC4.0 版の Type4 JDBC ドライバ及び JDBC4.3 版の Type4 JDBC ドライバでは例外を投入します。HiRDB SQL Tuning Advisor 用アクセスパス情報ファイルについては，「HiRDB SQL Tuning Advisor 用アクセスパス情報ファイル」を参照してください。

《留意事項》

- HiRDB SQL Tuning Advisor のダイナミックブラウジング機能を使用する場合は，この環境変数の指定は無効になります。
- この環境変数を指定している場合，プロセス間メモリ通信機能は使用できません。クライアント環境定義の PDIPC オペランドに MEMORY を指定していても，DEFAULT を指定した場合の動作となります。

(94) PDTAAPINFMODE= {0 | 1}

HiRDB SQL Tuning Advisor 用のアクセスパス情報ファイルを出力する場合に，アクセスパス情報ファイルのファイル名の形式を指定します。

0：

ファイル名は pdtaapinf1 及び pdtaapinf2 となります。

1：

接続ごとに，pdtaapinfHHMMSSmmm_XXX_1 及び pdtaapinfHHMMSSmmm_XXX_2 のファイル名で出力します。

HHMMSSmmm：

接続した時刻（SQL トレースに出力される，該当する CONNECT の接続開始時間と同じ）

XXXXXXXXXX：

接続通番（最大 10 けた）

(95) PDTAAPINFSIZE=アクセスパス情報ファイルサイズ

～＜符号なし整数＞((100000～2000000000))《409600》（単位：バイト）

HiRDB SQL Tuning Advisor 用のアクセスパス情報ファイルを出力する場合に，アクセスパス情報ファイルのファイルサイズを指定します。ここで指定したファイルサイズを超えると，出力先をもう一方のファイルに切り替えます。以降，これを繰り返しながら二つのファイルを交互に使用します。

(96) PDSTJTRNOUT= {YES | NO}

・0904 互換モードの場合：OLTP 環境下ではトランザクションごとに統計ログファイルに出力します。それ以外の環境下では，コネクションごとに統計ログファイルに出力します。

UAP に関する統計情報を，トランザクションごとに統計ログファイルに出力するかどうかを指定します。

YES：

UAP に関する統計情報を，トランザクションごとに統計ログファイルに出力します。

NO：

UAP に関する統計情報を，コネクションごとに統計ログファイルに出力します。

UAP に関する統計情報の出力の開始は，システム定義の pdstbegin オペランド，又は pdstbegin コマンドで指定します。pdstbegin オペランドについてはマニュアル「HiRDB システム定義」を，pdstbegin コマンドについてはマニュアル「HiRDB コマンドリファレンス」を参照してください。

(97) PDLOCKLIMIT=ユーザ当たりの最大排他資源要求数

～<符号なし整数>((0～200000000))《0》（単位：件）

一つのサーバに対して，UAP から発行する排他要求の上限値（排他資源数の上限値）を指定します。

0 を指定するか省略した場合は，排他要求上限数のチェックをしません。この場合，できる限りの排他要求が発行されます。

排他資源数が不足した場合は，SQL エラーとなります。

《見積もり方法》

見積もり方法には，次の 2 種類があります。

- 一つの UAP で使用する排他資源数から，上限値を算出します。
  排他資源数は SQL によって異なります。排他制御による排他資源数を概算し，その値からこの指定値を算出してください。排他資源数の見積もりについては，マニュアル「HiRDB システム定義」を参照してください。また，排他制御については，「排他制御」を参照してください。
- UAP を接続する HiRDB サーバ側が持つ排他資源数の総数のうち，一つの UAP で使用できる排他資源数から，上限値を算出します。
  排他資源数の見積もりについては，マニュアル「HiRDB システム定義」を参照してください。

(98) PDDLKPRIO={96 | 64 | 32}

UAP のデッドロックプライオリティ値を指定します。

この環境変数は，システム定義の pd_deadlock_priority_use オペランドに Y を指定した場合に有効になります。

この環境変数で指定した値の小さい方が，デッドロックが発生した場合にプログラムが優先的に処理されます。また，値が一番大きいと，デッドロックが発生した場合にプログラムはエラーとなり，ロールバックします。

なお，デッドロックプライオリティ値が同一の場合，トランザクションの開始が早い方を優先的に処理します。デッドロックプライオリティ値は次のようになります。

PDDLKPRIO の指定値			デッドロック プライオリティ値
96			96
64			64
32			32
省略	X/Open XA インタフェース使用の場合		96
	X/Open XA インタフェース未使用の場合		64
−	ユティリティ		64
	運用コマンド	pddbchg，pdhold（-b 及び-s），pdorbegin，及び pdorend	システム定義の pd_command_deadlock_priority オペランドの指定値
		上記以外	64

（凡例）−：該当しません。

(99)　PDLOCKSKIP={YES ｜ NO}

無排他条件判定をするかどうかを指定します。

YES：無排他条件判定をします。

NO：無排他条件判定をしません。

この環境変数に YES を指定した場合，検索処理（DELETE，UPDATE 時の検索処理を含みます）の条件判定処理を無排他で実行します。無排他条件判定については，「無排他条件判定」を参照してください。

(100)　PDFORUPDATEEXLOCK={YES ｜ NO}

・0904 互換モードの場合：《NO》

UAP 中の FOR UPDATE 句を指定した（又は仮定された）SQL の排他オプションに，WITH EXCLUSIVE LOCK を適用するかどうかを指定します。適用する場合，クライアント環境定義 PDISLLVL の指定値は無効になります。

YES：

FOR UPDATE 句を指定した SQL の排他オプションに，WITH EXCLUSIVE LOCK を適用します。

NO：

FOR UPDATE 句を指定した SQL の排他オプションに，PDISLLVL の指定値を適用します。

ルーチン中の SQL に対して，PDFORUPDATEEXLOCK の指定は無効になります。ルーチン中の FOR UPDATE 句を指定した SQL に，WITH EXCLUSIVE LOCK を適用する場合，ルーチン定義時の SQL コンパイルオプションで指定します。

(101)　PDISLLVL=データ保証レベル

～<符号なし整数>((0～2))

SQL 文のデータ保証レベルを指定します。データ保証レベルとは，トランザクションのどの時点までデータの内容を保証するかのレベルのことをいいます。この環境変数は，SQL 文に指定する排他オプションに相当します。

この環境変数を指定すると，UAP 中の SQL 文の排他オプションを一括して決定できます。ただし，SQL 文中で排他オプションを指定している場合は，SQL 文中の排他オプションの指定が優先されます。

なお，このオペランドの指定，SQL 文中の排他オプションの指定，及びシステム定義の pd_isolation_level オペランド指定の優先順位は次のとおりです。

1. SQL 文中の排他オプションの指定（SQL 文単位で制御する場合に指定）
2. このオペランドの指定（UAP 単位で制御する場合に指定）
3. システム定義の pd_isolation_level オペランド指定（システム単位で制御する場合に指定）

上記三つをすべて省略した場合，排他オプションの WITH SHARE LOCK が仮定されます。排他オプションについては，マニュアル「HiRDB SQL リファレンス」を参照してください。

データ保証レベルについては，「データ保証レベル」を参照してください。

0：

ほかのユーザが更新中のデータでも更新完了を待たないで参照できます。そのため，より同時実行性を向上できますが，同一トランザクション中で同じ行を 2 度検索した場合，同じデータを受け取れないときがあります。例えば，SELECT * FROM ZAIKO で在庫表を検索する場合，ほかのユーザが在庫表を更新中でも，排他待ちをしないで検索できます。これは，SELECT 文の WITHOUT LOCK NOWAIT 指定に相当します。

なお，更新を伴うカーソル宣言に対しては 0 を指定しても無効になり，1 が仮定されます。

1：

検索処理が終了するまで（HiRDB がページ又は行を見終わるまで），検索しているデータの内容をほかのユーザに更新させない場合に指定します。検索が終了すれば，そのトランザクションが終了していなくてもほかのユーザからの参照，及び更新を許可します。そのため，より同時実行性を向上できますが，同一トランザクション中で同じ行を 2 度検索した場合，同じデータを受け取れないときがあります。例えば，SELECT * FROM ZAIKO で在庫表を検索する場合，検索終了後にトランザクションの終了を待たないで，ほかのユーザからの在庫表の更新，又は参照を許可します。これは，SELECT 文の WITHOUT LOCK WAIT 指定に相当します。

2：

検索処理のトランザクションが終了するまで，検索しているデータの内容をほかのユーザに更新させない場合に指定します。例えば，SELECT * FROM ZAIKO で在庫表を検索する場合，トランザクションが終了するまで在庫表の内容を保証します。これは，SELECT 文の WITH SHARE LOCK 指定に相当します。

なお，更新を伴うカーソル宣言に対しては，WITH EXCLUSIVE LOCK が仮定されます。

《留意事項》

1. 手続き（ストアドプロシジャ）中の SQL 文のデータ保証レベルは，CREATE PROCEDURE，CREATE TYPE，ALTER PROCEDURE，又は ALTER ROUTINE の指定で決まり，手続き実行時にはこの環境変数によって影響を受けることはありません。
2. この環境変数を省略し，かつ SQL 文中の排他オプションも省略している場合は，その SQL 文に対して排他オプションの WITH SHARE LOCK が仮定されます。排他オプションについては，マニュアル「HiRDB SQL リファレンス」を参照してください。

(102) PDSQLOPTLVL=SQL 最適化オプション［，SQL 最適化オプション］...

～<識別子，又は符号なし整数>

データベースの状態を考慮して，最も効率的なアクセスパスを決定するための最適化の方法を指定します。

SQL 最適化オプションには，識別子（文字列）で指定する方法と，数値で指定する方法がありますが，通常時は識別子で指定する方法をお勧めします。

識別子で指定する場合：

```
PDSQLOPTLVL=”識別子”［，”識別子”］…
```

＜指定例＞

- ネストループジョイン優先とグループ分け高速化処理を適用する場合
  PDSQLOPTLVL="PRIOR_NEST_JOIN","RAPID_GROUPING"
- すべての最適化を適用しない場合
  PDSQLOPTLVL="NONE"

＜規則＞

1. 識別子は一つ以上指定してください。
2. 識別子を二つ以上指定する場合は，コンマで区切ってください。
3. 識別子に指定できる内容（最適化方法）については，《SQL 最適化オプションの指定値》を参照してください。
4. すべての最適化を適用しない場合は，識別子に"NONE"を指定してください。ただし，同時に"NONE"以外の識別子を指定すると，"NONE"は無効になります。
5. 識別子は大文字及び小文字で指定できます。
6. 同じ識別子を二つ以上指定しても，一つ指定したものとみなされますが，なるべく同じ識別子は指定しないようにしてください。
7. "識別子"［，"識別子"］...に指定できる文字列は，最大 575 バイトです。

数値で指定する場合：

```
PDSQLOPTLVL=符号なし整数［，符号なし整数］…
```

＜指定例＞

- 複数の SQL オブジェクト作成，AND の複数インデクス利用の抑止，及び複数インデクス利用の強制を適用する場合

  符号なし整数をコンマで区切って指定する場合：

  PDSQLOPTLVL=4,10,16

  符号なし整数の和を指定する場合：

  PDSQLOPTLVL=30

- 既に 14（4+10）を指定していて，新たに 16 を追加する場合

  PDSQLOPTLVL=14,16

- すべての最適化を適用しない場合

  PDSQLOPTLVL=0

＜規則＞

1. バージョン 06-00 より前の HiRDB から，バージョン 06-00 以降の HiRDB にバージョンアップする場合，バージョン 06-00 より前の合計値指定も有効になります。最適化オプションを変更する必要がない場合は，バージョン 06-00 以降の HiRDB にバージョンアップしたときにこのオペランドの指定値を変更する必要はありません。
2. 符号なし整数は一つ以上指定してください。
3. 符号なし整数を二つ以上指定する場合は，コンマで区切ってください。
4. 符号なし整数に指定できる内容（最適化方法）については，《SQL 最適化オプションの指定値》を参照してください。
5. すべての最適化を適用しない場合は，符号なし整数に 0 を指定してください。ただし，同時に 0 以外の識別子を指定すると，0 は無効になります。
6. 同じ符号なし整数を二つ以上指定しても，一つ指定したものとみなされますが，なるべく同じ符号なし整数は指定しないようにしてください。
7. 複数の最適化方法を指定する場合，その符号なし整数の和を指定することもできます。ただし，同じ最適化方法の値は二つ以上足さないでください（足した結果が別の最適化方法とみなされることもあるため）。
8. 複数の最適化方法の値を足して指定する場合，どの最適方法を指定しているのか分かりにくくなるため，コンマで区切って指定する方法をお勧めします。また，既に複数の最適化方法の値を足して指定している場合で，新たに別の最適化方法が必要になったときは，追加する値をコンマで区切って後ろに指定できます。
9. 符号なし整数［，符号なし整数］…に指定できる文字列は，最大 575 バイトです。

《システム定義との関係》

1. この環境変数を省略するとシステム定義の pd_optimize_level オペランドの指定値が仮定されます。pd_optimize_level オペランドについては，マニュアル「HiRDB システム定義」を参照してください。
2. システム定義の pd_floatable_bes オペランド，又は pd_non_floatable_bes オペランドを指定している場合，「フロータブルサーバ対象拡大（データ取り出しバックエンドサーバ）」及び「フロータブルサーバ対象限定（データ取り出しバックエンドサーバ）」の指定は無効になります。
3. システム定義の pd_indexlock_mode オペランドに KEY を指定している場合（インデクスキー値排他の場合），「更新 SQL の作業表作成抑止」の指定は無効になります。

《SQL との関係》

ストアドルーチン中の SQL 文の SQL 最適化オプションは，CREATE PROCEDURE，CREATE TYPE，ALTER PROCEDURE，又は ALTER ROUTINE の指定で決まり，PDSQLOPTLVL の指定によって影響を受けることはありません。

SQL 文中に SQL 最適化指定を指定している場合は，SQL 最適化オプションよりも SQL 最適化指定が優先されます。SQL 最適化指定については，マニュアル「HiRDB SQL リファレンス」を参照してください。

《SQL 最適化オプションの指定値》

SQL 最適化オプションの指定値を次の表に示します。

表 6–39　SQL 最適化オプションの指定値

項番	最適化方法	指定値	
		識別子	符号なし整数
1	ネストループジョイン強制	"FORCE_NEST_JOIN"	4
2	複数の SQL オブジェクト作成	"SELECT_APSL"	10
3	フロータブルサーバ対象拡大（データ取り出しバックエンドサーバ）※1※2	"FLTS_INC_DATA_BES"	16
4	ネストループジョイン優先	"PRIOR_NEST_JOIN"	32
5	フロータブルサーバ候補数の拡大※2	"FLTS_MAX_NUMBER"	64
6	OR の複数インデクス利用の優先	"PRIOR_OR_INDEXES"	128
7	自バックエンドサーバでのグループ化，ORDER BY，DISTINCT 集合関数処理※2	"SORT_DATA_BES"	256
8	AND の複数インデクス利用の抑止	"DETER_AND_INDEXES"	512
9	グループ分け高速化処理	"RAPID_GROUPING"	1024
10	フロータブルサーバ対象限定（データ取り出しバックエンドサーバ）※1※2	"FLTS_ONLY_DATA_BES"	2048

項番	最適化方法	指定値	
		識別子	符号なし整数
11	データ収集用サーバの分離機能※1※2	"FLTS_SEPARATE_COLLECT_SVR"	2064
12	インデクス利用の抑止（テーブルスキャン強制）	"FORCE_TABLE_SCAN"	4096
13	複数インデクス利用の強制	"FORCE_PLURAL_INDEXES"	32768
14	更新 SQL の作業表作成抑止	"DETER_WORK_TABLE_FOR_UPDATE"	131072
15	探索高速化条件の導出	"DERIVATIVE_COND"	262144
16	スカラ演算を含むキー条件の適用	"APPLY_ENHANCED_KEY_COND"	524288
17	プラグイン提供関数からの一括取得機能	"PICKUP_MULTIPLE_ROWS_PLUGIN"	1048576
18	導出表の条件繰り込み機能	"MOVE_UP_DERIVED_COND"	2097152

注※1

フロータブルサーバ対象拡大（データ取り出しバックエンドサーバ），及びフロータブルサーバ対象限定（データ取り出しバックエンドサーバ）を共に指定した場合，それぞれの最適化方法は有効にはなりません。代わりに，データ収集用サーバの分離機能として動作します。

注※2

HiRDB/シングルサーバの場合，指定しても無効になります。

《指定値の目安》

指定値の目安を「更新 SQL の作業表作成抑止」の適用条件を満たす場合と満たさない場合に分けて説明します。

なお，指定値の目安で項番 XX と表記していますが，これは表「SQL 最適化オプションの指定値」の項番のことを示しています。

●HiRDB/シングルサーバ

≪「更新 SQL の作業表作成抑止」の適用条件を満たさない場合≫

項番 4，6，8，9，16，及び 18 を指定してください。識別子で指定した場合の例を次に示します。

```
PDSQLOPTLVL＝”PRIOR_NEST_JOIN”,
             ”PRIOR_OR_INDEXES”,
             ”DETER_AND_INDEXES”,
             ”RAPID_GROUPING”,
             ”APPLY_ENHANCED_KEY_COND”,
             ”MOVE_UP_DERIVED_COND”
```

≪「更新 SQL の作業表作成抑止」の適用条件を満たす場合≫

項番 4，6，8，9，14，16，及び 18 を指定してください。識別子で指定した場合の例を次に示します。

```
PDSQLOPTLVL＝”PRIOR_NEST_JOIN”,
            ”PRIOR_OR_INDEXES”,
            ”DETER_AND_INDEXES”,
            ”RAPID_GROUPING”,
            ”DETER_WORK_TABLE_FOR_UPDATE”,
            ”APPLY_ENHANCED_KEY_COND”,
            ”MOVE_UP_DERIVED_COND”
```

●HiRDB/パラレルサーバ

≪「更新 SQL の作業表作成抑止」の適用条件を満たさない場合≫

表 6–40　HiRDB/パラレルサーバの SQL 最適化オプションの指定値の目安（1/2）

条件		指定値
SQL 処理になるべく多くのバックエンドサーバを使用し，個々の SQL 処理を速くしたい場合	大量検索の SQL を速くしたい場合	項番 3〜9，16，及び 18 を指定してください。 （識別子で指定した例） PDSQLOPTLVL＝”FLTS_INC_DATA_BES”, ”PRIOR_NEST_JOIN”, ”FLTS_MAX_NUMBER”, ”PRIOR_OR_INDEXES”, ”SORT_DATA_BES”, ”DETER_AND_INDEXES”, ”RAPID_GROUPING”, ”APPLY_ENHANCED_KEY_COND”, ”MOVE_UP_DERIVED_COND”
	結果が数十件程度の検索を速くしたい場合	項番 3，4，6〜9，16，及び 18 を指定してください。 （識別子で指定した例） PDSQLOPTLVL＝”FLTS_INC_DATA_BES”, ”PRIOR_NEST_JOIN”, ”PRIOR_OR_INDEXES”, ”SORT_DATA_BES”, ”DETER_AND_INDEXES”, ”RAPID_GROUPING”, ”APPLY_ENHANCED_KEY_COND”, ”MOVE_UP_DERIVED_COND”
業務ごとにバックエンドサーバを切り分けて使用したい場合	大量検索の SQL を速くしたい場合	項番 4〜10，16，及び 18 を指定してください。 （識別子で指定した例） PDSQLOPTLVL＝”PRIOR_NEST_JOIN”, ”FLTS_MAX_NUMBER”, ”PRIOR_OR_INDEXES”, ”SORT_DATA_BES”, ”DETER_AND_INDEXES”, ”RAPID_GROUPING”, ”FLTS_ONLY_DATA_BES”, ”APPLY_ENHANCED_KEY_COND”,

条件		指定値
		”MOVE_UP_DERIVED_COND”
	数十件程度に絞り込む検索を速くしたい場合	項番 4，6～10，16，及び 18 を指定してください。 （識別子で指定した例） PDSQLOPTLVL＝”PRIOR_NEST_JOIN”, ”PRIOR_OR_INDEXES”, ”SORT_DATA_BES”, ”DETER_AND_INDEXES”, ”RAPID_GROUPING”, ”FLTS_ONLY_DATA_BES”, ”APPLY_ENHANCED_KEY_COND”, ”MOVE_UP_DERIVED_COND”
上記の条件に該当しない場合		項番 4，6～9，16，及び 18 を指定してください。 （識別子で指定した例） PDSQLOPTLVL＝”PRIOR_NEST_JOIN”, ”PRIOR_OR_INDEXES”, ”SORT_DATA_BES”, ”DETER_AND_INDEXES”, ”RAPID_GROUPING”, ”APPLY_ENHANCED_KEY_COND”, ”MOVE_UP_DERIVED_COND”

≪「更新 SQL の作業表作成抑止」の適用条件を満たす場合≫

表 6–41　HiRDB/パラレルサーバの SQL 最適化オプションの指定値の目安（2/2）

条件		指定値
SQL 処理になるべく多くのバックエンドサーバを使用し，個々の SQL 処理を速くしたい場合	大量検索の SQL を速くしたい場合	項番 3～9，14，16，及び 18 を指定してください。 （識別子で指定した例） PDSQLOPTLVL＝”FLTS_INC_DATA_BES”, ”PRIOR_NEST_JOIN”, ”FLTS_MAX_NUMBER”, ”PRIOR_OR_INDEXES”, ”SORT_DATA_BES”, ”DETER_AND_INDEXES”, ”RAPID_GROUPING”, ”DETER_WORK_TABLE_FOR_UPDATE”, ”APPLY_ENHANCED_KEY_COND”, ”MOVE_UP_DERIVED_COND”
	結果が数十件程度の検索を速くしたい場合	項番 3，4，6～9，14，16，及び 18 を指定してください。 （識別子で指定した例） PDSQLOPTLVL＝”FLTS_INC_DATA_BES”, ”PRIOR_NEST_JOIN”,

条件		指定値
		”PRIOR_OR_INDEXES”, ”SORT_DATA_BES”, ”DETER_AND_INDEXES”, ”RAPID_GROUPING”, ”DETER_WORK_TABLE_FOR_UPDATE”, ”APPLY_ENHANCED_KEY_COND”, ”MOVE_UP_DERIVED_COND”
業務ごとにバックエンドサーバを切り分けて使用したい場合	大量検索の SQL を速くしたい場合	項番 4～10，14，16，及び 18 を指定してください。 （識別子で指定した例） PDSQLOPTLVL＝”PRIOR_NEST_JOIN”, ”FLTS_MAX_NUMBER”, ”PRIOR_OR_INDEXES”, ”SORT_DATA_BES”, ”DETER_AND_INDEXES”, ”RAPID_GROUPING”, ”FLTS_ONLY_DATA_BES”, ”DETER_WORK_TABLE_FOR_UPDATE”, ”APPLY_ENHANCED_KEY_COND”, ”MOVE_UP_DERIVED_COND”
	数十件程度に絞り込む検索を速くしたい場合	項番 4，6～10，14，16，及び 18 を指定してください。 （識別子で指定した例） PDSQLOPTLVL＝”PRIOR_NEST_JOIN”, ”PRIOR_OR_INDEXES”, ”SORT_DATA_BES”, ”DETER_AND_INDEXES”, ”RAPID_GROUPING”, ”FLTS_ONLY_DATA_BES”, ”DETER_WORK_TABLE_FOR_UPDATE”, ”APPLY_ENHANCED_KEY_COND”, ”MOVE_UP_DERIVED_COND”
上記の条件に該当しない場合		項番 4，6～9，14，16，及び 18 を指定してください。 （識別子で指定した例） PDSQLOPTLVL＝”PRIOR_NEST_JOIN”, ”PRIOR_OR_INDEXES”, ”SORT_DATA_BES”, ”DETER_AND_INDEXES”, ”RAPID_GROUPING”, ”DETER_WORK_TABLE_FOR_UPDATE”, ”APPLY_ENHANCED_KEY_COND”, ”MOVE_UP_DERIVED_COND”

《各最適化方法の内容》

1．ネストループジョイン強制

結合条件の列にインデクスを定義してある場合，結合処理にネストループジョインだけを使用します。ネストループジョインの結合処理方式については，「結合方式」を参照してください。

ただし，次のどれかの条件に当てはまる場合，ネストループジョイン以外の結合処理をすることがあります。

- 結合条件にスカラ演算など，列指定以外を指定している。
- 結合条件が＝述語以外である。
- 結合条件の列が，インデクスの先頭構成列でない。かつ，結合条件の列がインデクスの第 n 構成列の場合は，第 1 構成列から第 n−1 構成列まで，=述語，又は IS NULL 述語の制限条件を指定していない。
- 外結合で，結合条件を ON 探索条件に指定していない。
- 探索条件中に，結合する両方の表に対して，それぞれインデクスを使用した検索をするプラグイン提供関数，又は構造化繰返し述語を指定している。
- HiRDB/パラレルサーバで，分割表を内表とする外結合に対して，内表の分割列を結合条件に指定していない。
- HiRDB/パラレルサーバで，フレキシブルハッシュ分割表を内表とする外結合である。

＜ネストループジョイン強制の留意事項＞

1. 結合表をネストループジョインで処理する場合は，SQL の外表に指定した表を外表とします。
2. 結合条件の一方の列にだけインデクスが定義してある結合をネストループジョインで処理する場合，インデクスを定義してある表の列を内表とします。
3. 結合表以外で，結合条件の両辺の列にインデクスが定義してある結合をネストループジョインで処理する場合，ネストループジョインの外表，内表は HiRDB が判断して決定します。ただし，FROM 句にビュー表又は WITH 句問合せ名の指定がなく，探索条件に結合条件だけを指定している場合は，次の規則に従って外表，内表を決定します。

   (i) HiRDB/パラレルサーバの分割表の結合の場合，一方の表のすべての分割列を結合条件に指定していて，その結合相手の表の分割列のうち，結合条件に指定していない列があるときは，すべての分割列を結合条件に指定してある表が内表になります。

   (ii) (i)に当てはまらない場合は，FROM 句の先に指定した表が外表になります。
4. HiRDB/パラレルサーバで「ネストループジョイン強制」を適用する場合，大量データのジョインをするときには，できる限り結合列で表を分割するようにしてください。

2．複数の SQL オブジェクト作成

あらかじめ複数の SQL オブジェクトを作成し，実行時に埋込み変数，又は?パラメタの値によって，最適な SQL オブジェクトを選択します。

3．フロータブルサーバ対象拡大（データ取り出しバックエンドサーバ）

通常はデータ取り出しに使用しないバックエンドサーバをフロータブルサーバとして使用しています。この最適化方法を適用すると，データ取り出しに使用するバックエンドサーバについてもフロータブルサーバとして使用します。

ただし，フロータブルサーバとして使用するバックエンドサーバ数は HiRDB が計算して求めるのですべてのバックエンドサーバを使用するとは限りません。すべてのバックエンドサーバを使用したい場合は，「フロータブルサーバ候補数の拡大」とともに指定してください。

フロータブルサーバの割り当て方法については，「フロータブルサーバの割り当て方法（HiRDB/パラレルサーバ限定）」を参照してください。

この指定は，HiRDB/パラレルサーバのときだけ有効になります。

4．ネストループジョイン優先

結合条件の列にインデクスを定義してある場合，結合処理にネストループジョインを優先して使用します。ネストループジョインの結合処理方式については，「結合方式」を参照してください。

「ネストループジョイン強制」との違いは，ネストループジョイン強制は結合条件にインデクスが定義してあれば，絞り込み条件がなくても（制限に該当する場合を除く）必ずネストループジョインします。これに対して，ネストループジョイン優先は，絞り込み条件を指定している場合は必ずネストループジョインしますが，絞り込み条件がないときは結合方式を HiRDB が判断します。ただし，次のどれかの条件に当てはまる場合，絞り込み条件を指定していても，ネストループジョイン以外の結合処理をすることがあります。

- 結合条件にスカラ演算など，列指定以外を指定している。
- 結合条件が＝述語以外である。
- 結合条件の列が，インデクスの先頭構成列でない。かつ，結合条件の列がインデクスの第 n 構成列の場合は，第 1 構成列から第 n−1 構成列まで=述語，又は IS NULL 述語の制限条件を指定していない。
- 外結合で結合条件を ON 探索条件に指定していない。
- 探索条件中に結合する両方の表に対して，それぞれインデクスを使用した検索をするプラグイン提供関数，又は構造化繰返し述語を指定している。
- HiRDB/パラレルサーバで，分割表を内表とする外結合に対して，内表の分割列を結合条件に指定していない。
- HiRDB/パラレルサーバで，フレキシブルハッシュ分割表を内表とする外結合である。
- 最適化情報収集ユティリティ（pdgetcst）を実行している。
- 絞り込み条件が，定義長 256 バイト以上の CHAR，VARCHAR，MCHAR，MVARCHAR の列，若しくは定義長 128 文字以上の NCHAR，NVARCHAR の列，又は BLOB 型の列を含む探索条件だけである。
- 絞り込み条件が，否定，又は OR 演算子を含む探索条件だけである。

＜ネストループジョイン優先の留意事項＞

1. 結合表をネストループジョインで処理する場合は，SQL の外表に指定した表を外表とします。
2. 結合条件の一方の列にだけインデクスが定義してある結合をネストループジョインで処理する場合，インデクスを定義してある表の列を内表とします。
3. 結合表以外で，結合条件の両辺の列にインデクスが定義してある結合をネストループジョインで処理する場合，ネストループジョインの外表，内表は HiRDB が判断して決定します。ただし，FROM 句にビュー表及び WITH 句の問合せ名の指定がなく，探索条件に結合条件だけを指定している場合は，次の規則に従って外表，内表を決定します。

・HiRDB/パラレルサーバの分割表の結合時には，一方の表のすべての分割列を結合条件に指定して，その結合相手の表の分割列のうち，結合条件に指定していない列がある場合は，すべての分割列を結合条件に指定している表が内表になります。

・上記に該当しない場合は，FROM 句の最初に指定した表が外表になります。

4.「ネストループジョイン強制」とともに指定した場合，「ネストループジョイン優先」は無効になります。

5．フロータブルサーバ候補数の拡大

通常使用するフロータブルサーバ数は，利用できるフロータブルサーバから必要数を HiRDB が計算して割り当てます。この最適化方法を適用すると，利用できるフロータブルサーバをすべて利用します。ただし，データ取り出しに使用するバックエンドサーバはフロータブルサーバとして使用できません。

データ取り出しに使用するバックエンドサーバもフロータブルサーバとして使用したいときは，「フロータブルサーバ対象拡大（データ取り出しバックエンドサーバ）」とともに指定してください。

フロータブルサーバの割り当て方法については，「フロータブルサーバの割り当て方法（HiRDB/パラレルサーバ限定）」を参照してください。

この指定は，HiRDB/パラレルサーバのときだけ有効になります。

6．OR の複数インデクス利用の優先

OR の複数インデクスを利用して検索する方法を，優先して適用したい場合に指定します。

OR の複数インデクス利用とは，探索条件中の OR で結ばれた複数の条件に対して，それぞれの条件をインデクスを使用して検索し，検索結果の和集合をとることで探索条件を評価する方式をいいます。

WHERE 句又は ON 探索条件に A OR B OR C ... OR Z を指定して検索している場合で，OR で結ばれたすべての条件に対して=を使用して絞り込んでいるときに，「OR の複数インデクス利用の優先」を適用すると，高速に検索できます。

「OR の複数インデクス利用の優先」を指定しない場合でも，OR の数が少ないときは，HiRDB は OR の複数インデクス利用を適用して検索しますが，OR の数が多くなると HiRDB が内部的に計算している検索コストが大きくなるため，OR の複数インデクス利用が適用されなくなることもあります。そこで，「OR の複数インデクス利用の優先」を指定して，OR の数が多くなっても常に OR の複数インデクス利用が適用されるようにします。

＜ OR の複数インデクス利用の優先の留意事項＞

1. OR と並列に AND で条件を指定していて，その条件がインデクスを使用して絞り込める場合は，このインデクスを使用することがあります。
2. OR で指定したすべての条件が，比較述語の=で絞り込んでいる場合に適用されます。また，=で絞り込まれたすべての列に対して，単一列インデクス，又は複数列インデクスの第 1 構成列となるインデクスを定義していることが前提となります。
3. 2 表以上の結合検索時には，結合列のインデクスを使用して検索した方が高速に検索できると HiRDB が判断した場合，適用されないことがあります。
4. SQL 文によっては，OR の複数インデクス利用ではなく，和集合を伴った AND の複数インデクス利用が適用される場合があります。この場合も OR の複数インデクス利用と同様に，高速に検

索できます。ただし，AND を伴った条件を指定していると，積集合と和集合を組み合わせて AND の複数インデクス利用を適用する場合があります。

積集合を伴った AND の複数インデクス利用を適用していて性能が良くない場合には，次のどちらかの方法で改善できることがあります。

(a) 「OR の複数インデクス利用の優先」と同時に，「AND の複数インデクス利用の抑止」を指定してください。積集合だけ抑止されるようになります。

(b) AND で連結した複数の列の条件が絞り込める場合，これらの条件列を含む複数列インデクスを定義してください。

5. SQL 拡張最適化オプションでコストベース最適化モード 2 の適用を使用しない場合は，結合検索時に複数インデクスを利用しません。ただし，複数インデクス利用を適用しないと評価できない条件がある場合には，このオプションの指定に関係なく複数インデクスを利用します。

7．自バックエンドサーバでのグループ化，ORDER BY，DISTINCT 集合関数処理

通常，グループ化，ORDER BY，及び DISTINCT 集合関数処理はフロータブルサーバを使用して処理しますが，1 表検索の場合にはこの最適化を適用することで，表が定義されているバックエンドサーバ（自バックエンドサーバ）でグループ化，ORDER BY，及び DISTINCT 集合関数処理をします。

グループ分け処理方式については，「グループ分け処理方式（HiRDB/パラレルサーバ限定）」を参照してください。

グループ分け高速化処理を適用する場合や，インデクスを使用して検索した結果，グループ化，ORDER BY，及び DISTINCT 集合関数のためのソートをする必要がないと HiRDB が判断した場合には，より高速な処理方式が選択されます。

8．AND の複数インデクス利用の抑止

AND の複数インデクス利用をするアクセスパスを常に使わないようにします。

AND の複数インデクス利用とは，探索条件に AND で結ばれた条件が複数あり，それぞれの列に異なるインデクスが定義してある場合（例えば，SELECT ROW FROM T1 WHERE C1 = 100 AND C2 = 200），それぞれのインデクスを使って条件を満たす行の作業表を作成し，これらの積集合を求める方式です。

AND の複数インデクス利用中に OR が含まれる場合，AND の部分については複数インデクス利用を抑止しますが，OR の部分については抑止しません。

積集合は，データの特性によって有効な場合と，性能的に悪くなる場合とがあります。複数のインデクスを利用する場合に，それぞれのインデクスを使用することで，ある程度の件数に絞り込めて，更に積集合をとることで重なっている部分が少なくなる場合には有効になります。

AND の複数インデクスの利用が有効でないと考えられる場合は，この最適化を適用してください。

ただし，一つの問合せ指定中に，同一表の列を含む条件を次の箇所に複数指定したときは，AND の複数インデクス利用の抑止はできません。

- 構造化繰返し述語の探索条件中
- インデクスを使用した検索をするプラグイン提供関数の第 1 引数

9．グループ分け高速化処理

SQL の GROUP BY 句で指定したグループ分けを，ハッシングを使って高速に処理します。

グループ分け高速化機能については，「グループ分け高速化機能」を参照してください。

10．フロータブルサーバ対象限定（データ取り出しバックエンドサーバ）

通常はデータ取り出しに使用しないバックエンドサーバをフロータブルサーバとして使用しています。この最適化方法を適用すると，データ取り出しに使用するバックエンドサーバだけをフロータブルサーバとして使用します。この適用は，HiRDB/パラレルサーバのときだけ有効になります。

フロータブルサーバの割り当て方法については，「フロータブルサーバの割り当て方法（HiRDB/パラレルサーバ限定）」を参照してください。

11．データ収集用サーバの分離機能

「フロータブルサーバ対象拡大（データ取り出しバックエンドサーバ）」，又は「フロータブルサーバ対象限定（データ取り出しバックエンドサーバ）」を指定した場合は，データ収集用サーバの分離機能となります。

データ収集用サーバの分離機能を適用した場合，複数のバックエンドサーバから 1 か所のバックエンドサーバにデータを集める必要がある SQL に対しては，データ転送元以外のバックエンドサーバをデータ収集用に割り当てます。これ以外の用途のフロータブルサーバとしては，データ収集用以外のバックエンドサーバ（データ取り出しバックエンドサーバも含みます）を割り当てます。

フロータブルサーバの割り当て方法については，「フロータブルサーバの割り当て方法（HiRDB/パラレルサーバ限定）」を参照してください。

12．インデクス利用の抑止（テーブルスキャン強制）

通常はインデクスの利用判定を HiRDB が決定します。この最適化方法を適用すると，インデクスを使用しない方式を強制的に使用するようにします。

ただし，ジョインを使用してネストループ結合になる場合，構造化繰返し述語を探索条件に指定した場合，又はインデクス型プラグイン専用関数の条件の場合は，インデクス利用を抑止できません。

13．複数インデクス利用の強制

AND の複数インデクス利用を強制的に選択して，表を検索する場合に指定します。

AND で結ばれた条件を複数指定している場合，この最適化を指定しないと AND の複数インデクス利用が選択されたときでも，通常は最大二つ程度のインデクスしか使用しません。使用するインデクスの数は，表定義，インデクス定義，及び探索条件によって変わります。

この最適化を指定すると，インデクスを使用することでサーチ範囲が絞り込める条件をすべて使用するようになります。

AND の複数インデクス利用は，それぞれのインデクスを使用することで，ある程度の件数に絞り込めて，更に積集合をとることで重なっている部分が少なくなる場合に有効になります。

2 表以上の結合検索時には，結合列のインデクスを使用して検索した方が高速に検索できると HiRDB が判断した場合，適用されないことがあります。

SQL 拡張最適化オプションでコストベース最適化モード 2 を使用しない場合は，結合検索時に複数インデクスを利用しません。ただし，複数インデクス利用を適用しないと評価できない条件がある場合には，このオプションの指定に関係なく複数インデクスを利用します。

14．更新 SQL の作業表作成抑止

この機能は，カーソルを使って検索している表に対して行の更新，削除，及び挿入を行う場合に，検索で使用しているインデクスの構成列を更新すると，同一行を複数回検索，更新するおそれがあります。そのため，次に示す適用条件を満たす場合にだけ適用してください。

なお，この機能は，SQL 最適化オプションの指定を省略した場合にもデフォルトで適用されます。既にこの機能を使用していて，上記のケースに当てはまる場合は，《注意事項》の 10.を参照して対策してください。

＜適用条件＞

次に示す条件を満たす場合には，この機能の適用をお勧めします。

- **検索で使用するインデクスの構成列を更新※する可能性がない**

  注※　UPDATE，DELETE，又は INSERT 文の実行です。

検索で使用するインデクスの構成列を更新する可能性がある場合は，この機能を適用しないでください。

インデクスキー値無排他を適用している場合にこの最適化を指定すると，FOR UPDATE 句を指定した検索，UPDATE 文，又は DELETE 文に対してインデクスを使用したときでも，HiRDB は内部処理のための作業表を作成しません。したがって，高速に SQL 文を処理できます。

なお，インデクスキー値無排他を適用していない場合，作業表は作成されます。

インデクスを使用しているかどうかについては，アクセスパス表示ユティリティで確認できます。

更新 SQL の作業表作成抑止を指定して，更にインデクスキー値無排他機能を使用している場合，カーソル使用時の表操作の制限が緩和されます。

作業表を作成する SQL 文と，更新 SQL の作業表作成抑止との関係を次の表に示します。

表 6–42　作業表を作成する SQL 文と，更新 SQL の作業表作成抑止との関係

SQL 文		インデクスを使用する場合		インデクスを使用しない場合
		この最適化を適用する	この最適化を適用しない	
SELECT 文	FOR UPDATE※1	×	○	×
	FOR UPDATE OF	×	○※2	×
	FOR READ ONLY	○	○	○
	ORDER BY	○※4※5	○※4	○
	上記の指定なし	×※5	×	×
UPDATE 文	SET 句の更新値に値指定だけを指定している	×	○※3	×
	SET 句の更新値に値指定以外の指定がある	○※3	○※3	×
DELETE 文		×	○	×

（凡例）

○：作業表を作成します。

×：作業表を作成しません。

注※1

この SELECT 文のカーソルを使用して更新した場合に，仮定される FOR UPDATE を含みます。

注※2

FOR UPDATE OF 列名に，使用するインデクスの構成列を指定していない場合，作業表は作成されません。

注※3

SET 句の左辺の更新する列名に，使用するインデクスの構成列を指定していない場合，作業表は作成されません。

注※4

インデクスの利用によって，ORDER BY のための作業表が作成されない場合があります。

注※5

カーソルを使用して検索中の表に対して，別の SQL での表の更新ができます。ただし，カーソルの検索で使用しているインデクスを更新した場合，カーソルでの検索結果は保証されません。

15．探索高速化条件の導出

この最適化を指定すると，探索高速化条件の導出をします。

探索高速化条件とは，WHERE 句の探索条件，FROM 句の ON 探索条件から，CNF 変換，又は条件推移で新たに導出される条件のことをいいます。探索高速化条件を導出すると，検索する行が早い段階で絞り込まれるため，検索性能が向上します。ただし，探索高速化条件の生成及び実行に時間が掛かったり，アクセスパスが意図したとおりにならなかったりする場合があるため，できるだけこの最適化は指定しないで，探索高速化条件を直接 SQL 文中に指定するようにしてください。探索高速化条件の導出については，「探索高速化条件の導出」を参照してください。

16．スカラ演算を含むキー条件の適用

この最適化を指定すると，スカラ演算を指定した制限条件のうち，スカラ演算中に含まれる列がすべてインデクス構成列である場合に，インデクスのキー値ごとに条件を判定して絞り込みをします。この条件はキー条件として評価されます。

＜スカラ演算を含むキー条件の適用を指定した場合の HiRDB の動作＞

HiRDB は，インデクスを使用した検索の場合，次の順序で評価をします。

1. インデクスのサーチ範囲を絞り込みます（サーチ条件）。
2. 1.で絞り込んだ結果に対して，インデクスのキー値ごとに条件を判定して更に絞り込みます（キー条件）。
3. 2.で真となったキー値に対して，行識別子（ROWID）を使用してデータページを参照し，条件を評価します。

スカラ演算を含む条件は，この最適化を指定しない場合は 3.で評価されます。この最適化を指定した場合は，2.で評価されるため，データページを参照する行数が少なくなり，入出力を削減できます。サーチ条件，及びキー条件については，マニュアル「HiRDB コマンドリファレンス」を参照してください。

＜スカラ演算を含むキー条件の適用についての留意事項＞

1. この最適化を指定すると，HiRDB はインデクスでの絞り込みが有効に行われていると判断します。その結果，インデクスがより使用されやすくなるため，スカラ演算を含む条件のインデクスでの絞り込みがあまり期待できない場合には，この最適化は指定しないでください。
2. 次のどれかの条件に該当する場合，その条件はキー条件として評価されません。
   ・インデクス構成列以外の列を含む場合
   ・システム定義スカラ関数を含む場合
   ・システム組込みスカラ関数 IS_USER_CONTAINED_IN_HDS_GROUP を含む場合
   ・関数呼出しを含む場合
   ・添字が整数の繰返し列を含む場合
3. スカラ演算を含む構造化繰返し述語は，インデクスを使用しないと評価できないため，エラーとなります。そのため，この最適化を指定しなくても，キー条件が適用されます。

17．プラグイン提供関数からの一括取得機能

探索条件にプラグイン提供関数を指定し，HiRDB がプラグインインデクスを使用して検索する場合，通常，HiRDB はプラグイン提供関数からの返却結果（行位置情報と，必要に応じて受渡し値）を 1 行ごとに取得します。

この最適化を適用すると，プラグイン提供関数の返却結果を複数行まとめて取得できるため，プラグイン提供関数の呼び出し回数を削減できます。このため，検索性能も向上します。なお，プラグイン提供関数からの一括取得機能を適用する場合，HiRDB が内部的に作業表を作成します。

この最適化を指定していなくても，HiRDB が常にプラグイン提供関数からの一括取得機能を適用した方が高速に検索できると判断した場合には，無条件にプラグイン提供関数からの一括取得機能を適用することがあります。プラグイン提供関数からの一括取得機能の適用有無を次に示します。

指定した SQL 文の種類		プラグイン提供関数からの一括取得機能の指定	
		なし	あり
実表の検索結果に対して作業表が必要な SQL 文※	一括取得に対応していない関数	×	×
	一括取得に対応している関数	○	○
実表の検索結果に対して作業表が不要な SQL 文※	一括取得に対応していない関数	×	×
	一括取得に対応している関数	×	△

（凡例）

○：無条件にプラグイン提供関数からの一括取得機能を適用します。

△：新たに作業表を作成して，プラグイン提供関数からの一括取得機能を適用します。

×：プラグイン提供関数からの一括取得機能を適用しません。

注※

作業表用ファイルを必要とする SQL については，マニュアル「HiRDB システム導入・設計ガイド」の「作業表用ファイルの概要」を参照してください。

＜プラグイン提供関数からの一括取得機能の留意事項＞

1. プラグイン提供関数の返却結果を取得する場合，HiRDB が内部的に作業表を作成する必要があります。通常は，行ごとに返却結果を受け取る時間よりも，作業表を作成する時間の方が短いため，検索性能が向上します。ただし，この最適化を指定することで検索性能が低下する場合もあるため，性能低下による影響が大きいときにはこの最適化を指定しないでください。
2. 作業表を作成しない検索の場合に，この最適化を指定すると，1 件目の FETCH までの時間が遅くなります。これは，1 行取り出すごとにクライアントへ結果を返却していた処理が，プラグイン提供関数を指定した探索条件を満たすすべての行を取り出して作業表を作成した後に，クライアントへ結果を返却するように変わるためです。1 件目の FETCH の性能低下が問題になる場合は，この最適化を指定しないでください。
3. この最適化を適用すると，プラグイン提供関数の返却結果を複数行まとめて取得するため，メモリ所要量が増加します。メモリ所要量については，マニュアル「HiRDB システム導入・設計ガイド」を参照してください。

18．導出表の条件繰り込み機能

通常，導出表のための作業表には，導出表の列に対する探索条件に一致しない行も格納します。この最適化方法を適用すると，導出表の列に対する探索条件に一致しない行を除いた後に，導出表のための作業表を作成します。これによって，作業表の容量，及び作業表への入出力回数を削減できます。また，このようなアクセスパスを使用することによって，検索する行をより有効に絞り込めるインデクスを利用できる場合があります。

バージョン 08-02 以降の HiRDB を初めて導入する場合は，この機能の使用をお勧めします。

＜導出表の条件繰り込み機能の留意事項＞

バージョン 08-02 より前の HiRDB からバージョンアップする場合，この機能を使用することでアクセスパスが変わることがあります。アクセスパスが変わることによって，検索する行を有効に絞り込めないインデクスを選択し，性能が劣化するおそれがあります。その場合は，この機能を使用する前後でアクセスパスを確認し，最適なインデクスを使用しているかどうかを確認してください。最適なインデクスを使用できていない場合は，使用インデクスの SQL 最適化指定で使用するインデクスを指定するか，この機能の使用をやめてください。

《注意事項》

注意事項を次に示します。

1. インデクスを定義していない表については，次の最適化を指定しても影響はありません。

「ネストループジョイン強制」

「複数の SQL オブジェクト作成」

「ネストループジョイン優先」

「OR の複数インデクス利用の優先」

「AND の複数インデクス利用の抑止」

「インデクス利用の抑止（テーブルスキャン強制）」

「複数インデクス利用の強制」

「更新 SQL の作業表作成抑止」

「スカラ演算を含むキー条件の適用」

2. ASSIGN LIST 文（副問合せ中を除く）については，必ずインデクスを使用して検索するため，次の最適化を指定しても影響はありません。

   「AND の複数インデクス利用の抑止」

   「インデクス利用の抑止（テーブルスキャン強制）」

3. 最適化情報収集ユティリティ（pdgetcst）で最適化情報を取得しない場合は，「複数の SQL オブジェクト作成」を指定しても無効になります。

4. グループ分け高速化処理を利用する場合，グループ分けされるグループ数が多いときは効果がないことがあります。この場合は，PDAGGR を見積もって必要な大きさの値を指定します。見積もり値以上の値を指定しても効果はありません。なお，指定値に応じてプロセス固有メモリの使用量が多くなるので注意が必要です。

5. 「フロータブルサーバ対象拡大（データ取り出しバックエンドサーバ）」を指定しなくてもすべてのバックエンドサーバをデータ取り出しに使用する SQL に対しては，データ取り出しに使用するバックエンドサーバをフロータブルサーバとして使用します。

6. 「OR の複数インデクス利用の優先」と「インデクス利用の抑止（テーブルスキャン強制）」を同時に指定した場合，「OR の複数インデクス利用の優先」は無効になります。

7. 「AND の複数インデクス利用の抑止」と「複数インデクス利用の強制」を同時に指定した場合，AND の部分については複数インデクス利用を抑止し，OR の部分については複数インデクスを強制的に利用します。

8. 「複数インデクス利用の強制」と「インデクス利用の抑止（テーブルスキャン強制）」を同時に指定した場合，「複数インデクス利用の強制」は無効になります。

9. ディクショナリ表 SQL_ROUTINES に格納される SQL 最適化オプションの値は，10 進数の形式（各最適化方法の符号なし整数の和）となります。

10. 「更新 SQL の作業表作成抑止」を適用した場合，次のすべての条件を満たすときは，同一行を複数回更新してデータベース不正になる，又は同一行を延々と更新し続けてトランザクションが終了しないことがあります。

    （発生条件）

    - **(a)** インデクスキー値無排他を使用している
    - **(b)** カーソルを使って検索している表に対して，UPDATE，INSERT，又は DELETE 文を実行する
    - **(c)** (b)の検索の WHERE 句にインデクス構成列に対する条件を指定している
    - **(d)** (c)の条件を HiRDB がインデクスを用いて評価する※
    - **(e)** (b)が UPDATE の場合，(c)のインデクス構成列の値を変更する

      注※

      アクセスパス表示ユティリティ（pdvwopt）を使用することで HiRDB が検索で使用するインデクスを確認できます。

（例）

```
<インデクス定義>
  CREATE INDEX X1 ON T1(C1) ;

<カーソル宣言>
  DECLARE CR1 CURSOR FOR SELECT C1 FROM T1 WHERE C1>0 FOR UPDATE ;

上記のカーソルを使用して，次のFETCH文，UPDATE文を繰り返すと，C1=C1+10に更新した行が再度検索され，再度C1=C1+10に更新されます。

  WHILE(SQLCODE == 0){
    FETCH CR1 INTO :XX ;
    UPDATE T1 SET C1=C1+10 WHERE CURRENT OF CR1 ;
  }
```

（対策方法）

- 二重更新が発生しないように UAP を修正してください。修正例を次に示します。

  更新後の値が，更新対象のすべての行の更新前の値と異なる場合

  UPDATE 文の更新後の値が検索の探索条件を満たさないように探索条件を変更してください。

  （例）C1 の更新前の値が 0 以上 10 未満の場合，WHERE C1>0 AND C1<10 を追加する。

  UPDATE 文の更新値が，更新列の値を単純に増加させる操作の場合

  SELECT 文に ORDER BY 更新列 DESC を指定し，大きい順に更新してください。

  （例）C1=C1+10 の場合，SELECT C1 FROM T1 WHERE C1>0 ORDER BY C1 DESC FOR UPDATE とする。

  UPDATE 文の更新値が，更新列の値を単純に減少させる操作の場合

  SELECT 文に ORDER BY 更新列 ASC を指定し，小さい順に更新してください。

  （例）C1=C1-10 の場合，SELECT C1 FROM T1 WHERE C1>0 ORDER BY C1 ASC FOR UPDATE とする。

- 同一行が複数回検索されると困る UAP については，「更新 SQL の作業表作成抑止」を適用しないようにしてください。ストアドルーチンの場合は，ルーチン定義時の SQL 最適化オプションで「更新 SQL の作業表作成抑止」を指定しないで，ルーチンを定義し直してください。
- 検索に使用するインデクスの構成列から，該当する列を削除してください。ただし，インデクス構成列の一部を削除した場合，その列が探索条件で十分に絞り込める列のときは，性能が劣化するため注意してください。また，インデクスの一部を削除した場合，インデクスキーの重複数が多くなり，排他待ち及びデッドロックが多発する可能性があります。したがって，この対策方法はあまりお勧めできる対処でないため，採用する場合には十分に検証してください。

また，UPDATE 文の SET 句の列名に，この UPDATE 文で使用するインデクスの構成列を指定し，その更新値に WHERE 句の探索条件を満たす値を指定した場合も，同一行を複数回更新することがあります。

(103) PDADDITIONALOPTLVL=SQL 拡張最適化オプション［，SQL 拡張最適化オプション］...

～<識別子，又は符号なし整数>

データベースの状態を考慮して，最も効率的なアクセスパスを決定するための最適化の方法を指定します。

SQL 拡張最適化オプションは，識別子（文字列）で指定する方法と，数値で指定する方法があります。

識別子で指定する場合：

```
PDADDITIONALOPTLVL=”識別子”［，”識別子”］…
```

＜指定例＞

- 「コストベース最適化モード 2 の適用」及び「ハッシュジョイン，副問合せのハッシュ実行」を適用する場合

  PDADDITIONALOPTLVL="COST_BASE_2","APPLY_HASH_JOIN"

- すべての最適化を使用しない場合

  PDADDITIONALOPTLVL="NONE"

＜規則＞

1. 識別子は一つ以上指定してください。
2. 識別子を二つ以上指定する場合は，コンマで区切ってください。
3. 識別子に指定できる内容（最適化方法）については，《SQL 拡張最適化オプションの指定値》を参照してください。
4. すべての機能を使用しない場合は，識別子に"NONE"を指定してください。
5. 識別子は大文字及び小文字で指定できます。
6. 同じ識別子を二つ以上指定しても，一つ指定したものとみなされますが，なるべく同じ識別子は指定しないようにしてください。
7. "識別子"［，"識別子"］…に指定できる文字列は，最大 575 バイトです。

数値で指定する場合：

```
PDADDITIONALOPTLVL=符号なし整数［，符号なし整数］…
```

＜指定例＞

- 「コストベース最適化モード 2 の適用」及び「ハッシュジョイン，副問合せのハッシュ実行」を適用する場合

  PDADDITIONALOPTLVL=1,2

- すべての最適化を使用しない場合

  PDADDITIONALOPTLVL=0

＜規則＞

1. 符号なし整数は一つ以上指定してください。
2. 符号なし整数を二つ以上指定する場合は，コンマで区切ってください。

3. 符号なし整数に指定できる内容（最適化方法）については，《SQL 拡張最適化オプションの指定値》を参照してください。
4. すべての機能を使用しない場合は，符号なし整数に 0 を指定してください。
5. 同じ符号なし整数を二つ以上指定しても，一つ指定したものとみなされますが，なるべく同じ符号なし整数は指定しないようにしてください。
6. 符号なし整数［，符号なし整数］...に指定できる文字列は，最大 575 バイトです。

《システム定義との関係》

この環境変数を省略するとシステム定義の pd_additional_optimize_level オペランドの指定値が仮定されます。pd_additional_optimize_level オペランドについては，マニュアル「HiRDB システム定義」を参照してください。

《SQL との関係》

ストアドルーチン中の SQL 文の SQL 拡張最適化オプションは，CREATE PROCEDURE，CREATE TYPE，ALTER PROCEDURE，又は ALTER ROUTINE の指定で決まり，PDADDITIONALOPTLVL の指定によって影響を受けることはありません。

SQL 文中に SQL 最適化指定を指定している場合は，SQL 最適化拡張オプションよりも SQL 最適化指定が優先されます。SQL 最適化指定については，マニュアル「HiRDB SQL リファレンス」を参照してください。

《SQL 拡張最適化オプションの指定値》

SQL 拡張最適化オプションの指定値を次の表に示します。

表 6–43　SQL 拡張最適化オプションの指定値

項番	最適化方法	指定値	
		識別子	符号なし整数
1	コストベース最適化モード 2 の適用	"COST_BASE_2"	1
2	ハッシュジョイン，副問合せのハッシュ実行	"APPLY_HASH_JOIN"	2
3	値式に対する結合条件適用機能	"APPLY_JOIN_COND_FOR_VALUE_EXP"	32
4	スカラ演算を含む条件に対するサーチ条件適用	"APPLY_SRCH_COND_FOR_VALUE_EXP"	64
5	パラメタを含む XMLEXISTS 述語への部分構造インデクスの有効化	"ENABLE_INDEX_XMLEXISTS_PARAM"	256
6	プラグインインデクスでの他インデクス絞り込み結果の利用の抑止	"DETER_IS_TRUE_AND"	2048
7	FROM 句の導出表のマージ適用	"MERGE_FROM_DERIVED_TABLE"	4096
8	外結合内結合変換機能	"CONVERT_OUTER_INNER_JOIN"	8192

注

項番 2～8 は，「コストベース最適化モード 2 の適用」を指定した場合，有効になります。

《各最適化方法の内容》

1.コストベース最適化モード 2 の適用

コストベース最適化モード 2 を使用して最適化処理をします。コストベース最適化モード 2 については，「SQL 最適化モード」を参照してください。

留意事項

- HiRDB を初めて導入した場合

  この機能の使用をお勧めします。この機能を使用する場合，最適化の精度を更に向上させるために，必要に応じて最適化情報収集ユティリティを実行してください。最適化情報収集ユティリティの実行要否については，マニュアル「HiRDB コマンドリファレンス」を参照し，性能について十分に検証するようにしてください。

  最適化情報収集ユティリティを実行する場合，テスト環境と本番環境で DB 規模が異なると（表の行数が異なると），テスト環境と本番環境でアクセスパスが変わることがあります。テスト環境では，本番での表の行数(NROWS)を最適化情報パラメタファイルに記述して，最適化情報収集ユティリティに-s オプションを指定して実行してください。

- バージョン 06-00 より前のバージョンから HiRDB をバージョンアップした場合

  この機能を使用するかどうかを検討してください。バージョンアップ前と同じ動作環境にするならば，この機能を使用しないでください。ただし，バージョン 06-00 以降にサポートする SQL 文によっては，常にこの機能を使用して最適化処理をすることがあります。

2.ハッシュジョイン，副問合せのハッシュ実行

結合検索の場合にハッシュジョインを適用して最適化をします。副問合せを伴った検索については，ハッシングによって副問合せを処理します。この最適化を適用するかどうかについては，結合方式，外への参照のない副問合せの実行方式，及び外への参照のある副問合せの実行方式を考慮して決めてください。なお，これらの詳細については，それぞれ「結合方式」，「外への参照のない副問合せの実行方式」，及び「外への参照のある副問合せの実行方式」を参照してください。

また，この最適化を適用する場合は，事前にシステム定義を指定しておく必要があります。ハッシュジョイン，副問合せのハッシュ実行を適用する場合の準備については，「ハッシュジョイン，副問合せのハッシュ実行を適用する場合の準備」を参照してください。

留意事項

- ハッシュジョインを使用しない場合は，この機能を指定する必要はありません。

3.値式に対する結合条件適用機能

この機能を指定した場合，値式を含む条件に対して結合条件を作成します。

（例）

次の式の場合，結合条件を作成します。

substr(t1.c1,1,100)=t2.c1

値式を含む結合条件しかない場合，この機能によってアクセスパスが直積からネストループジョイン，ハッシュジョイン，又はマージジョインになり，SQL 実行の高速化が期待できます。

留意事項

- バージョン 08-04 以降の HiRDB を初めて導入した場合

  この機能の使用をお勧めします。

- バージョン 08-04 より前の HiRDB をバージョンアップした場合

  この機能を使用することで結合条件が増えるため，結合順序と結合方式が変わることがあります。結合順序と結合方式が変わると，絞り込みが有効に行われない結合順序，又は性能が悪い結合方式を選択するおそれがあり，これによって，性能が低下することがあります。また，有効に行が絞り込めない結合順序を選択した場合，ジョインの途中結果を作業表に入れることができなくなり，結果がエラーになるおそれがあります。

  このため，この機能を適用する前後でアクセスパスを確認し，この機能の適用によって性能の低下やエラーが発生しないことを確認してください。性能の低下やエラーが発生した場合は，この機能の使用をやめるか，結合表の INNER 又は LEFT OUTER で結合順序を指定し，結合方式の SQL 最適化指定で適切な結合方式を指定してください。

4.スカラ演算を含む条件に対するサーチ条件適用

この機能を指定した場合，強制的にコストベース最適化モード 2 が適用されます。スカラ演算を含む探索条件のうち，次の条件をすべて満たす場合にスカラ演算をあらかじめ演算し，その結果を使ってインデクスのサーチ条件として評価し，絞り込みます。

1. 次の述語を指定している。
   ・比較述語
   ・IN 述語
   ・SIMILAR 述語
   ・BETWEEN 述語
   ・構造化繰返し述語中の上記のどれかの述語
2. 1.の述語の左辺※1 が，その問合せの FROM 句に指定した表の列指定※2 である。
3. 2.で指定した列指定にインデクスを定義している。
4. 1.の述語の右辺※1 が，その問合せの FROM 句に指定した表の列指定を含まないスカラ演算である。

注※1

比較述語の場合は左辺と右辺の指定が逆でも適用されます。

注※2

繰返し列を指定する場合は，添字として ANY を指定した場合に適用されます。

この機能を指定した場合の HiRDB の動作

HiRDB は，インデクスを使用した検索の場合，次の順序で評価をします。

1. インデクスのサーチ範囲を絞り込みます（サーチ条件）。
2. 1.で絞り込んだ結果に対して，インデクスのキー値ごとに条件を判定して更に絞り込みます（キー条件）。
3. 2.で真となったキー値を持つ行のデータページを参照し，残りの探索条件を評価します。

スカラ演算を含む条件は，この最適化を指定しない場合は 2.（「スカラ演算を含むキー条件の適用」指定時）又は 3.で評価されますが，この最適化を指定した場合は，1.で評価されます。スカラ演算を含む探索条件の評価タイミングが 2.から 1.に変わる場合には，インデクスのサーチ範囲を絞り込めます。3.から 1.に変わる場合には，データページを参照する行数が少なくなり入出力を削減できます。サーチ条件及びキー条件については，マニュアル「HiRDB コマンドリファレンス」を参照してください。

留意事項

- バージョン 09-50 以降の HiRDB を初めて導入した場合

  この機能の使用をお勧めします。

- バージョン 09-50 より前の HiRDB をバージョンアップした場合

  この機能は SQL 拡張最適化オプション省略時にデフォルトで適用されます。

  この機能を使用することで，この機能を適用できる SQL で使用するインデクスやサーチ条件種別が変わり，アクセスパスが変更になることがあります。それによって，SQL 文の検索性能が低下するおそれがあります。この機能を使用する前後でアクセスパスを確認し，アクセスパスの変更によって性能低下が見られる場合は，この機能の使用を取りやめてください。

  この機能を使用することで，強制的にコストベース最適化モード 2 が適用されます。そのため，バージョンアップ前にコストベース最適化モード 1 で運用していた場合に，すべての SQL で使用するインデクスやサーチ条件種別などのアクセスパスが変更になることがあり，SQL 文の検索性能が低下するおそれがあります。この機能を使用する前後でアクセスパスを確認し，アクセスパスの変更によって性能低下が見られる場合は，この機能の使用を取りやめてください。

5.パラメタを含む XMLEXISTS 述語への部分構造インデクスの有効化

次の二つの条件を満たす検索をするときに部分構造インデクスを使用できるようにします。

- XMLEXISTS 述語の XML 問い合わせ関数に CAST 指定を指定して，かつ CAST 指定の値式に?パラメタ，SQL パラメタ，又は SQL 変数を指定
- 定義されているインデクスの部分構造指定と，XMLEXISTS 述語の XQuery 問い合わせ中で条件として指定した部分構造パスが一致

部分構造インデクスが使用できることで，検索処理の性能向上が期待できます。部分構造インデクスの使用条件については，マニュアル「HiRDB システム導入・設計ガイド」を参照してください。

部分構造インデクスを使用して検索処理の性能を向上させる場合の例を次に示します。

SQL 例

- 表定義

  CREATE TABLE T1(C1 XML);

- インデクス定義

  CREATE INDEX T1C1 ON T1(C1) KEY USING UNIQUE TAG FROM ‘/a/b/c’ AS VARCHAR(30);

- データ挿入

  INSERT INTO T1(C1) VALUES(XMLPARSE(DOCUMENT '<?xml version="1.0" encoding="Shift_JIS"?><a><c>aaa</c>'));

- 検索

  SELECT XMLSERIALIZE(C1 AS VARCHAR(256)) FROM T1 WHERE
  XMLEXISTS('/a/b[c=$AAA]' PASSING BY VALUE C1,CAST(? AS VARCHAR(256)) AS AAA);

留意事項

- バージョン 09-03 以降の HiRDB を初めて導入した場合

  この機能の使用をお勧めします。

- バージョン 09-03 より前の HiRDB をバージョンアップした場合

  この機能を使用すると，HiRDB が XMLEXISTS 述語を指定して検索する際に部分構造インデクスを使用する可能性があり，一部の検索処理で性能が低下するおそれがあります。

  このため，この機能の使用前後でアクセスパスを確認し，使用後にアクセスパスが変更されていないか確認してください。アクセスパスが変更され，性能低下が見られる場合は，この機能の使用を中止してください。

6.プラグインインデクスでの他インデクス絞り込み結果の利用の抑止

プラグインインデクスで他インデクス絞り込み結果を利用しないアクセスパスにする場合，この最適化を指定してください。プラグインインデクスでの他インデクス絞り込み結果の利用については，「プラグインインデクスでの他インデクス絞り込み結果の利用」を参照してください。

7.FROM 句の導出表のマージ適用

この機能を指定した場合，強制的にコストベース最適化モード 2 が適用されます。FROM 句の導出表を，上位問合せにマージして作業表を作成しないことで性能を向上させます。

マージを適用する条件については，マニュアル「HiRDB SQL リファレンス」の「表参照」の「FROM 句の導出表についての規則」を参照してください。

留意事項

- バージョン 09-50 以降の HiRDB を初めて導入した場合

  この機能の使用をお勧めします。

- バージョン 09-50 より前の HiRDB をバージョンアップした場合

  この機能は SQL 拡張最適化オプション省略時にデフォルトで適用されます。

  この機能を使用することによって，従来使用していたインデクスを使用しなくなり，検索方式や結合方式が変化するため，性能が低下するおそれがあります。

  そのため，この機能を使用する前後で意図したインデクスを使用しているかどうかをアクセスパスで確認してください。

  アクセスパスの変更によって性能低下が見られるときは，この機能の使用を取りやめてください。

  この機能を使用することで，強制的にコストベース最適化モード 2 が適用されます。そのため，バージョンアップ前にコストベース最適化モード 1 で運用していた場合に，すべての SQL で使用するインデクスやサーチ条件種別などのアクセスパスが変更になることがあり，SQL 文の検索性能が低下するおそれがあります。この機能を使用する前後でアクセスパスを確認し，アクセスパスの変更によって性能低下が見られるときは，この機能の使用を取りやめてください。

8.外結合内結合変換機能

この機能を指定した場合，強制的にコストベース最適化モード 2 が適用されます。この機能を指定した場合，外結合を含む問合せ指定の WHERE 句に指定した探索条件をチェックし，外結合でナル拡張して生成される行（結合条件を満たさない外表の行）が確実に選択されないことが分かるとき，外結合から内結合に変換します。

外結合から内結合に変換することで不要な処理を省略できるだけでなく，WHERE 句に指定した内表の列に対する探索条件へのインデクスの適用ができるようになり，実行性能の向上が期待できます。

また，導出表の条件繰り込み機能と組み合わせて使用することで，変換対象の外結合の内表として指定した表に対して WHERE 句で指定した探索条件の導出表作成前の評価ができるようになり，実行性能の向上が期待できます。

留意事項

- バージョン 09-50 以降の HiRDB を初めて導入した場合

  この機能の使用をお勧めします。

- バージョン 09-50 より前の HiRDB をバージョンアップした場合

  この機能は SQL 拡張最適化オプション省略時にデフォルトで適用されます。

  この機能を使用することで，アクセスパスが変わった結果として選択されたインデクスが，検索する行を有効に絞り込めなくなり，性能が低下するおそれがあります。

  そのため，この機能を使用する前後でアクセスパスを確認し，この機能を使用した後に，検索する行を有効に絞り込めないインデクスを使用していないことを確認してください。

  検索する行を有効に絞り込めないインデクスを使用している場合は，使用インデクスの SQL 最適化指定で使用するインデクスを指定するか，又はこの機能の使用をやめるなどの対策をしてください。

  この機能を使用することで，強制的にコストベース最適化モード 2 が適用されます。そのため，バージョンアップ前にコストベース最適化モード 1 で運用していた場合に，すべての SQL で使用するインデクスやサーチ条件種別などのアクセスパスが変更になることがあり，SQL 文の検索性能が低下するおそれがあります。この機能を使用する前後でアクセスパスを確認し，アクセスパスの変更によって性能低下が見られるときは，この機能の使用を取りやめてください。

(104) PDHASHTBLSIZE=ハッシュジョイン，副問合せのハッシュ実行適用時のハッシュ表サイズ

32 ビットモードの場合

～<符号なし整数>((128～524288))（単位：キロバイト）

64 ビットモードの場合

～<符号なし整数>((128～2097152))（単位：キロバイト）

SQL 拡張最適化オプションで「ハッシュジョイン，副問合せのハッシュ実行」を適用した場合，ハッシュ表のサイズを指定します。

指定値は，128 の倍数で指定してください。指定値が 128 の倍数でない場合，128 の倍数に切り上げられます。

サーバ側が 32 ビットモードの場合，524289～2097152 の値を指定すると，上限値 524288 が仮定されます。

《指定値の目安》

「ハッシュジョイン，副問合せのハッシュ実行を適用する場合の準備」を参照してください。

《システム定義との関係》

この環境変数を省略した場合，システム定義の pd_hash_table_size オペランドの指定値が仮定されます。

(105)　PDDFLNVAL={USE｜NOUSE}

表中のデータを埋込み変数に取り出す場合，取り出した値がナル値のときに埋込み変数に既定値を設定するかどうかを指定します。

USE：ナル値の既定値設定機能を使用します。

NOUSE：ナル値の既定値設定機能を使用しません。

ナル値の既定値設定機能については，マニュアル「HiRDB SQL リファレンス」を参照してください。

Java ストアドプロシジャで動作する Type4 内部 JDBC ドライバの場合は，Java ストアドプロシジャ呼び出し元のこの環境変数の指定と同じ指定をしてください。

(106)　PDAGGR=グループ分けのときに発生するグループ数

32 ビットモードの場合

～〈符号なし整数〉((0～30000000))《1024》

64 ビットモードの場合

～〈符号なし整数〉((0～2147483647))《1024》

GROUP BY 処理に使用するメモリ量を決定するため，サーバごとに発生するグループ数の最大値を指定します。なお，この環境変数は，SQL 最適化オプションでグループ分け高速化処理を指定した場合に有効になります。

《見積もり方法》

- 1024 以上のグループ数が発生する場合，又は期待していた性能が得られない場合
  この環境変数の値を大きくしてください。ただし，メモリ所要量との兼ね合いを考えて，少しずつ指定値を大きくしてください。
  1024 を指定すると，メモリが足りなくなるユーザはメモリに合わせて指定してください。
- 1024 未満のグループ数が発生する場合，又はメモリが足りなくなる場合

この環境変数の値を小さくしてください。また，メモリ所要量が多くなり，必要な大きさの値が指定できない場合，グループ数以下の値でも指定できる最大値を指定してください。

《留意事項》

指定値が大き過ぎると，メモリ不足となることがあります。また，指定値を超えたグループ数が発生した場合，割り当てられたメモリが不十分なため，処理が遅くなることがあります。なお，グループ分け高速化機能で使用するメモリ所要量の計算式については，マニュアル「HiRDB システム導入・設計ガイド」を参照してください。

(107) PDCMMTBFDDL={YES | NO}

操作系 SQL を実行しているトランザクションで定義系 SQL を実行する場合，自動的にコミットしてから定義系 SQL を実行するときに指定します。なお，定義系 SQL 実行前に自動的にコミットをした場合，開いているホールダブルカーソルは閉じて，前処理した SQL 文の結果は無効になります。

YES：

操作系 SQL を実行しているトランザクションを自動的にコミットしてから，定義系 SQL を実行します。また，開いているホールダブルカーソルは閉じて，前処理した SQL 文の結果は無効になります。

NO：

操作系 SQL を実行しているトランザクションを明示的に決着してから，定義系 SQL を実行します。

(108) PDPRPCRCLS = {YES | NO}

開いているカーソルで使用している SQL 識別子を再度 PREPRARE 文で使用する場合，開いているカーソルを自動的にクローズするかどうかを指定します。

この環境変数は，プリプロセス時に-Xe オプションを指定しない場合に有効になります。プリプロセスについては，「プリプロセス」を参照してください。

YES：開いているカーソルを自動的にクローズします。

NO：開いているカーソルを自動的にクローズしません。

(109) PDAUTOCONNECT = {ON | OFF}

HiRDB と接続していない状態で SQL 文を実行した場合，自動的に CONNECT するかどうかを指定します。

ON：自動的に CONNECT してから SQL 文を実行します。

OFF：自動的に CONNECT しません。このとき SQL 文はエラー（SQLCODE=-563）となります。

HiRDB サーバと接続していない状態で SET SESSION AUTHORIZATION 文を実行した場合，この環境変数の指定に関係なく常にエラー（SQLCODE=-563）となります。

UAP を開発する場合は，HiRDB に正しく CONNECT しているかどうかを判断する必要があるため，この環境変数には OFF を指定することをお勧めします。

(110)　PDDDLDEAPRPEXE=｛YES｜NO｝

先行するトランザクション（以降，先行トランザクションと呼びます）が，定義系 SQL を実行するトランザクション（以降，定義系トランザクションと呼びます）で操作する表，ストアドルーチン，抽象データ型などを既に使用していると，通常，定義系トランザクションは先行トランザクションの決着を待ちます。この環境変数に YES を指定すると，先行トランザクションの前処理結果を無効にし，定義系トランザクションを優先的に実行できます。

参考

次に示す環境変数を指定することで，定義系トランザクションを実行するタイミングを指定できます。

1. PDDDLDEAPRPEXE

   先行トランザクションの前処理結果を無効にし，定義系トランザクションを実行できます。

2. PDDDLDEAPRP

   先行トランザクションのホールダブルカーソルが閉じた後，そのホールダブルカーソルを含むトランザクションの決着後に定義系トランザクションを実行できます。定義系トランザクションが実行されると，ホールダブルカーソルの前処理結果は無効になります。

3. PDLCKWAITTIME

   排他待ち限界経過時間を指定できます。PDDDLDEAPRP と PDLCKWAITTIME を組み合わせて使用すると，先行トランザクションの決着までに定義系トランザクションがタイムアウトエラーになることを防止できます。

また，PDDDLDEAPRPEXE 及び PDDDLDEAPRP を組み合わせた場合の，先行トランザクションの前処理結果無効の可否を次に示します。

クライアント環境定義		ホールダブルカーソルで使用する前処理結果	ホールダブルカーソルで使用しない前処理結果
PDDDLDEAPRPEXE	PDDDLDEAPRP		
YES	YES	○	○
	NO	○	○
NO	YES	△	×
	NO	×	×

（凡例）

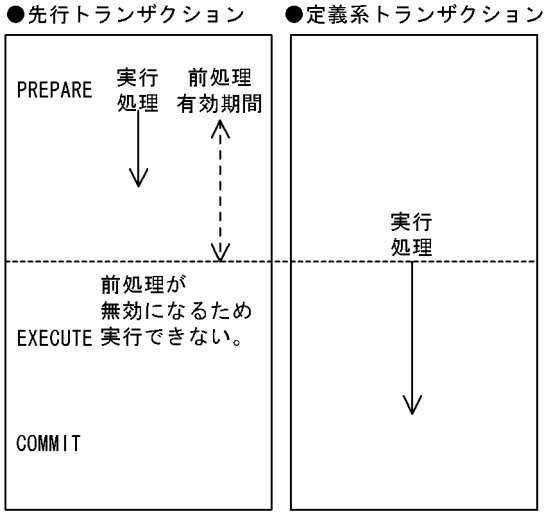
○：無効にできます。

△：先行トランザクションの決着後，無効にできます。

×：無効にできません。

YES：

先行トランザクションの前処理結果を無効にし，定義系トランザクションの実行を優先します。YES を指定した場合の例を次に示します。



[説明]

定義系トランザクションは，先行トランザクションの決着を待ちません。

定義系トランザクションを実行すると，先行トランザクションの前処理結果が無効になり，先行トランザクションは実行できなくなり，SQLCODE=-1512 エラーとなります。

《留意事項》

1. 前処理結果を無効にできるのは，定義系トランザクションが次の操作をした場合です（ただし，先行トランザクションが前処理実行中の場合を除く）。
   - 定義系 SQL を実行した場合
   - ストアドルーチンの SQL オブジェクトを格納するデータディクショナリ LOB 用 RD エリアをクローズした場合・
   - データベース構成変更ユティリティ（pdmod）で次に示す操作をした場合

     ・RD エリアの削除（remove rdarea 文）で解析情報表（SQL_DB_STATE_ANALYZED），及び運用履歴表（SQL_DB_MANAGEMENT）を格納するデータディクショナリ用 RD エリアを削除したとき

     ・ディクショナリ表の属性定義変更（alter system 文）で参照権限，又は混在文字データの使用有無を変更したとき

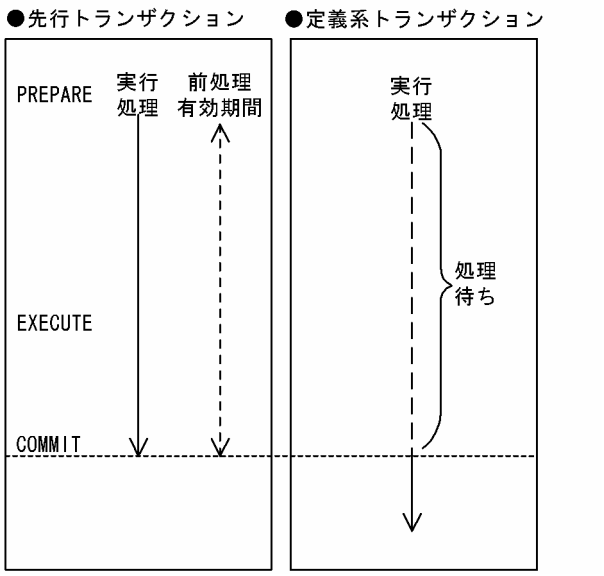
     ・RD エリアの属性変更（alter rdarea 文）で RD エリアの名称を変更したとき

   ただし，先行トランザクションで定義系 SQL（リバランスユティリティなども含む）が実行中の場合，その定義変更中の表，手続き，関数などを使用する定義系トランザクションはエラーになります。
2. 前処理結果を無効にした場合，先行トランザクションでは再度前処理を行う必要があります。
3. Java EE アプリケーションサーバのステートメントプーリング／ステートメントキャッシュ機能を使用している場合，無効になった Statement オブジェクトを再利用して SQL エラーとなる場合がありま

す。詳細はマニュアル「HiRDB システム導入・設計ガイド」の「ステートメントプーリング／ステートメントキャッシュ」の注意事項を参照してください。

NO：

定義系トランザクションの実行を優先しません。NO を指定した場合の例を次に示します。



［説明］

定義系トランザクションは，先行トランザクションで実行中の SQL 文に関係なく，先行トランザクションの決着を待ちます。

先行トランザクションの決着後に，定義系トランザクションが実行されます。

《指定値の目安》

先行するトランザクションの終了を待たないで，先行するトランザクション中に使用する表に対する定義変更，インデクス定義変更などを行う場合の，クライアント環境定義 PDDDLDEAPRPEXE と PDDDLDEAPRP の指定値の目安を次に示します。

使用している機能		無効にする前処理結果の SQL 形式	PDDDLDEAPRPEXE の指定値の目安	PDDDLDEAPRP の指定値の目安
接続プーリング機能	ステートメントプーリング機能	ホールダブルカーソル		
○	○	○	YES	YES
○	×	○	NO	YES
○	○	×	YES	NO
○	×	×	NO	NO
×	×	○	NO	YES
×	×	×	NO	NO

（凡例）

○：使用しています。

×：使用していません。

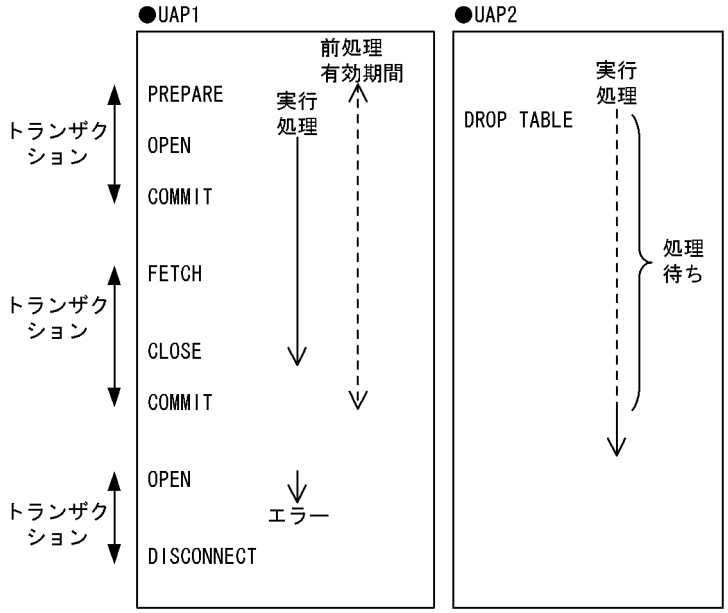
(111) PDDDLDEAPRP={YES｜NO}

閉じているホールダブルカーソルで使用している表の定義情報を，トランザクション間に他 UAP からの変更を許可するかどうかを指定します。なお，定義系 SQL が実行されると，ホールダブルカーソルの前処理は無効になります。

YES：

ホールダブルカーソルを使用している UAP のトランザクション間に，他 UAP から表の定義情報の変更を許します。

YES を指定した場合の例を次に示します。



［説明］

UAP2 で実行した定義系 SQL は，UAP1 のホールダブルカーソルが閉じた後，そのホールダブルカーソルを含んだトランザクションが決着すると，実行できます。また，UAP1 のホールダブルカーソルを再度開くと，SQLCODE=-1512 エラーとなります（前処理は無効になります）。

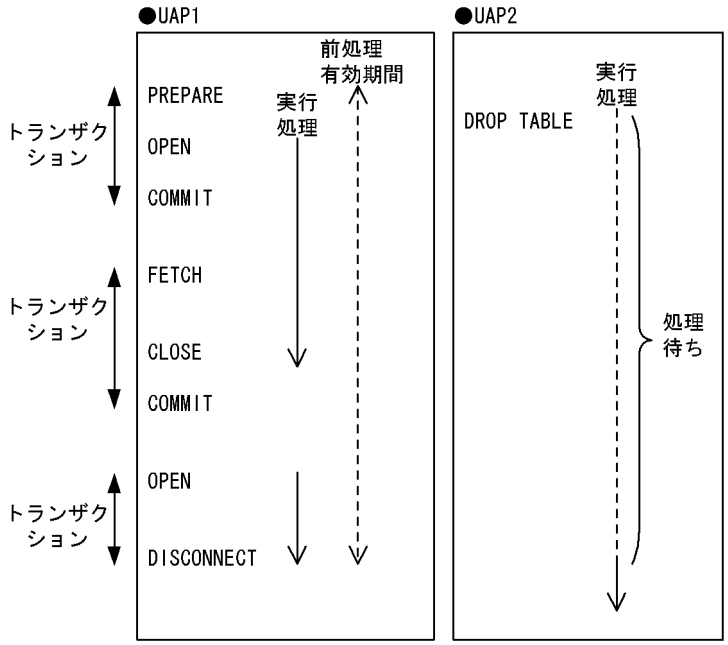
《留意事項》

定義系 SQL 実行後，そのホールダブルカーソルの前処理が無効になるため，後続のトランザクションでは再度ホールダブルカーソルを前処理する必要があります。

NO：

ホールダブルカーソルを使用している UAP のトランザクション間に，他 UAP から表の定義情報の変更を許しません。

NO を指定した場合の例を次に示します。



［説明］

UAP2 で実行した定義系 SQL は，UAP1 の DISCONNECT 後に実行できます。

《指定値の目安》

HiRDB の接続を保持したまま，サービスの要求を待機する Web アプリケーションで，ホールダブルカーソル使用中に表の定義変更を行う SQL 文を実行する場合は，YES を指定してください。

《ほかの環境変数との関係》

PDLCKWAITTIME で排他待ち限界経過時間を指定できます。PDDDLDEAPRP と PDLCKWAITTIME を組み合わせて使用すると，トランザクションの決着までに定義系 SQL がタイムアウトエラーになることを防止できます。

(112)　PDLCKWAITTIME=排他待ち限界経過時間

～<符号なし整数>((0～65535))《システム定義 pd_lck_wait_timeout の値》（単位：秒）

排他要求が待ち状態になってから解除されるまでの最大監視時間を指定します。指定した最大時間を過ぎても排他が解除されない場合，SQL はエラーリターンします。0 を指定した場合，待ち状態を監視しないで，排他が解除されるまで待ち続けます。

(113)　PDCURSORLVL= {0 | 1 | 2}

カーソルを使用した検索をする場合に，HiRDB クライアントから HiRDB サーバに対するカーソルオープン・クローズの要求を，どのタイミングで行うかを指定します。この環境変数を指定することで，アプリケーションからカーソルオープンの要求を受けた場合に，HiRDB サーバに対しては要求をしないで，初回取り出し時にカーソルオープン要求をします。また，検索データなし（SQLCODE=100）を検知した時点で，カーソルクローズをします。これによって，通信オーバヘッドを削減できます。

0：

アプリケーションから，カーソルオープン・クローズの要求を受けた場合，HiRDB クライアントは HiRDB サーバに対して，そのまま実行要求をします。

1：

検索データがない場合，HiRDB サーバは SQLCODE=100 の返却と同時に，HiRDB クライアントからの要求なしでカーソルをクローズします。HiRDB クライアントは，アプリケーションからのカーソルクローズ要求を受けた場合に，既に SQLCODE=100 を検知していれば，HiRDB サーバに対してカーソルクローズ要求をしません。SQLCODE=100 を検知していない場合にだけ，カーソルクローズ要求をします。

カーソルオープン要求については，0 を指定した場合と同じです。

2：

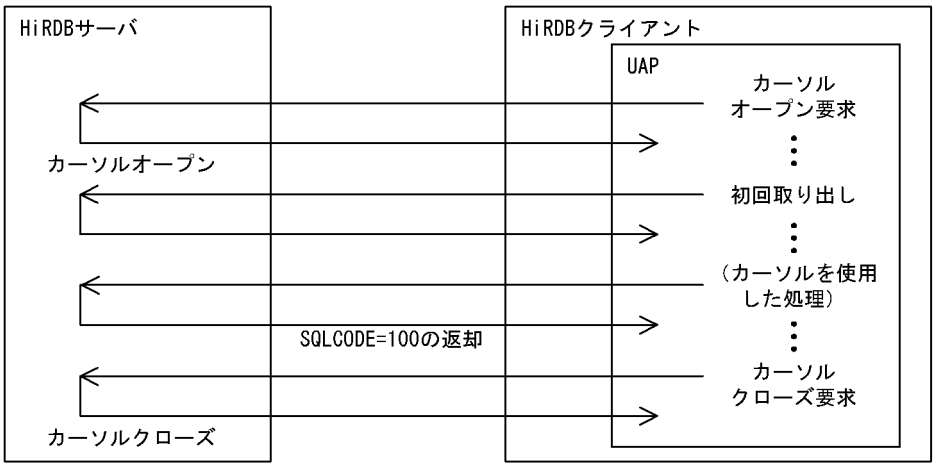
アプリケーションからのカーソルオープン要求を受けた場合に，HiRDB サーバに対しては実行を要求しないで，初回取り出し要求と同時にカーソルオープン要求を行います。

カーソルクローズ要求については，1 を指定した場合と同じです。

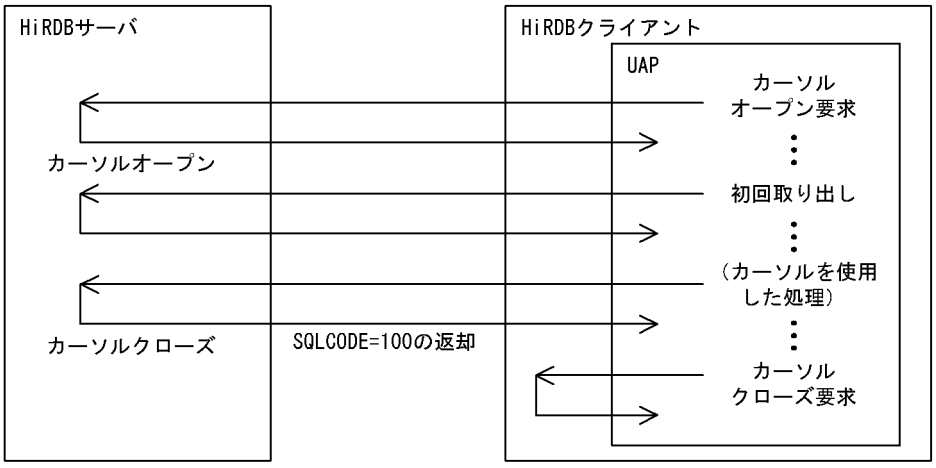
指定値ごとの処理概要を次の図に示します。

図 6-7　PDCURSORLVL の指定値ごとの処理概要

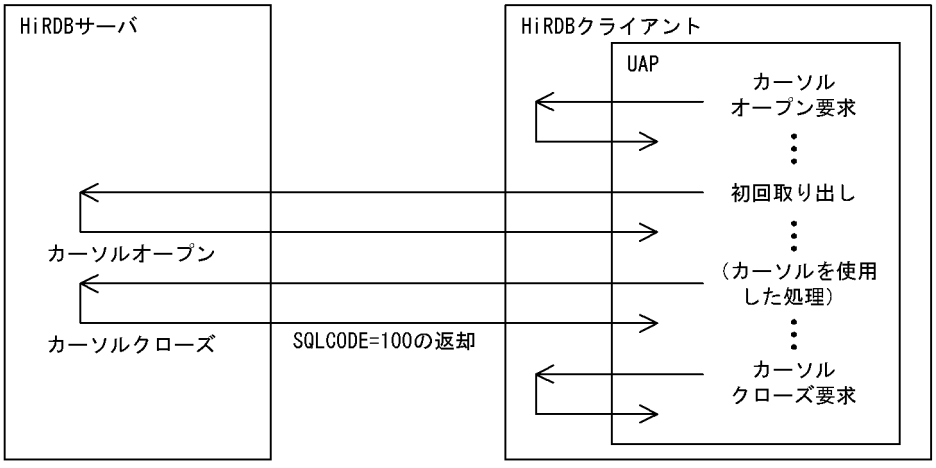
・0の場合



・1の場合



・2の場合



《注意事項》

- この環境変数に 1 又は 2 を指定した場合でも，手続きから返却された結果集合に対するカーソルのクローズ要求を受けたときは，HiRDB サーバに対して実行要求を行います。

- この環境変数に 1 又は 2 を指定した場合のカーソルクローズは，UAP に関する統計情報の SQL 実行回数には加算されますが，SQL に関する統計情報には出力されません。また，この環境変数に 2 を指定した場合のカーソルオープンも，UAP に関する統計情報の SQL 実行回数に加算されますが，SQL に関する統計情報には出力されません。
- この環境変数に 1 又は 2 を指定した場合でも，SQL トレースにはカーソルオープン・クローズのオペレーションコードが出力されます。また，FETCH 文でカーソルオープン・クローズした場合，カーソルオープン・クローズの SQL に関する統計情報，アクセスパス情報，及び SQL 実行時の中間結果情報は FETCH 側に出力されます。
- HiRDB/パラレルサーバの場合，カーソルオープン実行後から初回取り出しを実行するまでの間が長いと，初回取り出しに時間が掛かることがあります。
- この環境変数に 2 を指定した場合に，初回取り出しのコール前のオープン中カーソルに対して，再度 PREPARE 文を実行しても，HiRDB サーバへのカーソルオープン実行要求は行われていないため，エラーになりません。再度 PREPARE 文を実行した場合は，その PREPARE 文の情報をカーソル情報として使用するため，カーソルのオープンも再実行する必要があります。
- この環境変数に 1 又は 2 を指定した場合，SQLCODE=100 を検知後，CLOSE 文を実行しないで PREPARE 文又は OPEN 文を実行しても，カーソルは既にクローズされているため，エラーになりません。また，SQLCODE=100 を検知後に続けて FETCH 文を実行した場合は，検索データなしにはならないで，カーソルが開かれていないことを示す SQLCODE=-501 が返却されます。

(114)　PDDELRSVWDFILE=SQL 予約語削除ファイル名

～＜識別子＞((最大 8 文字))

SQL 予約語削除機能を使用する場合に，SQL 予約語削除ファイル名を指定します。SQL 予約語削除ファイルには，SQL の予約語から削除するキーワードを記述します。

《システム定義との関連》

PDDELRSVWDFILE を指定する場合，システム定義の pd_delete_reserved_word_file オペランドで SQL 予約語削除ファイルを指定しておく必要があります。SQL 予約語削除ファイルについては，マニュアル「HiRDB システム定義」を参照してください。

《注意事項》

Windows 版の HiRDB の場合，SQL 予約語削除ファイル名は，大文字，小文字が区別されません。したがって，大文字，小文字が異なるだけのファイル名は，同一ファイルとして扱われるため注意してください。

(115)　PDCALCMDWAITTIME=CALL COMMAND 文の最大待ち時間

～<符号なし整数>((0～65535))《PDCWAITTIME の指定値》（単位：秒）

CALL COMMAND 文によってコマンド，又はユティリティを開始してから終了するまでの，HiRDB クライアントの最大待ち時間を指定します。最大待ち時間を経過してもサーバから応答がない場合は，UAP にエ

ラーを返し，サーバ側のプロセスをキャンセルします。0 を指定した場合，HiRDB クライアントは HiRDB サーバからの応答があるまで待ち続けます。

(116) PDSTANDARDSQLSTATE=｛YES ｜ NO｝

SQLSTATE の値を詳細に出力するかどうかを指定します。

YES：

SQLSTATE の値を詳細に出力します。

NO：

SQLSTATE の値を詳細に出力しません。

《システム定義との関連》

この環境変数を省略した場合，システム共通定義の pd_standard_sqlstate オペランドの指定値が仮定されます。ただし，サーバとの接続が完了するまでにエラーが発生した場合，HiRDB はこの環境変数を省略したときには NO が指定されたものとして動作し，環境変数を指定したときには指定した内容で動作します。

SQLSTATE については，マニュアル「HiRDB メッセージ」を参照してください。

なお，Java ストアドプロシジャで使用するドライバが Type4 JDBC ドライバの場合，Java ストアドプロシジャに返される SQLSTATE の値を詳細に設定するかどうかは，クライアント（Java ストアドプロシジャ呼び出し元）の PDSTANDARDSQLSTATE，システム共通定義の pd_standard_sqlstate の値，及び Java ストアドプロシジャの PDSTANDARDSQLSTATE の指定値の組み合わせによって決まります。指定値の組み合わせを次に示します。

クライアント環境変数 PDSTANDARDSQLSTATE の指定値	システム共通定義 pd_standard_sqlstate の値	Java ストアドプロシジャ PDSTANDARDSQLSTATE の指定値	SQLSTATE の値を詳細に設定するかどうか
YES	−	YES	詳細に設定する
省略	Y	YES	
NO	−	NO 又は省略	詳細に設定しない
省略	N	NO 又は省略	

（凡例）

−：pd_standard_sqlstate の値は SQLSTATE の詳細設定に影響しません。

上記以外の組み合わせは指定できません。

ODBC ドライバから返却する SQLSTATE はこの環境変数，及びシステム共通定義の pd_standard_sqlstate の指定に関係なく，ODBC の規格に従った値となります。

(117)　PDBLKF=ブロック転送の行数

～<符号なし整数>((1～4096))《1》

サーバからクライアントに検索結果を転送する場合の，一回の転送処理で送信する行数を指定します。

なお，実際に送信する行数は，クライアント環境定義 PDBLKBUFFSIZE の指定値によって変わります。送信する行数については，「1 回の通信で転送する行数」を参照してください。

この値を大きくすると通信オーバヘッドが減り，検索時間を短縮できますが，その分メモリが余計に必要となります。したがって，メモリとの兼ね合いを考慮して値を決めてください。

サーバ側で必要なメモリの計算式については，マニュアル「HiRDB システム導入・設計ガイド」の「ブロック転送又は配列 FETCH で必要なメモリ所要量」を参照してください。クライアント側で必要なメモリの計算式を次に示します。

```
メモリ計算式（単位：キロバイト）
＝↑(600＋19×検索列数＋(7×検索列数＋Σ　列の定義長※)×PDBLKFの値)
　÷4096↑×4

注※　単位はバイトです。
```

(118)　PDBINARYBLKF={YES ｜ NO}

定義長が 32,001 バイト以上の BINARY 型の選択式がある表を検索する場合，ブロック転送機能を適用するかどうかを指定します。ブロック転送機能については，「ブロック転送機能」を参照してください。

YES：

ブロック転送機能を適用します。

NO：

ブロック転送機能を適用しません。

この場合，クライアント環境定義 PDBLKF に 2 以上，PDBLKBUFFSIZE に 1 以上を指定しても，1 件ずつの転送となります。

(119)　PDBLKBUFFSIZE=通信バッファサイズ

～＜符号なし整数＞（単位：キロバイト）

・Type4 JDBC ドライバ使用時：((0～2000000))《0》

・上記以外：

　・PDBLKF を省略した場合：((0～2000000))《10》

　・PDBLKF を指定した場合：((0～2000000))《0》

・0904 互換モードの場合：((0～2000000))《0》

ブロック転送機能で使用する，サーバ，クライアント間の通信バッファのサイズを指定します。

0 を指定した場合，クライアント環境定義 PDBLKF の値と 1 行の最大長から，HiRDB が通信バッファサイズ（単位：バイト）を算出します。

PDBLKBUFFSIZE で指定した値は，次のバッファサイズ，及び行数に影響します。

- 検索結果の転送で使用するサーバ，クライアント間の通信バッファサイズ
- シングルサーバ又はフロントエンドサーバからクライアントに送信する，1 回の通信での検索結果の行数

(120) PDBINDRETRYCOUNT=bind システムコールのリトライ回数

～＜整数＞((-1～1000))《10》

UNIX ドメインでの bind システムコールで EADDRINUSE が返却された場合のリトライ回数を指定します。

クライアントサーバ間が同一ホストの場合，HiRDB クライアントでは UNIX ドメイン通信の準備として名前付きファイルをソケットに割り当てるため，bind システムコールを発行します。

-1 を指定した場合は，EADDRINUSE が返却されなくなるまでリトライします。0 を指定した場合は，リトライをしないで UAP にエラーを返します。

《指定値の目安》

KFPA11723-E メッセージのネットワーク障害（reason=NETWORK）で，クライアントエラーログファイルに KFPZ02444-E メッセージが出力されている場合，内容が func=bind，errno=EADDRINUSE で count と interval が表示されているときは，この環境変数と PDBINDRETRYINTERVAL の値を大きくしてください。

(121) PDBINDRETRYINTERVAL=bind システムコールのリトライ間隔

～＜符号なし整数＞((0～1000))《0》（単位：ミリ秒）

UNIX ドメインでの bind システムコールで EADDRINUSE が返却された場合のリトライ間隔を指定します。0 を指定した場合は，即時にリトライを行います。

(122) PDDBACCS=アクセスする RD エリアの世代番号

～＜符号なし整数＞((0～10))

インナレプリカ機能を使用している場合，インナレプリカグループ内でカレント RD エリアでない RD エリアをアクセスしたいときに，その RD エリアの世代番号を指定します。0 はオリジナル RD エリアとなります。省略した場合は，UAP 環境定義 PDDBACCS の値が仮定されます。

この環境変数は，HiRDB に定義されているすべてのインナレプリカグループに対して適用されます。この環境変数で指定した世代のレプリカ RD エリアが定義されていない場合，該当するインナレプリカグループ内のカレントの RD エリアが処理対象となります。このため，レプリカ RD エリアを使用するテスト環境を設定する場合，誤って本番用の RD エリアをアクセスしないように，アクセスするすべての RD エリアについて，指定する世代のレプリカ RD エリアが定義されているか確認する必要があります。

(123)　PDDBORGUAP={YES ｜ NO}

レプリカ RD エリアでのオンライン業務中に，オリジナル RD エリアに対して UAP を実行する場合に指定します。

YES：

オンライン再編成閉塞のオリジナル RD エリアに対して，UAP を実行する場合に指定します。

NO：

オンライン再編成閉塞のオリジナル RD エリアに対して，UAP を実行しない場合に指定します。

(124)　PDSPACELVL ＝ {0 ｜ 1 ｜ 3}

データの格納，比較，及び検索時の，空白変換レベルを指定します。なお，定義系 SQL 実行時には，空白変換はされません。

0：

空白は変換しません。

1：

操作系 SQL での定数，埋込み変数，又は？パラメタのデータの空白を，次のように変換します。

- 文字列定数を各国文字列定数とみなした場合，半角空白 2 バイトを全角空白 1 文字に変換します。このとき，半角空白が 1 バイト単独で現れる場合，変換はしません。
- 混在文字列定数は，全角空白 1 文字を半角空白 2 バイトに変換します。
- 各国文字列型の列へのデータの格納時，及び各国文字列型の値式との比較時は，埋込み変数，又は？パラメタの半角空白 2 バイトを全角空白 1 文字に変換します。このとき，半角空白が 1 バイト単独で現れる場合，変換しません。
- 混在文字列型の列へのデータの格納時，及び混在文字列型の値式との比較時は，埋込み変数，又は？パラメタの全角空白 1 文字を半角空白 2 バイトに変換します。

3：

空白変換レベル 1 に加えて，各国文字列型の値式のデータを検索するときに，全角空白 1 文字を半角空白 2 バイトに変換します。

《システム定義との関連》

この環境変数を省略した場合，システム共通定義の pd_space_level オペランドの指定値が仮定されます。

《注意事項》

1. 空白変換レベルを変更した場合，変更の前後で UAP の結果が異なる場合があります。UAP の結果を同じにしたい場合には，空白変換レベルを変更しないでください。
2. 空白変換レベル 3 を指定してソートをした場合，HiRDB はソートの結果に対して空白変換をするため，期待した結果を得られないことがあります。
3. クラスタキーの列へデータを格納する場合，空白変換によってユニークエラーとなることがあります。この場合，空白変換をしないでデータを格納するか，又は既存のデータベースの空白を統一（データベース再編成ユティリティで空白変換）してください。
4. 各国文字列の空白変換は，先頭から 2 バイト単位で変換します。
5. 空白変換レベル 1，又は 3 を指定した場合，ハッシュ分割した表に対して UAP で表分割ハッシュ関数を使用して格納先 RD エリアを求めるときは，表分割ハッシュ関数の引数に空白変換レベルを指定しないと，表分割ハッシュ関数の結果が不正になることがあります。表分割ハッシュ関数については，「表分割ハッシュ関数」を参照してください。
6. 空白変換レベル 1，又は 3 を指定した場合，キーレンジ分割した表に対して UAP でキーレンジ分割処理をしていて分割キーに各国文字列型，又は混在文字列型の列があるときは，その分割キー値を空白変換関数で変換しないとキーレンジ分割の結果が不正になることがあります。空白変換関数については，「空白変換関数」を参照してください。

(125)　PDCLTRDNODE=XDM/RD E2 のデータベース識別子

～＜識別子＞

XDM/RD E2 接続機能使用時に，接続する XDM/RD E2 のデータベース識別子を指定します。データベース識別子とは，XDM のサブシステム定義で指定する RD ノード名のことです。

(126)　PDTP1SERVICE=｛YES｜NO｝

XDM/RD E2 接続機能使用時に，XDM/RD E2 に OpenTP1 のサービス名称を通知するかどうかを指定します。

Windows 版の HiRDB クライアントライブラリで cltdll.dll を使用している場合，この環境変数は指定できません。ほかの HiRDB クライアントライブラリ（pdcltm32.dll など）と再リンケージすれば指定できます。

YES：

OpenTP1 のサービス名称を XDM/RD E2 に通知します。

OpenTP1 のサービス名称を XDM/RD E2 に通知すると，XDM/RD E2 の統計情報をサービス単位に分析できます。なお，XDM/RD E2 のバージョンが 09-01 以降であることが前提です。

OpenTP1 を使用しない場合，及び OpenTP1 のサービスではない場合（SUP など），YES を指定してもサービス名称は通知されません。

NO：

OpenTP1 のサービス名称の通知はしません。

(127) PDRDCLTCODE={SJIS ｜ UTF-8}

この環境変数は，Windows 版クライアントの場合に有効になります。UNIX 版クライアントの場合は指定しても無効になります。

XDM/RD E2 接続機能使用時に，クライアントで使用する文字コード種別を指定します。

SJIS：

シフト JIS 漢字コードを使用します。

UTF-8：

Unicode（UTF-8）を使用します。UTF-8 を指定する場合，クライアント環境定義 PDCLTCNVMODE には NOUSE を指定するか，又は省略してください。

《UTF-8 指定時の規則》

1. 埋込み変数で扱う入出力データ，及び？パラメタで扱うデータに Unicode（UTF-8）を使用できます。
2. UAP で記述する SQL 文には，ASCII コードだけ指定できます。SQL 文中で ASCII コード以外の文字（漢字，半角片仮名，外字など）を指定する場合，PREPARE 文又は EXECUTE IMMEDIATE 文を使用して，埋込み変数で SQL 文を指定してください。
3. XDM/RD E2 から返却される，SQL 連絡領域に格納されるエラーメッセージ，列名記述領域に格納される列名，型名記述領域に格納されるデータ型名など，Unicode（UTF-8）となります。このため，これらの値に ASCII コード以外の文字が含まれている場合，シフト JIS 漢字コードとして出力すると，正しく表示されないことがあります。
4. XDM/RD E2 側で，文字コードを Unicode（UTF-8）から EBCDIK コード若しくは KEIS コード，又は EBCDIK コード若しくは KEIS コードから Unicode（UTF-8）に変換する場合，データの長さが変化することがあります。このため，埋込み変数の定義長などに注意してください。

(128) PDCNSTRNTNAME={LEADING ｜ TRAILING}

参照制約，及び検査制約を定義する場合，制約名定義の位置を指定します。

LEADING：

制約名定義を，制約定義の前に指定します。

TRAILING：

制約名定義を，制約定義の後に指定します。

《システム定義との関係》

省略した場合は，システム定義の pd_constraint_name オペランドの値が仮定されます。

(129) PDTMPTBLRDAREA=RD エリア名［，RD エリア名...］

～<文字列>((最大 3299 バイト))

使用する一時表及び一時インデクスを格納する一時表用 RD エリア名の候補を指定します。

一時表のデータ有効期間外に INSERT 文を実行した場合，この環境変数に指定した RD エリアの中からデータ格納先の一時表用 RD エリアを決定します。この環境変数を省略した場合，HiRDB が自動的にデータ格納先の一時表用 RD エリアを決定します。一時表の格納先 RD エリアの決定規則については，マニュアル「HiRDB システム導入・設計ガイド」を参照してください。

《指定値の規則》

1. 指定値には，最大 100 個の RD エリア名を指定できます。
2. RD エリア名は，重複して指定できません。

(130) PDBESCONHOLD={YES ｜ NO}

この環境変数は，HiRDB/パラレルサーバの場合に指定できます。

バックエンドサーバ接続保持機能を使用するかどうかを指定します。バックエンドサーバ接続保持機能については，マニュアル「HiRDB システム運用ガイド」を参照してください。

YES：

バックエンドサーバ接続保持機能を使用します。

NO：

バックエンドサーバ接続保持機能を使用しません。

《システム定義との関係》

省略した場合は，システム定義の pd_bes_connection_hold オペランドの値が仮定されます。

(131) PDBESCONHTI=バックエンドサーバ接続保持期間

～＜符号なし整数＞((0～3600))（単位：秒）

バックエンドサーバ接続保持機能を使用する場合，バックエンドサーバ接続保持期間を指定します。

バックエンドサーバ接続保持機能を使用した場合，バックエンドサーバはトランザクション終了後，次のトランザクションが実行されるまでの時間を監視します。次のトランザクションが実行されるまでの時間が PDBESCONHTI の指定値の範囲内であれば，バックエンドサーバ接続保持機能を続行します。
PDBESCONHTI の指定値を超えている場合は，トランザクション終了後にフロントエンドサーバとの接続を切り離します。

《留意事項》

1. 0 を指定した場合，時間監視はしません。SQL の DISCONNECT（XA ライブラリ使用時は xa_close），クライアント環境定義 PDCWAITTIME のタイムオーバなどで，フロントエンドサーバとクライアントの接続が切り離された場合にだけ，フロントエンドサーバとバックエンドサーバの接続を切り離します。
2. PDBESCONHTI は，PDBESCONHOLD に YES を指定している場合に有効になります。

(132) PDODBSTATCACHE={0 | 1}

ODBC 関数の SQLColumns()関数，SQLStatistics()関数で，1 度発行して取得したカラム情報，又はインデクス情報をキャッシュするかどうかを指定します。

0：

キャッシュしません。

SQLColumns()関数，SQLStatistics()関数を呼び出すごとにサーバにアクセスして，カラム情報，又はインデクス情報を取得します。

1：

１度取得したカラム情報，及びインデクス情報をキャッシュします。

ただし，サーバと接続中のときはキャッシュのリフレッシュはしません。したがって，接続中にテーブル定義を変更した場合は，実際の定義と異なるカラム情報，又はインデクス情報を返すことになるため，1 度サーバとの接続を切断する必要があります。

《利点》

SQLColumns()関数，SQLStatistics()関数が同一のパラメタで呼び出された場合は，キャッシュ上の検索結果を AP に返すことで，サーバとの通信回数を削減できます。

《留意事項》

このオプションの指定が効果的かどうかを判断するには，ODBC のトレースを取得し同一接続中に同じパラメタの SQLColumns()関数，SQLStatistics()関数が発行されているかどうかを調査する必要があります。

キャッシュできる行数を次に示します。

SQLColumns()：

約 60000/(50 ＋表オーナ名長＋表名長＋列名長＋コメント長)行数文

SQLStatistics()：

約 60000/(50 ＋表オーナ名長＋表名長＋インデクス名長＋列名長)行数文

(133) PDODBESCAPE={0 | 1}

カタログ系の ODBC 関数(SQLTables()，SQLColumns()など)の検索で，パターン文字に対して ESCAPE 文字('&')を指定するかどうかを指定します。

0：パターン文字に対して ESCAPE 文字を('&')を指定しません。

1：パターン文字に対して ESCAPE 文字を('&')を指定します。

《留意事項》

1. ディクショナリ表の列属性が CHAR 型(データベース初期設定ユティリティで dictionary datatype mchar nouse を指定)で，かつ表名，列名にコード'0x26'を含む 2 バイト文字を使用している場合，このオプションには 0 を指定してください。1 を指定して ODBC 経由でアクセスした場合，特定の表，列が認識されないことがあります。

2. 表名などの識別子にアンダスコア(_)を使用している場合，このオプションには，1 を指定してください。0 を指定した場合，一部の ODBC 対応ソフトからアンダスコア(_)を使用した識別子にアクセスできないことがあります。

(134) PDGDATAOPT={YES | NO}

ODBC 関数の SQLGetData 関数を使用してデータを取り出す場合，データ取得が完了した列に対して繰り返しデータを取得する場合に指定します。

通常，データ取得が完了した列に対して繰り返しデータを取得すると，戻り値として SQL_NO_DATA が返ります。

YES：

SQLGetData 関数でデータ取得が完了した列に対して，繰り返しデータを取得できます。

NO：

SQLGetData 関数でデータ取得が完了した列に対して，繰り返しデータ取得をすると，戻り値として SQL_NO_DATA が返ります。

《適用基準》

同じ列に対して複数回データ取得をしたい場合に指定します。例えば，複数回のデータ取得で SQL_SUCCESS を期待する上位 AP などを使用する場合など，この環境変数を指定してください。

《留意事項》

Internet Banking Server を使用している場合，HiRDB クライアント側の HiRDB.ini に PDGDATAOPT=YES を設定してください。設定しないと，Internet Banking Server の顧客情報管理ユティリティ，及び取引履歴管理ユティリティを使用して HiRDB へログインした後，「顧客登録」「顧客情報の更新」「顧客情報の参照」などの機能選択ボタンを押した後，［戻る］ボタン以外の画面操作ができなくなることがあります。

(135) PDODBLOCATOR={YES | NO}

DB アクセス部品を使用して，BLOB 型，又は BINARY 型の列を検索する場合に，位置付け子機能を使用してデータを分割取得するかどうかを指定します。DB アクセス部品とは，ODBC ドライバ，OLE DB プロバイダ，及び HiRDB データプロバイダ for .NET Framework を示します。

YES：

DB アクセス部品を使用して，BLOB 型，又は BINARY 型の列を検索する場合に，位置付け子機能を使用してデータを分割取得します。

ただし，ODBC ドライバ，OLE DB プロバイダを使用して，出力パラメタに BLOB 型又は BINARY 型を指定した call 文を実行する場合は，YES を指定できません。

NO：

DB アクセス部品を使用して，BLOB 型，又は BINARY 型の列を検索する場合に，位置付け子機能を使用しません。

《適用基準》

NO を指定した場合（省略時仮定値の場合），DB アクセス部品側が列の定義長分のデータ受信領域を確保します。また，HiRDB クライアント側でも列の定義長分のデータ受信領域を必要とします。

列の定義長が大きいと，実行時にメモリ不足になったりすることがあるため，メモリ不足になることを回避する場合は YES を指定してください。なお，YES を指定すると，HiRDB サーバとの通信回数が分割取得回数分増えます。

(136) PDODBSPLITSIZE=分割取得サイズ

～＜符号なし整数＞((4～2097152))《100》（単位：キロバイト）

PDODBLOCATOR=YES を指定している場合に，分割取得のサイズを指定します。

《指定値の目安》

実データ長の分布を考慮して，分割取得回数が少なくなり，かつメモリ不足が発生しない値を指定してください。

(137) PDODBCWRNSKIP={YES｜NO}

ODBC，OLE DB 接続時のワーニングを回避するかどうかを指定します。なお，ODBC，OLE DB 接続以外の場合，この環境変数を指定しても無効になります。

YES：

ODBC ドライバ使用時に SQLFetch()の延長で SQLWARN が設定された場合でも，ODBC ドライバで SQLFetch()の戻り値として SQL_SUCCESS を返却します。

OLE DB プロバイダ使用時に SQLWARN が設定された場合でも，エラーオブジェクトの生成をスキップします。

NO：

ODBC ドライバ使用時に SQLFetch()の延長で SQLWARN が設定された場合，ODBC ドライバで SQLFetch()の戻り値として SQL_SUCCESS_WITH_INFO を返却します。

OLE DB プロバイダ使用時に SQLWARN が設定された場合，エラーオブジェクトを生成します。

《適用基準》

ODBC ドライバでは，検索処理で HiRDB の SQL 連絡領域の SQLWARN が設定された場合，SQLFetch()の戻り値として SQL_SUCCESS_WITH_INFO を返却します。しかし，ODBC ドライバを呼び出す上位アプリケーション※によっては，SQL_SUCCESS_WITH_INFO で検索処理を打ち切るものがあります。この環境変数に YES を指定すると，検索処理で SQL 連絡領域の SQLWARN が設定された場合でも，SQLFetch()の戻り値を SQL_SUCCESS とすることで，検索処理が続行できるようになります。

OLE DB プロバイダでは，SQL 実行時に HiRDB の SQL 連絡領域の SQLWARN が設定された場合，エラーオブジェクトを生成します。しかし，OLE DB プロバイダを呼び出す上位アプリケーションによっては，エラーオブジェクトの生成に伴い例外が発生するときがあります。この機能を適用すれば，SQL 実行時に警告が発生した場合でもエラーオブジェクトの生成をスキップすることで，上位アプリケーションによる例外の発生を回避できるようになります。

注※

例えば，ADO.NET を使用して ODBC 経由で HiRDB に接続している場合，
SQL_SUCCESS_WITH_INFO で検索処理が打ち切られることがあります。

(138) PDJETCOMPATIBLE={YES | NO}

ODBC3.5 ドライバを，ODBC3.5 の規格ではなく Microsoft Access 互換モードで動作させるかどうかを指定します。

YES：

ODBC3.5 ドライバは，Microsoft Access 互換モードで動作します。

NO：

ODBC3.5 ドライバは，ODBC3.5 の規格どおりに動作します。

《適用基準》

Microsoft Access を使用して HiRDB にアクセスする場合に指定します。この環境変数を指定しないと，検索結果が"#Delete"と表示されたり，挿入したデータが不正に変換されることがあります。なお，Microsoft 社が提供するほかの製品やインタフェースでこのような現象が発生した場合，この環境変数を指定することで回避できることがあります。

(139) PDODBGINFOSUPPRESS={YES | NO}

この環境変数は，HiRDB ODBC3.5 ドライバが ODBC 関数の SQLGetInfoW()関数で ODBC 規格に準拠した情報を返却するかどうかを指定します。ODBC 規格に準拠した情報を返却する HiRDB ODBC3.5 ドライバのバージョンを次に示します。

- 07-02-65 以降
- 07-03-47 以降
- 08-01-45 以降
- 08-02-41 以降
- 08-03-26 以降
- 08-04-43 以降
- 08-05-30 以降
- 09-00 以降

上記より前のバージョンの HiRDB ODBC3.5 ドライバを，上記のバージョンにバージョンアップした場合，ADO 及び ADO.NET のインターフェースから ODBC3.5 ドライバを経由した SQL の実行でエラーが発生するおそれがありますが，この環境変数に YES を指定することでエラーを回避できることがあります。発生するエラーの詳細については，「ADO 及び ADO.NET から ODBC を経由して HiRDB サーバにアクセスする場合の留意事項」を参照してください。

通常は，この環境変数を省略するか，又は NO を指定してください。

YES：

HiRDB ODBC3.5 ドライバが SQLGetInfoW()関数で返却する情報を制限します。SQLGetInfoW()関数の第 3 引数 InfoValuePtr に NULL が指定された場合，第 5 引数 StringLengthPtr がポイントする領域に 0 を設定します。

NO：

HiRDB ODBC3.5 ドライバが SQLGetInfoW()関数で返却する情報を制限しないで，ODBC 規格に準拠した情報を返却します。SQLGetInfoW()関数の第 3 引数 InfoValuePtr に NULL が指定された場合，第 5 引数 StringLengthPtr がポイントする領域に取得対象となる情報の長さを設定します。

(140) PDODBSTANDARDARGSIZE= {YES | NO}

64 ビットモードで動作する ODBC3.5 ドライバについて，一部の ODBC 関数の引数に指定されたポインタが指すバッファを 4 バイト整数として扱うか，8 バイト整数として扱うかを指定します。

YES：

ODBC3.5 ドライバの一部の ODBC 関数について，引数に指定されたポインタが指すバッファを 8 バイト整数として扱います（ODBC 規格に準拠）。

NO：

ODBC3.5 ドライバの一部の ODBC 関数について，引数に指定されたポインタが指すバッファを 4 バイト整数として扱います（HiRDB 独自）。

.NET Framework 4 の.NET Framework Data Provider for ODBC から 64 ビットモードで動作する ODBC3.5 ドライバを経由して HiRDB にアクセスする場合，この環境変数に YES を指定してください。

この環境変数の指定値と，8 バイト整数として扱う ODBC 関数の引数の関係を次の表に示します。「パラメタの名称」列のパラメタに「パラメタの属性」列のパラメタのどれかを指定した場合，「8 バイト整数として扱う引数の名称」列の引数に指定したポインタが指すバッファが 8 バイト整数として扱われます。

表 6–44 環境変数 PDODBSTANDARDARGSIZE の指定値と 8 バイト整数として扱う ODBC 関数の引数の関係

ODBC 関数名	8 バイト整数として扱う引数の名称	パラメタの名称	パラメタの属性	
			環境変数の指定値が PDODBSTANDARDARGSIZE=NO 又は指定を省略した場合	環境変数の指定値が PDODBSTANDARDARGSIZE=YES の場合
SQLColAttribute	NumericAttribute	FieldIdentifier	SQL_DESC_DISPLAY_SIZE SQL_DESC_LENGTH SQL_DESC_OCTET_LENGTH	SQL_DESC_DISPLAY_SIZE SQL_DESC_LENGTH SQL_DESC_OCTET_LENGTH SQL_DESC_AUTO_UNIQUE_VALUE SQL_DESC_CASE_SENSITIVE SQL_DESC_CONCISE_TYPE SQL_DESC_FIXED_PREC_SCALE SQL_DESC_NULLABLE

ODBC 関数名	8 バイト整数として扱う引数の名称	パラメタの名称	パラメタの属性	
			環境変数の指定値が PDODBSTANDARDARGSIZE=NO 又は指定を省略した場合	環境変数の指定値が PDODBSTANDARDARGSIZE=YES の場合
				SQL_DESC_NUM_PREC_RADIX SQL_DESC_PRECISION SQL_DESC_SCALE SQL_DESC_SEARCHABLE SQL_DESC_TYPE SQL_DESC_UNNAMED SQL_DESC_UNSIGNED SQL_DESC_UPDATABLE
SQLGetDescField	ValuePtr	FieldIdentifier	SQL_DESC_ARRAY_SIZE SQL_DESC_ARRAY_STATUS_PTR SQL_DESC_BIND_OFFSET_PTR SQL_DESC_DATA_PTR SQL_DESC_INDICATOR_PTR SQL_DESC_OCTET_LENGTH_PTR SQL_DESC_ROWS_PROCESSED_PTR	SQL_DESC_ARRAY_SIZE SQL_DESC_ARRAY_STATUS_PTR SQL_DESC_BIND_OFFSET_PTR SQL_DESC_DATA_PTR SQL_DESC_DISPLAY_SIZE SQL_DESC_INDICATOR_PTR SQL_DESC_LENGTH SQL_DESC_OCTET_LENGTH SQL_DESC_OCTET_LENGTH_PTR SQL_DESC_ROWS_PROCESSED_PTR
SQLGetDiagField	DiagInfoPtr	DiagIdentifier	SQL_DIAG_CURSOR_ROW_COUNT SQL_DIAG_ROW_COUNT SQL_DIAG_ROW_NUMBER	SQL_DIAG_CURSOR_ROW_COUNT SQL_DIAG_ROW_COUNT SQL_DIAG_ROW_NUMBER
SQLGetInfo	InfoValuePtr	InfoType	SQL_DRIVER_HENV SQL_DRIVER_HDBC SQL_DRIVER_HLIB SQL_DRIVER_HSTMT SQL_DRIVER_HDESC	SQL_DRIVER_HENV SQL_DRIVER_HDBC SQL_DRIVER_HLIB SQL_DRIVER_HSTMT SQL_DRIVER_HDESC
SQLGetStmtAttr	ValuePtr	Attribute	SQL_ATTR_APP_PARAM_DESC SQL_ATTR_APP_ROW_DESC SQL_ATTR_IMP_PARAM_DESC SQL_ATTR_IMP_ROW_DESC SQL_ATTR_MAX_LENGTH SQL_ATTR_MAX_ROWS SQL_ATTR_ROW_ARRAY_SIZE	SQL_ATTR_APP_PARAM_DESC SQL_ATTR_APP_ROW_DESC SQL_ATTR_IMP_PARAM_DESC SQL_ATTR_IMP_ROW_DESC SQL_ATTR_MAX_LENGTH SQL_ATTR_MAX_ROWS SQL_ATTR_ROW_ARRAY_SIZE SQL_ATTR_ROWS_FETCHED_PTR SQL_ATTR_ASYNC_ENABLE SQL_ATTR_CONCURRENCY

ODBC 関数名	8 バイト整数として扱う引数の名称	パラメタの名称	パラメタの属性	
			環境変数の指定値が PDODBSTANDARDARGSIZE=NO 又は指定を省略した場合	環境変数の指定値が PDODBSTANDARDARGSIZE=YES の場合
			SQL_ATTR_ROWS_FETCHED_PTR SQL_ATTR_PARAM_STATUS_PTR SQL_ATTR_PARAMS_PROCESSED_PTR SQL_ATTR_PARAMSET_SIZE SQL_ATTR_RETRIEVE_DATA SQL_ATTR_ROW_OPERATION_PTR SQL_ATTR_ROW_STATUS_PTR	SQL_ATTR_CURSOR_SCROLLABLE SQL_ATTR_CURSOR_SENSITIVITY SQL_ATTR_CURSOR_TYPE SQL_ATTR_ENABLE_AUTO_IPD SQL_ATTR_METADATA_ID SQL_ATTR_NOSCAN SQL_ATTR_PARAM_OPERATION_PTR SQL_ATTR_PARAM_STATUS_PTR SQL_ATTR_PARAMS_PROCESSED_PTR SQL_ATTR_PARAMSET_SIZE SQL_ATTR_RETRIEVE_DATA SQL_ATTR_ROW_OPERATION_PTR SQL_ATTR_ROW_STATUS_PTR

(141) PDODBSTANDARDSQLSTATE={YES | NO}

HiRDB サーバでの SQL エラー発生時，ODBC3.5 ドライバから ODBC 規格に準拠した SQLSTATE を返却するかどうかを指定します。

YES：

ODBC3.5 ドライバから ODBC 規格に準拠した SQLSTATE が返却されます。

NO：

ODBC3.5 ドライバから HiRDB 独自の SQLSTATE が返却されます。

この環境変数に YES を指定すると，ODBC3.5 ドライバは，HiRDB サーバから返却された SQLSTATE について，対応する ODBC 規格の SQLSTATE にマッピングします。このため，この機能を使用する場合，HiRDB サーバで SQLSTATE の値の詳細を出力する設定が必要です。

SQLSTATE の値の詳細を出力するには，環境変数 PDSTANDARDSQLSTATE に YES を指定するか，又はシステム共通定義の pd_standard_sqlstate の値を Y にしてください。

この環境変数の指定値，システム共通定義の値，及び ODBC3.5 ドライバが返却する SQLSTATE の関係を次の表に示します。

表 6–45　環境変数 PDSTANDARDSQLSTATE，システム共通定義の値，及び ODBC3.5 ドライバが返却する SQLSTATE の関係

環境変数 PDSTANDARDSQLSTATE の指定値	システム共通定義 pd_standard_sqlstate の値	ODBC3.5 ドライバが返却する SQLSTATE	
		環境変数の指定値が PDODBSTANDARDSQLSTATE=YES の場合	環境変数の指定値が PDODBSTANDARDSQLSTATE=NO 又は指定を省略した場合
YES	−	ODBC 準拠	HiRDB 独自
NO	−	HiRDB 独自	
省略	Y	ODBC 準拠	
	N	HiRDB 独自	

(凡例)

−：pd_standard_sqlstate の値は返却される SQLSTATE に影響しません。

HiRDB 独自：HiRDB 独自の SQLSTATE が返却されます（ODBC 規格に非準拠）。

ODBC 準拠：ODBC 規格に準拠した SQLSTATE が返却されます。

ODBC3.5 ドライバを呼び出す他製品を利用している場合，この環境変数に YES を指定してください。SQLSTATE については，「ODBC3.5 ドライバが返却する SQLSTATE」を参照してください。

(142)　PDODBSTANDARDDESCCOL={YES | NO}

ODBC3.5 ドライバの SQLDescribeCol 関数の検索対象列が数データ型の場合，返却する列のサイズとして，ODBC 規格に準拠した値を返却するかどうかを指定します。

YES：

SQLDescribeCol 関数の検索対象列が数データ型の場合，ColumnSizePtr に指定されたポインタが指すバッファに対して，ODBC 規格に準拠した列のサイズ（数データの最大けた数）を返却します。

NO：

SQLDescribeCol 関数の検索対象列が数データ型の場合，ColumnSizePtr に指定されたポインタが指すバッファに対して，HiRDB 独自の列のサイズ（データ型が SQL_DECIMAL の場合は数データの精度，SQL_DECIMAL 以外の場合は数データのバイト数）を返却します。

検索した数データについて，ODBC3.5 ドライバを呼び出す上位アプリケーションが文字列形式で扱う場合，この環境変数に YES を指定してください。通常の運用では，この環境変数には NO を指定するか，指定を省略してください。

この環境変数の指定値と，SQLDescribeCol 関数で返却する数データ型の列のサイズの関係を次の表に示します。

表 6–46　環境変数 PDODBSTANDARDDESCCOL の指定値と SQLDescribeCol 関数で返却する数データ型の列のサイズの関係

ODBC のデータ型	HiRDB のデータ型	SQLDescribeCol 関数で返却する列のサイズ	
		環境変数の指定値が PDODBSTANDARDDESCCOL=NO 又は指定を省略した場合	環境変数の指定値が PDODBSTANDARDDESCCOL=YES の場合
SQL_DECIMAL	DECIMAL，NUMERIC	精度（全体のけた数）	
SQL_SMALLINT	SMALLINT	2	5
SQL_INTEGER	INTEGER	4	10
SQL_REAL	SMALLFLT，REAL	4	7
SQL_FLOAT	FLOAT，DOUBLE PRECISION	8	15
SQL_DOUBLE	FLOAT，DOUBLE PRECISION	8	15

(143)　PDODBSTANDARDGTYPEINFO=｛YES ｜ NO｝

ODBC3.5 ドライバの SQLGetTypeInfo 関数で，ODBC 規格に準拠した値を返却するかどうかを指定します。

DataStage を使用して HiRDB ODBC3.5 ドライバを使用する場合は，このクライアント環境定義に YES を指定してください。

YES：

SQLGetTypeInfo 関数の第 2 引数に指定されたデータ型に対して，ODBC 規格に準拠した結果セットを返却します。

NO：

SQLGetTypeInfo 関数の第 2 引数に指定されたデータ型に対して，HiRDB 独自の結果セットを返却します。

この環境定義の指定値と，SQLGetTypeInfo 関数で返却する戻り値，結果セットの関係を次の表に示します。

表 6–47　環境変数 PDODBSTANDARDGTYPEINFO の指定値と SQLGetTypeInfo 関数で返却する戻り値の関係

第 2 引数への指定値	SQLGetTypeInfo 関数で返却する戻り値	
	環境変数の指定値が PDODBSTANDARDGTYPEINFO=NO の場合	環境変数の指定値が PDODBSTANDARDGTYPEINFO=YES の場合
SQL_GUID SQL_INTERVAL_DAY SQL_INTERVAL_DAY_TO_HOUR SQL_INTERVAL_DAY_TO_MINUTE SQL_INTERVAL_DAY_TO_SECOND SQL_INTERVAL_HOUR SQL_INTERVAL_HOUR_TO_MINUTE SQL_INTERVAL_HOUR_TO_SECOND SQL_INTERVAL_MINUTE SQL_INTERVAL_MINUTE_TO_SECOND SQL_INTERVAL_MONTH SQL_INTERVAL_SECOND SQL_INTERVAL_YEAR SQL_INTERVAL_YEAR_TO_MONTH	SQL_ERROR	SQL_SUCCESS

表 6–48　環境変数 PDODBSTANDARDGTYPEINFO の指定値と SQLGetTypeInfo 関数で返却する結果セットの関係

第 2 引数への指定値	SQLGetTypeInfo 関数で返却する結果セット	結果セットの差異内容	
		環境変数の指定値が PDODBSTANDARDGTYPEINFO=NO の場合	環境変数の指定値が PDODBSTANDARDGTYPEINFO =YES の場合
SQL_ALL_TYPE	DATA_TYPE 列に次のデータ型を含むかどうか • SQL_DOUBLE • SQL_NUMERIC • SQL_LONGVARCHAR • SQL_INTERVAL_HOUR_TO_SECOND • SQL_WCHAR • SQL_WVARCHAR • SQL_WLONGVARCHAR	含まない	含む
	TYPE_NAME 列が BINARY の DATE_TYPE 列と SQL_DATETIME_SUB 列の値	SQL_BINARY	SQL_LONGVARBINARY
	TYPE_NAME 列が DATE の DATA_TYPE 列の値	SQL_DATE	SQL_TYPE_DATE

第 2 引数への指定値	SQLGetTypeInfo 関数で返却する結果セット	結果セットの差異内容	
		環境変数の指定値が PDODBSTANDARDGTYPEINFO =NO の場合	環境変数の指定値が PDODBSTANDARDGTYPEINFO =YES の場合
	TYPE_NAME 列が DATE の SQL_DATETIME_SUB 列の値	SQL_TYPE_DATE	SQL_CODE_DATE
	TYPE_NAME 列が TIME の DATA_TYPE 列の値	SQL_TIME	SQL_TYPE_TIME
	TYPE_NAME 列が TIME の SQL_DATETIME_SUB 列の値	SQL_TYPE_TIME	SQL_CODE_TIME
	TYPE_NAME 列が TIMESTAMP の DATA_TYPE 列の値	SQL_TIMESTAMP	SQL_TYPE_TIMESTAMP
	TYPE_NAME 列が TIMESTAMP の SQL_DATETIME_SUB 列の値	SQL_TYPE_TIMESTAMP	SQL_CODE_TIMESTAMP
SQL_DOUBLE SQL_NUMERIC SQL_LONGVARCHAR SQL_INTERVAL_HOUR_TO_SECOND SQL_WCHAR SQL_WVARCHAR SQL_WLONGVARCHAR	結果セットの返却	返却する	返却しない
SQL_BINARY	結果セットの返却	返却する	返却しない
SQL_LONGVARBINARY	DATA_TYPE 列に次のデータ型を含むかどうか • BLOB • BINARY	BLOB だけ含む	含む
SQL_DATE SQL_TYPE_DATE	DATA_TYPE 列の値	SQL_DATE	SQL_TYPE_DATE
	SQL_DATETIME_SUB 列の値	SQL_TYPE_DATE	SQL_CODE_DATE
SQL_TIME SQL_TYPE_TIME	DATA_TYPE 列の値	SQL_TIME	SQL_TYPE_TIME
	SQL_DATETIME_SUB 列の値	SQL_TYPE_TIME	SQL_CODE_TIME
SQL_TIMESTAMP SQL_TYPE_TIMESTAMP	DATA_TYPE 列の値	SQL_TIMESTAMP	SQL_TYPE_TIMESTAMP
	SQL_DATETIME_SUB 列の値	SQL_TYPE_TIMESTAMP	SQL_CODE_TIMESTAMP

(144) PDODBERRTRC = {YES | NO}

ODBC3.5 ドライバ内でエラーを検知した場合に，ODBC エラーログを出力するかどうかを指定します。

ODBC エラーログの詳細については，「ODBC エラーログ」を参照してください。

YES：

ODBC エラーログを出力します。

NO：

ODBC エラーログを出力しません。

《ほかの環境変数との関係》

ODBC エラーログは，PDCLTPATH で指定したディレクトリに出力されます。PDCLTPATH の指定がない場合，UAP を起動したときのカレントディレクトリに出力されます。

ODBC エラーログの最大ファイルサイズはクライアント環境定義 PDODBCERRTRCSIZE で指定できます。

(145) PDODBERRTRCSIZE ＝ {ODBC エラーログファイルのサイズ}

～<符号なし整数>((65535～2000000000))《65535》（単位：バイト）

PDODBERRTRC=YES を指定している場合に ODBC エラーログの最大ファイルサイズを指定します。指定値のサイズを超えると出力先が切り替わります。

(146) PDODBSYSUNSIGNED= {YES ｜ NO}

ODBC3.5 ドライバが提供するシステム情報取得関数で返却される結果セットの数値型を返す列の，列が符号なしかどうかを取得する SQLColAttribute（SQL_DESC_UNSIGNED）関数発行時，返却値を変更したい場合に指定します。

YES：

ODBC3.5 ドライバは，システム情報取得関数で返却される結果セットの数値型を返す列の SQLColAttribute（SQL_DESC_UNSIGNED）関数発行時に SQL_FALSE を返却します。

NO：

ODBC3.5 ドライバは，システム情報取得関数で返却される結果セットの数値型を返す列の SQLColAttribute（SQL_DESC_UNSIGNED）関数発行時に SQL_TRUE を返却します。

Microsoft の Excel や Power BI を使用し，ODBC3.5 ドライバを経由して HiRDB にアクセスしエラーが発生する場合，このクライアント環境定義に YES を指定してください。エラー（例えば「Int32 型の値が大き過ぎる，又は小さ過ぎます。」等）を回避できることがあります。

このクライアント環境定義の指定値のシステム情報取得関数が生成する結果セットの属性値の差異を次の表に示します。

表 6–49　PDODBSYSUNSIGNED の指定値の戻り値，結果セットの差異一覧

カタログ関数名	結果セット列名	クライアント環境定義 PDODBSYSUNSIGNED	
		YES	NO or 省略
SQLColumns	TABLE_CAT	SQL_TRUE	SQL_TRUE

カタログ関数名	結果セット列名	クライアント環境定義 PDODBSYSUNSIGNED	
		YES	NO or 省略
	TABLE_SCHEMA	SQL_TRUE	SQL_TRUE
	TABLE_NAME	SQL_TRUE	SQL_TRUE
	COLUMN_NAME	SQL_TRUE	SQL_TRUE
	DATA_TYPE	SQL_FALSE	SQL_TRUE
	TYPE_NAME	SQL_TRUE	SQL_TRUE
	COLUMN_SIZE	SQL_FALSE	SQL_TRUE
	BUFFER_LENGTH	SQL_FALSE	SQL_TRUE
	DECIMAL_DIGITS	SQL_FALSE	SQL_TRUE
	NUM_PREC_RADIX	SQL_FALSE	SQL_TRUE
	NULLABLE	SQL_FALSE	SQL_TRUE
	REMARKS	SQL_TRUE	SQL_TRUE
	COLUMN_DEF	SQL_TRUE	SQL_TRUE
	SQL_DATA_TYPE	SQL_FALSE	SQL_TRUE
	SQL_DATETIME	SQL_FALSE	SQL_TRUE
	CHAR_OCTET_LENGTH	SQL_FALSE	SQL_TRUE
	ORDINAL_POSITION	SQL_FALSE	SQL_TRUE
	IS_NULLABLE	SQL_TRUE	SQL_TRUE
SQLForeignKeys	PKTABLE_CAT	SQL_TRUE	SQL_TRUE
	PKTABLE_SCHEM	SQL_TRUE	SQL_TRUE
	PKTABLE_NAME	SQL_TRUE	SQL_TRUE
	PKCOLUMN_NAME	SQL_TRUE	SQL_TRUE
	FKTABLE_CAT	SQL_TRUE	SQL_TRUE
	FKTABLE_SCHEM	SQL_TRUE	SQL_TRUE
	FKTABLE_NAME	SQL_TRUE	SQL_TRUE
	FKCOLUMN_NAME	SQL_TRUE	SQL_TRUE
	KEY_SEQ	SQL_FALSE	SQL_TRUE
	UPDATE_RULE	SQL_FALSE	SQL_TRUE
	DELETE_RULE	SQL_FALSE	SQL_TRUE
	FK_NAME	SQL_TRUE	SQL_TRUE

カタログ関数名	結果セット列名	クライアント環境定義 PDODBSYSUNSIGNED	
		YES	NO or 省略
	PK_NAME	SQL_TRUE	SQL_TRUE
	DEFERRABILITY※	SQL_FALSE	SQL_FALSE
SQLPrimaryKey	TABLE_CAT	SQL_TRUE	SQL_TRUE
	TABLE_SCHEM	SQL_TRUE	SQL_TRUE
	TABLE_NAME	SQL_TRUE	SQL_TRUE
	COLUMN_NAME	SQL_TRUE	SQL_TRUE
	KEY_SEQ	SQL_FALSE	SQL_TRUE
	PK_NAME	SQL_TRUE	SQL_TRUE
SQLProcedures	PROCEDURE_CAT	SQL_TRUE	SQL_TRUE
	PROCEDURE_SCHEM	SQL_TRUE	SQL_TRUE
	PROCEDURE_NAME	SQL_FALSE	SQL_TRUE
	NUM_INPUT_PARAMS	SQL_TRUE	SQL_TRUE
	NUM_OUTPUT_PARAMS	SQL_TRUE	SQL_TRUE
	NUM_RESULT_SETS	SQL_TRUE	SQL_TRUE
	RERMARKS	SQL_TRUE	SQL_TRUE
	PROCEDURE_TYPE	SQL_TRUE	SQL_TRUE
SQLProcedureColumns	PROCEDURE_CAT	SQL_TRUE	SQL_TRUE
	PROCEDURE_SCHEM	SQL_TRUE	SQL_TRUE
	PROCEDURE_NAME	SQL_TRUE	SQL_TRUE
	COLUMN_NAME	SQL_TRUE	SQL_TRUE
	COLUMN_TYPE	SQL_FALSE	SQL_TRUE
	DATA_TYPE	SQL_FALSE	SQL_TRUE
	TYPE_NAME	SQL_TRUE	SQL_TRUE
	COLUMN_SIZE	SQL_FALSE	SQL_TRUE
	BUFFER_LENGTH	SQL_FALSE	SQL_TRUE
	DECIMAL_DIGITS	SQL_FALSE	SQL_TRUE
	NUM_PREC_RADIX	SQL_FALSE	SQL_TRUE
	NULLABLE	SQL_FALSE	SQL_TRUE
	REMARKS	SQL_TRUE	SQL_TRUE

カタログ関数名	結果セット列名	クライアント環境定義 PDODBSYSUNSIGNED	
		YES	NO or 省略
	COLUMN_DEF	SQL_TRUE	SQL_TRUE
	SQL_DATA_TYPE	SQL_FALSE	SQL_TRUE
	SQL_DATETIME_SUB	SQL_FALSE	SQL_TRUE
	CHAR_OCTET_LENGTH	SQL_FALSE	SQL_TRUE
	ORDINAL_POSITION	SQL_FALSE	SQL_TRUE
	IS_NULLABLE	SQL_TRUE	SQL_TRUE
SQLSpecialColumns	SCOPE	SQL_FALSE	SQL_TRUE
	COLUMN_NAME	SQL_TRUE	SQL_TRUE
	DATA_TYPE	SQL_FALSE	SQL_TRUE
	TYPE_NAME	SQL_TRUE	SQL_TRUE
	COLUMN_SIZE	SQL_FALSE	SQL_TRUE
	BUFFER_LENGTH	SQL_FALSE	SQL_TRUE
	DECIMAL_DIGITS	SQL_FALSE	SQL_TRUE
	PSEUDO_COLUMN	SQL_FALSE	SQL_TRUE
SQLStatistics	TABLE_CAT	SQL_TRUE	SQL_TRUE
	TABLE_SCHEM	SQL_TRUE	SQL_TRUE
	TABLE_NAME	SQL_TRUE	SQL_TRUE
	NON_UNIQUE	SQL_FALSE	SQL_TRUE
	INDEX_QUALIFIER	SQL_TRUE	SQL_TRUE
	INDEX_NAME	SQL_TRUE	SQL_TRUE
	TYPE	SQL_FALSE	SQL_TRUE
	ORDINAL_POSITION	SQL_FALSE	SQL_TRUE
	COLUMN_NAME	SQL_TRUE	SQL_TRUE
	ASC_OR_DESC	SQL_TRUE	SQL_TRUE
	CARDINALITY	SQL_FALSE	SQL_TRUE
	PAGES	SQL_FALSE	SQL_TRUE
	FILTER_CONDITION	SQL_TRUE	SQL_TRUE

注※

PDODBSYSKEYS=YES 指定時だけ返却します。

(147) PDODBSYSKEYS=｛YES ｜ NO｝

ODBC3.5 ドライバが提供する SQLPrimaryKeys 関数及び SQLForeignKeys 関数の ODBC 規格で規定された結果セットを返却する場合に指定します。

YES：

ODBC3.5 ドライバは，SQLPrimaryKeys 関数及び SQLForeignKeys 関数の該当するキー又は列名がある場合，1 件以上の結果セットを返します。該当するキー又は列名がない場合，行なしの結果セットを返します。

NO：

ODBC3.5 ドライバは，SQLPrimaryKeys 関数及び SQLForeignKeys 関数の該当するキー又は列名の有無に関係なく，常に行なしの結果セットを返します。

(148) PDPLGIXMK=｛YES ｜ NO｝

プラグインインデクスの遅延一括作成をするかどうかを指定します。プラグインインデクスの遅延一括作成については，マニュアル「HiRDB システム運用ガイド」を参照してください。

YES：

プラグインインデクスの遅延一括作成をします。

NO：

プラグインインデクスの遅延一括作成をしません。

(149) PDPLUGINNSUB

詳細については，各プラグインマニュアルを参照してください。

(150) PDPLGPFSZ=遅延一括作成用のインデクス情報ファイルの初期容量

～〈符号なし整数〉((1～1048574000))《8192》（単位：キロバイト）

プラグインインデクスの遅延一括作成用のインデクス情報ファイルの初期容量を指定します。なお，指定値は 4 の倍数に切り上げられます。

インデクス情報ファイルを HiRDB ファイルシステム領域に作成する場合に，この指定が有効になります。

この環境変数を指定する場合，同時に PDPLGIXMK=YES も指定してください。

(151) PDPLGPFSZEXP ＝遅延一括作成用のインデクス情報ファイルの増分値

～〈符号なし整数〉((1～1048573000))《8192》（単位：キロバイト）

プラグインインデクスの遅延一括作成用のインデクス情報ファイルの増分値を指定します。インデクス情報ファイルが満杯になった場合，ここで指定した値で増分します。なお，指定値は 4 の倍数に切り上げられます。

インデクス情報ファイルを HiRDB ファイルシステム領域に作成する場合に，この指定が有効になります。

この環境変数を指定する場合，同時に PDPLGIXMK=YES も指定してください。

(152)　PDHSICOPTIONS ＝集積モード［，集積モード］...

～〈識別子〉

HSIC を使用する運用の場合に，HSIC で集積する情報を指定します。

集積モードの指定値を次に示します。

ROOTAP：

この指定値は，Type4 JDBC ドライバを使用しているときだけ有効になります。

クライアントからサーバへの電文にルートアプリケーション情報を常に付加する場合に指定します。指定がない場合は，製品付加情報に変更が発生したときだけ電文に付加します。

UAPREP：

PDSQLTRACE が指定されていなくても，PDUAPREPLVL に ut が指定されていた場合，UAP 統計レポート情報をサーバからクライアントへの電文に付加します。

PDUAPREPLVL，PDSQLTRACE の指定値と UAPREP との対応を次の表に示します。

表 6–50　PDUAPREPLVL，PDSQLTRACE の指定値と UAPREP との対応

PDUAPREPLVL=ut	PDSQLTRACE	UAPREP	
		指定なし	指定あり
指定あり	指定あり	○	○
	指定なし	×	
指定なし	指定あり	×	
	指定なし		

（凡例）

○：電文に UAP 統計レポートを付加します。

×：電文に UAP 統計レポートを付加しません。

ALL：

上記のすべての指定値を適用したい場合に指定します。

PDHSICOPTIONS の指定値と HSIC で集積できる情報を次の表に示します。

表 6–51　PDHSICOPTIONS の指定値と HSIC で集積できる情報

PDHSICOPTIONS の指定値	クライアントライブラリ	Type4 JDBC ドライバ
未指定	なし	なし
UAPREP	UAP 統計レポート情報	UAP 統計レポート情報
ROOTAP	指定できません	ルートアプリケーション情報
ALL	UAP 統計レポート情報	UAP 統計レポート情報 および ルートアプリケーション情報

《注意事項》

- HSIC を使用しない場合は，このクライアント環境定義を指定しないでください。
  指定値については，HSIC のマニュアルを参照してください。
- UAPREP 又は ALL を指定している場合，pdclttrc コマンドを使用して SQL トレースを取得しないでください。UAPREP の情報が取得できなくなることがあります。pdclttrc コマンドの詳細は，マニュアル「HiRDB コマンドリファレンス」の「pdclttrc（SQL トレースの動的取得）」を参照してください。
- UAPREP 又は ALL を指定している場合，pdtrcmgr コマンドを使用して SQL トレースを取得するときは，-k オプションに ut を指定してください。pdtrcmgr コマンドの詳細は「SQL トレース動的取得機能」を参照してください。

(153)　PDJDBFILEDIR=Exception トレースログのファイル出力先

〜<パス名>((最大 256 バイト))《PDCLTPATH の指定値》

Type4 JDBC ドライバでの Exception トレースログ及び不正電文トレースのファイル出力先を指定します。ファイル出力先には，ディレクトリの絶対パスを 256 バイトまでで指定してください。この環境変数は，Type4 JDBC ドライバを使用する場合にだけ指定できます。

Exception トレースログの詳細は，「Exception トレースログ」を参照してください。不正電文トレースの詳細は，「不正電文トレース」を参照してください。そのほかの詳細は，「システムプロパティの設定」のシステムプロパティ「HiRDB_for_Java_FileDIR」を参照してください。

(154)　PDJDBFILEOUTNUM=Exception トレースログのファイルへの出力数

〜<符号なし整数>((1〜50))《5》

Type4 JDBC ドライバでの Exception トレースログのファイルへの出力数を指定します。この環境変数は，Type4 JDBC ドライバを使用する場合にだけ指定できます。

クライアント環境定義 PDJDBFILESIZE，又はシステムプロパティ HiRDB_for_Java_FileSize を指定している場合，このクライアント環境定義の値は無視されます。

Exception トレースログの詳細は，「Exception トレースログ」を参照してください。そのほかの詳細は，「システムプロパティの設定」のシステムプロパティ「HiRDB_for_Java_FileOutNUM」を参照してください。

(155)　PDJDBONMEMNUM=Exception トレースログのメモリ内取得情報数

～<符号なし整数>((500～10000))《1000》

Type4 JDBC ドライバでの Exception トレースログのメモリ内取得情報数を指定します。この環境変数は，Type4 JDBC ドライバを使用する場合にだけ指定できます。

Exception トレースログの詳細は，「Exception トレースログ」を参照してください。そのほかの詳細は，「システムプロパティの設定」のシステムプロパティ「HiRDB_for_Java_OnMemNUM」を参照してください。

(156)　PDJDBTRACELEVEL=Exception トレースログのトレース取得レベル

～<符号なし整数>((0～5))《1》

Type4 JDBC ドライバでの Exception トレースログのトレース取得レベルを指定します。0 を指定した場合，Exception トレースログを取得しません。この環境変数は，Type4 JDBC ドライバを使用する場合にだけ指定できます。

Exception トレースログの詳細は，「Exception トレースログ」を参照してください。そのほかの詳細は，「システムプロパティの設定」のシステムプロパティ「HiRDB_for_Java_TraceLevel」を参照してください。

(157)　PDJDBFILESIZE=Exception トレースログの最大ファイルサイズ

～<符号なし整数>((4096～2000000000))

Type4 JDBC ドライバでの Exception トレースログの最大ファイルサイズを指定します。この環境変数は，Type4 JDBC ドライバを使用する場合にだけ指定できます。

Exception トレースログの詳細は，「Exception トレースログ」を参照してください。そのほかの詳細は，「システムプロパティの設定」のシステムプロパティ「HiRDB_for_Java_FileSize」を参照してください。

(158)　PDDNDPCOMPATIBLE ＝ {NO ｜ ALL}

HiRDB データプロバイダ for .NET Framework では，HiRDB 独自の仕様から.NET Framework の規格に準拠した仕様へ変更したメソッドがあります。

このクライアント環境定義には，ADO.NET2.0 に対応した HiRDB データプロバイダ for .NET Framework のメソッドが，.NET Framework の規格に準拠した仕様とするか，又は以前の HiRDB 独自の仕様とするかを指定します。

NO：

メソッドの仕様が.NET Framework の規格に準拠した仕様となります。

ALL：

メソッドの仕様が以前の HiRDB 独自の仕様となります。指定値ごとの HiRDB データプロバイダ for .NET Framework のメソッドの仕様を次の表に示します。

表 6–52　PDDNDPCOMPATIBLE 指定値ごとのメソッドの仕様

対象クラス	対象メソッド	メソッドの仕様	
		環境変数の指定値が PDDNDPCOMPATIBLE =NO の場合	環境変数の指定値が PDDNDPCOMPATIBLE =ALL の場合
HiRDBCommand	ExecuteScalar	結果セットの最初の行の最初の列がナル値の場合，UAP に System.DBNull.Value を返却します。	結果セットの最初の行の最初の列がナル値の場合，次の値を UAP に返却します。 • Microsoft Visual C#の場合：null • Microsoft Visual Basic の場合：Nothing

6.6.5　HiRDB サーバと接続するための環境変数と接続形態との関係

HiRDB サーバと接続するための環境変数と接続形態との関係を次の表に示します。

表 6–53　環境変数と接続形態との関係

環境変数	HiRDB/シングルサーバ		HiRDB/パラレルサーバ				
			シングルフロントエンドサーバ		マルチフロントエンドサーバ		
	通常接続	高速接続	通常接続	高速接続	通常接続	フロントエンドサーバ指定接続	
						FES ホストダイレクト接続	高速接続
PDHOST	○	○	○	○	○	○	○
PDFESHOST	−	−	−	○※	−	○	○※
PDNAMEPORT	○	○	○	○	○	○	○
PDSERVICEPORT	−	○	−	○※	−	−	○※
PDSERVICEGRP	−	○	−	○※	−	○	○※
PDFESGRP	−	−	−	○※	−	−	○※

環境変数	HiRDB/シングルサーバ		HiRDB/パラレルサーバ				
			シングルフロントエンドサーバ		マルチフロントエンドサーバ		
	通常接続	高速接続	通常接続	高速接続	通常接続	フロントエンドサーバ指定接続	
						FES ホストダイレクト接続	高速接続
PDSRVTYPE	−	△	−	△	−	−	△

（凡例）

○：必ず指定します。

△：HiRDB サーバが Linux 版又は Windows 版の場合，指定が必要です。

−：指定する必要はありません。

注 1

高速接続，FES ホストダイレクト接続，通常接続の順に，必ず指定する環境変数がすべて指定されている接続形態が選択されます。このとき，不要な環境変数は使用されません。

注 2

推奨する接続形態を次に示します。

項番	運用形態	推奨する接続形態	備考
1	下記の項番 2，3 以外の場合	高速接続	−
2	マルチフロントエンドサーバを適用した構成で，フロントエンドサーバを効率良く利用したい場合	通常接続	通常接続，及び FES ホストダイレクト接続の場合，UAP がシステムマネジャプロセスやノードマネジャプロセスに対して接続を行います。このため，多数の UAP が同時に接続をした際など，システムマネジャプロセスやノードマネジャプロセスに負荷が掛かり，システム全体の性能に影響を与えるおそれがあります。
3	FES ホストダイレクト接続を適用した構成の場合	高速接続	通常接続から高速接続に変更する場合，表「環境変数と接続形態との関係」で示す環境変数のほかに，スケジューラのポート番号を設定する必要があります。 スケジューラのポート番号の設定については，マニュアル「HiRDB システム導入・設計ガイド」を参照してください。

注 3

HiRDB サーバへの接続時間，及び接続時に使用する TCP ポート数の関係を次に示します。

通常接続＞ FES ホストダイレクト接続＞高速接続

注※

高速接続で接続する場合，次の二つの指定方法があります。

- PDFESHOST，PDSERVICEGRP，及び PDSERVICEPORT で指定する

- PDFESGRP で指定する

PDFESHOST 又は PDFESGRP にホスト名を一つだけ指定する場合は，上記のどちらで指定しても同じ扱いになります。ホスト名を複数指定する場合は，次の表の条件によって指定方法が変わります。

項番	ホスト名を複数指定する場合の条件		指定する環境変数
	系切り替え機能	FES の数	
1	FES のあるユニットで系切り替え機能を使用していて，かつクライアントからの接続で使用する IP アドレスが現用系と予備系で異なる場合	一つ	PDFESHOST，PDSERVICEGRP，及び PDSERVICEPORT
2		複数	PDFESHOST，PDSERVICEGRP，及び PDSERVICEPORT
3	上記以外の場合	複数	PDFESGRP

項番 2 は FES の数が複数（マルチフロントエンドサーバ）ですが，PDFESHOST，PDSERVICEGRP，及び PDSERVICEPORT を指定し，接続先の FES は一つだけにしてください。PDFESGRP で複数の FES を指定すると，実行系で接続ユーザ数オーバー（KFPA11932-E）となったときに次の問題が発生します。

- むだに待機系にも接続を試みる。
- すべての FES グループで接続エラーとなった場合に，最後に接続を試みた FES グループが待機系のときは，待機系で発生したエラー情報だけ UAP に返却されるため，実行系で接続ユーザ数オーバーになったことが検知できない。

6.7　環境変数のグループ登録

クライアントの環境変数を，グループとして登録しておくことができます。環境変数を登録しておくと，接続ごとに環境変数を変更できるようになります。したがって，接続するたびに環境変数を変えるような場合に便利です。

登録先は UNIX 環境の場合は通常ファイル，Windows 環境の場合はレジストリ又はファイルです。登録した環境変数は，HiRDB サーバ接続時に情報を取得します。

なお，OLTP 下の X/Open に従った API を使用した UAP をクライアントとする場合で，かつオープン文字列を指定する場合，「OLTP 下の X/Open に従った API を使用した UAP をクライアントとする場合の指定方法」に従って指定した環境変数より，オープン文字列に指定した環境変数グループの環境変数が優先されます。オープン文字列については，マニュアル「HiRDB システム導入・設計ガイド」を参照してください。

6.7.1　UNIX 環境の場合

環境変数を通常ファイルに登録する場合は，次に示す規則に従ってください。

- １行１環境変数で設定してください。
- 「クライアント環境変数＝指定値」の形式で指定してください。
- コメントは，スラント及びアスタリスク（/*）とアスタリスク及びスラント（*/）で囲んでください。コメントの入れ子はできません。また，コメント中で改行をしないでください。
- 同一の環境変数を複数記述した場合は，最後に記述してある環境変数が有効になります。
- 指定値に空白を含む場合は，引用符（"）で指定値を囲んでください。囲まれていない場合は，空白が指定値から削除されます。
- 1 行目に[HIRDB]又は[HiRDB]を記述できます。
- odbc.ini の INIFLNAME にクライアント環境定義を指定する場合は，1 行目に[HIRDB]又は[HiRDB]を必ず記述してください。記述されていない場合，クライアント環境定義が読み込まれません。
- 指定値に引用符（ " ）で囲む値を含む場合の注意事項については，「クライアント環境定義の一覧」の「注※5」を参照してください。

通常ファイル（/HiRDB_P/Client/HiRDB.ini）への設定例を次に示します。

（例）

```
[HIRDB]
PDCLTPATH=トレースファイル格納ディレクトリ
PDHOST=システムマネジャのホスト名
PDUSER=”認可識別子”/”パスワード”
PDNAMEPORT=システムマネジャプロセスのポート番号
PDCLTAPNAME=実行するUAPの識別名称
```

注意事項

この方法では，Type4 JDBC ドライバを用いた UAP を除いて，PDJDB で始まるクライアント環境定義は有効になりません。

6.7.2 Windows 環境の場合（レジストリ登録）

環境変数をレジストリに登録する場合，HiRDB クライアント環境変数登録ツールを使用します。

HiRDB クライアント環境変数登録ツールを使用する場合，xxxx¥UTL¥pdcltadm.exe を実行します（xxxx は HiRDB サーバの場合は%PDDIR%¥client，HiRDB クライアントの場合は HiRDB クライアントのインストールディレクトリ）。

HiRDB クライアント環境変数登録ツールを実行するユーザの権限によって操作の可否が異なります。ユーザ権限と操作の可否を次の表に示します。

表 6–54　HiRDB クライアント環境変数登録ツールを実行するユーザの権限と操作の可否

項番	操作の対象	操作	HiRDB クライアント環境変数登録ツールを実行するユーザの権限		
			Administrators 権限	PowerUsers 権限	Users 権限
1	ユーザグループ	更新	○	○	○
2		参照	○	○	○
3	システムグループ	更新	○	×	×
4		参照	○	○	○
5	OLE DB プロバイダ用トレース	更新	○	×	×
6		参照	○	○	○

（凡例）

○：操作できます。

×：権限がないため操作できません。

ユーザアカウント制御（UAC）を有効にしたマシンである場合，HiRDB クライアント環境変数登録ツールは，管理者として実行してください。管理者として実行しないと，システムグループに環境変数を登録できません。

HiRDB クライアント環境変数登録ツールで環境変数をレジストリに登録する手順を次に示します。

なお，OLE DB 接続の場合，ユーザ環境変数，及び HIRDB.INI よりも，HiRDB クライアント環境変数登録ツールで登録した環境変数が優先されます。

Type4 JDBC ドライバを使用する場合，この方法で設定したクライアント環境定義は有効になりません。また，PDJDB で始まるクライアント環境定義は，この方法で設定しても有効になりません。

(1)　HiRDB クライアント環境変数登録ツールの起動

xxxx¥UTL¥pdcltadm.exe を実行してください。すると，次の「HiRDB クライアント環境変数登録ツール」画面が表示されます。



[説明]

ユーザグループ：

ユーザごとに，環境変数グループの追加，削除，又は構成ができます。この情報は，HKEY_CURRENT_USER 下に登録されます。

システムグループ：

コンピュータに対して，環境変数グループの追加，削除，又は構成ができます。この情報は，HKEY_LOCAL_MACHINE 下に登録されます。

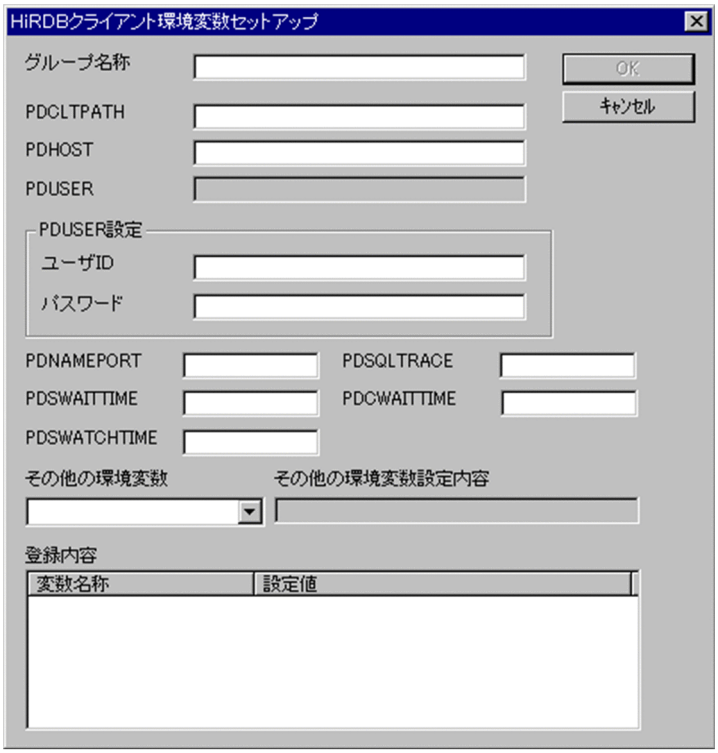
ユーザグループ，又はシステムグループのどちらかを選択したら，［追加］ボタンをクリックしてください。

注意事項：

- OLTP 下の X/Open に従った API を使用した UAP をクライアントとする場合で，オープン文字列に指定する環境変数グループ名称を「HiRDB クライアント環境変数登録ツール」で登録するときは，「システムグループ」を選択してください。オープン文字列については，マニュアル「HiRDB システム導入・設計ガイド」を参照してください。
- Windows 起動時に起動されるサービスによって自動起動される UAP の場合，「ユーザグループ」が読み込めないため，「システムグループ」を選択してください。

(2)　環境変数グループの登録

「HiRDB クライアント環境変数セットアップ」画面が表示されます。



[説明]

グループ名称：

グループ名称を最大 30 バイトで指定します。

各環境変数を指定してください：

各環境変数の内容については，「クライアント環境定義の設定内容」を参照してください。

設定が終了したら，[OK] ボタンをクリックしてください。

[OK] ボタンをクリックすると，クライアント環境変数の設定をレジストリに登録して，「HiRDB クライアント環境変数登録ツール」画面に戻ります。

[キャンセル] ボタンをクリックすると，クライアント環境変数の設定を無効にして，「HiRDB クライアント環境変数登録ツール」画面に戻ります。

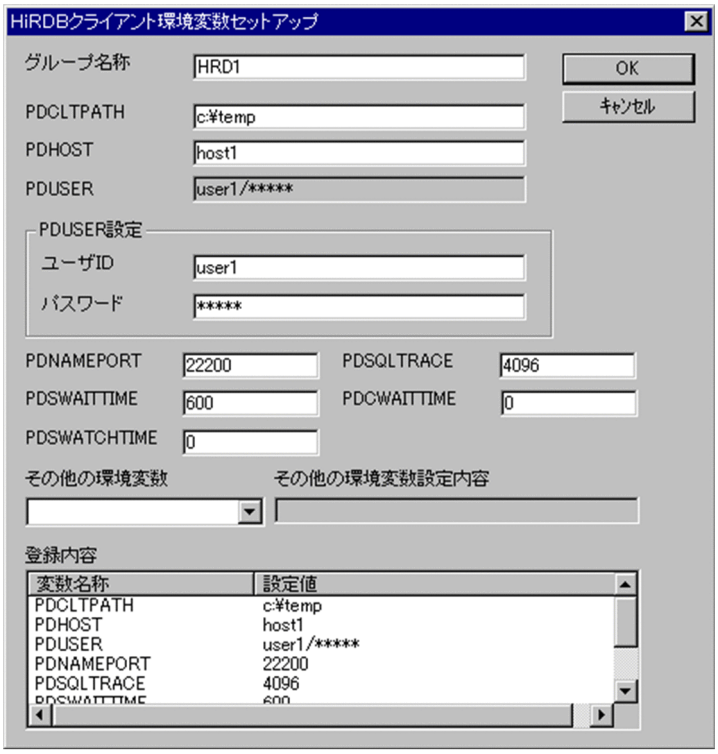
(3)　環境変数グループの設定内容の変更

環境変数グループを一つ以上登録すると，次の画面のように環境変数グループ名のリストが表示されます。



リスト中の環境変数グループ名を選択すると，［削除］，［構成］，［テスト］ボタンが有効になるので，［構成］ボタンをクリックしてください。又は，リスト中の環境変数グループ名をダブルクリックしてください。

すると，次の画面が表示されます。



環境変数の内容を変更したら，［OK］ボタンをクリックしてください。

［OK］ボタンをクリックすると，変更後のクライアント環境変数の設定をレジストリに登録して，「HiRDB クライアント環境変数登録ツール」画面に戻ります。

［キャンセル］ボタンをクリックすると，変更後のクライアント環境変数の設定を無効にして，「HiRDB クライアント環境変数登録ツール」画面に戻ります。

(4)　登録した環境変数グループでの HiRDB 接続の確認

登録した環境変数グループを使用して，HiRDB への接続確認ができます。

「HiRDB クライアント環境変数登録ツール」画面の環境変数グループ名のリストから，接続確認したい環境変数グループ名を選択して，［接続］ボタンをクリックしてください。

すると，次の画面が表示されます。



［はい］ボタンをクリックしてください。

HiRDB への接続が成功した場合には，次の画面が表示されます。



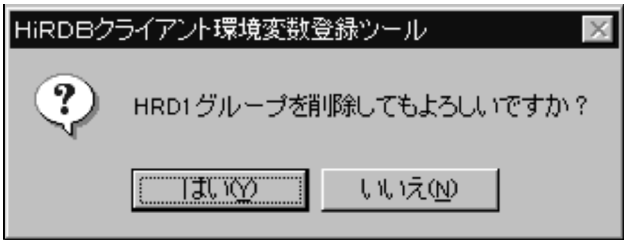
HiRDB への接続が失敗した場合には，次の画面が表示されます。エラー要因が環境変数の内容の場合は，環境変数グループの設定内容を変更してください。



(5)　環境変数グループの削除

「HiRDB クライアント環境変数登録ツール」画面の環境変数グループ名のリストから，削除したい環境変数グループ名を選択して，［削除］ボタンをクリックしてください。

すると，次の画面が表示されます。

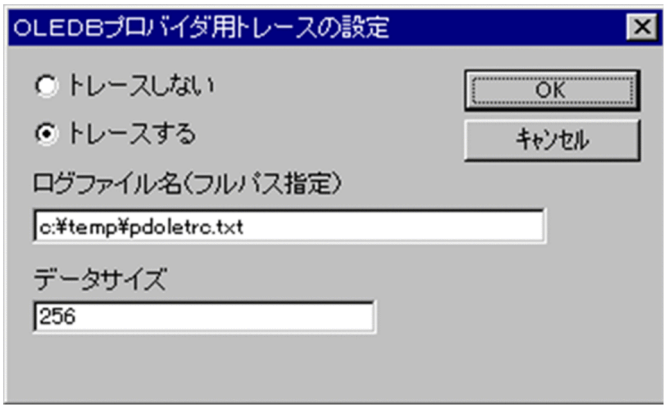


(6)　OLE DB プロバイダ用トレースの設定

OLE DB プロバイダ用トレースは，トラブルシュート用なので，トラブル調査以外では指定しないでください。トレースを取ると，極端に性能が劣化する場合があるので注意してください。

OLE DB 接続時に OLE DB プロバイダ用トレースの設定をする場合は，「HiRDB クライアント環境変数登録ツール」画面の［OLE DB トレース］ボタンをクリックしてください。

すると，次の画面が表示されます。



トレースを取る場合は「トレースする」を選択して，［OK］ボタンをクリックしてください。ただし，「トレースしない」を選択して［OK］ボタンをクリックするまで，トレースを取り続けるので注意してください。

ログファイル名は，必ず絶対パス名で指定してください。

データサイズは，void*型データのダンプ出力サイズをバイト単位で指定してください。

なお，「トレースする」を選択している場合，ログファイル名及びデータサイズの指定は必須です。

複数接続機能を使用した場合に無効になる環境変数：

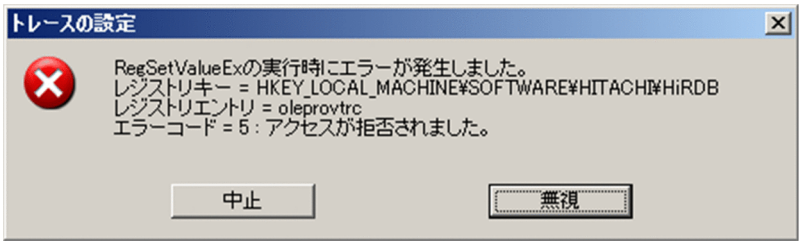
複数接続機能を使用する場合は，次に示す環境変数を接続先ごとに設定できません。次に示す環境変数を通常ファイル又はレジストリに登録して，接続先ごとに指定しても無効になります。

- HiRDB_PDHOST
- HiRDB_PDNAMEPORT
- HiRDB_PDTMID

- HiRDB_PDXAMODE
- PDTMID
- PDXAMODE
- PDTXACANUM

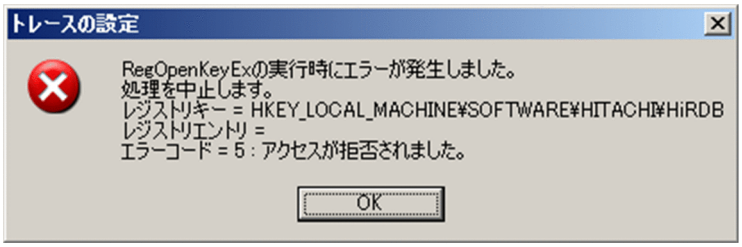
(7)　レジストリ登録でエラーが発生した場合

レジストリ登録時にレジストリ操作の API でエラーが発生した場合，次のどちらかの画面が表示されます。



［中止］ボタンをクリックすると，現在の操作を中止して元の画面に戻ります。

［無視］ボタンをクリックすると，エラーを無視して次の処理に進みます。



［OK］ボタンをクリックすると，現在の操作を中止して元の画面に戻ります。

画面の出力内容を次の表に示します。

項番	項目	内容
1	RegXXX	エラーが発生した API 名を示します。
2	レジストリキー	対象のレジストリキーを示します。64 ビットのレジストリビューが対象の場合は，先頭に「(64bit)」が付きます。
3	レジストリエントリ	対象のレジストリエントリを示します。レジストリキーが対象の場合は空となります。
4	エラーコード	API からの戻り値，及びエラーコードに対応する文字列（OS から取得）を示します。 エラーコードが 5 の場合は，表「HiRDB クライアント環境変数登録ツールを実行するユーザの権限と操作の可否」を参照してツールを実行するユーザの権限を見直してください。

6.7.3　Windows 環境の場合（ファイル登録）

クライアント環境定義をファイルに設定し，HiRDB サーバ接続時にファイルから環境変数を取得できます。

環境変数グループをファイルに登録する場合，1 行目に[HIRDB]又は[HiRDB]を記述する必要があります。odbc.ini の INIFLNAME にクライアント環境定義を指定する場合は，1 行目に[HIRDB]又は[HiRDB]を必ず記述してください。記述されていない場合，クライアント環境定義が読み込まれません。

指定値に引用符（ " ）で囲む値を含む場合の注意事項については，「クライアント環境定義の一覧」の「注※5」を参照してください。

ファイル（c:¥HiRDB_P¥Client¥HiRDB.ini）への設定例を次に示します。

（例）

```
[HIRDB]
PDCLTPATH=トレースファイル格納ディレクトリ
PDHOST=システムマネジャのホスト名
PDUSER=認可識別子/パスワード
PDNAMEPORT=システムマネジャプロセスのポート番号
PDCLTAPNAME=実行するUAPの識別名称
```

注意事項

Type4 JDBC ドライバを用いた UAP の場合は，「UNIX 環境の場合」に示す規則に従います。

6.8　クライアントライブラリのメモリ容量見積もり

6.8.1　クライアントライブラリのメモリ容量の見積もり方法

クライアントライブラリのメモリ容量の見積もりについて説明します。

1 接続当たりでの SQL 実行時に使用するクライアントライブラリのメモリ容量を次に示します。

```
クライアントライブラリのメモリ容量（単位：バイト）＝
  クライアントライブラリの管理情報長
  ＋複数接続機能のハンドル情報長
  ＋送信バッファ長
  ＋受信バッファ長
  ＋ブロック転送機能の管理バッファ長
  ＋カーソルの管理バッファ長
  ＋SQLトレース情報退避領域長
```

それぞれの項目について，以降で説明します。

(1)　クライアントライブラリの管理情報長

次の値を加算します。

- **32 ビットモードの環境の場合**：10970
- **64 ビットモードの環境の場合**：11390

(2)　複数接続機能のハンドル情報長

複数接続機能を使用する場合，次の値を加算します。

- **32 ビットモードの環境の場合**：2620
- **64 ビットモードの環境の場合**：2856

(3)　送信バッファ長

送信バッファ長の見積もり式を次に示します。

なお，複数の SQL 文を実行する場合は，SQL 文ごとに送信バッファ長を求め，その最大値を使用します。

```
送信バッファ長＝↑送信電文長÷4096↑×4096
```

送信電文長

見積もり式を次に示します。

送信電文長は最大 2 ギガバイト（2,147,483,647 バイト）まで指定できます。

(a) CONNECT 実行時

```
送信電文長＝
  ↑（896＋（106×ユーザ任意接続情報の指定数）
  ＋一時表用RDエリア情報長）÷4↑×4
```

ユーザ任意接続情報の指定数

CONNECT 文の実行前に DECLARE AUDIT INFO SET でユーザ任意接続情報を設定した場合に加算します。

一時表用 RD エリア情報長

クライアント環境変数 PDTMPTBLRDAREA を指定した場合に加算します。

見積もり式を次に示します。

```
一時表用RDエリア情報長＝
  6＋↑PDTMPTBLRDAREA指定長÷2↑×2
```

(b) そのほかの SQL 実行時

```
送信電文長＝
  ↑（920＋（106×ユーザ任意接続情報の指定数）
  ＋SQL文長
  ＋入力パラメタ情報長
  ＋出力パラメタ情報長
  ＋入力文字集合名記述領域長
  ＋出力文字集合名記述領域長
  ）÷4↑×4
```

ユーザ任意接続情報の指定数

SQL 文の実行前に DECLARE AUDIT INFO SET でユーザ任意接続情報を設定した場合に加算します。

SQL 文長

SQL 文長です。文字コード変換機能を使用する場合は，文字コード変換後の長さとなります。ただし，クライアント環境定義 PDCLTCNVMODE に UOC を指定した場合は，SQL 文長に PDCLTCNVBYTERATIO の値を乗算した値を加算します。

入力パラメタ情報長

入力パラメタがある場合に加算します。

クライアント環境定義 PDCURSORLVL に 2 を指定した場合で，SELECT 文に WHERE 句を指定しているときは，カーソルオープン後の 1 回目の FETCH 実行時に WHERE 句に指定した入力パラメタ情報長を加算してください。

見積もり式を次に示します。

$$
\begin{aligned}
&\text{入力パラメタ情報長＝}\\
&\left(16 + \sum_{i=1}^{\text{入力パラメタ数}} (16 + \text{入力データ長}_i)\right)\\
&\times \text{配列数}\\
&+ \text{標識変数長}
\end{aligned}
$$

入力パラメタ数

入力したパラメタの個数です。

入力データ長

入力データがある場合に，入力データの定義長を加算します。各データの定義長については，「SQLのデータ型とデータ記述」を参照してください。

なお，文字コード変換機能を使用する場合で，入力データのデータ型が CHAR，VARCHAR，NCHAR，NVARCHAR，MCHAR，又は MVARCHAR のときは，次の表に示す値を加算します。

PDCLTCNVMODE	加算する値
UTF8_TXT，UTF8MS_TXT	ユーザ定義長
UOC	ユーザ定義長×PDCLTCNVBYTERATIO の指定値
上記以外	ユーザ定義長×3

配列数

配列を使用した SQL を実行する場合に乗算します。

標識変数長

標識変数を指定したパラメタ数分を加算します。値は入力データのデータ型によって異なります。

- **BLOB，BINARY，BLOB 位置付け子，又は BINARY 位置付け子の場合**：4
- **上記以外の場合**：2

また，配列を使用した SQL を実行する場合は，配列数分を乗算します。

出力パラメタ情報長

出力パラメタがある場合に加算します。

見積もり式を次に示します。

```
出力パラメタ情報長＝
  16＋16×出力パラメタ数
```

出力パラメタ数

検索系 SQL 実行時の検索列数，又は OUT パラメタと INOUT パラメタの合計数となります。

入力文字集合名記述領域長

入力パラメタに対する文字集合名記述領域の指定がある場合に加算します。

見積もり式を次に示します。

```
入力文字集合名記述領域長＝
  32＋2×文字集合名記述領域のSQLCVARNの指定値
  ＋64×文字集合名記述領域のSQLCSNDの指定値
```

出力文字集合名記述領域長

出力パラメタに対する文字集合名記述領域の指定がある場合に加算します。

見積もり式は，入力文字集合名記述領域長の見積もり式と同じです。

(4)　受信バッファ長

受信バッファ長の見積もり式を次に示します。

なお，複数の SQL 文を実行する場合は，SQL 文ごとに受信バッファ長を求め，その最大値を使用します。

```
受信バッファ長＝↑受信電文長÷4096↑×4096
```

受信電文長

受信電文長は最大 2 ギガバイト（2,147,483,647 バイト）まで指定できます。

見積もり式を次に示します。

(a)　CONNECT 実行時

受信電文長：836

(b)　DISCONNECT 実行時

受信電文長：1388

(c)　そのほかの SQL 実行時

```
受信電文長＝
  ↑（196＋↑（656
  ＋入力パラメタ情報格納長
  ＋出力パラメタ情報格納長
  ＋入力文字集合名記述領域情報格納長
  ＋出力文字集合名記述領域情報格納長
  ＋列名記述領域情報格納長
  ＋型名記述領域情報格納長
  ）÷36864↑×36864※
）÷4↑×4
```

注※

アクセスパス情報又は中間結果情報を取得する場合は 36 キロバイトで切り上げます。

入力パラメタ情報格納長

入力パラメタ情報を受け取る場合に加算します。

見積もり式を次に示します。

```
入力パラメタ情報格納長＝
  16＋16×入力パラメタ情報格納数
```

入力パラメタ情報格納数

DESCRIBE INPUT 文又は PREPARE 文の INPUT 句に指定した SQL 記述領域の入力？パラメタ数（SQL 記述領域の SQLD の指定値）となります。

出力パラメタ情報格納長

出力データ又は出力パラメタ情報を受け取る場合に加算します。

見積もり式を次に示します。

```
出力パラメタ情報長＝
      出力パラメタ数
（16 ＋   Σ    （16＋出力データ長ｉ））
          i=1
  ×配列数
  ＋標識変数長
```

出力パラメタ数

検索系 SQL 実行時の検索列数，OUT パラメタ及び INOUT パラメタの合計数，DESCRIBE OUTPUT 文又は PREPARE 文の OUTPUT 句に指定した SQL 記述領域の出力？パラメタ数（SQL 記述領域の SQLD の指定値）となります。

出力データ長

出力データがある場合に，出力データの定義長を加算します。各データの定義長については，「SQL のデータ型とデータ記述」を参照してください。

なお，文字コード変換機能を使用する場合で，出力データのデータ型が CHAR，VARCHAR，NCHAR，NVARCHAR，MCHAR，又は MVARCHAR のときは，次の表に示す値を加算します。

PDCLTCNVMODE	加算する値
UTF8_TXT，UTF8MS_TXT， UCS2_UJIS，UCS2_UJIS2， UCS2_UTF8	ユーザ定義長
UJIS，UJIS2， UTF8，UTF8MS，UTF8_EX， UTF8_EX2	ユーザ定義長×3
UOC	ユーザ定義長×PDCLTCNVBYTERATIO の指定値

配列数

配列を使用した SQL を実行する場合に乗算します。

標識変数長

標識変数を取得するパラメタ数分を加算します。値は出力データのデータ型によって異なります。

- **BLOB，BINARY，BLOB 位置付け子，又は BINARY 位置付け子の場合**：4
- **上記以外の場合**：2

また，配列を使用した SQL を実行する場合は，配列数分を乗算します。

出力パラメタ情報格納長（ブロック転送機能使用時）

ブロック転送機能を使用する場合は，次に示す見積もり式で出力パラメタ格納情報長を算出します。

```
出力パラメタ情報格納長（ブロック転送機能使用時）＝
  （4×1回の通信で転送する行数×検索列数）
  ＋MAX（出力パラメタ情報格納長×PDBLKF，PDBLKBUFFSIZE×1024）
```

1 回の通信で転送する行数

「1 回の通信で転送する行数」を参照してください。

検索列数

検索系 SQL 実行時の検索列数です。

出力パラメタ情報格納長

前述の「出力パラメタ情報格納長」の見積もり式で算出します。

PDBLKF

クライアント環境変数 PDBLKF の指定値です。指定していない場合は 1 となります。

PDBLKBUFFSIZE

クライアント環境変数 PDBLKBUFFSIZE の指定値です。指定していない場合は 0 となります。

入力文字集合名記述領域情報格納長

入力パラメタ情報の文字集合名記述領域情報を受け取る場合に加算します。

見積もり式を次に示します。

```
入力文字集合名記述領域情報格納長＝
  32＋2×文字集合名記述領域のSQLCVARNの指定値
  ＋64×文字集合名記述領域のSQLCSNDの指定値
```

出力文字集合名記述領域情報格納長

出力パラメタ情報の文字集合名記述領域情報を受け取る場合に加算します。

見積もり式は，入力文字集合名記述領域情報格納長の見積もり式と同じです。

列名記述領域情報格納長

列名記述領域情報を受け取る場合に加算します。

見積もり式を次に示します。

```
列名記述領域情報格納長＝
  2＋32×SQL記述領域のSQLNの指定値
```

型名記述領域情報格納長

型名記述領域情報を受け取る場合に加算します。

見積もり式を次に示します。

```
型名記述領域情報格納長＝
  2＋42×SQL記述領域のSQLNの指定値
```

(5)　ブロック転送機能の管理バッファ長

ブロック転送機能を使用する場合に加算します。

見積もり式を次に示します。

```
ブロック転送機能の管理バッファ長＝
  （ブロック転送機能の管理情報
```

```
＋出力パラメタ情報長
＋出力文字集合名記述領域長
＋受信電文長）
×カーソル数
```

ブロック転送機能の管理情報

次の値を加算します。

- **32 ビットモードの環境の場合**：156
- **64 ビットモードの環境の場合**：184

出力パラメタ情報長

「送信バッファ長」の「そのほかの SQL 実行時」の「出力パラメタ情報長」の見積もり式で算出します。

出力文字集合名記述領域長

「送信バッファ長」の「そのほかの SQL 実行時」の「出力文字集合名記述領域長」の見積もり式で算出します。

受信電文長

「受信バッファ長」の「そのほかの SQL 実行時」の受信電文長の見積もり式で算出します。

カーソル数

同時にオープンするカーソル数です。

(6)　カーソルの管理バッファ長

クライアント環境定義 PDCURSORLVL に 1，又は 2 を指定した場合に加算します。

見積もり式を次に示します。

```
カーソルの管理バッファ長＝
　（カーソルオープン及びカーソルクローズの管理情報
　＋入力パラメタ情報長
　＋入力文字集合名記述領域長
　＋SQL文（SELECT文）長）
　×カーソル数
```

カーソルオープン及びカーソルクローズの管理情報

次の値を加算します。

- **32 ビットモードの環境の場合**：200
- **64 ビットモードの環境の場合**：216

入力パラメタ情報長

「送信バッファ長」の「そのほかの SQL 実行時」の「入力パラメタ情報長」の見積もり式で算出します。

入力文字集合名記述領域長

「送信バッファ長」の「そのほかの SQL 実行時」の「入力文字集合名記述領域長」の見積もり式で算出します。

SQL 文（SELECT 文）長

SQL 文（SELECT 文）長です。文字コード変換機能を使用する場合でも，文字コード変換前の長さとなります。

カーソル数

同時にオープンするカーソル数です。

(7) SQL トレース情報退避領域長

クライアント環境定義 PDSQLTRCTYPE に ERROR を指定した場合，クライアント環境定義 PDSQLTRCBUFFSIZE の指定値を加算します。

6.9　HiRDB クライアントの SELinux セキュリティコンテキスト設定及び解除

SELinux を Enforcing 又は Permissive モードで実行している Linux 環境では，HiRDB が使用するプロセスやファイルに適切な SELinux セキュリティコンテキストを設定しないと，HiRDB は正常に動作しません。ここでは，OS コマンドを使用して HiRDB クライアントに SELinux セキュリティコンテキストを設定及び解除する方法を示します。HiRDB サーバに SELinux セキュリティコンテキストを設定及び解除する方法については，マニュアル「HiRDB システム導入・設計ガイド」を参照してください。

6.9.1　前提条件

(1)　SELinux ポリシーバージョン

次の SELinux ポリシーバージョンを前提とします。

対象 OS	SELinux ポリシーバージョン
Red Hat Enterprise Linux 8 Oracle Linux 8	31
Red Hat Enterprise Linux 9 Oracle Linux 9 Red Hat Enterprise Linux 10 Oracle Linux 10	33

(2)　前提パッケージ

各対応 OS で SELinux の設定をするために必要な前提パッケージを次に示します。

対象 OS	前提パッケージ
Red Hat Enterprise Linux 8 Oracle Linux 8 Red Hat Enterprise Linux 9 Oracle Linux 9 Red Hat Enterprise Linux 10 Oracle Linux 10	• policycoreutils-python-utils • policycoreutils • libselinux-utils

(3)　SELinux ユーザ

HiRDB 管理者ユーザは，非制限ユーザ unconfined_u を前提とします。

6.9.2　SELinux セキュリティコンテキストを設定する方法

SELinux を Enforcing 又は Permissive モードへ変更する場合に，OS コマンドを使用して HiRDB クライアントのインストールディレクトリ配下に対して SELinux セキュリティコンテキストを設定する方法を，次に示します。

1. SELinux を Disabled モードから Permissive モードへ変更します。
2. OS コマンドを使用して，SELinux セキュリティコンテキストを設定します。
3. 必要に応じて，SELinux を Enforcing モードへ変更します。

(1)　実行する OS コマンド

root ユーザで次の OS コマンドを実行して SELinux セキュリティコンテキストを設定します。なお，semanage コマンド及び restorecon コマンドの詳細は，OS のマニュアルを参照してください。

1. /opt/HiRDB 配下の type を usr_t に変更します。

```
semanage fcontext -a -t usr_t /opt/HiRDB'(/.*)?'
```

2. /opt/HiRDB 配下の変更内容を確定します。

```
restorecon -FR /opt/HiRDB
```

(2)　設定後の確認

SELinux セキュリティコンテキストが設定できているかを確認する方法を，次に示します。

- /opt/HiRDB 配下のセキュリティコンテキストが"usr_t"になっているか確認してください。

```
ls -1Z /opt/HiRDB
```

実行結果の例

```
system_u:object_r:usr_t:s0 bin
system_u:object_r:usr_t:s0 client
 :
system_u:object_r:usr_t:s0 lib
```

6.9.3　SELinux セキュリティコンテキストを解除する方法

SELinux を Permissive モードへ変更した後，OS コマンドを使用して HiRDB クライアントのインストールディレクトリ配下に対して SELinux セキュリティコンテキストを解除する方法を，次に示します。

1. SELinux が Enforcing モード又は Disabled モードで動作している場合は，Permissive モードへ変更します。
2. OS コマンドを使用して，SELinux セキュリティコンテキストを解除します。

3. 必要に応じて，SELinux を Disabled モードへ変更します。

(1) 実行する OS コマンド

root ユーザで次の OS コマンドを実行して SELinux セキュリティコンテキストを解除します。なお，semanage コマンド及び restorecon コマンドの詳細は，OS のマニュアルを参照してください。

1. /opt/HiRDB 配下の type を変更前の状態に戻します。

```
semanage fcontext -d /opt/HiRDB'(/.*)?'
```

2. /opt/HiRDB 配下の変更内容を確定します。

```
restorecon -FR /opt/HiRDB
```

(2) 解除後の確認

SELinux セキュリティコンテキストが解除できているかを確認する方法を，次に示します。

- /opt/HiRDB 配下の SELinux コンテキストが変更前の状態になっているか確認してください。

```
ls -1Z /opt/HiRDB
```

実行結果の例

```
system_u:object_r:default_t:s0 bin
 :
```

7 UAP の作成

この章では，C 言語，又は COBOL 言語を使った埋込み型 UAP の作成方法について説明します。

7.1　埋込み型 UAP の概要

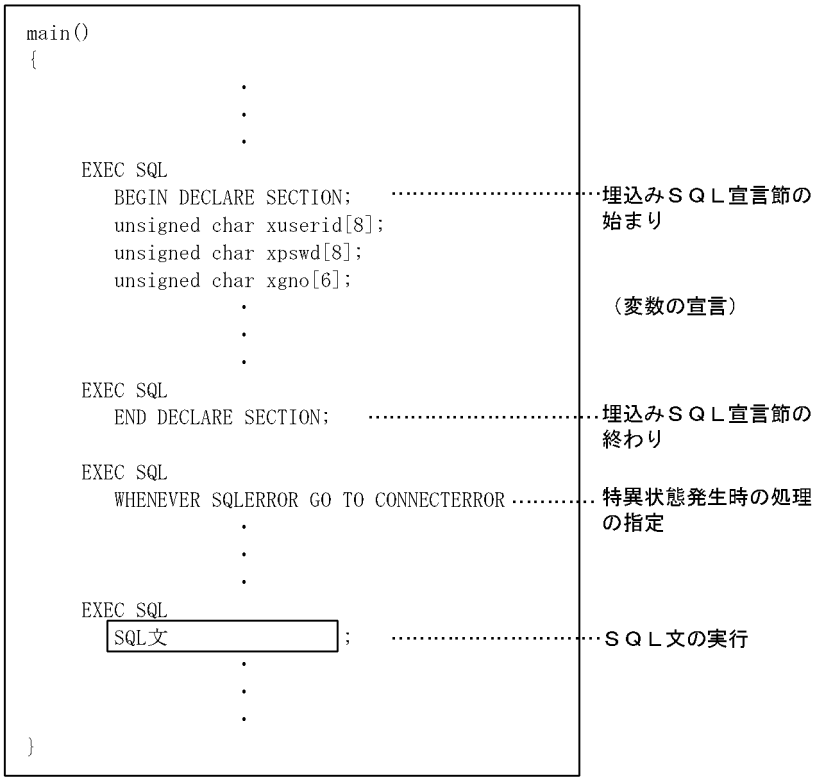
埋込み型の UAP を作成する場合，C 言語，又は COBOL 言語で記述されたソースプログラム中に SQL を埋め込みます。

ここでは，埋込み型の UAP の基本構成と規則について説明します。

7.1.1　UAP の基本構成

埋込み型で記述した UAP の基本構成の例として C 言語の記述例を次の図に示します。

図 7–1　埋込み型で記述した UAP の基本構成の例



7.1.2　UAP の構成要素

埋込み型の UAP の構成は，次に示す四つの要素を基本にしています。

- 埋込み変数，及び標識変数の宣言
- SQL 連絡領域の宣言
- 特異状態発生時の処理の指定
- SQL の実行

(1)　埋込み変数，及び標識変数の宣言

SQL 文で使用する埋込み変数，及び標識変数を宣言します。

埋込み変数，及び標識変数については，マニュアル「HiRDB SQL リファレンス」を参照してください。

(2)　SQL 連絡領域の宣言

HiRDB から返される情報（リターンコード）を受け取るための領域を宣言します。SQL 連絡領域は，UAP をプリプロセスすることでソースプログラム中に自動的に展開されるため，UAP 内に記述する必要はありません。

SQL 連絡領域については，「SQL 連絡領域」を参照してください。

(3)　特異状態発生時の処理の指定

SQL 実行後，HiRDB から返されるリターンコードによって，UAP が取らなければならない処理を WHENEVER 文で宣言します。

なお，WHENEVER 文を宣言しなくても，SQL 実行後に直接リターンコードを判定すると，特異状態発生時の処理を指定できます。

WHENEVER 文の宣言，及びリターンコードの判定については，「SQL のエラーの判定と処置」を参照してください。

(4)　SQL の実行

実行する SQL 文を指定します。

C 言語の記述規則については，「記述規則」を参照してください。また， COBOL 言語の記述規則については，「記述規則」を参照してください。

7.2　C 言語による UAP の作成

ここでは，C 言語による埋込み型 UAP の記述規則，及び作成例について説明します。

7.2.1　記述規則

UAP を作成するとき，SQL の文法の規則以外に，名標の付け方や SQL の記述についての規則があります。

(1)　名標の付け方の規則

名標を付ける場合，基本的に C 言語の規則に従います。

C 言語の規則以外に，使用できない名標を次に示します。

- 大文字の「SQL」で始まる名標
- 小文字の「p_」で始まる名標
- 小文字の「pd」で始まる名標
- 大文字の「PD」で始まる名標

なお，SQL 中で使用する埋込み変数，標識変数，及び分岐先ラベルの名称の付け方は，名標の付け方，及び C 言語の規則に従います。

(2)　SQL の記述規則

1. SQL は，一つの SQL 文ごとに SQL 先頭子（EXEC SQL）と SQL 終了子（;）とで囲みます。

   <正しい指定例>

```
EXEC SQL  SQL文  ;
```

2. 埋込み SQL 文，又はその一部に C 言語のマクロ機能を使用できません。

   <誤った指定例>

```
#define X USER.MEMBER

EXEC SQL
  SELECT NAME INTO :MANNAME FROM X;
```

   注　__部分が誤りです。

3. SQL の予約語は，大文字でも小文字でも使用できます。

   <指定例 1>

```
EXEC SQL
  SELECT MEM INTO :NAME FROM TABLE;
```

   <指定例 2>

```
exec sql
  select MEM into :NAME from TABLE;
```

<指定例 3>

```
exec SQL
  SELECT MEM Into :NAME From TABLE;
```

4. SQL 先頭子，埋込み SQL 開始宣言，及び埋込み SQL 終了宣言は，各 1 行で記述します。また，各々を構成する字句を区切るため，半角空白文字を挿入します。

   なお，改行文字の次の文字から次の改行文字までの文字列が 1 行です。ただし，プリプロセスする UAP のソースプログラムは，1 行の長さが 32000 文字を超える記述ができないので，1 行の長さを 32000 文字以内にしてください。

   <正しい指定例>

```
EXEC SQL
  BEGIN DECLARE SECTION;
    :

EXEC SQL
  END DECLARE SECTION;

EXEC SQL
  SELECT  ...    ;
```

   <誤った指定例>

```
EXEC SQL
  BEGIN__
  DECLARE SECTION;
    :

EXEC SQL
  END__
  DECLARE SECTION;

EXEC ¥
  SQL
  SELECT ...    ;
```

   注　__部分が誤りです。

5. 埋込み SQL 宣言節の定義は，埋込み変数，又は標識変数を使用する SQL よりも前に宣言します。

   <指定例>

```
EXEC SQL
  BEGIN DECLARE SECTION;
short URIAGE;
EXEC SQL
  END DECLARE SECTION;
     :

EXEC SQL
```

```
    SELECT KINGAKU INTO :URIAGE
      FROM TABLE;
```

6. 埋込み変数，及び標識変数を指定する場合，次の規則に従います。

   - 宣言文は複数行にわたって記述できます。また，同じ行に二つ以上の定義文を記述することもできます。

   <指定例>

```
short URIAGE,
   SURYO;
short URIAGE;     short SURYO;
```

   - 埋込み SQL 宣言節内で記述できる項目を次の表に示します。

     表 7–1　埋込み SQL 宣言節内で記述できる項目

記述する項目	埋込み宣言節内での記述
注釈	○
C 言語の命令文	×
C 言語の制御文	×
SQL 文	×
埋込み変数宣言	○
標識変数宣言	○

   （凡例）

   ○：記述できます。

   ×：記述できません。

   - 一つのソースファイル内では，同一名称の埋込み変数，及び標識変数を二つ以上宣言できません。
   - 一つの宣言文で複数の埋込み変数，又は標識変数を宣言できます。

   <指定例>

```
short URIAGE, SURYO ;      埋込み変数の宣言
short XURIAGE, XSURYO ;    標識変数の宣言
```

   - 埋込み変数に使用できるデータ型については，「SQL のデータ型とデータ記述」を参照してください。

7. 関数内で宣言した埋込み変数はローカル変数になり，関数外で宣言した埋込み変数はグローバル変数になります。

8. 関数ブロック内で C 言語の命令文を記述できる箇所には，埋込み SQL 文も記述できますが，ほかの SQL 文，及び C 言語で記述した文と同じ行には記述できません。

   なお，ラベルは SQL 先頭子の前に付けることができます。

   SQL 文を記述できる箇所を次の表に示します。

表 7–2　SQL 文を記述できる箇所

同一行内の記述箇所		SQL 文の記述
C 言語と命令文	前	×
	中	×
	後	×
C 言語制御文	前	×
	中	×
	後	×
ラベル	前	×
	後	○
注釈	前	○
	中	×
	後	○
SQL 文※	前	×
	中	×
	後	×

（凡例）

○：記述できます。

×：記述できません。

注※

SQL 先頭子で始まり，SQL 終了子で終わることを前提にします。

9. VisualC++コンパイラで MFC（MicrosoftFoundationClass）用ヘッダファイル（AFXxxxxx.H）を HiRDB の UAP ソースにインクルードする場合，次の SQL 文で MFC 用ヘッダファイルより後に HiRDB のヘッダファイルをインクルードしてください。HiRDB が提供するヘッダファイルを，この SQL 文を記述した箇所でインクルードします。

```
EXEC SQL INCLUDE HIRDB_HEADERS ;
```

- INCLUDE HIRDB_HEADERS は，ポストソースの先頭で自動的にインクルードしていた HiRDB のヘッダファイルを，指定した箇所でインクルードするものです。
- INCLUDE HIRDB_HEADERS は，C 言語，及び C++言語でだけ使用できます。ほかの言語では使用できません。
- INCLUDE HIRDB_HEADERS は，UAP 中で一回だけ使用できます。
- INCLUDE HIRDB_HEADERS を使用しない場合，ポストソースの先頭に HiRDB のヘッダファイルがインクルードされます。

VisualC++が提供している MFC 用のヘッダファイルには，インクルードする順番に順序関係があり，WINDOWS.H ヘッダファイル（HiRDB が使用しているヘッダファイル）が先にインクルードされているとエラーになることがあります。この場合，INCLUDE HIRDB_HEADERS を使用してください。

HiRDB は，次の VisualC++のヘッダファイルを使用しています。

- WINDOWS.H
- STRING.H

INCLUDE HIRDB_HEADERS の使用例を次に示します。

```
#include <afx.h>
EXEC SQL INCLUDE HIRDB_HEADERS ;
```

10. 注釈の規則を次に示します。

    - SQL 先頭子から SQL 終了子までの間に記述した囲み注釈（/*～*/）及び単純注釈（-- ～ 改行）は SQL プリプロセサが削除します。ただし，SQL 最適化指定（/*>>～<<*/）は削除しないで，SQL 文として扱います。また，引用符（"）やアポストロフィ（'）で囲まれた部分にある「/*～*/」及び「--～改行」は削除しません。
    - プリプロセスオプション-Xs 又は/Xs を指定した場合にだけ，SQL 先頭子から SQL 終了子までの間に単純注釈を記述できます。
    - 単純注釈と SQL 終了子を同じ行に記述することはできません。

    <単純注釈の正しい指定例>

    ```
    EXEC SQL -- 注釈1
        CALL PROC1(0) -- 注釈2
    ;
    ```

    <単純注釈の誤った指定例 1>

    ```
    EXEC SQL CALL PROC1(0) -- 注釈1 ;
    ```

    この場合，単純注釈の中にセミコロン（;）があるため，セミコロンが SQL 終了子としてみなされません。

    <単純注釈の誤った指定例 2>

    ```
    EXEC SQL CALL PROC1(0) ; -- 注釈1
    ```

    この場合，「-- ～ 改行」が SQL 先頭子から SQL 終了子（;）の範囲外にあるため，「-- ～ 改行」が注釈としてみなされません。

    SQL 文中での囲み注釈，単純注釈，及び SQL 最適化指定については，マニュアル「HiRDB SQL リファレンス」を参照してください。

11. ¥記号での行の継続は使用できません。

12. -E 又は/E オプションを使用すると，C コンパイラのプリプロセサ宣言文が有効となるため，#ifdef での SQL 文の切り替えができ，また，埋込み SQL 宣言節の定数をマクロ定数で記述できます。ただし，次の制限があります。

- SQL 先頭子から SQL 終了子までの間に，プリプロセサ宣言文を記述できません。
- SQL 先頭子と SQL 終了子のカラム位置が変わるマクロは使用できません。
- SQL 先頭子，及び SQL 終了子のマクロ定義はできません。

13. -E 又は/E オプションを使用すると，C 言語の文法に従って宣言された，SQL のデータ型に対応した変数であれば，埋込み SQL 宣言節で埋込み変数を宣言しなくても，埋込み変数を使用できます。同じ名称の変数がある場合は，C 言語の文法に従って，それぞれの変数の有効範囲を判定します。ただし，次の制限があります。
    - 変数名は，先頭から 63 文字までが区別されます。64 文字目以降は区別されません。なお，ユニバーサル文字名（¥uxxxx と¥Uxxxxxxxx）は 1 文字としてカウントされます。
    - ネストした構造体は使用できません。
    - 宣言文の中で，添字に式を使用している埋込み変数は使用できません。
    - const 型の埋込み変数は，入力用の変数としてだけ使用できます。
    - 変数名，関数名などの C 言語の識別子として，大文字，小文字に関係なく「varchar」は使用できません。
14. -E 又は/E オプションを使用すると，複数個の埋込み変数をメンバとして持つ構造体を，埋込み変数として宣言できます。すべてのメンバは，SQL のデータ型に対応した形式でなければなりません。構造体の中に，構造体及び共用体を含めることはできません。ただし，可変長文字列型，及び BINARY 型に対応する構造体は使用できます。
15. -E 又は/E オプションを使用すると，ポインタを埋込み変数として宣言できます。宣言の形式は C 言語の文法に従います。SQL 文中で使用する場合は，埋込み変数名の前にアスタリスクを付けないで，通常の埋込み変数と同じ形式で記述します。
16. -E 又は/E オプションを使用すると，構造体のメンバを埋込み変数として明示的に指定する場合は，構造体名で修飾して"：構造体．メンバ名"と記述します。構造体へのポインタを使用する場合は，ポインタで修飾して"：ポインタ名->メンバ名"と記述します。
17. Windows 版の場合に/E オプションを指定したときは，次の制限があります。
    - typedef で，同じ typedef 名を二回定義しても構文エラーになりませんが，定義内容が同じかどうかのチェックはされません。
    - 匿名構造体のメンバは，埋込み変数として使用できません。
    - 記憶クラス，及びデータ型なしで記述した宣言子は，埋込み変数として使用できません。
18. -E 又は/E オプション指定時にも，SQL の COPY 文を使用できます。COPY 文を使用できる条件については，表「プリプロセスオプション（UNIX 環境の C 言語の場合）」又は表「プリプロセスオプション（Windows 環境の C 言語の場合）」を参照してください。
19. 可変長配列は，埋込み変数として使用できません。
20. -E 又は/E オプション指定時には，2 文字表記「<:」，「<%」など，及び 3 文字表記「??(」，「??=」などの文字は使用できません。また，-E 又は/E オプションを指定しない場合でも，SQL 文と埋込み SQL 宣言節では使用できません。使用すると，通常の文字として扱われます。

21. -E 又は/E オプション指定時には，次に示す C の標準規格（C99）に規定されていない予約語を使用できます。

用途	処理	環境※1	予約語
型修飾子，又は型指定子※2	構文解析で型修飾子，又は型指定子として扱われる。	Windows	__int8，__int16，__int32，__int64，_int8，_int16，_int32，_int64
		UNIX	__volatile__，__builtin_va_list，__complex__，__signed__
その他	構文解析で意味のない語句として扱うため無視される。	Windows	__based，__cdecl，__export，__far，__fastcall，__forceinline，__inline，__near，__pascal，__ptr32，__ptr64，__stdcall，__unaligned，__w64，_based，_cdecl，_export，_far，_fastcall，_forceinline，_inline，_near，_pascal，_stdcall__declspec，__pragma，_declspec__try，__except，__finally__asm，_asm
		UNIX	__const，__const__，__extension__，__inline，__inline__，__restrict，__attribute__，__asm__

注※1

Windows 環境の予約語は，Visual C++で使用されるものです。2 文字の下線（__）で始まる予約語は，VisualC++6.0 以降で定義されているものです。下位バージョンの Visual C++で作成したソースのために，1 文字の下線（_）で始まる場合も含めます。

UNIX 環境の予約語は，gcc で使用されるものです。

注※2

型修飾子，又は型指定子として扱う予約語は，埋込み変数の宣言には使用できません。

22. -E 又は/E オプション指定時には，ビットフィールドは任意の整数型として宣言できます。ただし，ビットフィールドは埋込み変数の宣言には使用できません。

23. ソースにヘッダファイルを引き込む方法として，#include と SQL の COPY 文があります。それぞれヘッダファイル中に記述できる項目を次に示します。

項番	記述項目		ヘッダファイルを引き込む方法	
			#include	COPY 文
1	SQL 文	COPY	×	×
2		COPY 以外	×	○
3	埋込み変数宣言	「SQL TYPE IS」で始まる	×	○
4		「VARCHAR」で始まる	×	○

項番	記述項目		ヘッダファイルを引き込む方法	
			#include	COPY 文
5		「CHARACTER SET」を含む	×	○
6		C 言語による変数宣言※	△	○
7	標識変数宣言		△	○

（凡例）

○：記述できます。

×：記述できません。

△：-E3 又は/E3 オプション指定時にだけ記述できます。

注※

SQL プリプロセサが変換しなくても C コンパイラが構文解析できる変数宣言のことです。

なお，項番 3〜5 の埋込み変数宣言は SQL プリプロセサが変換しないと C コンパイラが構文解析できません。ただし，繰返し列用の埋込み変数宣言に使用する PD_MV_SINT などは C 言語のマクロとして定義しているため，繰返し列用の埋込み変数宣言はそのままで C コンパイラが構文解析できます。したがって，繰返し列用の埋込み変数宣言は C 言語による変数宣言になります。

7.2.2　プログラム例題

C 言語による埋込み型 UAP のプログラム例題を示します。

なお，SQL の文法の詳細については，マニュアル「HiRDB SQL リファレンス」を参照してください。

(1)　基本的な操作の例

(a)　PAD チャート

プログラム例題 1 の PAD チャートを次の図に示します。

図 7–2　プログラム例題 1 の PAD チャート（1/2）

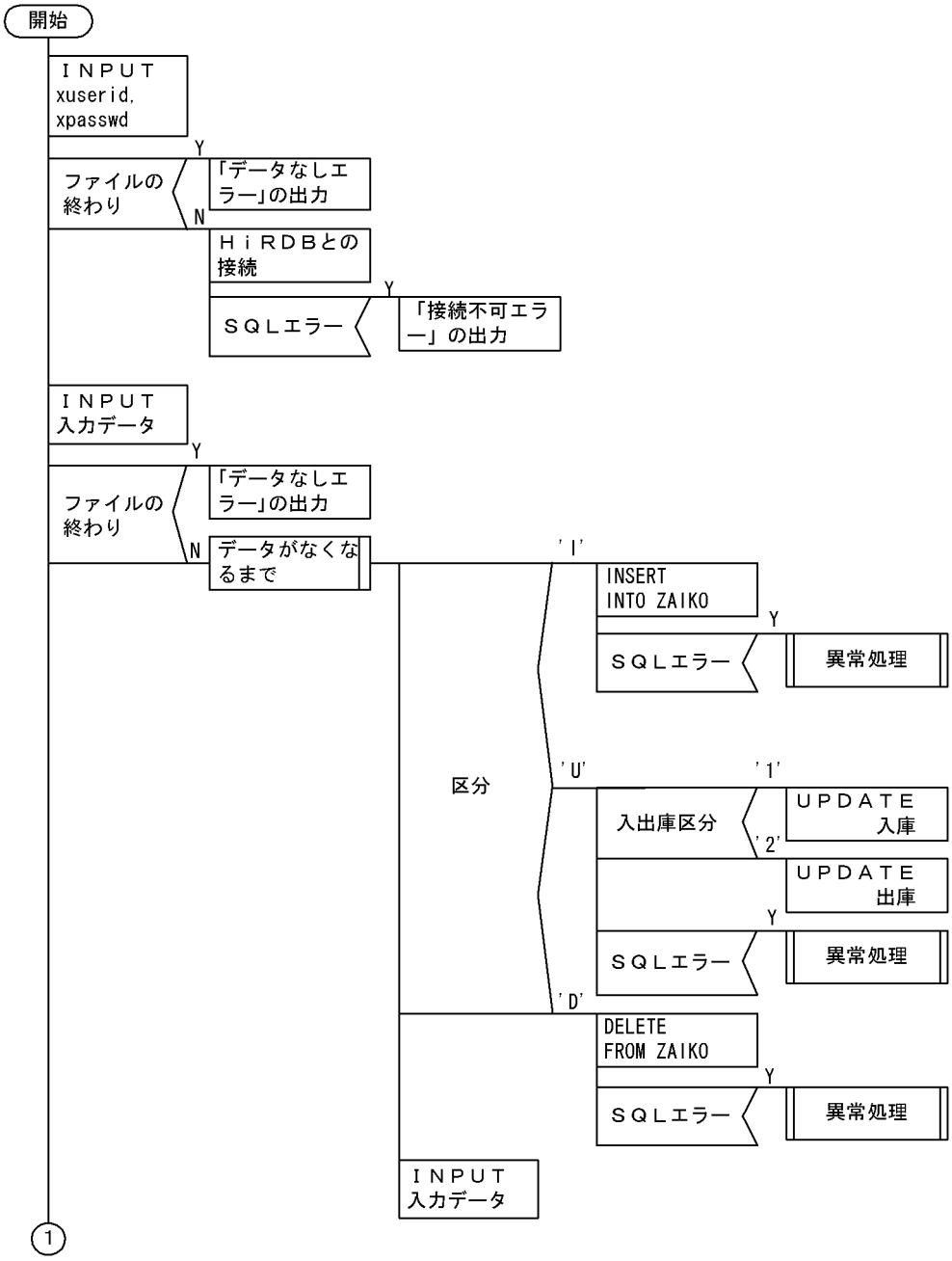
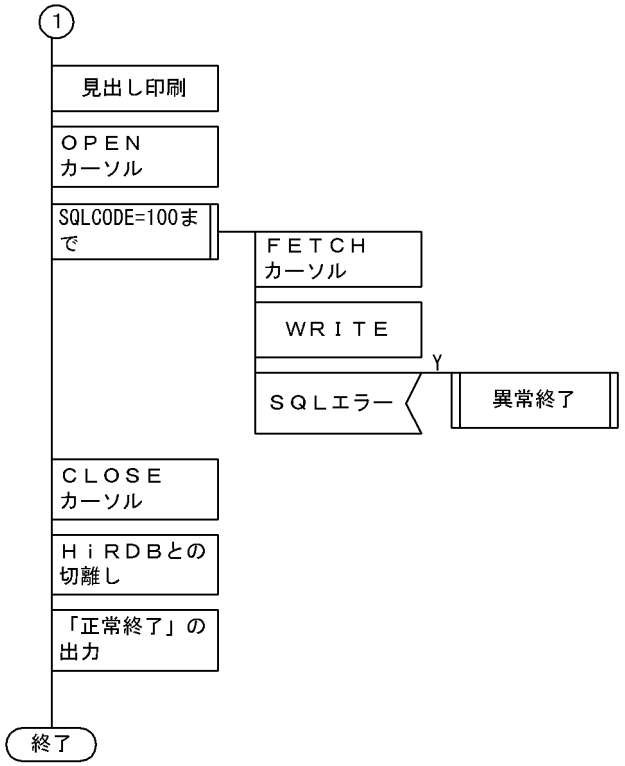
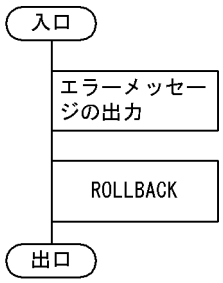


図 7–3　プログラム例題 1 の PAD チャート（2/2）



［異常処理］



(b)　コーディング例

プログラム例題 1 のコーディング例を次に示します。

```
 1  #include <string.h>
 2  #include <stdlib.h>
 3
 4  #define MAXCOLUMN 80           /* max column in one line */
 5  #define INFILE   "inputf1"     /* input data file name   */
 6
 7  /* declare functions */
 8  void abnormalend();
 9  void connecterror();
10
11  FILE *input = NULL;
12
13  main()
14  {
15    /* input data */
```

```
16    char indata[MAXCOLUMN + 1];
17
18    char in_userid[31];
19    char in_passwd[31];
20    char in_kubun;
21    char in_scode[5];
22    char in_sname[17];
23    char in_col[3];
24    int in_tanka;
25    int in_gryo;
26    char in_okubun;
27
28    /* variables for SQL */
29    EXEC SQL BEGIN DECLARE SECTION;                                    1
30      char xuserid[31];                                                1
31      char xpasswd[31];                                                1
32      char xscode[5];                                                  1
33      char xsname[17];                                                 1
34      char xcol[3];                                                    1
35      long xtanka;                                                     1
36      long xgryo;                                                      1
37    EXEC SQL END DECLARE SECTION;                                      1
38
39    /* input file open */                                              2
40    input = fopen(INFILE, "r");                                        2
41    if (input == NULL) {                                               2
42      /* input file open error */                                      2
43      fprintf(stderr, "can't open %s.", INFILE);                       2
44      goto FIN;                                                        2
45    }                                                                  2
46                                                                       2
47    /* get userid/passwd */                                            2
48    fgets(indata, 81, input);                                          2
49    sscanf(indata, "%30s %30s", xuserid, xpasswd);                     2
50    if (feof(input)) {                                                 2
51      fprintf(stderr, "*** error *** no data for connect ***");        2
52      goto FIN;                                                        2
53    }                                                                  2
54    printf("connect start,¥n");                                        2
55    EXEC SQL WHENEVER SQLERROR PERFORM connecterror;          (a)      2
56    EXEC SQL CONNECT USER :xuserid USING :xpasswd;            (b)      2
57    printf("connected,¥n");                                            2
58
59    /* read data from inputfile */
60    EXEC SQL WHENEVER SQLERROR PERFORM abnormalend;
61    fgets(indata, MAXCOLUMN, input);
62
63    while (!feof(input)) {
64      sscanf(indata, "%c %4s %16s %2s %8d %8d %c",
65        &in_kubun, in_scode, in_sname, in_col,
66        &in_tanka, &in_gryo, &in_okubun);
67      switch (in_kubun) {
68      case 'I':
69        strncpy(xscode, in_scode, 4);
70        strncpy(xsname, in_sname, 8);
71        strncpy(xcol, in_col, 2);
72        xtanka = in_tanka;
73        xgryo = in_gryo;
```

```
 74        EXEC SQL                                                    3
 75          INSERT INTO ZAIKO(SCODE,SNAME,COL,TANKA,ZSURYO)           3
 76            VALUES(:xscode,:xsname,:xcol,:xtanka,:xgryo);           3
 77        break;
 78      case 'U':
 79        strncpy(xscode, in_scode, 4);                               4
 80        xgryo = in_gryo;                                            4
 81        if (in_okubun == '1') {                                     4
 82          EXEC SQL                                            (a)  4
 83            UPDATE ZAIKO SET ZSURYO=ZSURYO+:xgryo             (a)  4
 84              WHERE SCODE=:xscode;                            (a)  4
 88        } else {                                                    4
 86          EXEC SQL                                            (b)  4
 87            UPDATE ZAIKO SET ZSURYO=ZSURYO-:xgryo             (b)  4
 88              WHERE SCODE=:xscode;                            (b)  4
 89        }
 90        break;
 91      case 'D':
 92        strncpy(xscode, in_scode, 4);
 93        EXEC SQL                                                    5
 94          DELETE FROM ZAIKO WHERE SCODE=:xscode;                    5
 95        break;
 96      }
 97      fgets(indata, MAXCOLUMN, input);
 98    }
 99
100    /* print zaiko list */
101    EXEC SQL                                                        6
102      DECLARE CR1 CURSOR FOR                                        6
103          SELECT SCODE,SNAME,COL,TANKA,ZSURYO FROM ZAIKO;           6
104    EXEC SQL OPEN CR1;                                              7
105
106    /* print midashi */
107    printf("¥n¥n");
108    printf("  ***** 在庫表 リスト *****¥n¥n");
109    printf("  商品コード  商品名      色 単価    現在庫量¥n");
110    printf("  ----   ---------------- -- -------- --------¥n");
111
112    /* FETCH */
113    SQLCODE = 0;
114    while (SQLCODE <= 100) {
115      EXEC SQL WHENEVER NOT FOUND GO TO OWARI;
116      EXEC SQL                                                      8
117        FETCH CR1 INTO :xscode,:xsname,:xcol,:xtanka,:xgryo;        8
118      EXEC SQL WHENEVER NOT FOUND CONTINUE;
119      printf("  %4s   %-16s %2s %8d %8d¥n",
120          xscode, xsname, xcol, xtanka, xgryo);
121    }
122
123  OWARI:
124    /* finish */
125    EXEC SQL CLOSE CR1;                                       (a)  9
126    EXEC SQL COMMIT;                                          (b)  9
127    printf(" *** normal ended ***¥n");
128
129  FIN:
130    if (input != NULL) {
131      fclose(input);
```

```
  }
  EXEC SQL WHENEVER SQLERROR  CONTINUE;
  EXEC SQL WHENEVER NOT FOUND CONTINUE;
  EXEC SQL WHENEVER SQLWARNING CONTINUE;
  EXEC SQL DISCONNECT;                                          10
  return(0);
}


void connecterror()
{

  printf("¥n************ error *** cannot connect ***¥n");
  fclose(input);
  EXEC SQL DISCONNECT;
  exit(1);
}


void abnormalend()
{
  int  wsqlcode;

  wsqlcode = -SQLCODE;
  printf("¥n*** HiRDB SQL ERROR SQLCODE = %d ¥n", wsqlcode);
  printf("SQLERRMC = %s¥n", SQLERRMC);

  EXEC SQL ROLLBACK;                                        (a)  11
  EXEC SQL DISCONNECT;                                      (b)  11
  exit(2);
}
```

＜説明＞

1. 埋込み SQL 宣言節の始まりと終わり

   UAP 中で使用する変数を BEGIN DECLARE SECTION と END DECLARE SECTION とで囲んで，埋込み SQL 宣言節の始まりと終わりを示します。

2. HiRDB との接続

   (a) 特異状態発生時の指定

   以下の SQL の実行後に，エラー（SQLERROR）が発生した場合の処理として，分岐先（connecterror）を指定します。

   (b) 接続

   HiRDB に認可識別子及びパスワードを連絡して，UAP が HiRDB を使用できる状態にします。

3. 在庫表への行の挿入

   在庫表の各列に，埋込み変数に読み込まれた値を挿入します。

4. 在庫表の行の更新

(a) 入庫

在庫表から，埋込み変数（:xgno）に読み込んだ品番をキーとして，更新する行を検索します。検索した行の数量（SURYO）の値に，埋込み変数（:xsuryo）に読み込んだ値を加算して，行を更新します。

(b) 在庫

在庫表から，埋込み変数(:xgno)に読み込んだ品番をキーとして，更新する行を検索します。検索した行の数量（SURYO）の値に，埋込み変数（:xsuryo）に読み込んだ値を減算して，行を更新します。

5. 在庫表の行の削除

在庫表から，埋込み変数(:xgno)に読み込んだ品番をキーとして，それと等しいキーを持つ行を削除します。

6. カーソル CR1 の宣言

在庫表（ZAIKO）の行を検索するために，カーソル CR1 を宣言します。

7. カーソル CR1 のオープン

在庫表（ZAIKO）の検索行の直前にカーソルを位置づけて，行を取り出せる状態にします。

8. 在庫表の行の取り出し

在庫表（ZAIKO）から，カーソル CR1 の示す行を１行取り出し，各埋込み変数に設定します。

9. カーソル CR1 のクローズとトランザクションの終了

(a) カーソル CR1 のクローズ

カーソル CR1 を閉じます。

(b) トランザクションの終了

現在のトランザクションを正常終了させて，そのトランザクションによるデータベースへの追加，更新，削除の結果を有効にします。

10. HiRDB の切り離し

UAP を HiRDB から切り離します。

11. トランザクションの取り消し

(a) トランザクションの無効化

現在のトランザクションを取り消して，そのトランザクションによるデータベースへの追加，更新，削除の結果を無効にします。

(b) HiRDB の切り離し

UAP を HiRDB から切り離します。

(2)　ユーザ定義の SQL 記述領域を使用した例

(a)　PAD チャート

プログラム例題 2 の PAD チャートを次の図に示します。

図 7–4　プログラム例題 2 の PAD チャート（1/4）

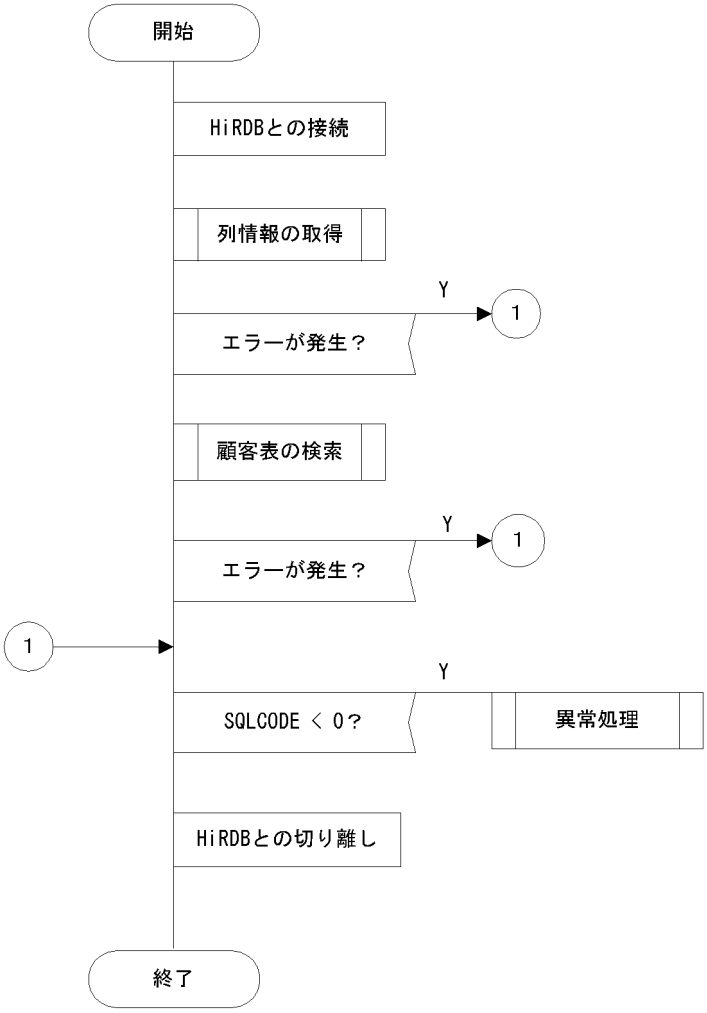


図 7–5　プログラム例題 2 の PAD チャート（2/4）

［列情報の取得］

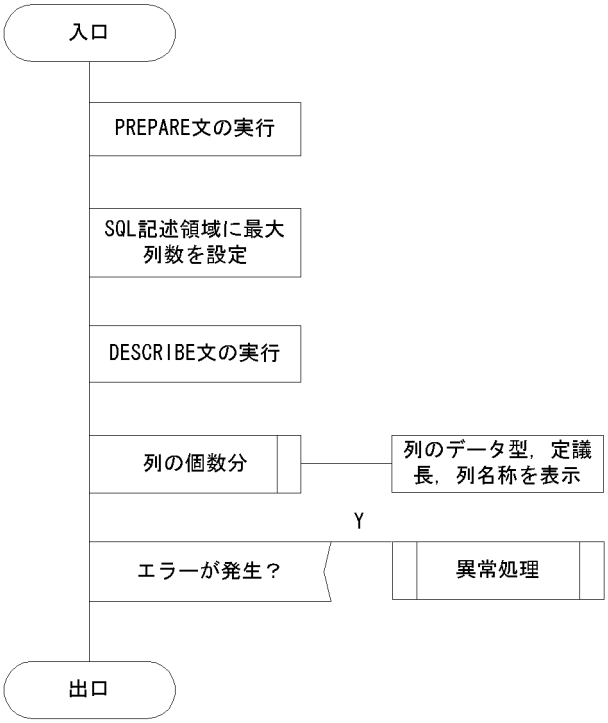


図 7–6　プログラム例題 2 の PAD チャート（3/4）

［顧客表の検索］

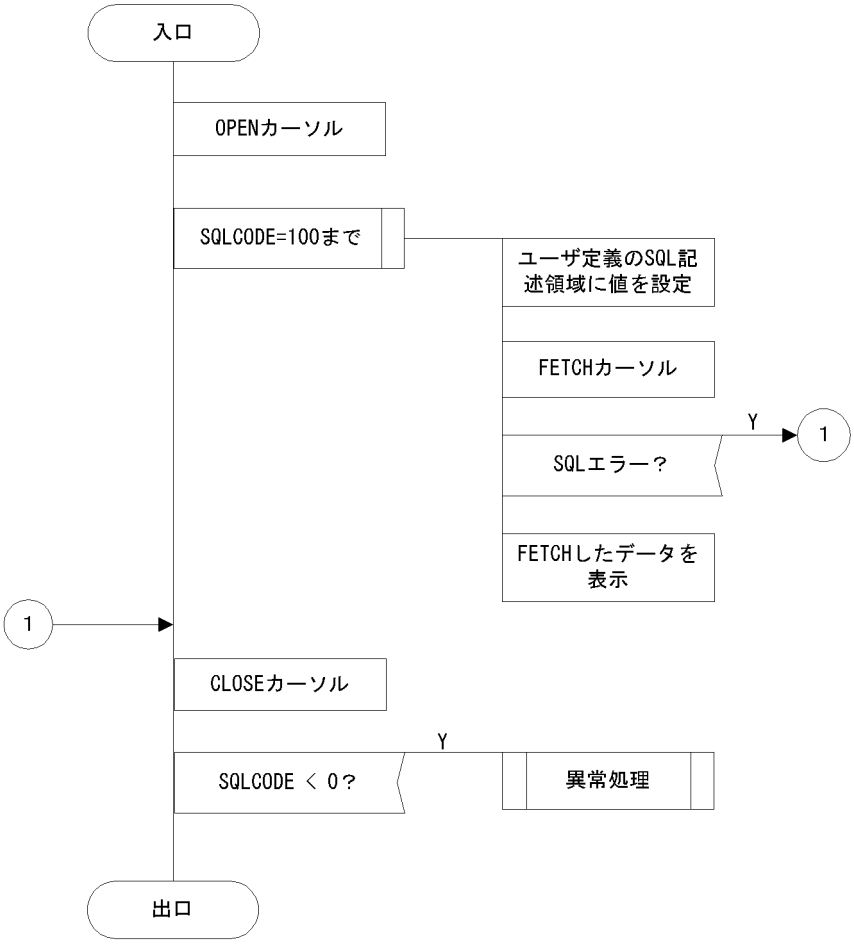
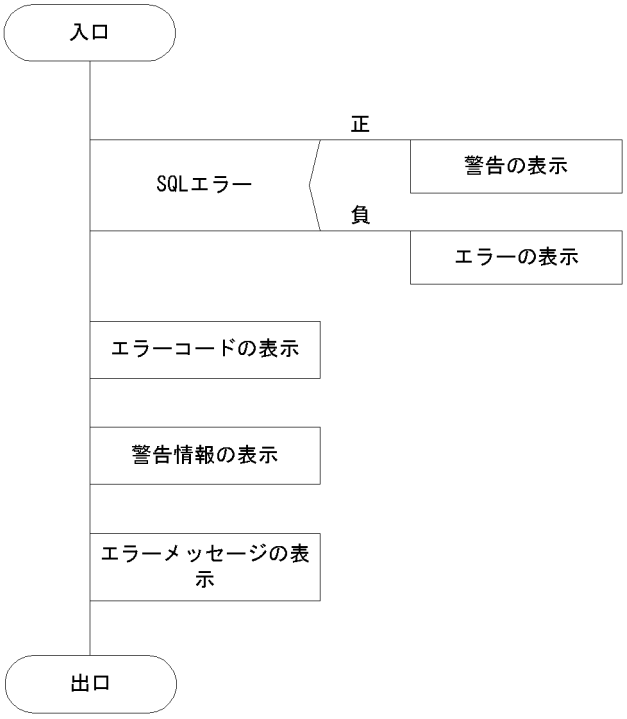


図 7–7　プログラム例題 2 の PAD チャート（4/4）

［異常処理］



(b)　コーディング例

プログラム例題 2 のコーディング例を次に示します。

```
 1  /*******************************************************/
 2  /*                                                     */
 3  /* ALL RIGHTS RESERVED,COPYRIGHT (C)1997,HITACHI,LTD. */
 4  /* LICENSED MATERIAL OF HITACHI,LTD.                   */
 5  /*                                                     */
 6  /* SQLDAを使用したFETCHのサンプル                      */
 7  /*                                                     */
 8  /*******************************************************/
 9
10
11  #include <stdio.h>
12  #include <stdlib.h>
13  #include <string.h>
14  #include "pdbsqlda.h"                                        1
15
16
17  static void Describe();
18  static void Fetch();
19  static void ClearSqlda(short);
20  static void errmsg();
21
22  /*************************************************************/
23  /*  GLOBAL VARIABLE                                          */
24  /*************************************************************/
25  short ErrFlg;
```

```
26
27  /*************************************************************/
28  /*  GLOBAL VARIABLE                                          */
29  /*************************************************************/
30
31  /* sqlda */
32  PDUSRSQLDA(10) xsqlda;                                        2
33
34  /* sqlcnda */                                                 3
35  struct {                                                      3
36      short    sqlnz;                                           3
37      struct {                                                  3
38           short    sqlnamel;                                   3
39           char     sqlnamec[30];                               3
40      } SQLNAME[10];                                            3
41  } ucnda;                                                      3
42
43
44  /*************************************************************/
45  /*                                                           */
46  /*  MAIN ROUTINE                                             */
47  /*                                                           */
48  /*************************************************************/
49  int main(
50  int   argc,
51  char  *argv[])
52  {
53
54  /*************************************************************/
55  /*  CONNECT                                                  */
56  /*************************************************************/
57      EXEC SQL
58        WHENEVER SQLERROR  GOTO :ERR_EXIT;
59
60      printf("***** connect start ¥n");
61      EXEC SQL
62         CONNECT;                                               4
63      printf("***** connect : END¥n");
64
65  /*************************************************************/
66  /*  DESCRIBE                                                 */
67  /*************************************************************/
68      Describe();                                               5
69      if(ErrFlg < 0){                                           5
70          goto ERR_EXIT;                                        5
71      }                                                         5
72                                                                5
73  /*************************************************************/  5
74  /*  FETCH                                                    */  5
75  /*************************************************************/  5
76      Fetch();                                                  5
77      if(ErrFlg < 0){                                           5
78          goto ERR_EXIT;                                        5
79      }                                                         5
80
81  /*************************************************************/
82  /*  END OF ALL                                               */
83  /*************************************************************/
```

```
 84  ERR_EXIT:
 85      if(SQLCODE < 0){
 86          errmsg();
 87          ErrFlg = -1;
 88      }
 89
 90      EXEC SQL
 91        WHENEVER SQLERROR  CONTINUE;
 92      EXEC SQL
 93        WHENEVER NOT FOUND CONTINUE;
 94      EXEC SQL
 95        WHENEVER SQLWARNING CONTINUE;
 96
 97      EXEC SQL
 98        DISCONNECT;                                          6
 99
100      return(ErrFlg);
101  }
102
103
104  /************************************************************/
105  /*                                                          */
106  /*  DYNAMIC CURSOR                                          */
107  /*                                                          */
108  /************************************************************/
109  static void Fetch()
110  {
111      EXEC SQL BEGIN DECLARE SECTION;
112      char XCUSTOM_CD[6];
113      char XCUSTOM_NAME[31];
114      char XTELNO[13];
115      char XZIPCD[4];
116      char XADDRESS[31];
117      EXEC SQL END DECLARE SECTION;
118
119      EXEC SQL
120        WHENEVER SQLERROR  GOTO :Exit_Fetch;
121
122      EXEC SQL
123         DECLARE CUR2 CURSOR FOR SEL1;                       7
124
125  /************************************************************/
126  /*  OPEN CURSOR                                             */
127  /************************************************************/
128      printf("***** DYNAMIC CURSOR open start¥n");
129      EXEC SQL
130        OPEN CUR2;                                           8
131      printf("***** DYNAMIC CURSOR open : END¥n");
132
133
134  /************************************************************/
135  /*  FETCH                                                   */
136  /************************************************************/
137      printf("***** fetch (use sqlda) start¥n");
138
139
140      EXEC SQL
141        WHENEVER NOT FOUND GOTO FETCH2_END;
```

```
142
143     for(;;) {
144         ClearSqlda(5);                                          9
145         PDSQLDATA(xsqlda, 0) = (void *)XCUSTOM_CD;        (a)  9
146         PDSQLCOD(xsqlda, 0) = PDSQL_CHAR;                 (a)  9
147         PDSQLLEN(xsqlda, 0) = sizeof(XCUSTOM_CD)-1;       (a)  9
148         PDSQLDATA(xsqlda, 1) = (void *)XCUSTOM_NAME;      (b)  9
149         PDSQLCOD(xsqlda, 1) = PDSQL_CHAR;                 (b)  9
150         PDSQLLEN(xsqlda, 1) = sizeof(XCUSTOM_NAME)-1;     (b)  9
151         PDSQLDATA(xsqlda, 2) = (void *)XTELNO;            (c)  9
152         PDSQLCOD(xsqlda, 2) = PDSQL_CHAR;                 (c)  9
153         PDSQLLEN(xsqlda, 2) = sizeof(XTELNO)-1;           (c)  9
154         PDSQLDATA(xsqlda, 3) = (void *)XZIPCD;            (d)  9
155         PDSQLCOD(xsqlda, 3) = PDSQL_CHAR;                 (d)  9
156         PDSQLLEN(xsqlda, 3) = sizeof(XZIPCD)-1;           (d)  9
157         PDSQLDATA(xsqlda, 4) = (void *)XADDRESS;          (e)  9
158         PDSQLCOD(xsqlda, 4) = PDSQL_CHAR;                 (e)  9
159         PDSQLLEN(xsqlda, 4) = sizeof(XADDRESS)-1;         (e)  9
160
161         memset(XCUSTOM_CD, 0, sizeof(XCUSTOM_CD));
162         memset(XCUSTOM_NAME, 0, sizeof(XCUSTOM_NAME));
163         memset(XTELNO, 0, sizeof(XTELNO));
164         memset(XZIPCD, 0, sizeof(XZIPCD));
165         memset(XADDRESS, 0, sizeof(XADDRESS));
166
167         EXEC SQL FETCH CUR2
168           USING DESCRIPTOR :xsqlda;                             10
169
170         printf("%s ", XCUSTOM_CD);
171         printf("%s ", XCUSTOM_NAME);
172         printf("%s ", XTELNO);
173         printf("%s ", XZIPCD);
174         printf("%s¥n", XADDRESS);
175     }
176 FETCH2_END:
177     printf("***** fetch : END¥n");
178
179 /*************************************************************/
180 /*  CLOSE CURSOR                                             */
181 /*************************************************************/
182     printf("*****  close start¥n");
183     EXEC SQL
184       WHENEVER NOT FOUND  CONTINUE;
185     EXEC SQL
186       CLOSE CUR2;                                             11
187     printf("*****  close : END¥n");
188
189 /*************************************************************/
190 /*                                                           */
191 /*************************************************************/
192 Exit_Fetch:
193     if(SQLCODE < 0){
194         errmsg();
195         ErrFlg = -1;
196     }
197     return;
198 }
199
```

```
200
201 /*************************************************************/
202 /*  DESCRIBE                                                 */
203 /*************************************************************/
204 static void Describe()
205 {
206     short  i;
207
208     EXEC SQL
209       WHENEVER SQLERROR  GOTO :Exit_Describe;
210
211 /*************************************************************/
212 /*  PREPARE                                                  */
213 /*************************************************************/
214     printf("***** prepare start¥n");
215     EXEC SQL                                                  12
216       PREPARE SEL1                                            12
217         FROM 'SELECT * FROM CUSTOM'                           12
218         WITH SQLNAME OPTION;                                  12
219     printf("***** prepare  : END¥n");
220
221 /*************************************************************/
222 /*  DESCRIBE                                                 */
223 /*************************************************************/
224     PDSQLN(xsqlda) = 10;
225     printf("***** describe start¥n");
226     EXEC SQL
227        DESCRIBE SEL1 INTO :xsqlda :ucnda;                     13
228     printf("***** describe : END¥n");
229
230     printf(" describe result¥n");
231     printf("  NUMBER OF DATA = %d¥n", PDSQLD(xsqlda));
232     printf("  NUMBER OF COLUMN NAME = %d¥n", ucnda.sqlnz);
233     for (i = 0 ; i < ucnda.sqlnz ; i++ ) {
234         printf(" [%d]", i );
235         printf(" DATA TYPE(%d)", PDSQLCOD(xsqlda, i));
236         printf(" DATA LENGTH(%d)", PDSQLLEN(xsqlda, i));
237         printf(" COLUMN NAME(%s)¥n", ucnda.SQLNAME[i].sqlnamec);
238     }
239
240 /*************************************************************/
241 /*                                                           */
242 /*************************************************************/
243 Exit_Describe:
244     if(SQLCODE < 0){
245         errmsg();
246         ErrFlg = -1;
247     }
248     return;
249 }
250
251
252 /*************************************************************/
253 /*  Clear SQLDA                                              */
254 /*************************************************************/
255 static void ClearSqlda(
256 short num)
257 {
```

```
258     PDSQLN(xsqlda) = num;                                          14
259     PDSQLD(xsqlda) = num;                                          14
260     while(num-->0){
261         PDSQLDATA(xsqlda, num) = NULL;                             15
262         PDSQLIND(xsqlda, num)  = NULL;                             15
263         PDSQLDIM(xsqlda, num) = 0;                                 15
264         PDSQLXDIM(xsqlda, num) = 1;                                15
265         PDSQLSYS(xsqlda, num) = 0;                                 15
266         PDSQLCOD(xsqlda, num) = 0;                                 15
267         PDSQLLEN(xsqlda, num) = 0;                                 15
268     }
269     return;
270 }
271
272
273 /************************************************************/
274 /*                                                          */
275 /*  WARNING                                                 */
276 /*                                                          */
277 /************************************************************/
278 static void errmsg()
279 {
280     int wsqlcode;
281
282     if(SQLCODE > 0){
283         printf(">>>警告¥n");
284     }
285     if(SQLCODE < 0){
286         printf(">>> 異常発生¥n");
287     }
288     wsqlcode = SQLCODE;
289     printf(">>> sqlcode = %d¥n", SQLCODE);
290     printf(">>> sqlwarn = %c", SQLWARN0);
291     printf("%c", SQLWARN1);
292     printf("%c", SQLWARN2);
293     printf("%c", SQLWARN3);
294     printf("%c", SQLWARN4);
295     printf("%c", SQLWARN5);
296     printf("%c", SQLWARN6);
297     printf("%c", SQLWARN7);
298     printf("%c", SQLWARN8);
299     printf("%c", SQLWARN9);
300     printf("%c", SQLWARNA);
301     printf("%c", SQLWARNB);
302     printf("%c¥n", SQLWARNC);
303
304 #if defined(HIUXWE2) || defined(WIN32)
305     printf(">>> message = %s¥n", SQLERRMC);
306 #else
307     printf(">>> message = %Fs¥n", SQLERRMC);
308 #endif
309     return;
310 }
```

＜説明＞

1. 提供ヘッダファイルのインクルード

SQL 記述領域に設定・参照するとき用いるデータコードの定数宣言や，SQL 記述領域自体のデータ型を宣言します。

2. SQL 記述領域の宣言

   UAP でユーザが独自に使用する SQL 記述領域を定義します。データ型は提供ヘッダファイルの中で定義されているものです。

3. 列名記述領域の宣言

   列名を DESCRIBE 文で取得するときに用いる，変数を定義します。

4. HiRDB への接続

   環境変数 PDUSER に定義されている認可識別子とパスワードを使用してサーバに接続します。

5. 顧客表（CUSTOM）の検索

   顧客表（CUSTOM）の各列の列名を取得し，表に格納されているすべての行をユーザ定義の SQL 記述領域を用いて検索して表示します。

6. HiRDB の切り離し

   UAP をサーバから切り離します。

7. カーソル CUR2 の宣言

   顧客表（CUSTOM）の行を検索するために，カーソル CUR2 を宣言します。

8. カーソル CUR2 のオープン

   顧客表（CUSTOM）の検索行の直前にカーソルを位置づけて，行を取り出せる状態にします。

9. ユーザ定義の SQL 記述領域の設定

   FETCH 文実行時に指定する，ユーザ定義の SQL 記述領域の設定をします。

   (a) １列目のデータを格納する領域のアドレスとデータコード，データ長を設定します。

   (b) ２列目のデータを格納する領域のアドレスとデータコード，データ長を設定します。

   (c) ３列目のデータを格納する領域のアドレスとデータコード，データ長を設定します。

   (d) ４列目のデータを格納する領域のアドレスとデータコード，データ長を設定します。

   (e) ５列目のデータを格納する領域のアドレスとデータコード，データ長を設定します。

10. 顧客表の行の取り出し

    顧客表（CUSTOM）から，カーソル CUR2 の示す行を１行取り出し，ユーザ定義の SQL 記述領域が示す領域に設定します。

11. カーソル CUR2 のクローズ

    カーソル CUR2 を閉じます。

12. SQL の動的実行の用意

    DESCRIBE 文で顧客表（CUSTOM）の各列の列名とデータ型，データ長を取得するために，表を検索する SELECT 文を用意します。

13. 列名とデータ型の取得

    顧客表（CUSTOM）の各列のデータ型，データ長を取り出して，ユーザ定義の SQL 記述領域に設定します。また，各列の列名を取り出して，ユーザ列名記述領域に設定します。

14. ユーザ定義の SQL 記述領域の列数の設定

ユーザ定義の SQL 記述領域に，SQL 記述領域の大きさと取得する列の個数を設定します。

15. ユーザ定義の SQL 記述領域のクリア

ユーザ定義の SQL 記述領域の中の，各列に対応した領域をクリアします。

(3) LOB データを操作する例

(a) PAD チャート

プログラム例題 3 の PAD チャートを次の図に示します。

図 7–8 プログラム例題 3 の PAD チャート（1/3）

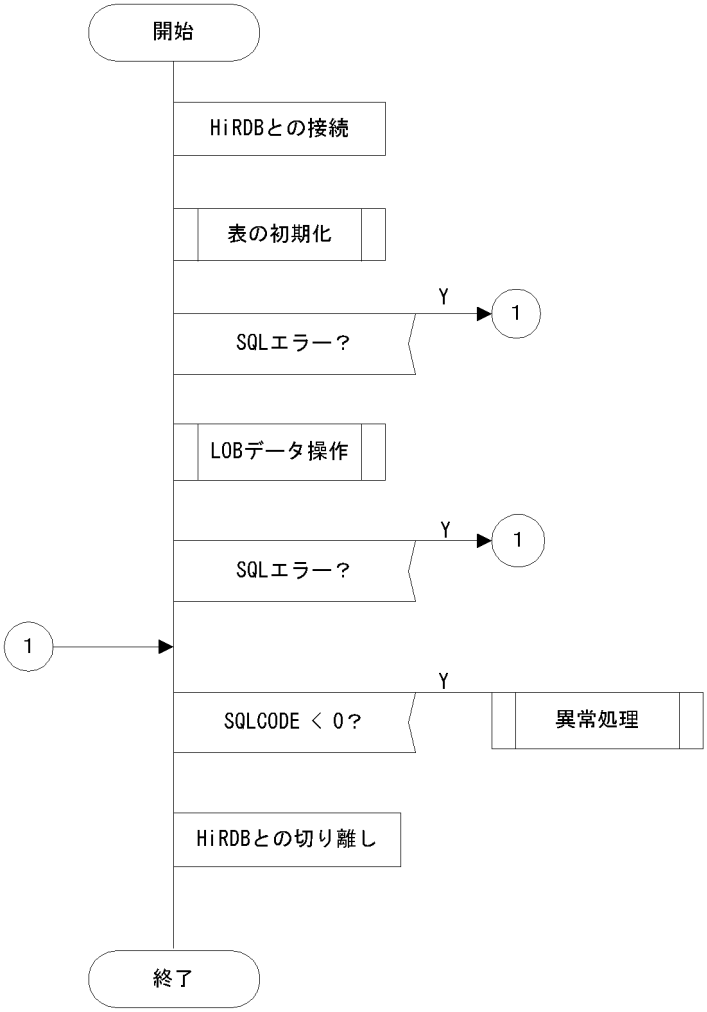
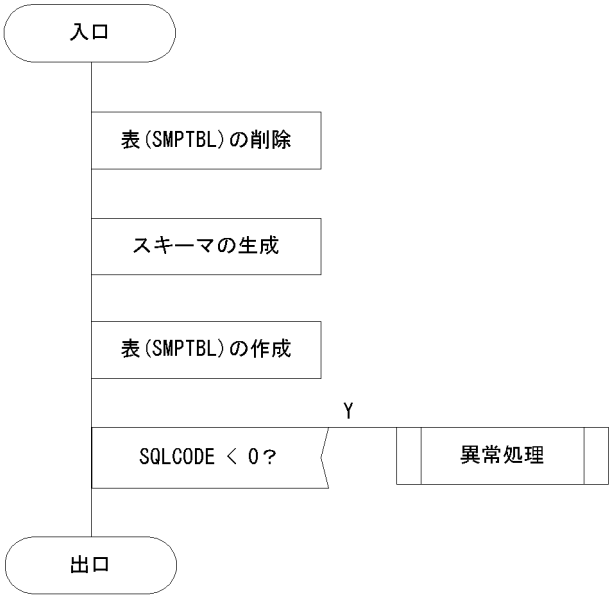


図 7–9　プログラム例題 3 の PAD チャート（2/3）

［表の初期化］



［異常処理］

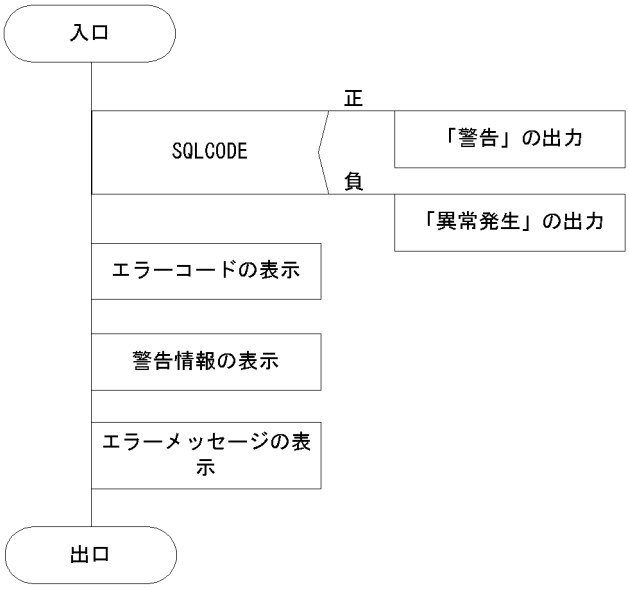
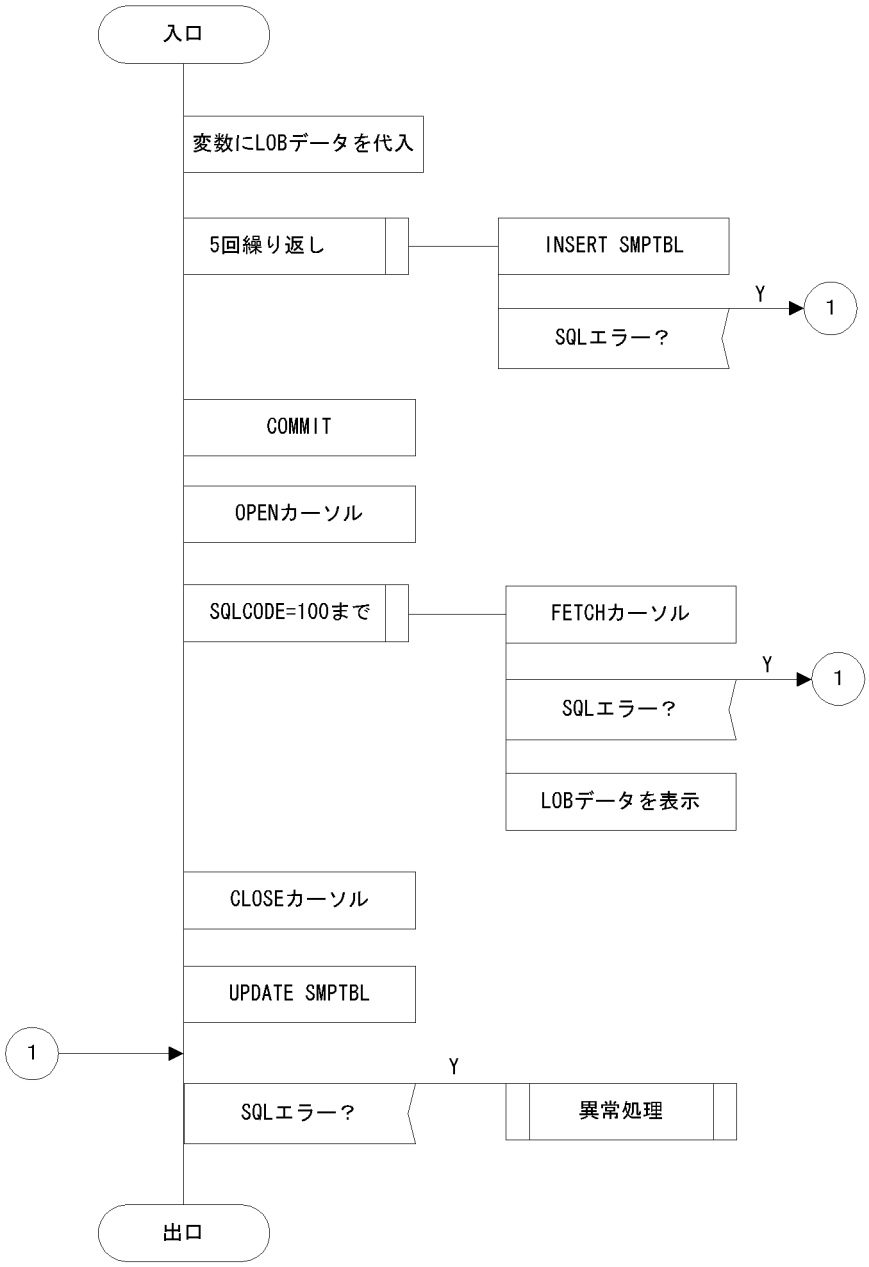


図 7–10　プログラム例題 3 の PAD チャート（3/3）



(b)　コーディング例

プログラム例題 3 のコーディング例を次に示します。

```
 1  /*******************************************************/
 2  /*                                                     */
 3  /* ALL RIGHTS RESERVED,COPYRIGHT (C)1997,HITACHI,LTD. */
 4  /* LICENSED MATERIAL OF HITACHI,LTD.                   */
 5  /*                                                     */
 6  /*******************************************************/
 7
 8
```

```
 9  #include <stdio.h>
10  #include <stdlib.h>
11  #include <stddef.h>
12  #include <ctype.h>
13  #include <string.h>
14
15  static void InitTable();
16  static void TestBlob();
17  static void warning();
18
19
20  /************************************************************/
21  /*  GLOBAL VARIABLE                                         */
22  /************************************************************/
23  short ErrFlg;
24
25  EXEC SQL BEGIN DECLARE SECTION;
26     short XSINT_IN;
27     short XSINT_OUT;
28     long  XINT_IN;
29     long  XINT_OUT;
30     SQL TYPE IS BLOB(16K) XBLOB_IN;                          1
31     SQL TYPE IS BLOB(16K) XBLOB_OUT;                         1
32  EXEC SQL END DECLARE SECTION;
33
34  /*
35   *   name = MAIN
36   *   func =  SAMPLE
37   *   io   =  argc : i :
38   *           argv : i :
39   *   return = 0,-1
40   *   note   =  This program needs "RDUSER02" rdarea on Server.
41   *   date =  98.04.24 by matsushiba
42   */
43  int main(
44  int   argc,
45  char  *argv[])
46  {
47      ErrFlg = 0;
48
49  /************************************************************/
50  /*                                                          */
51  /************************************************************/
52      EXEC SQL
53        WHENEVER SQLERROR goto ERREXIT;
54
55      EXEC SQL
56        WHENEVER SQLWARNING PERFORM :warning;
57
58      EXEC SQL CONNECT;                                        2
59
60
61  /************************************************************/
62  /*  INIT                                                    */
63  /************************************************************/
64      InitTable();                                             3
65      if(ErrFlg < 0){                                          3
66          goto ERREXIT;                                        3
```

```
 67      }                                                                3
 68
 69  /*************************************************/
 70  /*                                               */
 71  /*************************************************/
 72      TestBlob();                                                      4
 73      if(ErrFlg < 0){                                                  4
 74          goto ERREXIT;                                                4
 75      }                                                                4
 76
 77  /*****************************************************************/
 78  /*                                                               */
 79  /*****************************************************************/
 80  ERREXIT:
 81      if(SQLCODE < 0){
 82          printf(":> ERROR HAPPENED!!¥n");
 83          warning();
 84          ErrFlg = -1;
 85      }
 86
 87      EXEC SQL
 88        WHENEVER SQLERROR  CONTINUE;
 89      EXEC SQL
 90        WHENEVER NOT FOUND CONTINUE;
 91      EXEC SQL
 92        WHENEVER SQLWARNING CONTINUE;
 93
 94      EXEC SQL DISCONNECT;                                             5
 95
 96      return(ErrFlg);
 97  }
 98
 99
100  /*****************************************************************/
101  /*  INIT                                                         */
102  /*****************************************************************/
103  static void InitTable()
104  {
105
106  /*****************************************************************/
107  /*                                                               */
108  /*****************************************************************/
109      EXEC SQL
110        WHENEVER SQLERROR  CONTINUE;
111
112      EXEC SQL                                                         6
113       DROP TABLE SMPTBL;                                              6
114                                                                       6
115      EXEC SQL                                                         6
116        CREATE SCHEMA;                                                 6
117
118      printf("## CREATE TABLE¥n");
119
120      EXEC SQL
121        WHENEVER SQLERROR  GOTO INIT_ERROR;
122
123      printf("## CREATE SMPTBL¥n");
124      EXEC SQL                                                         7
```

```
125       CREATE TABLE SMPTBL(CLM1  BLOB(30K) IN RDUSER02,         7
126                           CLM2  SMALLINT,                      7
127                           CLM3  INTEGER);                      7
128
129     return;
130
131 INIT_ERROR:
132     warning();
133     ErrFlg = -1;
134     return;
135 }
136
137
138
139 /*************************************************************/
140 /*  TEST BLOB                                                */
141 /*************************************************************/
142 static void TestBlob()
143 {
144     short cnt;
145
146     EXEC SQL
147       WHENEVER  SQLERROR goto :ExitTestBlob;
148
149     EXEC SQL
150       WHENEVER  SQLWARNING PERFORM :warning;
151
152 /*************************************************************/
153 /*  INSERT                                                   */
154 /*************************************************************/
155     memset(XBLOB_IN.XBLOB_IN_data,
156            0x55,
157            sizeof(XBLOB_IN.XBLOB_IN_data));
158     XBLOB_IN.XBLOB_IN_length = sizeof(XBLOB_IN.XBLOB_IN_data);
159
160     printf("## INSERT ¥n");
161     for(cnt=1; cnt<5; cnt++){
162         XSINT_IN = cnt;
163         XINT_IN = 100+cnt;
164         EXEC SQL                                               8
165           INSERT INTO SMPTBL                                   8
166               VALUES(:XBLOB_IN, :XSINT_IN, :XINT_IN);          8
167     }
168     EXEC SQL COMMIT;
169
170 /*************************************************************/
171 /*  FETCH                                                    */
172 /*************************************************************/
173     printf("## FETCH ¥n");
174
175     EXEC SQL                                                   9
176       DECLARE CUR_BLOB CURSOR FOR                              9
177          SELECT * FROM SMPTBL;                                 9
178
179     EXEC SQL
180       OPEN CUR_BLOB;                                           10
181
182     EXEC SQL
```

```
183       WHENEVER NOT FOUND GOTO FETCH_END;
184
185     for(;;){
186         memset(XBLOB_OUT.XBLOB_OUT_data,
187                0,
188                sizeof(XBLOB_OUT.XBLOB_OUT_data));
189         XBLOB_OUT.XBLOB_OUT_length = 0;
190         EXEC SQL                                                  11
191           FETCH CUR_BLOB INTO :XBLOB_OUT,                         11
192                               :XSINT_OUT,                         11
193                               :XINT_OUT;                          11
194
195             printf("CLM1 XBLOB_length == %d¥n",
196                     XBLOB_OUT.XBLOB_OUT_length);
197             printf("CLM2 = %d¥n", XSINT_OUT);
198             printf("CLM3 = %ld¥n", XINT_OUT);
199     }
200 FETCH_END:
201     EXEC SQL
202       WHENEVER NOT FOUND CONTINUE;
203
204     EXEC SQL
205       CLOSE CUR_BLOB;                                           12
206
207 /*************************************************************/
208 /*  UPDATE                                                   */
209 /*************************************************************/
210     memset(XBLOB_IN.XBLOB_IN_data,
211            0x38,
212            sizeof(XBLOB_IN.XBLOB_IN_data));
213     XBLOB_IN.XBLOB_IN_length = sizeof(XBLOB_IN.XBLOB_IN_data);
214
215     printf("## UPDATE¥n");
216     EXEC SQL
217       UPDATE SMPTBL SET CLM1=:XBLOB_IN;                         13
218
219     EXEC SQL COMMIT;
220
221 /*************************************************************/
222 /*                                                           */
223 /*************************************************************/
224 ExitTestBlob:
225     if(SQLCODE < 0){
226         warning();
227         ErrFlg = -1;
228     }
229     return;
230 }
231
232
233 /*************************************************************/
234 /*  WARNING                                                  */
235 /*************************************************************/
236 static void warning()
237 {
238     if(SQLCODE < 0){
239         printf(">>>ERROR¥n");
240         printf(">>> sqlcode = %d¥n", SQLCODE);
```

```
241  #if defined(HIUXWE2) || defined(WIN32)
242          printf(":> message = %s¥n", SQLERRMC);
243  #else
244          printf(":> message = %Fs¥n", SQLERRMC);
245  #endif
246      }
247      else{
248          printf(">>>WARNING¥n");
249          printf(">>> sqlwarn = %c", SQLWARN0);
250          printf("%c", SQLWARN1);
251          printf("%c", SQLWARN2);
252          printf("%c", SQLWARN3);
253          printf("%c", SQLWARN4);
254          printf("%c", SQLWARN5);
255          printf("%c", SQLWARN6);
256          printf("%c", SQLWARN7);
257          printf("%c", SQLWARN8);
258          printf("%c", SQLWARN9);
259          printf("%c", SQLWARNA);
260          printf("%c", SQLWARNB);
261          printf("%c¥n", SQLWARNC);
262      }
263      return;
264  }
```

＜説明＞

1. LOB 型の埋込み変数の宣言

   書き込み用の LOB 型の埋込み変数(：XBLOB_IN)と，読み取り用の LOB 型の埋込み変数（:XBLOB_OUT）を宣言します。

2. HiRDB への接続

   環境変数 PDUSER に定義されている認可識別子とパスワードを使用してサーバに接続します。

3. 表の初期化

   LOB 型の列を持つ表（SMPTBL）を定義します。

4. LOB データの挿入・検索・更新

   空の表（SMPTBL）に LOB 型の列を含む行を挿入し，すべての行を検索した後，LOB 型の列の内容を新しい LOB データに更新します。

5. HiRDB の切り離し

   UAP をサーバから切り離します。

6. 表（SMPTBL）を作成するための準備

   LOB 型の列を含む表（SMPTBL）を作成するため，同名の表があった場合は削除し，スキーマがないときのためにスキーマを生成します。

7. LOB 型の列を含む表（SMPTBL）を作成する

   LOB 型の列を含む表（SMPTBL）を作成します。LOB データは LOB 専用の RD エリア（RDUSER02）に格納するように定義するので，サーバにユーザ LOB 用 RD エリアを作成しておかなければなりません。ユーザ LOB 用 RD エリアがない場合はエラーとなります。

8. LOB データの追加

   LOB 型の列を持つ表（SMPTBL）に，埋込み変数（:XBLOB_IN，:XINT_IN，:XSINT_IN）に設定した値を追加します。

9. カーソル CUR_BLOB の宣言

   LOB 型の列を含む表（SMPTBL）を検索するために，カーソル CUR_BLOB を宣言します。

10. カーソル CUR_BLOB のオープン

    LOB 型の列を含む表（SMPTBL）の検索行の直前にカーソルを位置づけて，各行を取り出せる状態にします。

11. LOB 型データの取り出し

    LOB 型の列を含む表（SMPTBL）のカーソル CUR_BLOB の示す行を１行取り出し，埋込み変数（:XBLOB_OUT，:XINT_OUT，:XSINT_OUT）に設定します。

12. カーソル CUR_BLOB のクローズ

    カーソル CUR_BLOB を閉じます。

13. LOB データの更新

    表（SMPTBL）の LOB 型の列の値を，埋込み変数（：XBLOB_IN）の値で更新します。

7.3　COBOL 言語による UAP の作成

ここでは，COBOL 言語による埋込み型 UAP の記述規則，及び作成例について説明します。

7.3.1　記述規則

UAP を作成するとき，SQL の文法の規則以外に，名標の付け方や SQL の記述についての規則があります。

(1)　名標の付け方の規則

名標を付ける場合，基本的に COBOL 言語の規則に従います。

COBOL 言語の規則以外に，名標を付けるときの規則を次に示します。

(a)　SQL の予約語

- 大文字でも小文字でも使用できます。
- 大文字と小文字を混在できます。

(b)　ホスト名

- 「SQL」で始まる名標は使用できません。
- ホスト名中のコロンの後ろに空白を記述できます。
- 大文字と小文字は同等に使用できます。
- 大文字と小文字は混在できます。
- 全角と半角の，英字，数字，記号，片仮名，及び空白は異なる文字として扱われます。

なお，使用する埋込み変数，標識変数，及び分岐先の名前の付け方は，名標の付け方の規則，及び COBOL 言語の規則に従います。また，次に示す名標は外部属性を持つため，使用できないので注意してください。

- 大文字の「SQL」で始まる名標
- 小文字の「p_」で始まる名標
- 小文字の「pd」で始まる名標

(2)　SQL の記述規則

1. SQL は，一つの SQL 文ごとに SQL 先頭子（EXEC SQL）と SQL 終了子（END-EXEC）とで囲みます。
   <指定例>

```
  EXEC SQL  SQL文    END-EXEC.
```

2. COBOL 言語と SQL とを同一行に混在できません。

3. SQL は，SQL 先頭子，及び SQL 終了子も含めてすべて B 領域（第 12 欄～72 欄）に記述します。



4. SQL の継続規則は，原則として COBOL 言語の「行のつなぎ」の規則に従います。

   SQL で空白を必ず挿入する箇所，又は空白を挿入できる箇所であれば，自由に行を変えて記述できます。また，複数行にわたって記述することもできます。

   SQL で空白を挿入できない箇所で行を変えるときは，次の行の標識領域にハイフン（-）を記述して，B 領域の任意の欄から行の続きを記述します。

   文字列定数の途中で行を変えるときは，必ず第 72 欄まで記述し，続きを次の行の B 領域に記述します。また，文字列の続きは，最初に引用符又はアポストロフィ（引用符又はアポストロフィの最初に記述した方）を記述し，その引用符又はアポストロフィの次の欄から記述します。

5. SQL 先頭子の直前に，段落の見出しを記述できます。

   なお，段落の見出しを同一行には記述できません。

   <正しい指定例>

```
  OWARI.
      EXEC SQL   SQL文     END-EXEC.
```

   <誤った指定例>

```
  OWARI.  EXEC SQL   SQL文
             END-EXEC.
```

   注　__部分が誤りです。

6. 一つの SQL は，COBOL 言語での一つの命令として扱われます。したがって，SQL が一つの完結文の最後の命令になる場合，SQL 終了子の後ろに終止符と空白を指定します。

   <一つの SQL が一つの完結文の場合の指定例>

```
  EXEC SQL
       SQL文
  END-EXEC.
```

   <SQL が完結文中の最後の命令の場合の指定例>

```
  IF U-IOKBN = '2'
  THEN
      EXEC SQL    SQL文    END-EXEC.
```

   <SQL が完結文の途中の命令の場合の指定例>

```
  IF U-IOKBN = '1'
  THEN
    EXEC SQL    SQL文
    END-EXEC
  ELSE  IF  U-IOKBN = '2'
    THEN  NEXT  SENTENCE.
```

7. SQL 中には注記行を記述できませんが，SQL 先頭子と SQL 終了子の間には，注記行を何行でも記述できます。

   <指定例>

```
EXEC SQL
*ZAIKO表を検索するSELECT文に対して      …注記行
*カーソルを宣言する                        …注記行
  SQL文
END-EXEC.
```

8. 埋込み変数の宣言は，次に示す規則に従います。

   - 埋込み SQL 宣言節は，次のどれかの節内で記述します。
     ・データ部（DATA DIVISION）のファイル節（FILESECTION）
     ・作業場所節（WORKING-STORAGE SECTION）
     ・局所記憶節（LOCAL-STORAGE SECTION）
     ・連絡節（LINKAGE SECTION）
   - SQL のデータ型に対応する埋込み変数については，「SQL のデータ型とデータ記述」を参照してください。
   - 埋込み変数のデータ記述項には，JUSTIFIED，BLANK，及び WHEN ZERO 句を指定できません。
   - レベル番号 66 の再命令項目，及びレベル番号 88 の条件名項目を埋込み変数としては使用できませんが，埋込み SQL 宣言節で定義できます。
   - 埋込み SQL 宣言節内のデータ記述項目の行のつなぎは，COBOL 言語の「行のつなぎ」の規則に従います。
   - 「FILLER」は埋込み変数として使用できません。
   - TYPE 句，TYPEDEF 句，及び SAME AS 句を使用したデータ項目は，埋込み変数として使用できます。
   - REDEFINES 句を使用した場合，再定義する項目と再定義される項目のけた詰めが同じかどうかはチェックしていません。領域の大きさは，どちらか大きい方に合わせます。
   - PICTURE 句を省略し，VALUE 句だけを指定したデータ項目は，埋込み変数として使用できません。
   - -E 又は/E オプションを使用すると，埋込み SQL 宣言節を使用しなくても，宣言したデータ項目を埋込み変数として使用できます。ただし，SQL 文で埋込み変数として使用できるのは，「SQL のデータ型とデータ記述」の形式で宣言したデータ項目だけです。そのほかの形式のデータ項目は，埋込み変数として使用できません。

     データ項目名は，COBOL 言語の文法に従って各データ項目名の有効範囲が判定されます。
   - COBOL2002 のクラス継承機能を使用して，親クラスから継承したデータ項目は，埋込み変数として使用できません。
   - COBOL2002 の Unicode 機能を使用する UAP では，SQL プリプロセサ実行時に-XU16 又は/XU16 オプションを指定すると，UTF-16 の文字データを格納する日本語項目を埋込み変数として使用できます。日本語項目を使用した埋込み変数は，SQL 文中で混在文字データ型（MCHAR 又は

MVARCHAR）に対応する埋込み変数が使用できる箇所なら，どこにでも埋め込むことができます。なお，使用できる文字の範囲は COBOL2002 の Unicode 機能がサポートする範囲内に限定されます。

COBOL2002 の Unicode 機能を使用した UAP の実行については「COBOL2002 の Unicode 機能を使用した UAP の実行」を参照してください。

9. 標識変数の宣言は，次に示す規則に従います。
   - 標識変数は，レベル番号 01～49 の基本項目，又はレベル番号 77 の独立項目であることが前提になります。
   - 標識変数のデータ記述項については，「SQL のデータ型とデータ記述」を参照してください。
   - 標識変数のデータ記述項目には，SIGN，JUSTIFIED，BLANK，及び WHEN ZERO 句を指定できません。
   - 「FILLER」は，標識変数名として使用できません。
   - -E 又は/E オプションを使用すると，埋込み SQL 宣言節を使用しなくても，宣言したデータ項目を標識変数として使用できます。ただし，SQL 文で標識変数として使用できるのは，「SQL のデータ型とデータ記述」の形式で宣言したデータ項目だけです。そのほかの形式のデータ項目は，標識変数として使用できません。

     データ項目名は，COBOL 言語の文法に従って各データ項目名の有効範囲が判定されます。
   - COBOL2002 のクラス継承機能を使用して，親クラスから継承したデータ項目は，標識変数として使用できません。
10. COBOL 言語での SQL を記述できる部（DIVISION）を次の表に示します。

表 7–3　COBOL 言語での SQL を記述できる部

SQL 文		データ部※	手続き部
定義系 SQL		×	○
操作系 SQL		×	○
制御系 SQL		×	○
埋込み言語	BEGIN DECLARE SECTION	○	×
	END DECLARE SECTION	○	×
	COPY	○	○
	WHENEVER	×	○
	DECLARE CONNECTION HANDLE UNSET	×	×
	COMMAND EXECUTE	×	×
	上記以外	×	○

（凡例）

○：記述できます。

×：記述できません。

注※

作業場所節，ファイル節，又は連絡節の中の一つを指します。

11. WHENEVER 文，及びカーソル宣言は宣言文なので，IF 命令，及び EVALUATE 命令中には記述できません。

12. SQL 先頭子と SQL 終了子で囲まれた SQL 文の途中には，コンパイルリスト出力制御（EJECT，SKIP1，SKIP2，SKIP3，又は TITLE）を記述しないでください。表名や列名として EJECT，SKIP1，SKIP2，SKIP3，又は TITLE を使用する場合は，必ず引用符で囲んでください。なお，表名や列名などが語句の一部分として EJECT，SKIP1，SKIP2，SKIP3，又は TITLE を含む場合は，囲む必要はありません。

13. 注釈の規則を次に示します。

- SQL 先頭子から SQL 終了子までの間に記述した囲み注釈（/*～*/）及び単純注釈（-- ～ 改行）は SQL プリプロセサが削除します。ただし，SQL 最適化指定（/*>>～<<*/）は削除しないで，SQL 文として扱います。また，引用符（"）やアポストロフィ（'）で囲まれた部分にある「/*～*/」及び「--～改行」は削除しません。
- SQL 先頭子がある行と SQL 終了子がある行の間にある注釈行，デバッグ行はポストソースから除かれます。
- 囲み注釈及び SQL 最適化指定を複数行にわたって記述している場合は，*/が現れるまで行の先頭は B 領域の先頭から始まっているとみなされます。
- プリプロセスオプション-Xs 又は/Xs を指定した場合にだけ，SQL 先頭子から SQL 終了子までの間に単純注釈を記述できます。
- 単純注釈の開始を示す「--」は B 領域に記述します。「--」から行の末尾までにあるすべての文字が注釈になります。行の末尾が見出し領域にあってもかまいません。
- 行末に囲み注釈又は単純注釈がある場合に，次の行の標識領域に継続標識「-」を指定しないでください。この場合，継続標識を指定しても，注釈の前にある字句と次の行にある字句を，空白を挟まずにつなぐことはできません。
- 単純注釈と SQL 終了子を同じ行に記述することはできません。

<単純注釈の正しい指定例>

```
EXEC SQL -- 注釈1
    CALL PROC1(0) -- 注釈2
END-EXEC.
```

<単純注釈の誤った指定例 1>

```
EXEC SQL CALL PROC1(0) -- 注釈1  END-EXEC.
```

この場合，単純注釈の中に END-EXEC があるため，END-EXEC が SQL 終了子としてみなされません。

<単純注釈の誤った指定例 2>

```
EXEC SQL CALL PROC1(0) END-EXEC. -- 注釈1
```

この場合，「-- ～ 改行」が SQL 先頭子から SQL 終了子（END-EXEC）の範囲外にあるため，「-- ～ 改行」が注釈としてみなされません。

SQL 文中での囲み注釈，単純注釈，及び SQL 最適化指定については，マニュアル「HiRDB SQL リファレンス」を参照してください。

14. 行内注記（*>）を使用できます。ただし，SQL 先頭子から SQL 終了子までの間は，行内注記を使用できません。使用した場合，行内注記ではなく文字列として扱われます。

15. 埋込み SQL 宣言節，及び SQL に記述した行では，タブコードを長さ 1 文字として扱います。また，-E2 又は/E2，及び-E3 又は/E3 オプションを使用した場合，すべてのデータ部が同様に，タブコードを長さ 1 文字として扱います。

16. COBOL2002 のオブジェクト指向機能を使用する場合，「SQL の記述規則」も適用されます。

17. ソースに登録集原文を引き込む方法として，COBOL の COPY 文又は INCLUDE 文と，SQL の COPY 文があります。それぞれ登録集原文中に記述できる項目を次に示します。

記述項目		登録集原文を引き込む方法	
		COBOL の COPY 文 又は INCLUDE 文	SQL の COPY 文
SQL 文	COPY	×	×
	COPY 以外	×	○
埋込み変数と標識変数の宣言		×	○

（凡例）

○：記述できます。

×：記述できません。

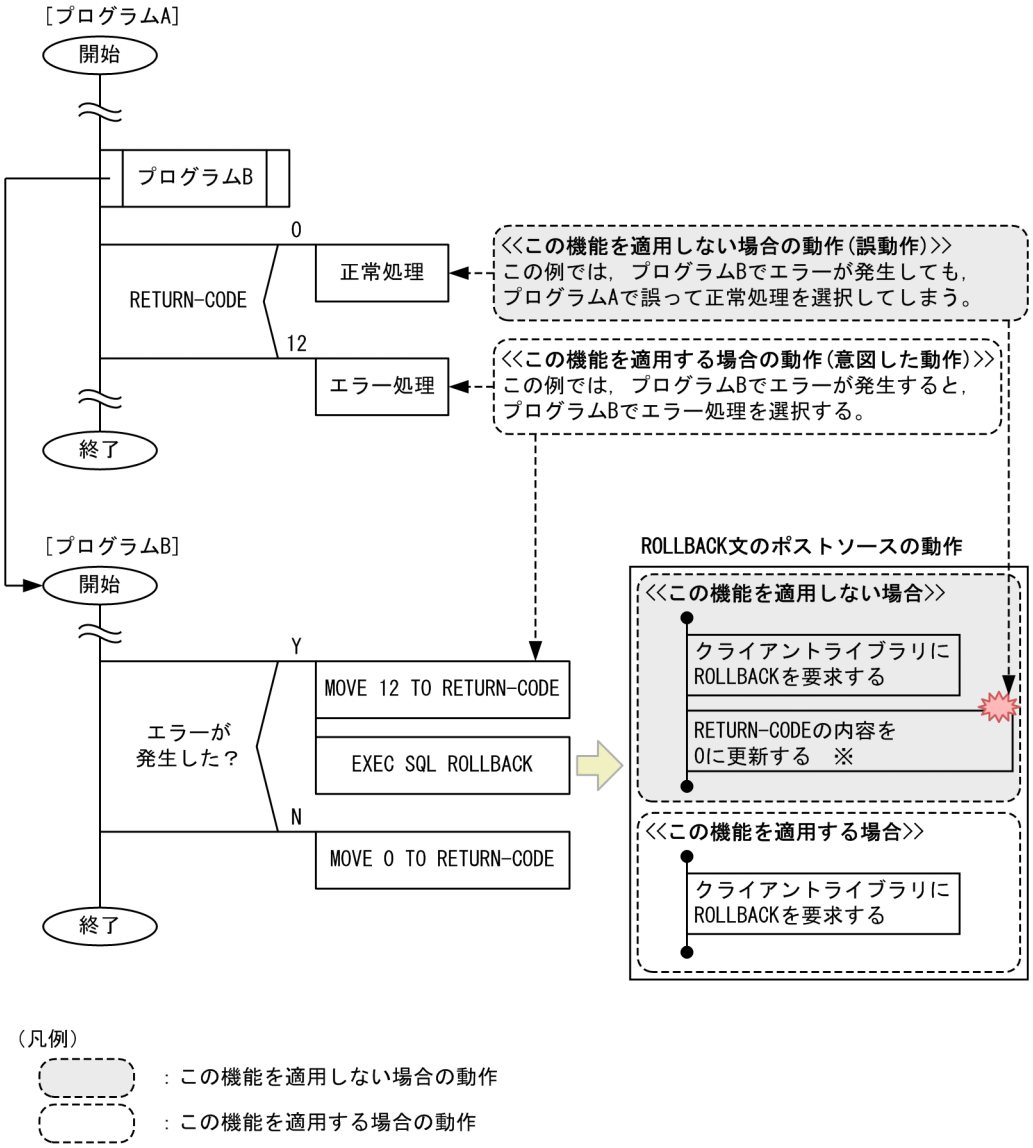
18. -Xr 又は/Xr オプションを指定しない場合，SQL 文の種類によって，SQL 文実行時の RETURN-CODE 特殊レジスタの更新有無が変わります。SQL 文の種類による RETURN-CODE 特殊レジスタの更新有無を次の表に示します。

表 7–4　SQL 文の種類による RETURN-CODE 特殊レジスタの更新有無

SQL 文		RETURN-CODE 特殊レジスタ更新有無	
分類	種類	-Xr 又は/Xr 指定無し	-Xr 又は/Xr 指定有り
定義系 SQL	すべて	更新する	更新しない
操作系 SQL	DECLARE CURSOR 文	更新しない	
	上記以外	更新する	
制御系 SQL	すべて	更新する	
埋込み言語	GET DIAGNOSTICS 文	更新する	
	DECLARE AUDIT INFO SET 文		
	上記以外	更新しない	

-Xr 又は/Xr オプション指定有無によるプログラム動作の差異の例を，次の図に示します。

図 7–11　-Xr 又は/Xr オプション指定有無によるプログラム動作の差異の例



注※　この例ではSQL文の実行時にRETURN-CODEの内容を0に更新していますが，
0以外の値に更新することもあります。

7.3.2　プログラム例題

COBOL 言語による埋込み型 UAP のプログラム例題を示します。

なお，SQL の文法の詳細については，マニュアル「HiRDB SQL リファレンス」を参照してください。

(1)　基本的な操作の例

(a)　PAD チャート

プログラム例題 4 の PAD チャートを次の図に示します。

図 7–12　プログラム例題 4 の PAD チャート（1/3）

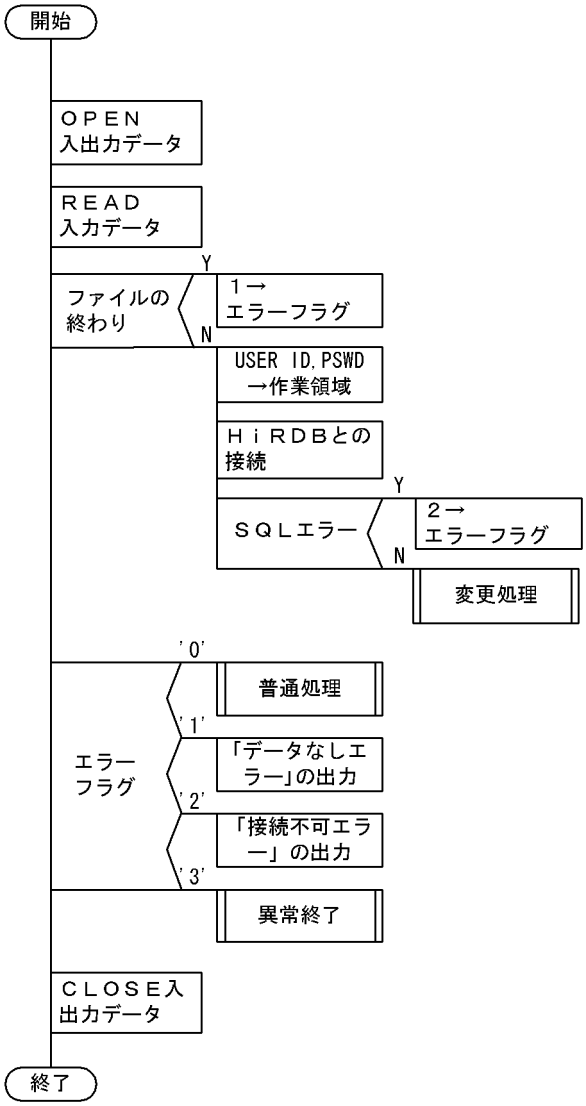


図 7–13　プログラム例題 4 の PAD チャート（2/3）

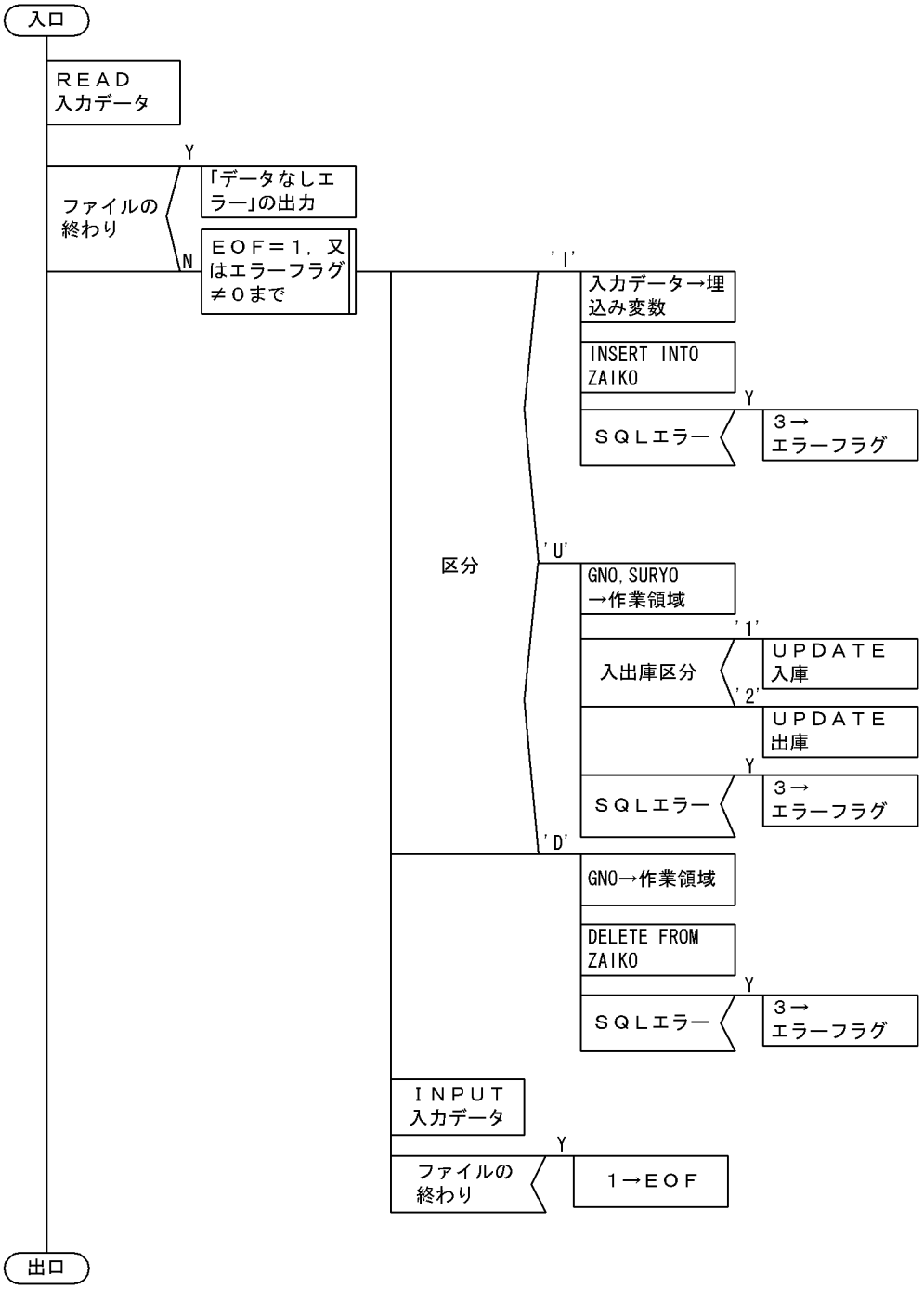
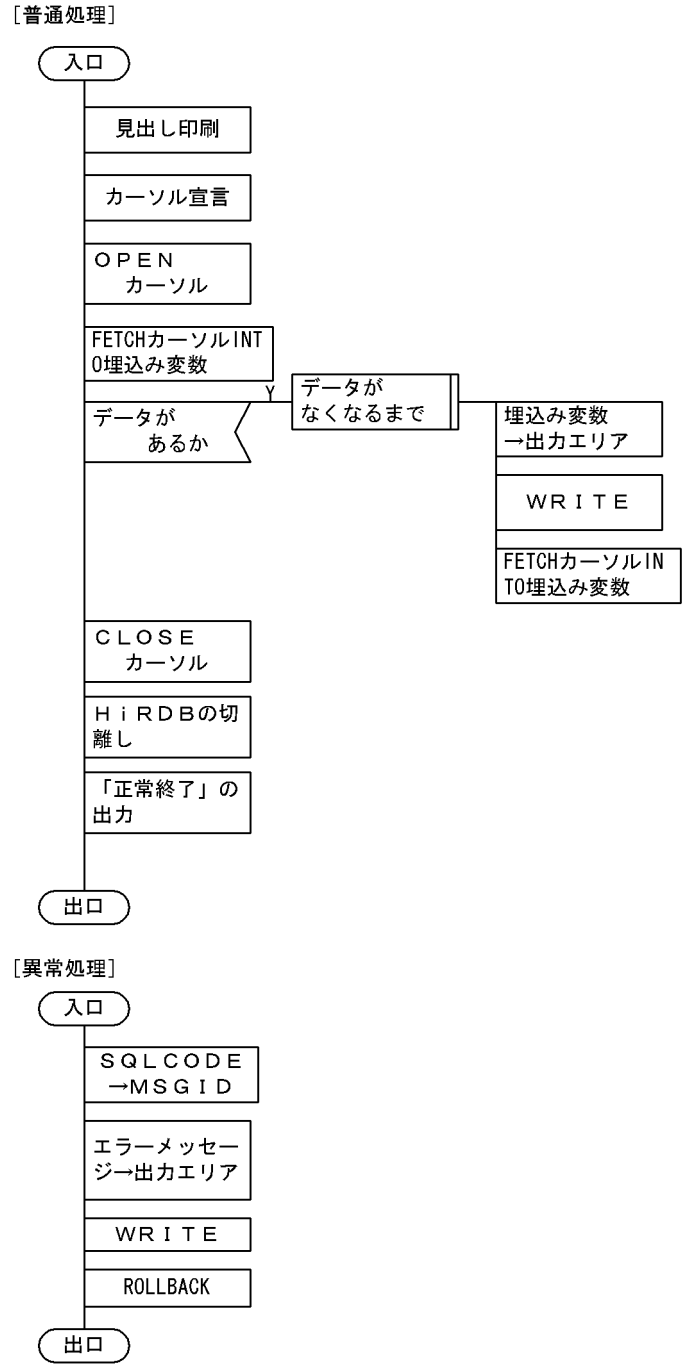


図 7-14　プログラム例題 4 の PAD チャート（3/3）



(b)　コーディング例

プログラム例題 4 のコーディング例を次に示します。

```
000010*ZAIKO KANRI PROG.
000020*
000030*
000040* ALL RIGHTS RESERVED,COPYRIGHT (C)1997 HITACHI,LTD.
000050* LICENSED MATERIAL OF HITACHI,LTD.
000060*
000070 IDENTIFICATION DIVISION.
000080 PROGRAM-ID. ECOBUAP.
```

```
000090*
000100 ENVIRONMENT DIVISION.
000110 CONFIGURATION SECTION.
000120 SOURCE-COMPUTER. HITAC.
000130 OBJECT-COMPUTER. HITAC.
000140 INPUT-OUTPUT SECTION.
000150 FILE-CONTROL.
000160     SELECT INPUT-CARD-FILE
000170       ASSIGN TO  DISK
000180       ORGANIZATION IS LINE SEQUENTIAL.
000190     SELECT PRINT-ZAIKO-FILE
000200       ASSIGN TO  LP.
000210*
000220 DATA DIVISION.
000230 FILE SECTION.
000240 FD  INPUT-CARD-FILE
000250                    DATA RECORD USER-CARD-REC I-ZAIKO-REC.
000260*
000270 01  USER-CARD-REC.
000280     02 IUSERID          PIC X(20).
000290     02 IPSWD            PIC X(20).
000300     02 FILLER           PIC X(40).
000310*
000320 01  I-ZAIKO-REC.
000330     02 IKUBUN           PIC X(1).
000340     02 FILLER           PIC X(2).
000350     02 ISCODE           PIC X(4).
000360     02 FILLER           PIC X(2).
000370     02 ISNAME           PIC N(8).
000380     02 ICOL             PIC N(1).
000390     02 ITANKA           PIC X(9).
000400     02 IGRYO            PIC X(9).
000410     02 IIOKBN           PIC X(1).
000420     02 FILLER           PIC X(34).
000430*
000440 FD  PRINT-ZAIKO-FILE RECORDING MODE IS F
000450                      LABEL RECORD IS OMITTED
000460                      DATA RECORD PRINT-ZAIKO-REC.
000470 01  PRINT-ZAIKO-REC      PIC X(132).
000480*
000490 WORKING-STORAGE SECTION.
000500*
000510     EXEC SQL                                          1
000520          BEGIN DECLARE SECTION                        1
000530     END-EXEC.                                         1
000540 77  XUSERID            PIC X(30).                     1
000550 77  XPSWD              PIC X(30).                     1
000560 77  XSCODE             PIC X(4)  VALUE '0000'.        1
000570 77  XSNAME             PIC N(8).                      1
000580 77  XCOL               PIC N(1).                      1
000590 77  XTANKA             PIC S9(9) COMP.                1
000600 77  XGRYO              PIC S9(9) COMP.                1
000610* INDICATOR VARIABLE                                   1
000620 77  XISCODE            PIC S9(4) COMP  VALUE 1040.    1
000630 77  XISNAME            PIC S9(4) COMP  VALUE 1050.    1
000640 77  XICOL              PIC S9(4) COMP  VALUE 1060.    1
000650 77  XITANKA            PIC S9(4) COMP  VALUE 1070.    1
000660 77  XIGRYO             PIC S9(4) COMP  VALUE 1080.    1
```

```
000670*                                                          1
000680*                                                          1
000690     EXEC SQL                                              1
000700          END DECLARE SECTION                              1
000710     END-EXEC.                                             1
000720*
000730 01  MIDASHI-REC.
000740     02 FILLER          PIC X(13) VALUE SPACE.
000750     02 FILLER          PIC X(32)
000760          VALUE '******  ZAIKO TABLE LIST  ******'.
000770     02 FILLER          PIC X(87) VALUE SPACE.
000780*
000790 01  RETSUMEI-REC.
000800     02 FILLER          PIC X(14) VALUE SPACE.
000810     02 FILLER          PIC X(9)  VALUE 'SCODE'.
000820     02 FILLER          PIC X(16) VALUE 'SNAME'.
000830     02 FILLER          PIC X(8)  VALUE 'COLOR'.
000840     02 FILLER          PIC X(8)  VALUE 'TANKA'.
000850     02 FILLER          PIC X(8)  VALUE 'SURYO'.
000860     02 FILLER          PIC X(69) VALUE SPACE.
000870*
000880 01  LINE-REC.
000890     02 FILLER          PIC X(14) VALUE SPACE.
000900     02 FILLER          PIC X(9)  VALUE '-----  '.
000910     02 FILLER          PIC X(16) VALUE '--------  '.
000920     02 FILLER          PIC X(8)  VALUE '------- '.
000930     02 FILLER          PIC X(8)  VALUE '------- '.
000940     02 FILLER          PIC X(8)  VALUE '------- '.
000950     02 FILLER          PIC X(69) VALUE SPACE.
000960*
000970 01  SELECT-OUT-REC.
000980     02 FILLER          PIC X(14) VALUE SPACE.
000990     02 O-SCODE         PIC X(5).
001000     02 FILLER          PIC X(2)  VALUE SPACE.
001010     02 O-KANJI         CHARACTER TYPE KEIS.
001020        03 O-SNAME         PIC N(8).
001030        03 FILLER          PIC X(2)  VALUE SPACE.
001040        03 O-COL           PIC N(1).
001050        03 FILLER          PIC X(6)  VALUE SPACE.
001060        03 O-TANKA         PIC X(8)  JUST RIGHT.
001070        03 FILLER          PIC X(2)  VALUE SPACE.
001080        03 O-GRYO          PIC X(8)  JUST RIGHT.
001090        03 FILLER          PIC X(69) VALUE SPACE.
001100 77  O-SCODE-NULL       PIC X(5) VALUE '*****'.
001110 77  O-SNAME-NULL       PIC N(8) VALUE NC'------'.
001120 77  O-COL-NULL         PIC N(1) VALUE NC'-'.
001130 77  O-TANKA-NULL       PIC X(8) VALUE '********'.
001140 77  O-GRYO-NULL        PIC X(8) VALUE '********'.
001150*
001160 01  I-CARD-ERROR-REC.
001170     02 FILLER          PIC X(14) VALUE SPACE.
001180     02 FILLER          PIC X(41)
001190          VALUE '***  ERROR  ***  NO CARD FOR CONNECT  ***'.
001200     02 FILLER          PIC X(77) VALUE SPACE.
001210*
001220 01  CONNECT-ERROR-REC.
001230     02 FILLER          PIC X(14) VALUE SPACE.
001240     02 FILLER          PIC X(45)
```

```
001250           VALUE '***  ERROR  ***  CANNOT CONNECT  ***  CODE = '.
001260     02 CNCT-EC          PIC X(5).
001270     02 FILLER            PIC X(68) VALUE SPACE.
001280*
001290 01  NORMAL-END-REC.
001300     02 FILLER            PIC X(14) VALUE SPACE.
001310     02 FILLER            PIC X(22)
001320           VALUE '***  NORMAL ENDED  ***'.
001330     02 FILLER            PIC X(96) VALUE SPACE.
001340*
001350 01  SQLERR-PRINT-REC.
001360     02 FILLER            PIC X(14) VALUE SPACE.
001370     02 FILLER            PIC X(34)
001380           VALUE '***  HiRDB SQL ERROR MESSAGE-ID = '.
001390     02 RC-MSGID          PIC X(8).
001400     02 FILLER            PIC X(14) VALUE ' SQLERRORMC ='.
001500     02 RC-SQLERRMC       PIC X(62).
001510*
001520 01  WSQLCODE             PIC -(10)9.
001530*
001540 01  WMSGID.
001550     02 FILLER            PIC X(8).
001560     02 MSGID             PIC X(3).
001570*
001580 01  ERRORMSGID.
001590     02 FILLER            PIC X(5) VALUE 'KFPA1'.
001600     02 E-MSGID           PIC X(4).
001610     02 FILLER            PIC X(2) VALUE '-E'.
001620*
001630 01  EOF                  PIC X(1) VALUE '0'.
001640 01  ERR-FLG              PIC X(1) VALUE '0'.
001650*
001660*
001670 PROCEDURE DIVISION.
001680 MAIN SECTION.
001690 M-1.
001700     OPEN INPUT  INPUT-CARD-FILE
001710                 OUTPUT PRINT-ZAIKO-FILE.
001720     READ INPUT-CARD-FILE
001730           AT END
001740             MOVE '1' TO ERR-FLG
001750             GO TO M-3
001760     END-READ.
001770     MOVE IUSERID TO XUSERID.
001780     MOVE IPSWD TO XPSWD.
001790*
001800     EXEC SQL                                          (a) 2
001810           WHENEVER SQLERROR                           (a) 2
001820              GO TO M-2                                (a) 2
001830     END-EXEC.                                         (a) 2
001840     EXEC SQL                                          (b) 2
001850           CONNECT USER :XUSERID USING :XPSWD          (b) 2
001860     END-EXEC.                                         (b) 2
001870     PERFORM HENKOU.
001880     GO TO M-3.
001890 M-2.
001900     MOVE '2' TO ERR-FLG.
001910*
```

```
001920 M-3.
001930     EVALUATE ERR-FLG
001940       WHEN '0'
001950         PERFORM FUTSUU
001960       WHEN '1'
001970         WRITE PRINT-ZAIKO-REC
001980           FROM I-CARD-ERROR-REC
001990           AFTER ADVANCING 2 LINES
002000       WHEN '2'
002010         MOVE SQLCODE  TO CNCT-EC
002020         WRITE PRINT-ZAIKO-REC
002030           FROM CONNECT-ERROR-REC
002040           AFTER ADVANCING 2 LINES
002050       WHEN '3'
002060         PERFORM IJYOU
002070     END-EVALUATE.
002080 M-4.
002090     CLOSE INPUT-CARD-FILE
002100           PRINT-ZAIKO-FILE.
002110 M-EX.
002120     EXEC SQL
002130        WHENEVER SQLERROR   CONTINUE
002140     END-EXEC.
002150     EXEC SQL
002160        WHENEVER NOT FOUND  CONTINUE
002170     END-EXEC
002180     EXEC SQL
002190        WHENEVER SQLWARNING CONTINUE
002200     END-EXEC.
002210     EXEC SQL
002220        DISCONNECT
002230     END-EXEC.
002240     GOBACK.
002250 HENKOU SECTION.
002260 H-1.
002270     READ INPUT-CARD-FILE
002280       AT END
002290         MOVE '1' TO ERR-FLG
002300     END-READ.
002310     EXEC SQL
002320          WHENEVER SQLERROR
002330            GO TO H-2
002340     END-EXEC.
002350     PERFORM UNTIL EOF = '1' OR ERR-FLG NOT = '0'
002360       EVALUATE IKUBUN
002370         WHEN 'I'
002380           PERFORM TSUIKA
002390         WHEN 'U'
002400           PERFORM KOUSHIN
002410         WHEN 'D'
002420           PERFORM SAKUJO
002430       END-EVALUATE
002440       READ INPUT-CARD-FILE
002450         AT END
002460           MOVE '1' TO EOF
002470       END-READ
002480     END-PERFORM.
002490     GO TO H-EX.
```

```
002500 H-2.
002510     MOVE '3' TO ERR-FLG.
002520 H-EX.
002530     EXIT.
002540*
002550 TSUIKA SECTION.
002560 T-1.
002570     MOVE ISCODE  TO XSCODE.
002580     MOVE ISNAME  TO XSNAME.
002590     MOVE ICOL    TO XCOL.
002600     MOVE ITANKA  TO XTANKA.
002610     MOVE IGRYO   TO XGRYO.
002620     EXEC SQL
002610           WHENEVER SQLERROR GO TO T-2
002620     END-EXEC.
002630     EXEC SQL                                                    3
002640           INSERT INTO ZAIKO(SCODE, SNAME, COL, TANKA, ZSURYO)   3
002650              VALUES(:XSCODE, :XSNAME, :XCOL, :XTANKA, :XGRYO)   3
002660     END-EXEC.                                                   3
002670     GO TO T-EX.
002680 T-2.
002690     MOVE '3' TO ERR-FLG.
002700 T-EX.
002710     EXIT.
002720 KOUSHIN SECTION.
002730 K-1.
002740     MOVE ISCODE TO XSCODE.
002750     MOVE IGRYO  TO XGRYO.
002760     EXEC SQL
002770           WHENEVER SQLERROR GO TO K-2
002780     END-EXEC.
002790     EVALUATE IIOKBN
002800       WHEN '1'                                                    4
002810         EXEC SQL                                            (a) 4
002820               UPDATE ZAIKO SET ZSURYO = ZSURYO + :XGRYO     (a) 4
002830                  WHERE SCODE=:XSCODE                        (a) 4
002840         END-EXEC                                            (a) 4
002850       WHEN '2'                                                    4
002860         EXEC SQL                                            (b) 4
002870               UPDATE ZAIKO SET ZSURYO = ZSURYO - :XGRYO     (b) 4
002880                  WHERE SCODE=:XSCODE                        (b) 4
002890         END-EXEC                                            (b) 4
002900     END-EVALUATE.
002910     GO TO K-EX.
002920 K-2.
002930     MOVE '3' TO ERR-FLG.
002940 K-EX.
002950     EXIT.
002960*
002970 SAKUJO SECTION.
002980 S-1.
002990     MOVE ISCODE TO XSCODE.
003010     EXEC SQL
003020           WHENEVER SQLERROR GO TO S-2
003030     END-EXEC.
003040     EXEC SQL                                                  5
003050           DELETE FROM ZAIKO                                   5
003060              WHERE SCODE=:XSCODE                              5
```

```
003070     END-EXEC.                                                  5
003080     GO TO S-EX.
003090 S-2.
003100     MOVE '3' TO ERR-FLG.
003110 S-EX.
003120     EXIT.
003130*
003140 FUTSUU SECTION.
003150 F-0.
003160     WRITE PRINT-ZAIKO-REC
003170        FROM MIDASHI-REC
003180        AFTER ADVANCING 4 LINES.
003190     WRITE PRINT-ZAIKO-REC
003200        FROM RETSUMEI-REC
003210        AFTER ADVANCING 2 LINES.
003220     WRITE PRINT-ZAIKO-REC
003230        FROM LINE-REC
003240        AFTER ADVANCING 2 LINES.
003250 F-1.
003260     EXEC SQL
003270           WHENEVER SQLERROR GO TO F-4
003280     END-EXEC.
003290     EXEC SQL                                                   (a) 6
003300           DECLARE CR1 CURSOR FOR                               (a) 6
003310             SELECT SCODE,SNAME,COL,TANKA,ZSURYO FROM ZAIKO     (a) 6
003320     END-EXEC.                                                  (a) 6
003330     EXEC SQL                                                   (b) 6
003340           OPEN CR1                                             (b) 6
003350     END-EXEC.                                                  (b) 6
003360 F-2.
003370     EXEC SQL                                                   (a) 7
003380           WHENEVER NOT FOUND                                   (a) 7
003390              GO TO F-3                                         (a) 7
003400     END-EXEC.                                                  (a) 7
003410     EXEC SQL                                                   (b) 7
003420           FETCH CR1                                            (b) 7
003430              INTO :XSCODE:XISCODE, :XSNAME:XISNAME,            (b) 7
003440                :XCOL:XICOL, :XTANKA:XITANKA, :XGRYO:XIGRYO     (b) 7
003450     END-EXEC.                                                  (b) 7
003460     EXEC SQL
003470          WHENEVER NOT FOUND
003480             CONTINUE
003490     END-EXEC.
003500     IF XISCODE IS >= 0 THEN
003510       MOVE XSCODE TO O-SCODE
003520     ELSE
003530       MOVE O-SCODE-NULL TO O-SCODE
003540     END-IF.
003550     IF XISNAME IS >= 0 THEN
003560       MOVE XSNAME TO O-SNAME
003570     ELSE
003580       MOVE O-SNAME-NULL TO O-SNAME
003590     END-IF.
003600     IF XICOL IS >= 0 THEN
003610       MOVE XCOL TO O-COL
003620     ELSE
003630       MOVE O-COL-NULL TO O-COL
003640     END-IF.
```

```
003650     IF XITANKA IS >= 0 THEN
003660       MOVE XTANKA TO O-TANKA
003670     ELSE
003680       MOVE O-TANKA-NULL TO O-TANKA
003690     END-IF.
003700     IF XIGRYO IS >= 0 THEN
003710       MOVE XGRYO TO O-GRYO
003720     ELSE
003730       MOVE O-GRYO-NULL TO O-GRYO
003740     END-IF.
003750     WRITE PRINT-ZAIKO-REC
003760           FROM SELECT-OUT-REC
003770           AFTER ADVANCING 2 LINES.
003780     GO TO F-2.
003790 F-3.
003800     EXEC SQL
003810        WHENEVER SQLERROR   CONTINUE
003820     END-EXEC.
003830     EXEC SQL
003840        WHENEVER NOT FOUND  CONTINUE
003850     END-EXEC
003860     EXEC SQL
003870        WHENEVER SQLWARNING CONTINUE
003880     END-EXEC.
003890     EXEC SQL                                              (a) 8
003900        CLOSE CR1                                          (a) 8
003910     END-EXEC.                                             (a) 8
003920*
003930     EXEC SQL                                              (b) 8
003940        COMMIT                                             (b) 8
003950     END-EXEC.                                             (b) 8
003960*
003970     WRITE PRINT-ZAIKO-REC
003980           FROM NORMAL-END-REC
003990           AFTER ADVANCING 2 LINES.
004000     GO TO F-EX.
004010 F-4.
004020     PERFORM IJYOU.
004030 F-EX.
004040     EXIT.
004050 IJYOU SECTION.
004060 I-1.
004070     MOVE SQLCODE TO WSQLCODE.
004080     MOVE WSQLCODE TO WMSGID.
004090     MOVE MSGID TO E-MSGID.
004100     MOVE ERRORMSGID TO RC-MSGID.
004110     MOVE SQLERRMC TO RC-SQLERRMC.
004120     WRITE PRINT-ZAIKO-REC
004130           FROM SQLERR-PRINT-REC
004140           AFTER ADVANCING 2 LINES.
004150     EXEC SQL                                          (a) 9
004160        WHENEVER SQLERROR   CONTINUE                   (a) 9
004170     END-EXEC.                                         (a) 9
004180     EXEC SQL                                          (a) 9
004190        WHENEVER NOT FOUND  CONTINUE                   (a) 9
004200     END-EXEC.                                         (a) 9
004210     EXEC SQL                                          (a) 9
004220        WHENEVER SQLWARNING CONTINUE                   (a) 9
```

```
004230     END-EXEC.                                                    (a) 9
004240     EXEC SQL                                                     (b) 9
004250        ROLLBACK                                                  (b) 9
004260     END-EXEC.                                                    (b) 9
004270 I-EX.
004280     EXIT.
```

＜説明＞

1. 埋込み SQL 宣言節の始まりと終わり

   UAP 中で使用する変数を，BEGIN DECLARE SECTION と END DECLARE SECTION とで囲んで，埋込み SQL 宣言節の始まりと終わりを示します。

2. HiRDB との接続

   (a) 特異状態発生時の指定

   以下の SQL の実行後に，エラー（SQLERROR）が発生した場合の処理として，分岐先（M-2）を指定します。

   (b) HiRDB への接続

   HiRDB に認可識別子（XUSERID）及びパスワード（XPSWD）を連絡して，UAP が HiRDB を使用できる状態にします。

3. 在庫表への行の追加

   在庫表の各列に，埋込み変数に読み込まれた値を追加します。

4. 在庫表の行の更新

   (a) 入庫

   在庫表から，埋込み変数（:XGNO）に読み込んだ品番をキーとして，更新する行を検索します。検索した行の数量（SURYO）の値に，埋込み変数（:XSURYO）に読み込んだ値を加算して，行を更新します。

   (b) 在庫

   在庫表から，埋込み変数(:XGNO)に読み込んだ品番をキーとして，更新する行を検索します。検索した行の数量（SURYO）の値に，埋込み変数（:XSURYO）に読み込んだ値を減算して，行を更新します。

5. 在庫表の行の削除

   在庫表から，埋込み変数(:XGNO)に読み込んだ品番をキーとして，それと等しいキーを持つ行を削除します。

6. カーソル CR1 の宣言とオープン

   (a) カーソル CR1 の宣言

   在庫表（ZAIKO）の行を検索するために，カーソル CR1 を宣言します。

   (b) カーソル CR1 のオープン

   在庫表（ZAIKO）の検索行の直前にカーソルを位置づけて，行を取り出せる状態にします。

7. 在庫表の行の取り出し

(a) 特異状態発生時の処理の指定

以下の在庫表の検索で，FETCH 文で取り出す行がない場合（NOT FOUND）の処理として，分岐先（M-３）を指定します。

(b) FETCH 文の実行

在庫表（ZAIKO）から，カーソル CR1 の示す行を１行取り出して，各埋込み変数に設定します。

8. トランザクションの終了

(a) カーソル CR1 のクローズ

カーソル CR1 を閉じます。

(b) トランザクションの終了

現在のトランザクションを正常終了させて，そのトランザクションによるデータベースへの追加，更新，削除の結果を有効にします。

9. トランザクションの取り消し

(a) 特異状態発生時の処理の指定

以下の SQL の実行でエラー（SQLERROR）や警告（SQLWARNING）が発生した場合，何もしないで次の命令に進むことを指定します。

(b) トランザクションの取り消し

現在のトランザクションを取り消して，そのトランザクションによるデータベースへの追加，更新，削除の結果を無効にします。

(2) 行インタフェースを使用した例

(a) PAD チャート

プログラム例題 5 の PAD チャートを次の図に示します。

図 7–15　プログラム例題 5 の PAD チャート（1/4）

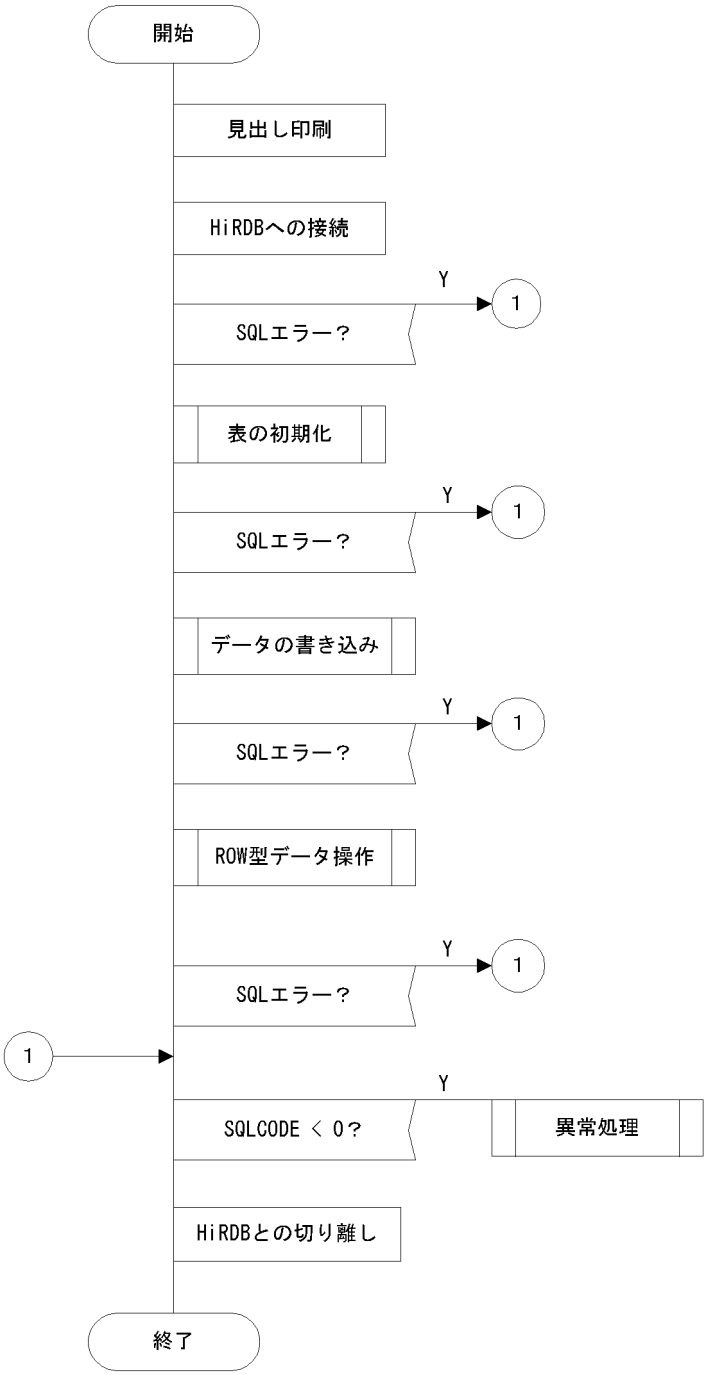
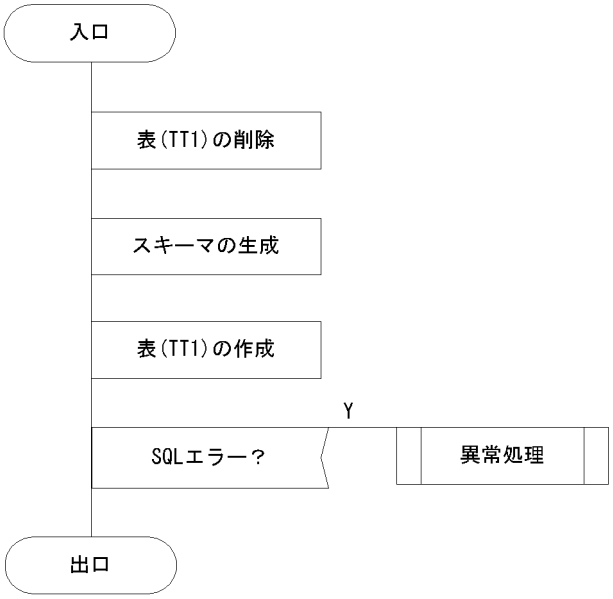


図 7–16　プログラム例題 5 の PAD チャート（2/4）

［表の初期化］



［異常処理］

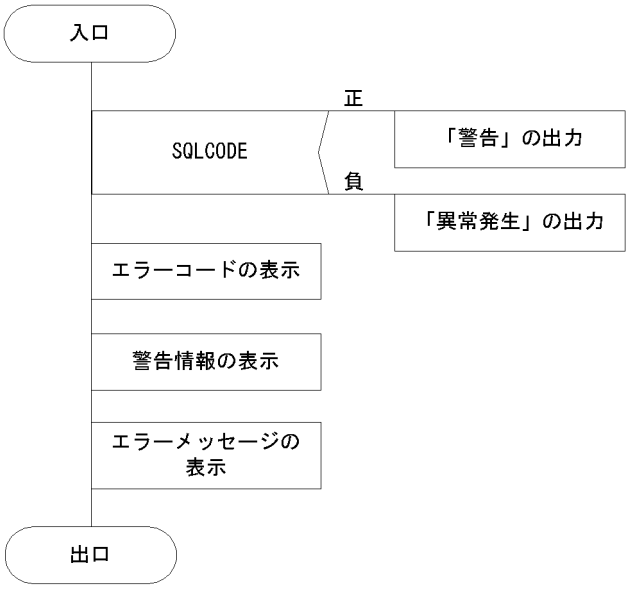


図 7–17　プログラム例題 5 の PAD チャート（3/4）

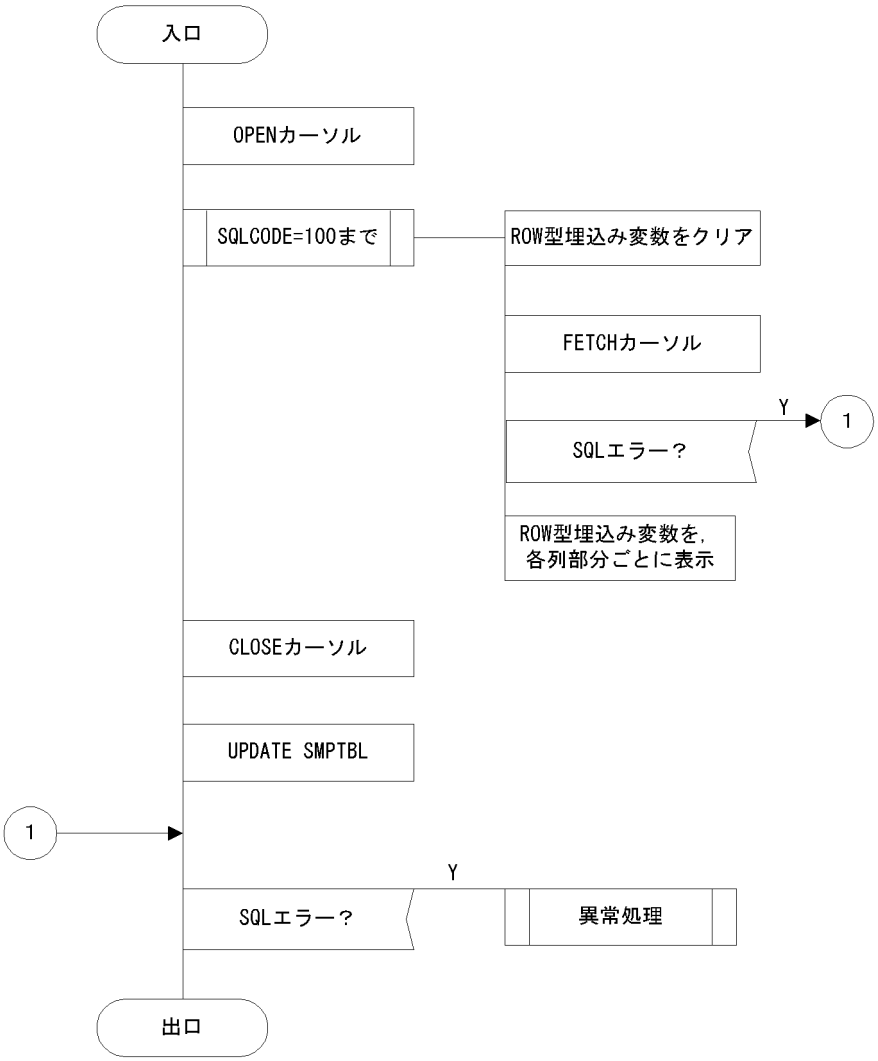
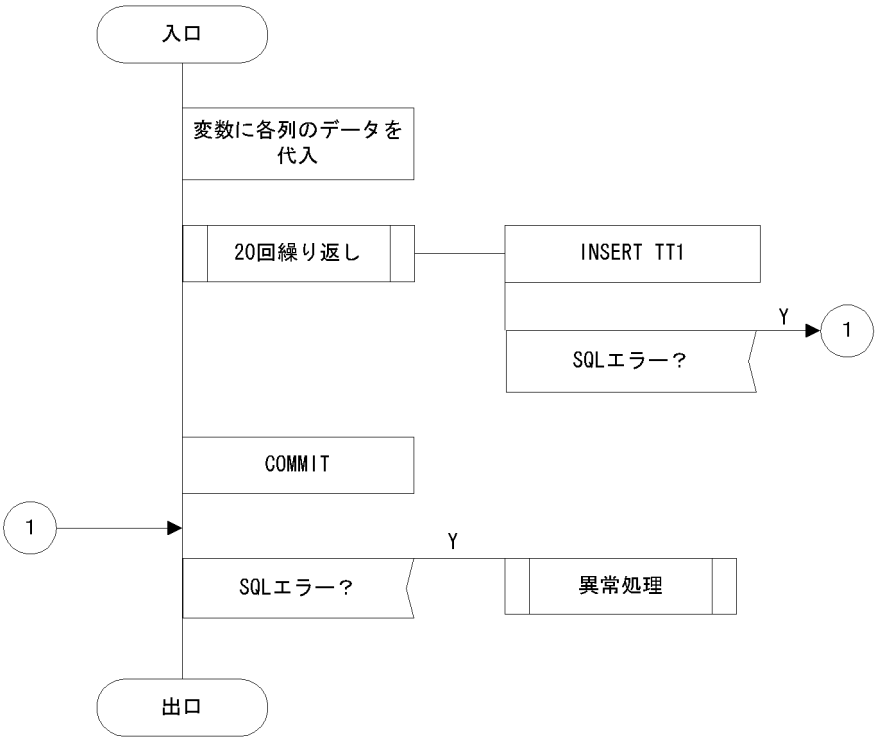


図 7-18　プログラム例題 5 の PAD チャート（4/4）

［データの挿入］



(b)　コーディング例

```
000010*****************************************************************
000020*                                                               *
000030*       埋込み型SQL  COBOL  UAP                                 *
000040*        ROW インタフェースサンプル                             *
000050*                            1997/11/27                         *
000060*****************************************************************
000070 IDENTIFICATION DIVISION.
000080 PROGRAM-ID.            ROW-SAMPLE.
000090 AUTHOR.                CLIENT.
000100 DATA-WRITTEN.          1997/11/27.
000110 DATA-COMPILED.         ROW-SAMPLE.
000120 REMARKS.
000130*
000140 ENVIRONMENT DIVISION.
000150 CONFIGURATION SECTION.
000160 SOURCE-COMPUTER. HITAC.
000170 OBJECT-COMPUTER. HITAC.
000180 INPUT-OUTPUT SECTION.
000190 FILE-CONTROL.
000200     SELECT OUTLIST  ASSIGN TO LP.
000210*
000220 DATA DIVISION.
000230 FILE SECTION.
000240 FD OUTLIST RECORDING MODE IS F
000250            LABEL RECORD IS OMITTED
000260            DATA RECORD OUTREC.
000270 01  OUTREC             PIC X(80).
```

```
000280*
000290 WORKING-STORAGE SECTION.
000300     EXEC SQL
000310       BEGIN DECLARE SECTION
000320     END-EXEC.
000330 01 IN-REC1 IS GLOBAL.
000340   02  IN-CHR1       PIC X(15)       VALUE 'EVA-00'.
000350   02  IN-INT1       PIC S9(9) COMP  VALUE 255.
000360   02  IN-INT2       PIC S9(9) COMP  VALUE 1.
000370
000380 01 XSQLROW  IS GLOBAL.                                          1
000390   02 ROW-CHR1 PIC X(30).                                        1
000400   02 ROW-INT1 PIC S9(9) COMP.                                   1
000410   02 ROW-INT2 PIC S9(9) COMP.                                   1
000420
000430     EXEC SQL
000440       END DECLARE SECTION
000450     END-EXEC.
000460
000470 01 DISP-REC IS GLOBAL.
000480   02  DISP-CHR1       PIC X(15).
000490   02  DISP-INT1       PIC S9(9).
000500   02  DISP-INT2       PIC S9(4).
000510 01  ERRFLG  PIC S9(4) COMP IS GLOBAL.
000520
000530 01  MSG-ERR      PIC X(10) VALUE '!! ERROR'.
000540 01  MSG-CODE IS GLOBAL.
000550     02  FILLER     PIC X(15) VALUE '!! SQLCODE ='.
000560     02  MSG-SQLCODE    PIC S9(9) DISPLAY.
000570 01  MSG-MC IS GLOBAL.
000580     02  FILLER     PIC X(15) VALUE '!! SQLERRMC ='.
000590     02  MSG-SQLERRMC   PIC X(100).
000600
000610 PROCEDURE DIVISION.
000620*************************************************
000630* DISPLAY TITLE
000640*************************************************
000650 MAIN SECTION.
000660    CALL 'DISPLAY-TITLE'.
000670    MOVE ZERO TO ERRFLG.
000680
000690*************************************************
000700* CONNECT
000710*************************************************
000720      EXEC SQL
000730        WHENEVER SQLERROR GOTO ERR-EXIT
000740      END-EXEC
000750
000760      DISPLAY '***** CONNECT '.
000770      EXEC SQL                                                  2
000780        CONNECT                                                 2
000790      END-EXEC.                                                 2
000800      DISPLAY '***** CONNECT : END'.
000810
000820*************************************************************
000830* INIT
000840*************************************************************
000850      DISPLAY '## テーブルの初期化を行います'.
```

```
000860      CALL 'INIT-TABLE'.
000870      IF ERRFLG < ZERO
000880        GO TO ERR-EXIT
000890      END-IF
000900      DISPLAY '## 正常です'.
000910
000920*************************************************************
000930* INSERT
000940*************************************************************
000950      DISPLAY '## DATAをINSERT'.
000960      CALL 'TEST-INSERT'.
000970      IF ERRFLG < ZERO
000980        GO TO ERR-EXIT
000990      END-IF
001000      DISPLAY '## 正常です'.
001010
001020*************************************************************
001030* ROW
001040*************************************************************
001050      DISPLAY '## ROW型のテストを行います'.
001060      CALL 'TEST-ROW'.
001070      IF ERRFLG < ZERO
001080        GO TO ERR-EXIT
001090      END-IF
001100      DISPLAY '## 正常です'.
001110
001120*************************************************************
001130* DISCONNECT
001140*************************************************************
001150 ERR-EXIT.
001160      IF SQLCODE < ZERO
001170        MOVE SQLCODE TO MSG-SQLCODE
001180        MOVE SQLERRMC TO MSG-SQLERRMC
001190        DISPLAY MSG-ERR
001200        DISPLAY MSG-CODE
001210        DISPLAY MSG-MC
001220        MOVE -1 TO ERRFLG
001230      END-IF
001240
001250      EXEC SQL
001260        WHENEVER SQLERROR  CONTINUE
001270      END-EXEC
001280      EXEC SQL
001290        WHENEVER NOT FOUND CONTINUE
001300      END-EXEC
001310      EXEC SQL
001320        WHENEVER SQLWARNING CONTINUE
001330      END-EXEC
001340
001350      DISPLAY '##DISCONNECT'
001360
001370      EXEC SQL                                                3
001380        DISCONNECT                                            3
001390      END-EXEC                                                3
001400      STOP RUN.
001410
001420*************************************************************
001430*  INSERT文のテスト
```

```
001440*************************************************************
001450 IDENTIFICATION DIVISION.
001460 PROGRAM-ID. TEST-INSERT.
001470 DATA DIVISION.
001480 WORKING-STORAGE SECTION.
001490   01 DCNT PIC S9(9) COMP.
001500 PROCEDURE DIVISION.
001510       EXEC SQL
001520         WHENEVER SQLERROR  GOTO :Exit-Test-Insert
001530       END-EXEC.
001540*************************************************************
001550* INSERT HOST
001560*************************************************************
001570       DISPLAY '***** 埋込み変数によるINSERT start'
001580       MOVE ZERO TO DCNT.
001590 INSERT-LOOP.
001600       COMPUTE IN-INT1 = DCNT
001610       COMPUTE IN-INT2 = DCNT + 100
001620       COMPUTE DCNT = DCNT + 1
001630       EXEC SQL                                                4
001640         INSERT INTO TT1(CLM1,                                 4
001650                         CLM2,                                 4
001660                         CLM3)                                 4
001670           VALUES (:IN-CHR1,                                   4
001680                   :IN-INT1,                                   4
001690                   :IN-INT2)                                   4
001700       END-EXEC                                                4
001710       IF DCNT < 20 THEN
001720         GO TO INSERT-LOOP
001730       END-IF
001740       DISPLAY '***** insert : SUCCESS'.
001750*************************************************************
001760*
001770*************************************************************
001780 EXIT-TEST-INSERT.
001790       IF SQLCODE < ZERO
001800         MOVE SQLCODE TO MSG-SQLCODE
001810         MOVE SQLERRMC TO MSG-SQLERRMC
001820         DISPLAY MSG-CODE
001830         DISPLAY MSG-MC
001840         MOVE -1 TO ERRFLG
001850       END-IF
001860       DISPLAY '>> TEST-INSERT <<'
001870       GOBACK.
001880*************************************************************
001890*  WARNING
001900*************************************************************
001910 INSERT-WARNING.
001920     DISPLAY 'WARINING'
001930     MOVE SQLCODE TO MSG-SQLCODE
001940     MOVE SQLERRMC TO MSG-SQLERRMC
001950     DISPLAY MSG-CODE
001960     DISPLAY MSG-MC.
001970 END PROGRAM TEST-INSERT.
001980
001990*************************************************************
002000*  ROWのテスト
002010*************************************************************
```

```
002020 IDENTIFICATION DIVISION.
002030 PROGRAM-ID. TEST-ROW.
002040 DATA DIVISION.
002050 WORKING-STORAGE SECTION.
002060 PROCEDURE DIVISION.
002070        DISPLAY '***** ROW CURSOR OPEN'
002080        EXEC SQL                                                5
002090          DECLARE CUR_ROW CURSOR FOR                            5
002100             SELECT ROW FROM TT1                                5
002110              WHERE CLM2 = 10                                   5
002120              FOR UPDATE OF CLM3                                5
002130        END-EXEC                                                5
002140*********************************************************
002150*  ROW CURSOR
002160*********************************************************
002170        DISPLAY '***** ROW CURSOR OPEN'.
002180        EXEC SQL
002190          WHENEVER SQLERROR  GOTO :Exit-Test-ROW
002200        END-EXEC
002210        EXEC SQL                                                6
002220          OPEN CUR_ROW                                          6
002230        END-EXEC                                                6
002240
002250*********************************************************
002260*  FETCH ROW CURSOR
002270*********************************************************
002280        DISPLAY '***** ROW CURSOR FETCH'
002290        EXEC SQL
002300          WHENEVER NOT FOUND GOTO :Exit-Test-ROW
002310        END-EXEC
002320        EXEC SQL
002330          WHENEVER SQLERROR GOTO :Exit-Test-ROW
002340        END-EXEC
002350        MOVE SPACE TO XSQLROW
002360        EXEC SQL                                                7
002370          FETCH CUR_ROW INTO :XSQLROW                           7
002380        END-EXEC                                                7
002390        DISPLAY '## FETCH DATA'
002400        MOVE ROW-CHR1 TO DISP-CHR1
002410        MOVE ROW-INT1 TO DISP-INT1
002420        MOVE ROW-INT2 TO DISP-INT2
002430        DISPLAY DISP-REC
002440
002450        DISPLAY '***** ROW UPDATE'
002460        MOVE 'ANGEL' TO ROW-CHR1
002470        EXEC SQL                                                8
002480          UPDATE TT1 SET ROW = :XSQLROW                         8
002490           WHERE CURRENT OF CUR_ROW                             8
002500        END-EXEC                                                8
002510
002520*********************************************************
002530*  FETCH ROW CURSOR
002540*********************************************************
002550        DISPLAY '***** ROW CURSOR CLOSE'
002560        EXEC SQL
002570          WHENEVER NOT FOUND CONTINUE
002580        END-EXEC
002590        EXEC SQL
```

```
002600           WHENEVER SQLERROR CONTINUE
002610         END-EXEC
002620         EXEC SQL                                                  9
002630           CLOSE  CUR_ROW                                          9
002640         END-EXEC.                                                 9
002650*************************************************
002660*
002670*************************************************
002680 EXIT-TEST-ROW.
002690         IF SQLCODE < ZERO THEN
002700           MOVE SQLCODE  TO MSG-SQLCODE
002710           MOVE SQLERRMC TO MSG-SQLERRMC
002720           DISPLAY MSG-CODE
002730           DISPLAY MSG-MC
002740           MOVE -1 TO ERRFLG
002750         END-IF
002760         EXEC SQL
002770           WHENEVER NOT FOUND CONTINUE
002780         END-EXEC
002790         EXEC SQL
002800           WHENEVER SQLERROR CONTINUE
002810         END-EXEC
002820         EXEC SQL
002830           COMMIT
002840         END-EXEC
002850         DISPLAY '>> TEST-ROW END <<'
002860         GOBACK.
002870
002880*************************************************
002890*  WARNING
002900*************************************************
002910 ROW-WARNING.
002920     DISPLAY 'WARINING'
002930     MOVE SQLCODE TO MSG-SQLCODE
002940     MOVE SQLERRMC TO MSG-SQLERRMC
002950     DISPLAY MSG-CODE
002960     DISPLAY MSG-MC.
002970 END PROGRAM TEST-ROW.
002980
002990
003000 *************************************************
003010 *  テーブルの初期化
003020 *************************************************
003030 IDENTIFICATION DIVISION.
003040 PROGRAM-ID. INIT-TABLE.
003050 DATA DIVISION.
003060 WORKING-STORAGE SECTION.
003070 PROCEDURE DIVISION.
003080       EXEC SQL
003090         WHENEVER SQLERROR CONTINUE
003100       END-EXEC
003110
003120*************************************************
003130* DROP TABLE
003140*************************************************
003150       DISPLAY '***** DROP TABLE'.
003160       EXEC SQL                                                   10
003170         DROP TABLE TT1                                           10
```

```
003180       END-EXEC                                                  10
003190       DISPLAY '***** CREATE SCHEMA'.
003200       EXEC SQL                                                  11
003210         CREATE SCHEMA                                           11
003220       END-EXEC                                                  11
003230
003240************************************************************
003250* COMMIT
003260************************************************************
003270       DISPLAY '***** COMMIT START'.
003280       EXEC SQL
003290         WHENEVER SQLERROR GOTO EXIT-INIT-TABLE
003300       END-EXEC
003310       EXEC SQL
003320         COMMIT
003330       END-EXEC
003340       DISPLAY '***** COMMIT : END'.
003350
003360************************************************************
003370* CREATE TABLE
003380************************************************************
003390       DISPLAY '***** create table'.
003400       EXEC SQL                                                  12
003410         CREATE FIX TABLE TT1(CLM1 CHAR(30),                     12
003420                              CLM2 INTEGER,                      12
003430                              CLM3 INTEGER)                      12
003440       END-EXEC                                                  12
003450
003460       DISPLAY '***** create table : SUCCESS'.
003470
003480************************************************************
003490*
003500************************************************************
003510 EXIT-INIT-TABLE.
003520       IF SQLCODE < ZERO THEN
003530         MOVE SQLCODE  TO MSG-SQLCODE
003540         MOVE SQLERRMC TO MSG-SQLERRMC
003550         DISPLAY MSG-CODE
003560         DISPLAY MSG-MC
003570         MOVE -1 TO ERRFLG
003580       END-IF
003590       GOBACK.
003600
003610************************************************************
003620*  WARNING
003630************************************************************
003640 INIT-TABLE-WARNING.
003650     DISPLAY 'WARINING'
003660     MOVE SQLCODE TO MSG-SQLCODE
003670     MOVE SQLERRMC TO MSG-SQLERRMC
003680     DISPLAY MSG-CODE
003690     DISPLAY MSG-MC.
003700 END PROGRAM INIT-TABLE.
003710
003720************************************************************
003730*  DISPLAY
003740************************************************************
003750 IDENTIFICATION DIVISION.
```

```
003760 PROGRAM-ID. DISPLAY-TITLE.
003770 DATA DIVISION.
003780 WORKING-STORAGE SECTION.
003790 PROCEDURE DIVISION.
003800     DISPLAY '##############################################'
003810     DISPLAY '#                                            #'
003820     DISPLAY '# このプログラムはROW型インタフェースの     #'
003830     DISPLAY '#   サンプルプログラムです                    #'
003840     DISPLAY '#                                            #'
003850     DISPLAY '##############################################'.
003860 END PROGRAM DISPLAY-TITLE.
003870 END PROGRAM ROW-SAMPLE.
```

＜説明＞

1. ROW 型の埋込み変数の宣言

   行インタフェースで使用する埋込み変数（:XSQLROW）を宣言します。

2. HiRDB への接続

   環境変数 PDUSER に定義されている認可識別子とパスワードを使用して，サーバに接続します。

3. HiRDB の切り離し

   UAP をサーバから切り離します。

4. 行の追加

   FIX 表（TT1）にデータを追加します。

5. カーソル CUR_ROW の宣言

   行インタフェースを使用して FIX 表（TT1）を検索するので，カーソル CUR_ROW を宣言します。

6. カーソル CUR_ROW のオープン

   FIX 表（TT1）の検索行の直前にカーソルを位置づけて，各行を取り出せる状態にします。

7. 行の取り出し

   FIX 表（TT1）から，カーソル CUR_ROW の示す行を１行取り出し，埋込み変数（:XSQLROW）に設定します。

8. 行の更新

   カーソル CUR_ROW が位置づけられている FIX 表（TT1）の行を，埋込み変数（:XSQLROW）の値で更新します。

9. カーソル CUR_ROW のクローズ

   カーソル CUR_ROW を閉じます。

10. 表（TT1）を削除する

    FIX 表（TT1）を作成するために，同名の表があった場合は削除します。

11. スキーマの生成

    スキーマがないときのために，スキーマを生成します。

12. FIX 表（TT1）の作成

FIX 表（TT1）を作成します。行インタフェースは FIX 属性の表に対してだけ使用できます。

(3) TYPE 句，TYPEDEF 句，及び SAME AS 句を使用した例

TYPE 句，TYPEDEF 句，及び SAME AS 句を使用したコーディング例を次に示します。

```
000100 IDENTIFICATION  DIVISION.
000200 PROGRAM-ID.     CBL001.
000300 DATA            DIVISION.
000400 WORKING-STORAGE SECTION.
000500     EXEC SQL BEGIN DECLARE SECTION END-EXEC.
000600* -- type declaration --
000700 01  VCHR20 TYPEDEF.
000800     05  LEN  PIC S9(4) COMP.
000900     05  STR  PIC X(20).
001000
001100* -- data declaration --
001200 01  D-4C.
001300     05  XCUT     TYPE VCHR20.
001400     05  XCOLOR   PIC X(10).
001500     05  XCLARITY SAME AS XCOLOR.
001600     05  XCARAT   PIC S9(4) COMP.
001700
001800     EXEC SQL END DECLARE SECTION END-EXEC.
   :                 :
   :                 :
002000 PROCEDURE DIVISION.
002100 CB_001 SECTION.
   :                 :
   :                 :
003400 INS-1.
003500       EXEC SQL
003600        INSERT INTO A_DIM (C1, C2, C3, C4)
003700            VALUES (:XCUT, :XCOLOR, :XCLARITY, :XCARAT)
003800       END-EXEC.
   :                 :
   :                 :
005000 INS-EX.
005100     EXIT.
005200 END PROGRAM CBL001.
```

7.4　C++言語による UAP の作成

ここでは，C++言語による埋込み型 UAP の記述規則について説明します。

7.4.1　記述規則

UAP を作成するとき，SQL の文法の規則以外に，名標の付け方や SQL の記述についての規則があります。

(1)　名標の付け方の規則

名標を付けるときの規則は，基本的に C 言語の場合の規則に従います。

C 言語の規則以外に，使用できない名標を次に示します。

- 大文字の「SQL」で始まる名標
- 小文字の「p_」で始まる名標
- 小文字の「pd」で始まる名標

なお，SQL 中で使用する埋込み変数，標識変数，及び分岐先ラベルの名称の付け方は，名標の付け方，及び C 言語の規則に従います。

(2)　SQL の記述規則

1. 注釈文として，//を使用できます。
2. 埋込み変数にオブジェクトのメンバを使用できません。
3. WHENEVER 文で，オブジェクトのメソッドを指定できません。
4. クラス定義内で SQL を記述できません。

上記以外の記述規則については，C 言語の場合と同じなので，「SQL の記述規則」を参照してください。

7.5　OOCOBOL 言語による UAP の作成

ここでは，OOCOBOL 言語による埋込み型 UAP の記述規則について説明します。

7.5.1　記述規則

UAP を作成するとき，SQL の文法の規則以外に，名標の付け方や SQL の記述についての規則があります。

(1)　名標の付け方の規則

名標を付けるときの規則は，基本的に COBOL 言語の場合の規則に従います。

COBOL 言語の規則以外に，名標を付けるときの規則を次に示します。

(a)　SQL の予約語

- 大文字でも小文字でも使用できます。
- 大文字と小文字を混在できます。

(b)　ホスト名

- 「SQL」で始まる名標は使用できません。
- ホスト名中のコロンの後ろに空白を記述できます。
- 大文字と小文字は同等に使用できます。
- 大文字と小文字は混在できます。

なお，使用する埋込み変数，標識変数，及び分岐先の名前の付け方は，名標の付け方の規則，及び COBOL 言語の規則に従います。また，次に示す名標は外部属性を持つため，使用できないので注意してください。

- 大文字の「SQL」で始まる名標
- 小文字の「p_」で始まる名標
- 小文字の「pd」で始まる名標

(2)　SQL の記述規則

1. 埋込み変数にオブジェクトのメンバを使用できません。
2. WHENEVER 文で，オブジェクトのメソッドを指定できません。
3. クラス定義内で SQL を記述できません。

上記以外の記述規則については，COBOL 言語の場合と同じなので，「SQL の記述規則」を参照してください。

7.6　64 ビットモードでの UAP の作成

HiRDB クライアントを使用した 64 ビットモード対応の UAP を作成する方法について説明します。

7.6.1　64 ビットモード対応の UAP で使用できる言語，及び機能

(1)　言語

C 言語，C++言語，及び COBOL 言語（COBOL2002）で UAP を作成できます。OOCOBOL 言語では作成できません。

(2)　機能

HiRDB クライアントの機能は基本的に使用できます。また，複数接続機能では，擬似スレッドではなく，リアルスレッドを提供しています。

7.6.2　SQL 連絡領域の違い

HiRDB を 64 ビットモードにすると，SQL 連絡領域の構成が変わります。また，次の表に示す連絡領域名の長さが変わります。SQL 連絡領域については，「SQL 連絡領域」を参照してください。

表 7–5　変更になる連絡領域名

連絡領域名	長さ（単位：バイト）	
	32 ビットモード	64 ビットモード
SQLCA	336	368
SQLCABC	4	8
SQLCODE	4	8
SQLERRD	4×6	8×6

7.6.3　SQL 記述領域の違い

HiRDB を 64 ビットモードにすると，SQL 記述領域の構成が変わります。また，次の表に示す記述領域名の長さとデータ型が変わります。SQL 記述領域については，「SQL 記述領域」を参照してください。

表 7–6　変更になる記述領域名

記述領域名	32 ビットモード		64 ビットモード	
	長さ（バイト）	データ型	長さ（バイト）	データ型
SQLDA	16 ＋ 16×n	−	24 ＋ 24×n	−
SQLDABC	4	−	8	−
SQLVAR	16×n	−	24×n	−
SQLVAR_LOB	16×n	−	24×n	−
SQLLOBLEN	−	long	−	int
SQLDATA	4	−	8	−
SQLIND	4	−	8	−
SQLLOBIND	4	long *	8	int *

（凡例）

n：記述領域名 SQLN に指定した SQLVAR の数です。

−：長さ又はデータ型の変更はありません。

7.6.4　SQL のデータ型と C 言語のデータ記述の違い

64 ビットモード対応の C 言語 UAP では，long 型のサイズが 8 バイトのため，long を使用していた埋込み変数は long の代わりに int を使用します。したがって，次の表に示す C 言語のデータ記述が変わります。C 言語のデータ記述については，「SQL のデータ型と C 言語のデータ記述」を参照してください。

表 7–7　変更になる C 言語のデータ記述

SQL のデータ型		C 言語のデータ記述	項目の記述	備考
INTEGER	単純形	int 変数名;	変数	なし
	配列形	int 変数名[n];	配列	1≦n≦4096
BLOB 用標識変数		int 標識変数名;	−	なし
SQL 文		struct{ int len; char str[n]; }変数名;	構造体	なし

（凡例）

n：長さ（単位：バイト）。

−：記述できません。

7.6.5　表分割ハッシュ関数を使用したUAPの作成方法の違い（Linux版限定）

表分割ハッシュ関数を使用したUAPのコンパイル及びリンケージをするときに指定する共用ライブラリが変わります。表分割ハッシュ関数を使用したUAPの作成方法については，「表分割ハッシュ関数」を参照してください。

7.6.6　32ビットモードから64ビットモードへのHiRDBクライアントへの移行

32ビットモードから64ビットモードへHiRDBクライアントを移行するには，HiRDBクライアントを64ビットモードにバージョンアップする必要があります（64ビットモードのHiRDBクライアントをインストールして，クライアントの環境設定をします）。クライアントの環境設定については，「クライアントの環境設定」を参照してください。

64ビットモードのHiRDBクライアントをインストールした場合，64ビットモード用のファイルが作成されます。インストールしたときに作成されるファイルについては，「HiRDBクライアントのディレクトリ及びファイル構成」を参照してください。

クライアントの環境設定が終わったら，次の手順でUAPを64ビットモード対応にしてください。

＜手順＞

1. 埋込み変数の宣言でlong型を使用している場合，その箇所をint型に置き換えます。
2. UAPのプリプロセスを実行します。このとき，64ビットモード用のポストソースを生成するオプション（UNIXの場合は-h64，Windowsの場合は/h64）を指定します。
3. UAPのコンパイルを実行します。このとき，64ビットモード用のオブジェクトを生成するオプションを指定します。
4. UAPのリンケージを実行します。このとき，リンケージするクライアントライブラリに64ビットモードのクライアントライブラリを指定します。

注

プリプロセス，及びコンパイル，リンケージについては，それぞれ「プリプロセス」，「コンパイルとリンケージ」を参照してください。

8 UAP 実行前の準備

この章では，UAP を実行する前の準備作業について説明します。

8.1 UAP の実行手順

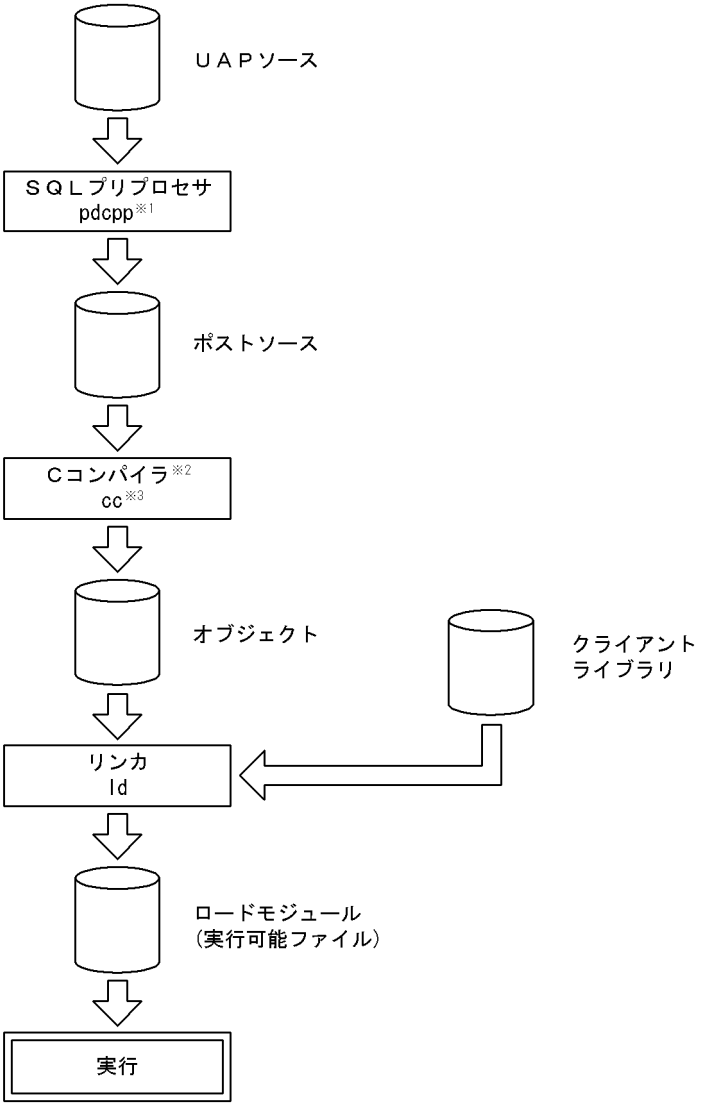
SQL を埋め込んで作成した UAP は，そのままでは実行できません。ここでは，作成した UAP を実行するための手順について説明します。

8.1.1 C 言語で作成した UAP の実行手順

C 言語で記述されたソースプログラム中に直接 SQL を埋め込んだ UAP は，SQL プリプロセサでポストソースに変換してください。変換されたポストソースは，専用の言語コンパイラでコンパイル，及びリンケージをするとロードモジュール（実行可能ファイル）になります。

C 言語で作成した UAP の実行までの手順を次の図に示します。

図 8–1　C 言語で作成した UAP の実行までの手順



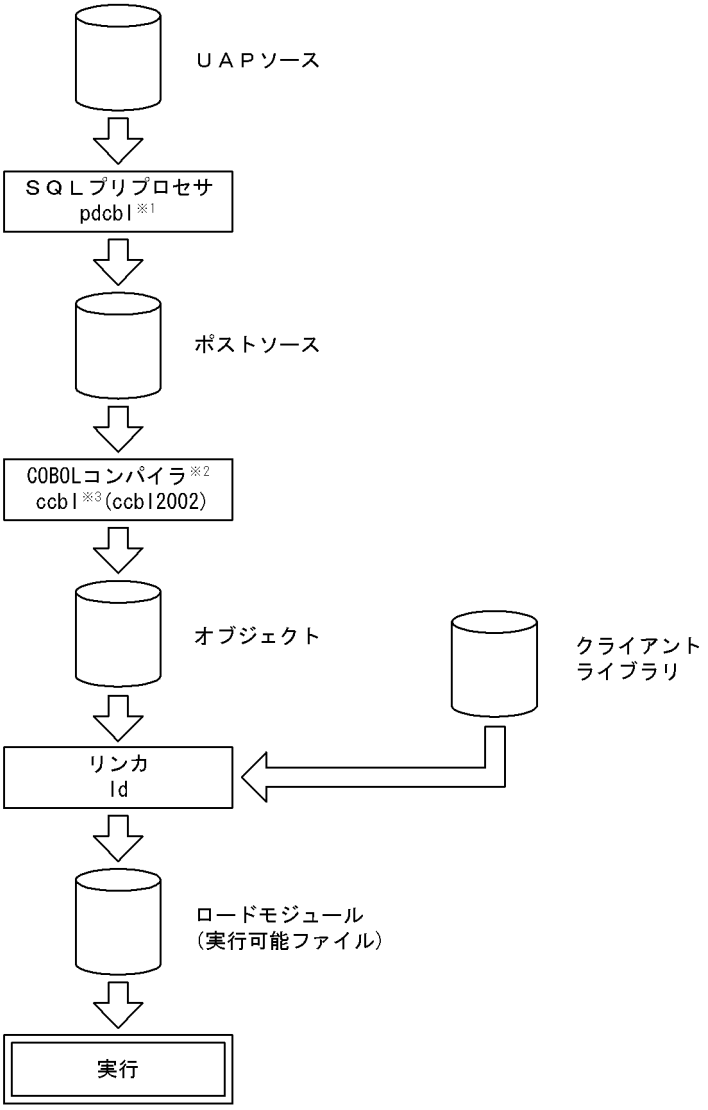
注※１　C++言語の場合，pdoccになります。
注※２　C++言語の場合，C++コンパイラになります。
注※３　C++言語の場合，CCになります。

8.1.2　COBOL 言語で作成した UAP の実行手順

COBOL 言語で記述されたソースプログラム中に直接 SQL を埋め込んだ UAP は，SQL プリプロセサでポストソースに変換してください。変換されたポストソースは，専用の言語コンパイラでコンパイル，及びリンケージをするとロードモジュール（実行可能ファイル）になります。

COBOL 言語で作成した UAP の実行までの手順を次の図に示します。

図 8–2　COBOL 言語で作成した UAP の実行までの手順



注※１　00COBOL言語の場合，pdocbになります。
注※２　00COBOL言語の場合，00COBOLコンパイラになります。
注※３　00COBOL言語の場合，ocblになります。

8.2 プリプロセス

8.2.1 プリプロセスの概要

(1) プリプロセスとは

SQL を埋め込んだ UAP のソースファイルは，そのままではコンパイルできないため，SQL プリプロセサを実行する必要があります。SQL プリプロセサを実行すると，埋め込まれた SQL 文は，高級言語用の記述に変換されます。この SQL プリプロセサを実行して，UAP ソースを言語コンパイラで変換できるポストソースに変換することを**プリプロセス**といいます。

SQL プリプロセサは，SQL 文に対応した高級言語の関数を生成し，ソース中に埋め込みます。このとき，変数のデータ型の妥当性，値の妥当性，及び各種名称のシンタクスについてチェックします。チェックした結果，入力ソースプログラム中にエラーを検出すると，メッセージを標準出力に出力します。

(2) SQL プリプロセサでチェックされない項目

SQL プリプロセサを実行しても，次に示す内容についてはチェックされません。

- サーバに問い合わせが必要な表名があるかどうか
- サーバに問い合わせが必要な列名があるかどうか
- サーバに問い合わせが必要な，ほかの識別子，データ型，関数があるかどうか
- 表に対するアクセス権限

(3) プリプロセスをする場合の注意事項

1. SQL プリプロセサは，ソースプログラムに使用している高級言語，及び環境によって，環境変数の設定，及びコマンドの指定方法が異なるため，使用している言語や環境に合わせる必要があります。
2. Windows 環境での SQL プリプロセサでは，/Xp を指定しないと厳密な SQL の構文チェックができないため，SQL 文に文法上のエラーがあっても発見できないことがあります。また，AIX 版のクライアントで文字コードが SJIS 以外の場合，厳密な SQL の構文チェックはできません。
3. Windows 環境では EUC コードを認識できないため，使用できる文字コードは sjis 又は lang-c だけです。HiRDB サーバの文字コード種別に sjis 又は lang-c 以外を指定している場合，HiRDB クライアントで UAP を実行するとエラーになります。

8.2.2　UNIX 環境でのプリプロセス

(1)　C 言語の場合

(a)　環境変数の設定

UAP をプリプロセスする前に，次に示す環境変数を必要に応じて設定します。

PDDIR：

HiRDB（サーバ，又はクライアント）のインストールディレクトリを絶対パス名で指定します。この環境変数を設定しないと，/HiRDB が仮定されます。

なお，インストール先が/HiRDB の場合，この変数を設定する必要はありません。

PDCLIB：

COPY 文でソースファイルに引き込まれるヘッダファイルを検索するディレクトリを指定します。複数のディレクトリを指定する場合，ディレクトリ同士をコロンで区切ります。この環境変数を省略すると，カレントディレクトリだけが検索されます。

LANG：

HiRDB クライアント環境の文字コード種別を設定します。

SQL プリプロセサは，この環境変数に設定されたロケール名から，UAP のソースファイルの文字コード種別を識別します。ただし，SQL プリプロセサがサポートしていないロケール名を指定すると，SQL プリプロセサは C（単一バイト文字コード）を指定したとみなします。UAP ソースの文字コード種別を示すロケール名を SQL プリプロセサがサポートしていない場合は，LANG の代わりに環境変数 PDCLTLANG を指定してください。

SQL プリプロセサがサポートしているロケール名を次の表に示します。

表 8–1　SQL プリプロセサがサポートしているロケール名

UAP ソースの文字コード種別	環境変数 LANG に設定するロケール名	
	AIX	Linux
シフト JIS 漢字コード	Ja_JP 又は未指定※	–
EUC 中国語漢字コード	–	–
EUC 日本語漢字コード	ja_JP	ja_JP.eucJP，ja_JP，ja_JP.ujis 又は未指定※
単一バイト文字コード	C	C
Unicode（UTF-8）	–	–
Unicode（IVS 対応 UTF-8）	–	–
中国語漢字コード（GB18030）	–	–

（凡例）

－：該当する文字コード種別を示すロケール名を，SQL プリプロセサがサポートしていません。LANG の代わりに PDCLTLANG を指定してください。

注※

PDCLTLANG と LANG が共に未指定の場合，SQL プリプロセサは該当する文字コード種別を仮定します。

PDCLTLANG を指定した場合，SQL プリプロセサは LANG の指定を無視します。ただし，-E プリプロセスオプションを指定した場合に，SQL プリプロセサが呼び出す C コンパイラの動作に LANG の指定が影響する可能性があります。C コンパイラの正常動作のために必要な場合は，PDCLTLANG の指定有無に関わらず，LANG に適切な値を設定してください。

PDCLTLANG：

UAP ソースの文字コード種別を LANG で指定できない場合，及び LANG で指定した文字コード種別を無視して，ほかの文字コード種別でプリプロセスする場合に指定します。詳細については「クライアント環境定義の設定内容」の「PDCLTLANG」を参照してください。

PDDEFAULTOPTION：

プリプロセスのオプションについて，省略時の動作を指定します。通常は，この環境変数を指定する必要はありません。互換モードを適用したい場合に，指定してください。詳細については，「クライアント環境定義の設定内容」の「PDDEFAULTOPTION」を参照してください。

＜例 1＞（sh（ボーンシェル）で環境設定をする場合）

- prdb ディレクトリがインストールディレクトリの場合

```
$ PDDIR=”/prdb”
$ export PDDIR
```

＜例 2＞（csh（C シェル）で環境設定をする場合）

- prdb ディレクトリがインストールディレクトリの場合

```
% setenv   PDDIR ”/prdb”
```

(b)　SQL プリプロセサの起動

SQL プリプロセサの起動は，pdcpp コマンド（C 言語の場合），又は，pdocc コマンド（C++言語の場合）を使用します。

SQL プリプロセサを起動するコマンドの入力形式を次に示します。

```
pdcpp 入力ファイル名称　［オプション［出力ファイル名称｜認可識別子］］
```

注　C++言語の場合，下線で示す部分を pdocc に置き換えてください。

入力ファイル名称：

UAP ソースファイルの名称を指定します。

ファイル識別子は，.ec（C 言語の場合），又は.EC（C++言語の場合）にします。

出力ファイル名称：

ポストソースファイルの名称を指定します。

出力ファイル名称を省略した場合，ファイル識別子は.c（C 言語の場合），又は.C（C++言語の場合）になります。

認可識別子：

SQL で認可識別子を省略した場合に仮定する認可識別子を指定します。認可識別子を省略した場合，CONNECT 時のユーザ識別子が仮定されます。

オプション：

必要に応じて次の表に示すオプションを指定します。なお，オプションは大文字，小文字を区別しません。

表 8–2　プリプロセスオプション（UNIX 環境の C 言語の場合）

プリプロセスオプション	内容
-s	構文チェックだけをして，ポストソースを出力しない場合に指定します。このオプションを省略すると，ポストソースが出力されます。
-o ファイル名	出力するポストソースのファイル名称を指定します。 このオプションを省略すると，入力ファイル名称のファイル識別子を.c（C 言語の場合），又は.C（C++言語の場合）に変更されたものが出力ファイル名称になります。
-A 認可識別子	静的 SQL で認可識別子を省略した場合，及び仮定する認可識別子を変更する場合に指定します。 静的 SQL とは，INSERT 文，UPDATE 文，DELETE 文，1 行 SELECT 文，OPEN 文（形式 1），CALL 文，LOCK 文，及び PURGE TABLE 文を示します。 認可識別子を引用符（"）で囲んで指定することはできません。指定した認可識別子は常に大文字と小文字を区別して扱います。SQL 文中に引用符で囲まないで指定した認可識別子は小文字を大文字として扱います。そのため，ほかの UAP などに記述した SQL 文中に引用符で囲まないで小文字を使って指定した認可識別子と同じ認可識別子をこのオプションに指定する場合は，すべての英字を大文字で指定してください。 最大 30 バイトの認可識別子を指定できます。 認可識別子を省略した場合に仮定する認可識別子については，マニュアル「HiRDB SQL リファレンス」の「名前の修飾」を参照してください。
-h64	64 ビットモード用のポストソースを作成する場合に指定します。ただし，32 ビット版のプリプロセサを使用した場合は指定できません。 long 型を使用した埋込み変数の宣言はエラーになります。
-P	SQL の構文チェックをしない場合に指定します。次のどれかの UAP をプリプロセスする場合に指定できます。 • XDM/RD E2 接続用の UAP • SQL 予約語削除機能を使用する UAP • 集合演算子 MINUS を使用する UAP

プリプロセスオプション	内容
	これらの UAP をプリプロセスするときにこのオプションを指定しないと，XDM/RD E2 又は HiRDB サーバで使用できる SQL が構文エラーとなることがあります。
-Xo	UAP から抽出した SQL 文を標準出力へ出力する場合に指定します。このとき，出力方法は次のようになります。 • SQL 文中の埋込み変数は?パラメタに置換します。 • 1 行 SELECT 文の INTO 句を削除します。 • SQL 文中の語句間の空白が 2 文字以上の場合，空白 1 文字に置き換えます。 • 複数行に分割して記述している SQL は 1 行にまとめます。 • 実行時にサーバに送られる SQL だけ出力します。実行されない SQL 文（WHENEVER 文，BEGIN DECLARE SECTION など）は出力しません。 • SQL の末尾にはセミコロン（;）を付けます。 • 埋込み変数の宣言は出力しません。 • 動的 SQL は，SQL がリテラルで指定されている場合にだけ出力します。そのほかの場合は出力しません。 • OPEN 文は，形式 1 のカーソルの場合だけ，問合せ式を出力します。 • ポストソースは生成しません。
-Xe {y ｜ n}	PREPRARE 文実行時のカーソルを，自動的にクローズするかどうかを指定します。 y： カーソルを自動的にクローズするポストソースを生成します。 n： カーソルを自動的にクローズしないポストソースを生成します。 省略した場合，クライアント環境定義 PDPRPCRCLS の指定値に従いポストソースを生成します。
-Xv	-E2 オプションを指定した場合に，VARCHAR 型，及び BINARY 型に対応する構造体を，通常の構造体として解析するときに指定します。VARCHAR 型，及び BINARY 型に対応する埋込み変数を宣言するためには，SQL TYPE IS〜を使用します。このオプションは，-E2 オプションと同時に指定する必要があります。なお，繰返し列のマクロを使用している場合，このオプションを指定しないでください。
-E {1 ｜ 2 ｜ 3} ["オプション文字列"]	UAP 中で使用しているプリプロセサ宣言文の有効化と，埋込み変数を埋込み SQL 宣言節で宣言しないで使用する場合に指定します。なお，このオプションは pdocc では無効となります。 -E1： プリプロセサ宣言文の有効化を指定します。 -E2： 埋込み変数を埋込み SQL 宣言節で宣言しないで使用する場合に指定します。また，ポインタでの埋込み変数指定，及び構造体の参照を使用する場合にも指定します。 -E3： -E1 と-E2 の両方を指定します。

プリプロセスオプション	内容
	"オプション文字列"： インクルードするファイルを検索するディレクトリのパス名を，C コンパイラに指定する-I オプションの形式で指定します。オプション文字列に複数のオプションを記述する場合は，セミコロンで区切ってください。また，任意の C コンパイラも指定できます。なお，-E2 オプション指定時には無視されます。 ≪留意事項≫ • -E1 又は-E3 を指定すると，内部的に C コンパイラが呼び出されるため，PATH 環境変数にコンパイラへのパス名を設定しておく必要があります。 • -E1 又は-E3 を指定すると UAP ソースファイルがあるディレクトリに作業用の一時ファイルを作成します。詳細は表「構文解析の対象ファイルとプリプロセスオプションの関係」の注 7 を参照してください。 • -E1 又は-E3 を指定し，UAP ソースファイル中で SQL の COPY 文を使用する場合は，-Ec オプションを同時に指定する必要があります。 SQL で指定するキーワード及び使用する C コンパイラの解析方法によっては，作成したポストソースのコンパイル時にエラーが発生するおそれがあります。この場合，COPY 文の使用有無に関係なく，Ec オプションを同時に指定することで，コンパイル時のエラーを回避できることがあります。 例えば，SQL のキーワードに NULL を指定し，RHEL8 又は RHEL9 の gcc コンパイラを使用してプリプロセスする場合，-Ec オプションを追加で指定する必要があります。-Ec オプションを指定しないと，コンパイル時に「不明な型名'EXEC'です」というエラーが表示されます。
-XU16[L ｜ B][T"型指定子"]	• UTF-16 のバイトオーダを指定します。 埋込み変数の宣言で文字集合名 UTF16 を指定した場合に，埋込み変数に格納する UTF-16 のバイトオーダを指定します。 -XU16L〔T"型指定子"〕： UTF-16 のバイトオーダをリトルエンディアンにします。 -XU16B〔T"型指定子"〕： UTF-16 のバイトオーダをビッグエンディアンにします。 L/B を省略，又はこのオプションを省略した場合，UTF-16 のバイトオーダを，プリプロセスを実行する OS のバイトオーダにします。 • UTF-16 の埋込み変数の展開に使用する型指定子を指定します。 SQL TYPE IS CHAR，又は SQL TYPE IS VARCHAR で始まるデータ記述で埋込み変数を宣言した場合に，ポストソース中に展開される C の宣言の中で UTF-16 の文字データを格納する変数，又は構造体メンバに付ける型指定子を指定します。 型指定子が示す型の配列要素 1 個に，UTF-16 の 2 バイトの文字データ 1 個を格納するので，sizeof（型指定子）==2 となる型指定子だけを指定できます。 <型指定子の指定例> 次に示す指定例は，sizeof(wchar_t)==2 となるコンパイラでだけ使用できます。 -XU16T"wchar_t"

プリプロセスオプション	内容
	次に示す指定例は，sizeof(unsigned short)==2 なので，どのコンパイラでも使用できます。 -XU16T"unsigned short" T"型指定子"を省略，又はこのオプションを省略した場合，型指定子に char が使用されます。このとき，char 型の配列要素 2 個に UTF-16 の 2 バイトの文字データ 1 個を格納します。
-g {c89 | c99}	-E2，-E3 オプションを指定した場合，SQL プリプロセサが UAP ソースを解析するとき準拠する C の標準規格を指定します。 -gc89： C89（ISO/IEC 9899:1990, Programming languages - C）に準拠します。 -gc99： C99（ISO/IEC 9899:1999, Programming languages - C）に準拠します。 省略した場合，-gc99 が仮定されます。 -E2，-E3 オプションを指定しない場合，このオプションを指定しても無視されます。
-Ec	-E1 又は-E3 オプションを指定した場合に，UAP ソースファイル中で SQL の COPY 文を使用できるようにします。 なお，それ以外の場合，このオプションの指定に関係なく COPY 文は使用できます（このオプションの指定は無視されます）。 ≪留意事項≫ • このオプションを指定すると UAP ソースファイルがあるディレクトリに作業用の一時ファイルを作成します。詳細は表「構文解析の対象ファイルとプリプロセスオプションの関係」の注 7 を参照してください。 • SQL の COPY 文の使用有無に関係なく，-Ec オプションを指定することで，プリプロセス時に発生する問題を回避できることがあります。詳細は-E オプションの説明を参照してください。
-Xs	UAP に埋め込んだ SQL 文中に単純注釈（-- ～ 改行）を記述する場合に指定します。 単純注釈の記述規則については，「SQL の記述規則」の 10.を参照してください。 このオプションは，省略時にも仮定されます。ただし，0904 互換モードを適用した場合は仮定されませんので，オプションを指定することを検討してください。
-Ei	Linux 環境で pdcpp コマンドの-E1 又は-E3 のオプション文字列で，gcc のオプション-finput-charset を指定する場合に指定します。 このオプションを指定する場合，環境変数 LANG に ja_JP.utf8 を指定する必要があります。 -E1 又は-E3 オプションを指定すると，pdcpp コマンドは内部で gcc のプリプロセスを実行し一時ファイルを作成します。-finput-charset で UTF-8 以外を指定すると，一時ファイルの文字コードが UTF-8 になりますが，pdcpp は一時ファイルの文字コードをソースファイルの文字コードと同じものとして処理するためエラーが発生するおそれがあります。このエラーを回避するため，このオプションでは，iconv コマンドを実行し一時ファイルの文字コードを UTF-8 からソースファイルの文字コードに変換します。

プリプロセスオプション	内容
	このオプションは，次のすべての条件を満たす場合に有効になります。条件を満たさない場合，指定は無視されます。 1. PDCLTLANG の指定が SJIS，UJIS，CHINESE，CHINESE-GB18030 のどれかであること 2. LANG の指定が ja_JP.utf8 であること 3. -E1 又は-E3 のオプション文字列で，C コンパイラの指定がなし，又は gcc であること 4. -E1 又は-E3 のオプション文字列で，-finput-charset を指定し，指定されている文字コードが次のどの UTF-8 指定とも一致しないこと ・ISO-10646/UTF-8 ・ISO-10646/UTF8 ・UTF-8 ・UTF8 このオプションを指定すると UAP ソースファイルがあるディレクトリに一時ファイルを作成します。詳細は表「構文解析の対象ファイルとプリプロセスオプションの関係」の注 7 を参照してください。

注 1

-E オプション指定時に使用できる機能を次に示します。

機能	省略	-E1	-E2	-E3
#define で定義したマクロを有効にする	×	○	×	○
#include でインクルードしたヘッダファイルを有効にする※	×	○	×	○
#if，#ifdef などの条件コンパイルを有効にする	×	○	×	○
UAP 中の任意の場所で宣言した変数を，埋込み変数として使用する	×	×	○	○
構造体を埋込み変数として使用する	×	×	○	○
ポインタを埋込み変数として使用する	×	×	○	○

（凡例）

○：該当する機能を使用できます。

×：該当する機能は使用できません。

注※

ヘッダファイル中に記述できる項目については，「SQL の記述規則」の 23.を参照してください。

注 2

-E オプションを指定した場合，プリプロセサは C コンパイラを内部的に呼び出します。プラットフォームごとの C コンパイラを次に示します。

プラットフォーム	コンパイラの種類	呼び出し時のロード名
AIX	C for AIX コンパイラ	xlc
Linux	gcc コンパイラ	gcc
Windows	Microsoft Visual C++コンパイラ	CL.EXE

これ以外の C コンパイラを使用する場合は，オプション文字列の先頭に，コンパイラのロードのディレクトリを含めた絶対パス名で指定します。ディレクトリ名，及びロード名には，空白及びセミコロンを含めることはできません。環境変数 PATH にパス名を追加している場合は，絶対パス名でなくてもかまいません。

ロード名を指定する場合は，ロード名とオプションの間をセミコロンで区切ってください。

使用するコンパイラは，-C オプションと-E オプションをサポートしている必要があります。プリプロセサは，#define，#include などの擬似命令を処理するために，内部的に C コンパイラに対して-C オプション，及び-E オプションを指定して，作業用の一時ファイルを作成します。Linux では，-C オプション，及び-E オプションのほかに，-xc オプションを使用しています。

それ以外のオプション文字列に指定できるオプションは，使用するコンパイラの仕様に依存しますが，-C オプション又は-E オプションに背反するオプションを指定すると，プリプロセサはエラーとなります。ヘルプなどを表示するオプションを使用した場合は，動作が保証されません。

例を次に示します。

例 1：デフォルトの C コンパイラを使用する場合

```
pdcpp connect.ec -E1"-I$PDDIR/include;-DDEBUG"
```

例 2：ユーザ指定の C コンパイラを使用する場合

```
pdcpp connect.ec -E1"/usr/bin/gcc;-I$PDDIR/include;-DDEBUG"
```

-E2，-E3 オプションを指定すると，プリプロセサは UAP 中の任意の箇所で宣言されている埋込み変数を認識するために，-g オプションで指定，又は仮定された C の標準規格（C89 又は C99）に準拠して構文を解析します。

構文解析の対象ファイルとプリプロセスオプションの関係を次の表に示します。

表 8–3　構文解析の対象ファイルとプリプロセスオプションの関係

項番	構文解析の対象となるファイル	プリプロセスオプション		
		-E2	-E3	-E2 及び-E3 の指定なし
1	UAP ソースファイル	○	○	×※
2	COPY 文で引き込むヘッダファイル	○	○	×※
3	#include で引き込むヘッダファイル	×	○	×

（凡例）

○：構文解析します。

×：構文解析しません。

注※

選択した規格に準拠した構文解析はしないで，埋込み SQL 宣言節の中だけを埋込み変数のデータ記述形式に合わせて解析します。

なお，構文解析の対象ファイルの中で選択した規格に準拠しない構文が使用されている場合，構文エラーになることがあります。

構文エラーを回避するには，構文解析の対象ファイルの中で選択した規格に準拠した構文を使用するようにします。コンパイラ製品に付属しているヘッダファイル中で選択した規格に準拠しない構文が使用されているために構文エラーになる場合は，選択した規格に準拠したコンパイルをするコンパイラオプションを-E3 オプション文字列に指定することで，問題を回避できることがあります。

例 3：IBM XL C V9.0 を使う場合に，C89 準拠オプション（-qlanglvl=extc89）を指定する場合

```
pdcpp connect.ec -E3”-qlanglvl=extc89;-I$PDDIR/include” -gc89
```

注 3

埋込み変数の宣言に文字集合名 UTF16 を指定した場合に，SQL プリプロセサは文字集合名記述領域に文字集合名を設定するソースコードをポストソース中に展開します。そのソースコードは，-XU16 オプションの指定に依存して，リトルエンディアンの場合には UTF-16LE を設定し，ビッグエンディアンの場合には UTF-16BE を設定します。

埋込み変数を使用しないで，SQL 記述領域と文字集合名記述領域を使用して UAP 実行時に入出力変数の文字集合名を動的に決定する場合には，-XU16 オプションで指定したバイトオーダは無効になります。この場合に，文字集合名を UTF16 にするとバイトオーダがビッグエンディアンになります。

注 4

-XU16 オプションで型指定子を指定する機能の対象となるデータ記述を，次に示します。

- SQL TYPE IS CHAR(2n) CHARACTER SET [IS] [MASTER.]UTF16 変数名;
- SQL TYPE IS CHAR(2n) CHARACTER SET [IS] [MASTER.]UTF16 変数名[m];
- SQL TYPE IS CHAR(2n) CHARACTER SET [IS] [MASTER.]UTF16 *変数名;
- SQL TYPE IS VARCHAR(2n) CHARACTER SET [IS] [MASTER.]UTF16 変数名;
- SQL TYPE IS VARCHAR(2n) CHARACTER SET [IS] [MASTER.]UTF16 変数名[m];
- SQL TYPE IS VARCHAR(2n) CHARACTER SET [IS] [MASTER.]UTF16 *変数名;

注 5

-g オプションの指定内容によって，C99 で追加された予約語の扱いが変わります。C99 で追加された予約語を次に示します。

restrict，inline，_Bool，_Complex，_Imaginary，_Pragma

ただし，restrict と inline については，C89 では変数名などの識別子として使用できましたが，C99 では識別子として使用できません。C89 に準拠している既存の UAP で，restrict と inline を識別子として使用している場合，その UAP を C99 に準拠して解析するとエラーになります。

C99 で追加された予約語の扱いを次の表に示します。

-g オプション	扱い
省略 -gc99 指定	C99 に準拠して予約語として扱います。
-gc89 指定	識別子として扱います。※

注※

先頭文字が下線文字（_）で 2 文字目が英大文字（A～Z），又は下線文字の名前が C89 の場合も，C のライブラリ用として予約されています。このため，_Bool，_Complex，_Imaginary，_Pragma を UAP 中で識別子として使用した場合，プリプロセサの動作は保証できません。

-g オプションの指定内容を次の表に示します。-g オプションの指定内容は，C99 で追加された予約語の扱い以外には影響しません。

C99 で追加された予約語の使用状況	-g オプションの指定内容
予約語として使用する	-g オプションを省略，又は-gc99 を指定します。
識別子として使用する	-gc89 を指定します。
予約語としても識別子としても使用しない	-g オプションを指定する必要はありません。コメントとして任意に指定できます。

-g オプションを指定する場合は，次の点に注意してください。

- C99 に準拠したコンパイルができるコンパイラ製品を使用する場合，コンパイラ製品に付属しているヘッダファイルの中で，C99 で追加された予約語が予約語として使用されていることがあります。-E3 オプションを指定すると，そのヘッダファイルが#include 文で引き込まれるおそれがあります。
- コンパイラ製品の中には，予約語の扱いを個別に指定できるものがあります（例えば，restrict は予約語とし，inline は予約語としない，と指定できます）。SQL プリプロセサは予約語の扱いを個別に指定できません。

なお，UNIX 環境では，トラブルを避けるためコンパイラが準拠する標準規格と同じ標準規格を-g オプションで指定することを推奨します。また，C99 で追加された予約語の扱いを個別に指定するコンパイラオプションは指定しないでください。

注 6

-E2，-E3 オプションを指定しない場合，埋込み SQL 宣言節で，C99 で追加された予約語を識別子として扱います。また，埋込み SQL 宣言節と SQL 文（EXEC SQL ～ ;）以外は解析しません。

注 7

-E1，-E3，又は-Ec オプションを指定すると，プリプロセサは UAP ソースファイルがあるディレクトリに一時ファイルを作成します。一時ファイルはプリプロセス終了時にプリプロセサが削除します。一時ファイルの名称は拡張子を除いて UAP ソースファイルと同じです。拡張子を次に示します。

- プリプロセスオプションが-E1 又は-E3 の場合：.i2
- プリプロセスオプションが-Ec の場合：.i3
- プリプロセスオプションが-Ei の場合：.i4

-E1 又は-E3 オプションと-Ec，-Ei オプションを同時に指定した場合は，それぞれの拡張子の一時ファイルが作成されます。

プリプロセス実行時に一時ファイルと同名のファイルが既に存在すると，プリプロセサはそのファイルを上書きしてから削除します。ただし，プリプロセサの実行ユーザにそのファイルを上書きする権限がない場合，プリプロセサは次のメッセージを出力して処理を終了します。

- 拡張子.i2 のファイルの場合：KFPZ13576-E メッセージ（内部コンパイラ処理でエラー）又は KFPZ13714-E メッセージ（内部 iconv コマンド処理でエラー）
- 拡張子.i3 のファイルの場合：KFPZ13706-E メッセージ（ファイルのオープンでエラー）
- 拡張子.i4 のファイルの場合：KFPZ13576-E メッセージ（内部コンパイラ処理でエラー）

また，プリプロセサの実行ユーザにそのファイルを削除する権限がない場合，プリプロセサはファイルの削除エラーを無視して処理を続行します。

誤って削除されないように，SQL プリプロセサ以外が作成するファイルの拡張子にはこれらの一時ファイルと同じ拡張子を使用しないことをお勧めします。同じ拡張子の同名のファイルを既に使用している場合は，それらを UAP ソースファイルがあるディレクトリに置かないように注意してください。

なお，プリプロセサが異常終了すると，一時ファイルが削除されないで残ることがあります。残った一時ファイルは，同じ UAP ソースファイルに対するプリプロセスが再実行され正常終了すれば，削除されます。

1. C 言語の場合のコマンド指定例

＜例 1＞

UAP ソースファイルの名称が sample で，ポストソースを出力しない場合

```
pdcpp sample.ec -s
```

＜例 2＞

UAP ソースファイルの名称が sample で，出力するポストソースのファイル名称を main にする場合

```
pdcpp sample.ec -o main.c
```

2. C++言語の場合のコマンド指定例

＜例 1＞

UAP ソースファイルの名称が sample で，ポストソースを出力しない場合

```
pdocc sample.EC -s
```

＜例 2＞

UAP ソースファイルの名称が sample で，出力するポストソースのファイル名称を main にする場合

```
pdocc sample.EC -o main.C
```

(c) SQL プリプロセサのリターンコード

SQL プリプロセサは，処理終了後にリターンコードを OS に返します。

リターンコードは，シェル変数$?（ボーンシェルの場合），又は$status（C シェルの場合）の内容を表示させることで参照できます。

リターンコードとその内容を次の表に示します。

表 8–4　SQL プリプロセサのリターンコード（UNIX 環境での C 言語の場合）

リターンコード	内　　容
0	正常終了
4，8	エラー発生（プリプロセスを最後まで続行）
12，16	エラー発生（プリプロセスを途中で終了）

(d)　エラーの出力

SQL プリプロセサは，SQL 文に文法上のエラーが発生した場合，その SQL 文を無視して処理を続行します。しかし，オプションの指定に誤りがある場合，処理を中断します。また，メモリ不足やファイル入出力エラーなど，システム上でエラーが発生し，それ以降の処理ができない場合，処理の途中でも終了します。

SQL 文に文法上のエラーが発生したとき，SQL プリプロセサはエラーメッセージを標準出力へ出力します。標準出力をリダイレクトすると，エラーメッセージをファイルに格納できます。ファイルを参照すると，エラー内容，UAP ソースファイル名，エラーが発生した箇所（SQL 文の行番号）などが分かります。

(2)　COBOL 言語の場合

(a)　環境変数の設定

UAP をプリプロセスする前に，次に示す環境変数を必要に応じて設定します。

PDDIR：

HiRDB（サーバ，又はクライアント）のインストールディレクトリを絶対パス名で指定します。この環境変数を設定しないと，/HiRDB が仮定されます。

なお，インストール先が/HiRDB の場合，この変数を設定する必要はありません。

PDCBLFIX：

UAP ソースファイルの規定の識別子以外に，任意のファイル識別子を使用する場合に指定します。

ファイル識別子は，ピリオドで始まる 4 文字までの任意の文字列を設定します。なお，この環境変数に設定したファイル識別子は，入力ファイルにだけ使用できます。

PDCBLLIB：

SQL の COPY 文でソースファイルに引き込まれる登録集原文を検索するディレクトリを指定します。複数のディレクトリを指定する場合，ディレクトリ同士をコロンで区切ります。この環境変数を省略すると，カレントディレクトリだけが検索されます。

LANG：

HiRDB クライアント環境の文字コード種別を設定します。

SQL プリプロセサは，この環境変数に設定されたロケール名から，UAP のソースファイルの文字コード種別を識別します。ただし，SQL プリプロセサがサポートしていないロケール名を指定すると，SQL プリプロセサは C(単一バイト文字コード)を指定したとみなします。UAP ソースの文字コード種別を示

すロケール名を SQL プリプロセサがサポートしていない場合は，LANG の代わりに環境変数 PDCLTLANG を指定してください。

SQL プリプロセサがサポートしているロケール名については，表「SQL プリプロセサがサポートしているロケール名」を参照してください。

PDCLTLANG を指定した場合，SQL プリプロセサは LANG の指定を無視します。

PDCLTLANG：

UAP ソースの文字コード種別を LANG で指定できない場合，及び LANG で指定した文字コード種別を無視して，ほかの文字コード種別でプリプロセスする場合に指定します。詳細については「クライアント環境定義の設定内容」の「PDCLTLANG」を参照してください。

PDDEFAULTOPTION：

プリプロセスのオプションについて，省略時の動作を指定します。通常は，この環境変数を指定する必要はありません。互換モードを適用したい場合に，指定してください。詳細については，「クライアント環境定義の設定内容」の「PDDEFAULTOPTION」を参照してください。

＜例 1＞（sh（ボーンシェル）で環境設定をする場合）

```
$ PDDIR=”/prdb”                                    … 1
$ PDCBLFIX=”.Cob”                                  … 2
$ PDCBLLIB=$HOME/cobol/include:$HOME/cobol/source … 3
$ export PDDIR PDCBLFIX PDCBLLIB                   … 4
```

＜説明＞

1. インストールディレクトリ（この例では/prdb）を指定します。
2. UAP ソースファイルの識別子として，.Cob も有効にします。
3. 登録集原文を検索するディレクトリ（この例では$HOME/cobol/include，及び$HOME/cobol/source）を指定します。
4. SQL プリプロセサで参照できるようにします。

＜例 2＞（csh（C シェル）で環境設定をする場合）

```
% setenv   PDDIR ”/prdb”                                  … 1
% setenv   PDCBLFIX ”.Cob”                                … 2
% setenv   PDCBLLIB $HOME/cobol/include:$HOME/cobol/source … 3
```

＜説明＞

1. インストールディレクトリ（この例では/prdb）を指定します。
2. UAP ソースファイルの識別子として，.Cob も有効にします。
3. 登録集原文を検索するディレクトリ（この例では$HOME/cobol/include，及び$HOME/cobol/source）を指定します。

(b)　SQL プリプロセサの起動

SQL プリプロセサの起動は，pdcbl コマンド（COBOL 言語の場合），又は pdocb コマンド（OOCOBOL 言語の場合）を使用します。

SQL プリプロセサを起動するコマンドの入力形式を次に示します。

```
pdcbl 入力ファイル名称 ［オプション［出力ファイル名称｜認可識別子］］
```

注　OOCOBOL 言語の場合，下線で示す部分を pdocb に置き換えてください。

入力ファイル名称：

UAP ソースファイルの名称を指定します。

ファイル識別子は，.ecb，.cob，.cbl のどれか（COBOL 言語の場合），又は.eoc（OOCOBOL 言語の場合）にします。

なお，環境設定で任意のファイル識別子を登録してある場合，その識別子を使用することもできます。

出力ファイル名称：

ポストソースファイルの名称を指定します。

出力ファイル名称を省略した場合，ファイル識別子は.cbl（COBOL 言語の場合），又は.ocb（OOCOBOL 言語の場合）になります。

認可識別子：

SQL で認可識別子を省略した場合に仮定する認可識別子を指定します。認可識別子を省略した場合，CONNECT 時のユーザ識別子が仮定されます。

オプション：

必要に応じて次の表に示すオプションを指定します。なお，オプションは大文字，小文字を区別しません。

表 8–5　プリプロセスオプション（UNIX 環境の COBOL 言語の場合）

プリプロセスオプション	内容
-s	構文チェックだけをして，ポストソースを出力しない場合に指定します。このオプションを省略すると，ポストソースが出力されます。
-o ファイル名	出力するポストソースのファイル名称を変更する場合に指定します。 このオプションを省略すると，入力ファイル名称のファイル識別子を.cbl（COBOL 言語の場合），又は.ocb（OOCOBOL 言語の場合）に変更されたものが出力ファイル名称になります。 入力ファイルの識別子が.cbl（COBOL 言語の場合），又は.ocb（OOCOBOL 言語の場合）の場合，必ずこのオプションを指定して，ポストソースのファイル名称を.cbl（COBOL 言語の場合）以外，又は.ocb 以外（OOCOBOL 言語の場合）の識別子に変更してください。
-Xc	次の英数字定数を囲む分離符を，引用符（"）に変更する場合に指定します。 このオプションを省略すると，次の英数字定数を囲む分離符はアポストロフィ（'）になります。 • SQL プリプロセサが生成するソースコード中の英数字定数 • SQL プリプロセサが生成する COPY 文でポストソースに引き込む登録集原文ファイル中の英数字定数

プリプロセスオプション	内容
-A 認可識別子	静的 SQL で認可識別子を省略した場合，及び仮定する認可識別子を変更する場合に指定します。 静的 SQL とは，INSERT 文，UPDATE 文，DELETE 文，1 行 SELECT 文，OPEN 文（形式 1），CALL 文，LOCK 文，及び PURGE TABLE 文を示します。 認可識別子を引用符（"）で囲んで指定することはできません。指定した認可識別子は常に大文字と小文字を区別して扱います。SQL 文中に引用符で囲まないで指定した認可識別子は小文字を大文字として扱います。そのため，ほかの UAP などに記述した SQL 文中に引用符で囲まないで小文字を使って指定した認可識別子と同じ認可識別子をこのオプションに指定する場合は，すべての英字を大文字で指定してください。 最大 30 バイトの認可識別子を指定できます。 認可識別子を省略した場合に仮定する認可識別子については，マニュアル「HiRDB SQL リファレンス」の「名前の修飾」を参照してください。
-h64	64 ビットモード用のポストソースを作成する場合に指定します。ただし，32 ビット版のプリプロセサを使用した場合は指定できません。
-P	SQL の構文チェックをしない場合に指定します。次のどれかの UAP をプリプロセスする場合に指定できます。 • XDM/RD E2 接続用の UAP • SQL 予約語削除機能を使用する UAP • 集合演算子 MINUS を使用する UAP これらの UAP をプリプロセスするときにこのオプションを指定しないと，XDM/RD E2 又は HiRDB サーバで使用できる SQL が構文エラーとなることがあります。
-Xo	UAP から抽出した SQL 文を標準出力へ出力する場合に指定します。このとき，出力方法は次のようになります。 • SQL 文中の埋込み変数は?パラメタに置換します。 • 1 行 SELECT 文の INTO 句を削除します。 • SQL 文中の語句間の空白が 2 文字以上の場合，空白 1 文字に置き換えます。 • 複数行に分割して記述している SQL は 1 行にまとめます。 • 実行時にサーバに送られる SQL だけ出力します。実行されない SQL 文（WHENEVER 文，BEGIN DECLARE SECTION など）は出力しません。 • SQL の末尾にはセミコロン（;）を付けます。 • 埋込み変数の宣言は出力しません。 • 動的 SQL は，SQL がリテラルで指定されている場合にだけ出力します。そのほかの場合は出力しません。 • OPEN 文は，形式 1 のカーソルの場合だけ，問合せ式を出力します。 • ポストソースは生成しません。
-Xe {y | n}	PREPRARE 文実行時のカーソルを，自動的にクローズするかどうかを指定します。 y： カーソルを自動的にクローズするポストソースを生成します。 n： カーソルを自動的にクローズしないポストソースを生成します。 省略した場合，クライアント環境定義 PDPRPCRCLS の指定値に従いポストソースを生成します。

プリプロセスオプション	内容
-E2	埋込み変数を埋込み SQL 宣言節で宣言しないで使用する場合に指定します。
-XU16[L ｜ B]	COBOL2002 の Unicode 機能を使用する場合に指定します。COBOL2002 の Unicode 機能を使用した UAP の実行については，「COBOL2002 の Unicode 機能を使用した UAP の実行」を参照してください。 日本語項目の文字コード（UTF-16）のバイトオーダを指定します。 -XU16L： UTF-16 のバイトオーダをリトルエンディアンにします。 -XU16B： UTF-16 のバイトオーダをビッグエンディアンにします。 -XU16： UTF-16 のバイトオーダを，プリプロセスを実行する OS のバイトオーダにします。 • Windows，Linux の場合はリトルエンディアンにします。 • AIX の場合はビッグエンディアンにします。 上記以外の OS でこのオプションを指定した場合は，プリプロセサの動作は保証できません。 COBOL2002 の Unicode 機能を使用しない場合は，このオプションを指定しないでください。指定すると，埋込み変数の宣言に日本語項目を使用できなくなります。
-Xm	複数接続機能を使用した UAP をプリプロセスする場合に指定します。詳細は，「COBOL 言語で複数接続機能を使用する場合の注意事項」を参照してください。 このオプションは，省略時にも仮定されます。ただし，0904 互換モードを適用した場合は仮定されませんので，オプションを指定することを検討してください。
-Xs	UAP に埋め込んだ SQL 文中に単純注釈（-- ～ 改行）を記述する場合に指定します。 単純注釈の記述規則については，「SQL の記述規則」の 13.を参照してください。 このオプションは，省略時にも仮定されます。ただし，0904 互換モードを適用した場合は仮定されませんので，オプションを指定することを検討してください。
-Xb	Linux 環境で，COBOL2002 のコンパイラオプション-BigEndian,Bin を使用して 2 進項目のバイトオーダをビッグエンディアンにする場合に指定します。詳細は，「COBOL のビッグエンディアンオプションへの対応」を参照してください。 Linux 以外の UNIX 環境では，このオプションを指定できません。
-Xr	SQL 文の実行時に COBOL の RETURN-CODE 特殊レジスタを更新しないポストソースを生成します。 UAP が呼び出し元プログラムに返すリターンコードを RETURN-CODE 特殊レジスタに設定してから，呼び出し元プログラムに制御を戻すまでの間に SQL 文を実行する場合，このオプションを指定してください。 省略した場合，SQL 文の種類によって RETURN-CODE 特殊レジスタの更新有無が変わります。SQL 文の種類による RETURN-CODE 特殊レジスタの更新有無については，「SQL の記述規則」の 18.を参照してください。

1. COBOL 言語の場合のコマンド指定例

＜例 1＞

UAP ソースファイルの名称が sample で，ポストソースを出力しない場合

```
pdcbl sample.ecb -s
```

＜例 2＞

UAP ソースファイルの名称が sample で，出力するポストソースのファイル名称を main にする場合

```
pdcbl sample.ecb -o main.cbl
```

2. OOCOBOL 言語の場合のコマンド指定例

＜例 1＞

UAP ソースファイルの名称が sample で，ポストソースを出力しない場合

```
pdocb sample.eoc -s
```

＜例 2＞

UAP ソースファイルの名称が sample で，出力するポストソースのファイル名称を main にする場合

```
pdocb sample.eoc -o main.ocb
```

(c)　SQL プリプロセサのリターンコード

SQL プリプロセサは，処理終了後にリターンコードを OS に返します。

リターンコードは，シェル変数$?（ボーンシェルの場合），又は$status（C シェルの場合）の内容を表示させることで参照できます。

リターンコードとその内容を次の表に示します。

表 8–6　SQL プリプロセサのリターンコード（UNIX 環境での COBOL 言語の場合）

リターンコード	内　　容
0	正常終了
4，8	エラー発生（プリプロセスを最後まで続行）
12，16	エラー発生（プリプロセスを途中で終了）

(d)　エラーの出力

SQL プリプロセサは，SQL 文に文法上のエラーが発生した場合，その SQL 文を無視して処理を続行します。しかし，オプションの指定に誤りがある場合，処理を中断します。また，メモリ不足やファイル入出力エラーなど，システム上でエラーが発生し，それ以降の処理ができない場合，処理の途中でも終了します。

SQL 文に文法上のエラーが発生したとき，SQL プリプロセサはエラーメッセージを標準出力へ出力します。標準出力をリダイレクトすると，エラーメッセージをファイルに格納できます。ファイルを参照すると，エラー内容，UAP ソースファイル名，エラーが発生した箇所（SQL 文の行番号）などが分かります。

8.2.3　Windows 環境でのプリプロセス

(1)　C 言語の場合

(a)　環境変数の設定

UAP をプリプロセスする前に，必要に応じて HiRDB.INI ファイルに次に示す環境変数を設定します。

なお，HiRDB.INI ファイルは，%windir%ディレクトリに格納されています。

PDCLIB：

COPY 文でソースファイルに引き込まれるヘッダファイルを検索するディレクトリを指定します。複数のディレクトリを指定する場合，ディレクトリ同士をセミコロンで区切ります。この環境変数を省略すると，カレントディレクトリだけが検索されます。

PDCLTLANG：

特に文字コード種別を指定してプリプロセスする場合に指定します。省略した場合は，sjis が仮定されます。詳細については，「クライアント環境定義の設定内容」の「PDCLTLANG」を参照してください。

PDDEFAULTOPTION：

プリプロセスのオプションについて，省略時の動作を指定します。通常は，この環境変数を指定する必要はありません。互換モードを適用したい場合に，指定してください。詳細については，「クライアント環境定義の設定内容」の「PDDEFAULTOPTION」を参照してください。

(b)　SQL プリプロセサの起動

SQL プリプロセサの起動は，次に示す三つの方法があります。

- アイコンの重ね合わせによる実行
- ファイル名の指定による実行
- コマンドプロンプト又は MS-DOS プロンプトからの実行

- **アイコンの重ね合わせによる実行**

  エクスプローラでプリプロセスしたいファイルをドラッグして，プリプロセサのファイル（PDCPP.EXE（C 言語の場合），又は PDOCC.EXE（C++言語の場合））に重ねると実行できます。

- **ファイル名の指定による実行**

  プリプロセサのアイコン（PDCPP.EXE（C 言語の場合），又は PDOCC.EXE（C++言語の場合））をクリックし，次に示す手順で実行します。

  1. ファイルメニューから「ファイル名を指定して実行」を選択します。
  2. コマンドラインにファイル名，及びオプションを指定して実行します。

- **コマンドプロンプト又は MS-DOS プロンプトからの実行**

  コマンドプロンプト又は MS-DOS プロンプトを起動し，PDCPP.EXE（C 言語の場合），又は PDOCC.EXE（C++言語の場合）コマンドを入力して実行します。

コマンドラインに入力するコマンドの形式を次に示します。

```
PDCPP.EXE　入力ファイル名称　［オプション［出力ファイル名称］］
```

注　C++言語の場合，下線で示す部分を PDOCC.EXE に置き換えてください。

入力ファイル名称：

UAP ソースファイルの名称を指定します。

ファイル識別子は，.EC（C 言語の場合），又は.ECP（C++言語の場合）にします。

出力ファイル名称：

ポストソースファイルの名称を指定します。

出力ファイル名称を省略した場合，ファイル識別子は.C（C 言語の場合），又は.CPP（C++言語の場合）になります。

認可識別子：

SQL で認可識別子を省略した場合に仮定する認可識別子を指定します。認可識別子を省略した場合，CONNECT 時のユーザ識別子が仮定されます。

オプション：

必要に応じて次の表に示すオプションを指定します。なお，オプションは大文字，小文字を区別しません。

表 8–7　プリプロセスオプション（Windows 環境の C 言語の場合）

プリプロセスオプション	内容
/S	構文チェックだけをして，ポストソースを出力しない場合に指定します。このオプションを省略すると，ポストソースが出力されます。 なお，同時に/Xp も指定しないと，SQL プリプロセサは厳密な SQL の構文チェックができないため，SQL 文に文法上のエラーが発生していても発見できない場合があるので注意が必要です。
/O ファイル名	出力するポストソースのファイル名称を指定します。 このオプションを省略すると，入力ファイル名称のファイル識別子を.C（C 言語の場合），又は.CPP（C++言語の場合）に変更されたものが出力ファイル名称になります。
/A 認可識別子	静的 SQL で認可識別子を省略した場合，及び仮定する認可識別子を変更する場合に指定します。 静的 SQL とは，INSERT 文，UPDATE 文，DELETE 文，1 行 SELECT 文，OPEN 文（形式 1），CALL 文，LOCK 文，及び PURGE TABLE 文を示します。 認可識別子を引用符（"）で囲んで指定することはできません。指定した認可識別子は常に大文字と小文字を区別して扱います。SQL 文中に引用符で囲まないで指定した認可識別子は小文字を大文字として扱います。そのため，ほかの UAP などに記述した SQL 文中に引用符で囲まないで小文字を使って指定した認可識別子と同じ認可識別子をこのオプションに指定する場合は，すべての英字を大文字で指定してください。 最大 30 バイトの認可識別子を指定できます。 認可識別子を省略した場合に仮定する認可識別子については，マニュアル「HiRDB SQL リファレンス」の「名前の修飾」を参照してください。

プリプロセスオプション	内容
/h64	64 ビットモード用のポストソースを作成する場合に指定します。ただし，32 ビット版のプリプロセサを使用した場合は指定できません。 long 型を使用した埋込み変数の宣言はエラーになります。
/Xe {y | n}	PREPRARE 文実行時のカーソルを，自動的にクローズするかどうかを指定します。 y： カーソルを自動的にクローズするポストソースを生成します。 n： カーソルを自動的にクローズしないポストソースを生成します。 省略した場合，クライアント環境定義 PDPRPCRCLS の指定値に従いポストソースを生成します。
/Xv	/E2 オプションを指定した場合に，VARCHAR 型，及び BINARY 型に対応する構造体を，通常の構造体として解析するときに指定します。VARCHAR 型，及び BINARY 型に対応する埋込み変数を宣言するためには，SQL TYPE IS～を使用します。このオプションは，/E2 オプションと同時に指定する必要があります。なお，繰返し列のマクロを使用している場合，このオプションを指定しないでください。
/XA	X/Open に準じた API を使用して，UAP を作成する場合に指定してください。
/Xo	UAP から抽出した SQL 文を標準出力へ出力する場合に指定します。このとき，出力方法は次のようになります。 • SQL 文中の埋込み変数は?パラメタに置換します。 • 1 行 SELECT 文の INTO 句を削除します。 • SQL 文中の語句間の空白が 2 文字以上の場合，空白 1 文字に置き換えます。 • 複数行に分割して記述している SQL は 1 行にまとめます。 • 実行時にサーバに送られる SQL だけ出力します。実行されない SQL 文（WHENEVER 文，BEGIN DECLARE SECTION など）は出力しません。 • SQL の末尾にはセミコロン（;）を付けます。 • 埋込み変数の宣言は出力しません。 • 動的 SQL は，SQL がリテラルで指定されている場合にだけ出力します。そのほかの場合は出力しません。 • OPEN 文は，形式 1 のカーソルの場合だけ，問合せ式を出力します。 • ポストソースは生成しません。
/E {1 | 2 | 3} ["オプション文字列"]	UAP 中で使用しているプリプロセサ宣言文の有効化と，埋込み変数を埋込み SQL 宣言節で宣言しないで使用する場合に指定します。なお，このオプションは PDOCC.EXE では無効となります。 /E1： プリプロセサ宣言文の有効化を指定します。 /E2： 埋込み変数を埋込み SQL 宣言節で宣言しないで使用する場合に指定します。また，ポインタでの埋込み変数指定，及び構造体の参照を使用する場合にも指定します。 /E3： /E1 と/E2 の両方を指定します。

プリプロセスオプション	内容
	"オプション文字列"： インクルードするファイルを検索するディレクトリのパス名を，C コンパイラに指定する/I オプションの形式で指定します。オプション文字列に複数のオプションを記述する場合は，セミコロンで区切ってください。また，任意の C コンパイラも指定できます。なお，/E2 オプション指定時には無視されます。 ≪留意事項≫ • /E1 又は/E3 を指定すると，内部的に C コンパイラが呼び出されるため，PATH 環境変数にコンパイラへのパス名を設定しておく必要があります。 • /E1 又は/E3 を指定すると，UAP ソースファイルがあるディレクトリに作業用の一時ファイルを作成します。詳細は表「構文解析の対象ファイルとプリプロセスオプションの関係」の注 7 を参照してください。 • /E1 又は/E3 を指定し，UAP ソースファイル中で SQL の COPY 文を使用する場合は，-Ec オプションを同時に指定する必要があります。 • -E3 オプション指定時にインクルードするヘッダファイルについては，ISO/IEC 9899:1999 で定義された標準ヘッダファイルをサポートしています。それ以外のヘッダファイルについては，プリプロセス実行時にエラーとなることがあります。 • SQL で指定するキーワード及び使用する C コンパイラの解析方法によっては，作成したポストソースのコンパイル時にエラーが発生するおそれがあります。この場合，COPY 文の使用有無に関係なく，Ec オプションを同時に指定することで，コンパイル時のエラーを回避できることがあります。
/Xp	厳密な SQL の構文チェックをする場合に指定してください。ただし，次の UAP をプリプロセスする場合は指定しないでください。 • XDM/RD E2 接続用の UAP • SQL 予約語削除機能を使用する UAP • 集合演算子 MINUS を使用する UAP これらの UAP をプリプロセスするときにこのオプションを指定すると，XDM/RD E2 又は HiRDB サーバで使用できる SQL が構文エラーとなることがあります。
/XU16[L ｜ B][T"型指定子"]	• UTF-16 のバイトオーダを指定します。 埋込み変数の宣言で文字集合名 UTF16 を指定した場合に，埋込み変数に格納する UTF-16 のバイトオーダを指定します。 /XU16L[T"型指定子"]： UTF-16 のバイトオーダをリトルエンディアンにします。 /XU16B[T"型指定子"]： UTF-16 のバイトオーダをビッグエンディアンにします。 L/B を省略，又はこのオプションを省略した場合は，UTF-16 のバイトオーダを，プリプロセスを実行する OS のバイトオーダにします。 • UTF-16 の埋込み変数の展開に使用する型指定子を指定します。 SQL TYPE IS CHAR，又は SQL TYPE IS VARCHAR で始まるデータ記述で埋込み変数を宣言した場合に，ポストソース中に展開される C の宣言の中で UTF-16 の文字データを格納する変数，又は構造体メンバに付ける型指定子を指定します。 型指定子が示す型の配列要素 1 個に，UTF-16 の 2 バイトの文字データ 1 個を格納するので，sizeof(型指定子)==2 となる型指定子だけが指定できます。

プリプロセスオプション	内容
	<型指定子の指定例> 次に示す指定例は，sizeof(wchar_t)==2 となるコンパイラでだけ使用できます。 /XU16T"wchar_t" 次に示す指定例は，sizeof(unsigned short)==2 なので，どのコンパイラでも使用できます。 /XU16T"unsigned short" T"型指定子"を省略，又はこのオプションを省略した場合，型指定子に char が使用されます。このとき，char 型の配列要素 2 個に UTF-16 の 2 バイトの文字データ 1 個を格納します。
/g｛c89 ｜ c99｝	/E2，/E3 オプションを指定した場合，SQL プリプロセサが UAP ソースを解析するとき準拠する C の標準規格を指定します。 /gc89： C89（ISO/IEC 9899:1990, Programming languages - C）に準拠します。 /gc99： C99（ISO/IEC 9899:1999, Programming languages - C）に準拠します。 省略した場合，/gc89 が仮定されます。 /E2，/E3 オプションを指定しない場合，このオプションを指定しても無視されます。
/Ec	/E1 又は/E3 オプションを指定した場合に，UAP ソースファイル中で SQL の COPY 文を使用できるようにします。 なお，それ以外の場合，このオプションの指定に関係なく COPY 文は使用できます（このオプションの指定は無視されます）。 ≪留意事項≫ • このオプションを指定すると UAP ソースファイルがあるディレクトリに作業用の一時ファイルを作成します。詳細は表「構文解析の対象ファイルとプリプロセスオプションの関係」の注 7 を参照してください。 • SQL の COPY 文の使用有無に関係なく，-Ec オプションを指定することで，プリプロセス時に発生する問題を回避できることがあります。詳細は/E オプションの説明を参照してください。
/Xs	UAP に埋め込んだ SQL 文中に単純注釈（-- ～ 改行）を記述する場合に指定します。 単純注釈の記述規則については，「SQL の記述規則」の 10.を参照してください。 このオプションは，省略時にも仮定されます。ただし，0904 互換モードを適用した場合は仮定されませんので，オプションを指定することを検討してください。

注 1

/E オプション指定時に使用できる機能を次に示します。

機能	省略	/E1	/E2	/E3
#define で定義したマクロを有効にする	×	○	×	○
#include でインクルードしたヘッダファイルを有効にする※	×	○	×	○
#if，#ifdef などの条件コンパイルを有効にする	×	○	×	○
UAP 中の任意の場所で宣言した変数を，埋込み変数として使用する	×	×	○	○

機能	省略	/E1	/E2	/E3
構造体を埋込み変数として使用する	×	×	○	○
ポインタを埋込み変数として使用する	×	×	○	○

（凡例）

○：該当する機能を使用できます。

×：該当する機能は使用できません。

注※

ヘッダファイル中に記述できる項目については，「SQL の記述規則」の 23.を参照してください。

注 2

/E オプションを指定した場合，プリプロセサは Microsoft Visual C++コンパイラ（呼び出し時のロード名：CL.EXE）を内部的に呼び出します。

これ以外の C コンパイラを使用する場合は，オプション文字列の先頭に，コンパイラのロードのディレクトリを含めた絶対パス名で指定します。ディレクトリ名，及びロード名には，空白及びセミコロンを含めることはできません。環境変数 PATH にパス名を追加している場合は，絶対パス名でなくてもかまいません。

ロード名を指定する場合は，ロード名とオプションの間をセミコロンで区切ってください。

使用するコンパイラは，/C オプションと/E オプションをサポートしている必要があります。プリプロセサは，#define，#include などの擬似命令を処理するために，内部的に C コンパイラに対して/C オプション，及び/E オプションを指定して，作業用の一時ファイルを作成します。それ以外のオプション文字列に指定できるオプションは，使用するコンパイラの仕様に依存しますが，/C オプション又は/E オプションに背反するオプションを指定すると，プリプロセサはエラーとなります。ヘルプなどを表示するオプションを使用した場合は，動作が保証されません。

/E2，/E3 オプションを指定すると，プリプロセサは UAP 中の任意の箇所で宣言されている埋込み変数を認識するために，/g オプションで指定，又は仮定された C の標準規格（C89 又は C99）に準拠して構文を解析します。

構文解析の対象ファイルとプリプロセスオプションの関係を次の表に示します。

表 8–8　構文解析の対象ファイルとプリプロセスオプションの関係

項番	構文解析の対象となるファイル	プリプロセスオプション		
		-E2	-E3	-E2 及び-E3 の指定なし
1	UAP ソースファイル	○	○	×※
2	COPY 文で引き込むヘッダファイル	○	○	×※
3	#include で引き込むヘッダファイル	×	○	×

（凡例）

○：構文解析します。

×：構文解析しません。

注※

選択した規格に準拠した構文解析はしないで，埋込み SQL 宣言節の中だけを埋込み変数のデータ記述形式に合わせて解析します。

なお，構文解析の対象ファイルの中で選択した規格に準拠しない構文が使用されている場合，構文エラーになることがあります。

構文エラーを回避するには，構文解析の対象ファイルの中で選択した規格に準拠した構文を使用するようにします。コンパイラ製品に付属しているヘッダファイル中で選択した規格に準拠しない構文が使用されているために構文エラーになる場合は，選択した規格に準拠したコンパイルをするコンパイラオプションを/E3 オプション文字列に指定することで，問題を回避できることがあります。

注 3

埋込み変数の宣言に文字集合名 UTF16 を指定した場合に，SQL プリプロセサは文字集合名記述領域に文字集合名を設定するソースコードをポストソース中に展開します。そのソースコードは，/XU16 オプションの指定に依存して，リトルエンディアンの場合には UTF-16LE を設定し，ビッグエンディアンの場合には UTF-16BE を設定します。

埋込み変数を使用しないで，SQL 記述領域と文字集合名記述領域を使用して UAP 実行時に入出力変数の文字集合名を動的に決定する場合には，/XU16 オプションで指定したバイトオーダは無効になります。この場合に，文字集合名を UTF16 にするとバイトオーダがビッグエンディアンになります。

注 4

/XU16 オプションで型指定子を指定する機能の対象となるデータ記述を，次に示します。

- SQL TYPE IS CHAR(2n) CHARACTER SET [IS] [MASTER.]UTF16 変数名;
- SQL TYPE IS CHAR(2n) CHARACTER SET [IS] [MASTER.]UTF16 変数名[m];
- SQL TYPE IS CHAR(2n) CHARACTER SET [IS] [MASTER.]UTF16 *変数名;
- SQL TYPE IS VARCHAR(2n) CHARACTER SET [IS] [MASTER.]UTF16 変数名;
- SQL TYPE IS VARCHAR(2n) CHARACTER SET [IS] [MASTER.]UTF16 変数名[m];
- SQL TYPE IS VARCHAR(2n) CHARACTER SET [IS] [MASTER.]UTF16 *変数名;

注 5

/g オプションの指定内容によって，C99 で追加された予約語の扱いが変わります。C99 で追加された予約語を次に示します。

restrict，inline，_Bool，_Complex，_Imaginary，_Pragma

ただし，restrict と inline については，C89 では変数名などの識別子として使用できましたが，C99 では識別子として使用できません。C89 に準拠している既存の UAP で，restrict と inline を識別子として使用している場合，その UAP を C99 に準拠して解析するとエラーになります。

C99 で追加された予約語の扱いを次の表に示します。

/g オプション	扱い
省略	識別子として扱います。※
/gc99 指定	C99 に準拠して予約語として扱います。

/g オプション	扱い
/gc89 指定	識別子として扱います。※

注※

先頭文字が下線文字（_）で 2 文字目が英大文字（A～Z），又は下線文字の名前が C89 の場合も，C のライブラリ用として予約されています。このため，_Bool，_Complex，_Imaginary，_Pragma を UAP 中で識別子として使用した場合，プリプロセサの動作は保証できません。

/g オプションの指定内容を次の表に示します。/g オプションの指定内容は，C99 で追加された予約語の扱い以外には影響しません。

C99 で追加された予約語の使用状況	/g オプションの指定内容
予約語として使用する	/gc99 を指定することで，SQL プリプロセサを実行できます。ただし，Visual C++(バージョン 2008 まで)は C99 に対応していないので，ポストソースをコンパイルできません。Visual C++以外のコンパイラを使用した場合，動作は保証できません。
識別子として使用する	/g オプションを省略，又は/gc89 を指定します。
予約語としても識別子としても使用しない	/g オプションを指定する必要はありません。コメントとして任意に指定できます。

/g オプションを指定する場合は，次の点に注意してください。

- C99 に準拠したコンパイルができるコンパイラ製品を使用する場合，コンパイラ製品に付属しているヘッダファイルの中で，C99 で追加された予約語が予約語として使用されていることがあります。/E3 オプションを指定すると，そのヘッダファイルが#include 文で引き込まれるおそれがあります。
- コンパイラ製品の中には，予約語の扱いを個別に指定できるものがあります（例えば，restrict は予約語とし，inline は予約語としない，と指定できます）。SQL プリプロセサは予約語の扱いを個別に指定できません。

注 6

/E2，/E3 オプションを指定しない場合，埋込み SQL 宣言節で，C99 で追加された予約語を識別子として扱います。また，埋込み SQL 宣言節と SQL 文（EXEC SQL ～ ;）以外は解析しません。

注 7

-E1，-E3，又は-Ec オプションを指定すると，プリプロセサは UAP ソースファイルがあるディレクトリに一時ファイルを作成します。一時ファイルはプリプロセス終了時にプリプロセサが削除します。一時ファイルの名称は拡張子を除いて UAP ソースファイルと同じです。拡張子を次に示します。

- プリプロセスオプションが-E1 又は-E3 の場合：.I2
- プリプロセスオプションが-Ec の場合：.I3

-E1 又は-E3 オプションと-Ec オプションを同時に指定した場合は，両方の拡張子の一時ファイルが作成されます。

プリプロセス実行時に一時ファイルと同名のファイルが既に存在すると，プリプロセサはそのファイルを上書きしてから削除します。ただし，プリプロセサの実行ユーザにそのファイルを上書きする権限がない場合，プリプロセサは次のメッセージを出力して処理を終了します。

- 拡張子.I2 のファイルの場合：KFPZ13576-E メッセージ（内部コンパイラ処理でエラー）
- 拡張子.I3 のファイルの場合：KFPZ13706-E メッセージ（ファイルのオープンでエラー）

また，プリプロセサの実行ユーザにそのファイルを削除する権限がない場合，プリプロセサはファイルの削除エラーを無視して処理を続行します。

誤って削除されないように，SQL プリプロセサ以外が作成するファイルの拡張子にはこれらの一時ファイルと同じ拡張子を使用しないことをお勧めします。同じ拡張子の同名のファイルを既に使用している場合は，それらを UAP ソースファイルがあるディレクトリに置かないように注意してください。

なお，プリプロセサが異常終了すると，一時ファイルが削除されないで残ることがあります。残った一時ファイルは，同じ UAP ソースファイルに対するプリプロセスが再実行され正常終了すれば，削除されます。

1. C 言語の場合のコマンド指定例

＜例 1＞

UAP ソースファイルの名称が SAMPLE で，ポストソースを出力しない場合

```
PDCPP SAMPLE.EC /S
```

＜例 2＞

UAP ソースファイルの名称が SAMPLE で，出力するポストソースのファイル名称を MAIN にする場合

```
PDCPP SAMPLE.EC /O MAIN.C
```

2. C++言語の場合のコマンド指定例

＜例 1＞

UAP ソースファイルの名称が SAMPLE で，ポストソースを出力しない場合

```
PDOCC.EXE SAMPLE.ECP /S
```

＜例 2＞

UAP ソースファイルの名称が SAMPLE で，出力するポストソースのファイル名称を MAIN にする場合

```
PDOCC.EXE SAMPLE.ECP /O MAIN.CPP
```

(c) SQL プリプロセサのリターンコード

SQL プリプロセサは，処理終了後にリターンコードを OS に返します。

リターンコードは，OS のバッチコマンド ERRORLEVEL で参照できます。

リターンコードとその内容を次の表に示します。

表 8–9　SQL プリプロセサのリターンコード（Windows 環境での C 言語の場合）

リターンコード	内　　容
0	正常終了
4，8	エラー発生（プリプロセスを最後まで続行）
12，16	エラー発生（プリプロセスを途中で終了）

(d)　エラーの出力

SQL プリプロセサは，SQL 文に文法上のエラーが発生した場合，その SQL 文を無視して処理を続行します。しかし，オプションの指定に誤りがある場合，処理を中断します。また，メモリ不足やファイル入出力エラーなど，システム上でエラーが発生し，それ以降の処理ができない場合，処理の途中でも終了します。

SQL 文に文法上のエラーが発生したとき，SQL プリプロセサはエラーメッセージを標準出力へ出力します。標準出力をリダイレクトすると，エラーメッセージをファイルに格納できます。ファイルを参照すると，エラー内容，UAP ソースファイル名，エラーが発生した箇所（SQL 文の行番号）などが分かります。

(2)　COBOL 言語の場合

(a)　環境変数の設定

UAP をプリプロセスする前に，必要に応じて HIRDB.INI ファイルに次に示す環境変数を設定します。

なお，HIRDB.INI ファイルは，%windir%ディレクトリに格納されています。

PDCBLFIX：

UAP ソースファイルの規定の識別子以外に，任意のファイル識別子を使用する場合に指定します。ファイル識別子は，ピリオドで始まる 4 文字までの任意の文字列を設定します。ファイル識別子で設定する英字は大文字で設定する必要があります。なお，この環境変数に設定したファイル識別子は，入力ファイルにだけ使用できます。

PDCBLLIB：

SQL の COPY 文でソースファイルに引き込まれる登録集原文を検索するディレクトリを指定します。複数のディレクトリを指定する場合，ディレクトリ同士をセミコロンで区切ります。この環境変数を省略すると，カレントディレクトリだけが検索されます。

PDCLTLANG：

特に文字コード種別を指定してプリプロセスする場合に指定します。省略した場合は，sjis が仮定されます。詳細については，「クライアント環境定義の設定内容」の「PDCLTLANG」を参照してください。

PDDEFAULTOPTION：

プリプロセスのオプションについて，省略時の動作を指定します。通常は，この環境変数を指定する必要はありません。互換モードを適用したい場合に，指定してください。詳細については，「クライアント環境定義の設定内容」の「PDDEFAULTOPTION」を参照してください。

＜例＞

```
[HiRDB]                          … 1
PDCBLFIX=.AAA                    … 2
PDCBLLIB=E:¥USER¥COPY            … 3
```

＜説明＞

1. [HiRDB]と記述します。
2. UAP ソースファイルの識別子として，.AAA も有効にします。
3. COPY 文で引き込む登録集原文を検索するディレクトリ（この例では E:¥USER¥COPY）を指定します。

(b)　SQL プリプロセサの起動

SQL プリプロセサの起動は，次に示す三つの方法があります。

- アイコンの重ね合わせによる実行
- ファイル名の指定による実行
- コマンドプロンプト又は MS-DOS プロンプトからの実行

- **アイコンの重ね合わせによる実行**
  エクスプローラでプリプロセスしたいファイルをドラッグして，プリプロセサのファイル（PDCBL.EXE（COBOL 言語の場合），又は PDOCB.EXE（OOCOBOL 言語の場合））に重ねると実行できます。
- **ファイル名の指定による実行**
  プリプロセサのアイコン（PDCBL.EXE（COBOL 言語の場合），又は PDOCB.EXE（OOCOBOL 言語の場合））をクリックし，次に示す手順で実行します。
  1. ファイルメニューから「ファイル名を指定して実行」を選択します。
  2. コマンドラインにファイル名，及びオプションを指定して実行します。
- **コマンドプロンプト又は MS-DOS プロンプトからの実行**
  コマンドプロンプト又は MS-DOS プロンプトを起動し，PDCBL.EXE（COBOL 言語の場合），又は PDOCB.EXE（OOCOBOL 言語の場合）コマンドを入力して実行します。

コマンドラインに入力するコマンドの形式を次に示します。

```
PDCBL.EXE 入力ファイル名称 ［オプション［出力ファイル名称｜認可識別子］］
```

注　OOCOBOL 言語の場合，下線で示す部分を PDOCB.EXE に置き換えてください。

入力ファイル名称：

UAP ソースファイルの名称を指定します。ファイル識別子は，.ECB，.COB，.CBL のどれか（COBOL 言語の場合），又は.EOC（OOCOBOL 言語の場合）を指定します。

出力ファイル名称：

ポストソースファイルの名称を指定します。出力ファイル名称を省略した場合，ファイル識別子は.CBL（COBOL 言語の場合），又は.OCB（OOCOBOL 言語の場合）を指定します。

認可識別子：

SQL で認可識別子を省略した場合に仮定する認可識別子を指定します。認可識別子を省略した場合，CONNECT 時のユーザ識別子が仮定されます。

オプション：

必要に応じて次の表に示すオプションを指定します。なお，オプションは大文字，小文字を区別しません。

表 8–10　プリプロセスオプション（Windows 環境の COBOL 言語の場合）

プリプロセスオプション	内容
/S	構文チェックだけをして，ポストソースを出力しない場合に指定します。このオプションを省略すると，ポストソースファイルが出力されます。なお，同時に/Xp も指定しないと，SQL プリプロセサは厳密な SQL の構文チェックができないため，SQL 文に文法上のエラーが発生していても発見できない場合があるので注意が必要です。
/O ファイル名	出力するポストソースのファイル名称を変更する場合に指定します。 このオプションを省略すると，入力ファイル名称のファイル識別子を.CBL（COBOL 言語の場合），又は.OCB（OOCOBOL 言語の場合）に変更されたものが出力ファイル名称になります。 入力ファイルの識別子が.CBL（COBOL 言語の場合），又は.OCB（OOCOBOL 言語の場合）の場合，必ずこのオプションを指定して，ポストソースのファイル名称を.CBL 以外（COBOL 言語の場合），又は.OCB 以外（OOCOBOL 言語の場合）の識別子に変更してください。
/XC	次の英数字定数を囲む分離符を，引用符（"）に変更する場合に指定します。 このオプションを省略すると，次の英数字定数を囲む分離符はアポストロフィ（'）になります。 • SQL プリプロセサが生成するソースコード中の英数字定数 • SQL プリプロセサが生成する COPY 文でポストソースに引き込む登録集原文ファイル中の英数字定数
/A 認可識別子	静的 SQL で認可識別子を省略した場合，及び仮定する認可識別子を変更する場合に指定します。 静的 SQL とは，INSERT 文，UPDATE 文，DELETE 文，1 行 SELECT 文，OPEN 文（形式 1），CALL 文，LOCK 文，及び PURGE TABLE 文を示します。 認可識別子を引用符（"）で囲んで指定することはできません。指定した認可識別子は常に大文字と小文字を区別して扱います。SQL 文中に引用符で囲まないで指定した認可識別子は小文字を大文字として扱います。そのため，ほかの UAP などに記述した SQL 文中に引用符で囲まないで小文字を使って指定した認可識別子と同じ認可識別子をこのオプションに指定する場合は，すべての英字を大文字で指定してください。 最大 30 バイトの認可識別子を指定できます。 認可識別子を省略した場合に仮定する認可識別子については，マニュアル「HiRDB SQL リファレンス」の「名前の修飾」を参照してください。
/h64	64 ビットモード用のポストソースを作成する場合に指定します。ただし，32 ビット版のプリプロセサを使用した場合は指定できません。

プリプロセスオプション	内容
/XD	DLL を作成する場合に指定します。 DLL を作成する場合，コンパイラは COBOL85 Version4.0 04-02 以降が前提となります。/XD オプションを指定してプリプロセスした UAP と，指定しないでプリプロセスした UAP を混在させてアプリケーションを作成しないでください。実行時に COBOL のランタイムライブラリでエラー（KCCBO204R-S）が発生します。
/Xe {y ｜ n}	PREPRARE 文実行時のカーソルを，自動的にクローズするかどうかを指定します。 y： カーソルを自動的にクローズするポストソースを生成します。 n： カーソルを自動的にクローズしないポストソースを生成します。 省略した場合，クライアント環境定義 PDPRPCRCLS の指定値に従いポストソースを生成します。
/XAD	X/Open に準じた API を使用した UAP を，DLL として作成する場合に指定してください。
/XA	X/Open に準じた API を使用して，UAP を作成する場合に指定してください。
/Xo	UAP から抽出した SQL 文を標準出力へ出力する場合に指定します。このとき，出力方法は次のようになります。 • SQL 文中の埋込み変数は?パラメタに置換します。 • 1 行 SELECT 文の INTO 句を削除します。 • SQL 文中の語句間の空白が 2 文字以上の場合，空白 1 文字に置き換えます。 • 複数行に分割して記述している SQL は 1 行にまとめます。 • 実行時にサーバに送られる SQL だけ出力します。実行されない SQL 文（WHENEVER 文，BEGIN DECLARE SECTION など）は出力しません。 • SQL の末尾にはセミコロン（;）を付けます。 • 埋込み変数の宣言は出力しません。 • 動的 SQL は，SQL がリテラルで指定されている場合にだけ出力します。そのほかの場合は出力しません。 • OPEN 文は，形式 1 のカーソルの場合だけ，問合せ式を出力します。 • ポストソースは生成しません。
/E2	埋込み変数を埋込み SQL 宣言節で宣言しないで使用する場合に指定します。
/Xp	厳密な SQL の構文チェックをする場合に指定してください。ただし，次の UAP をプリプロセスする場合は指定しないでください。 • XDM/RD E2 接続用の UAP • SQL 予約語削除機能を使用する UAP • 集合演算子 MINUS を使用する UAP これらの UAP をプリプロセスするときにこのオプションを指定すると，XDM/RD E2 又は HiRDB サーバで使用できる SQL が構文エラーとなることがあります。
/XU16[L ｜ B]	COBOL2002 の Unicode 機能を使用する場合に指定します。COBOL2002 の Unicode 機能を使用した UAP の実行については，「COBOL2002 の Unicode 機能を使用した UAP の実行」を参照してください。 日本語項目の文字コード（UTF-16）のバイトオーダを指定します。

プリプロセスオプション	内容
	/XU16L： UTF-16 のバイトオーダをリトルエンディアンにします。 /XU16B： UTF-16 のバイトオーダをビッグエンディアンにします。 /XU16： UTF-16 のバイトオーダを，プリプロセスを実行する OS のバイトオーダにします。 • Windows，Linux の場合はリトルエンディアンにします。 • AIX の場合はビッグエンディアンにします。 上記以外の OS でこのオプションを指定した場合は，プリプロセサの動作は保証できません。 COBOL2002 の Unicode 機能を使用しない場合は，このオプションを指定しないでください。指定すると，埋込み変数の宣言に日本語項目を使用できなくなります。
/Xm	複数接続機能を使用した UAP をプリプロセスする場合に指定します。詳細は，「COBOL 言語で複数接続機能を使用する場合の注意事項」を参照してください。 このオプションは，省略時にも仮定されます。ただし，0904 互換モードを適用した場合は仮定されませんので，オプションを指定することを検討してください。
/Xs	UAP に埋め込んだ SQL 文中に単純注釈（-- ～ 改行）を記述する場合に指定します。 単純注釈の記述規則については，「SQL の記述規則」の 13.を参照してください。 このオプションは，省略時にも仮定されます。ただし，0904 互換モードを適用した場合は仮定されませんので，オプションを指定することを検討してください。
/Xb	COBOL2002 のコンパイラオプション-BigEndian,Bin を使用して 2 進項目のバイトオーダをビッグエンディアンにする場合に指定します。詳細は，「COBOL のビッグエンディアンオプションへの対応」を参照してください。
/Xr	SQL 文の実行時に COBOL の RETURN-CODE 特殊レジスタを更新しないポストソースを生成します。 UAP が呼び出し元プログラムに返すリターンコードを RETURN-CODE 特殊レジスタに設定してから，呼び出し元プログラムに制御を戻すまでの間に SQL 文を実行する場合，このオプションを指定してください。 省略した場合，SQL 文の種類によって RETURN-CODE 特殊レジスタの更新有無が変わります。SQL 文の種類による RETURN-CODE 特殊レジスタの更新有無については，「SQL の記述規則」の 18.を参照してください。

1. COBOL 言語の場合のコマンド指定例

＜例 1＞

UAP ソースファイルの名称が SAMPLE で，ポストソースを出力しない場合

```
PDCBL SAMPLE.ECB /S
```

＜例 2＞

UAP ソースファイルの名称が SAMPLE で，出力するポストソースのファイル名称を MAIN にする場合

```
PDCBL SAMPLE.ECB /O MAIN.CBL
```

2. OOCOBOL 言語の場合のコマンド指定例

＜例 1＞

UAP ソースファイルの名称が SAMPLE で，ポストソースを出力しない場合

```
PDOCB.EXE SAMPLE.EOC /S
```

＜例 2＞

UAP ソースファイルの名称が SAMPLE で，出力するポストソースのファイル名称を MAIN にする場合

```
PDOCB.EXE SAMPLE.EOC /O MAIN.OCB
```

(c) SQL プリプロセサのリターンコード

SQL プリプロセサは，処理終了後にリターンコードを OS に返します。

リターンコードは，OS のバッチコマンド ERRORLEVEL で参照できます。

リターンコードとその内容を次の表に示します。

表 8–11 SQL プリプロセサのリターンコード（Windows 環境での COBOL 言語の場合）

リターンコード	内　　容
0	正常終了
4，8	エラー発生（プリプロセスを最後まで続行）
12，16	エラー発生（プリプロセスを途中で終了）

(d) エラーの出力

SQL プリプロセサは，SQL 文に文法上のエラーが発生した場合，その SQL 文を無視して処理を続行します。しかし，オプションの指定に誤りがある場合，処理を中断します。また，メモリ不足やファイル入出力エラーなど，システム上でエラーが発生し，それ以降の処理ができない場合，処理の途中でも終了します。

SQL 文に文法上のエラーが発生したとき，SQL プリプロセサはエラーメッセージを標準出力へ出力します。標準出力をリダイレクトすると，エラーメッセージをファイルに格納できます。ファイルを参照すると，エラー内容，UAP ソースファイル名，エラーが発生した箇所（SQL 文の行番号）などが分かります。

8.2.4 プリプロセサ宣言文の有効化

(1) 概要

プリプロセサでは，オプションで C コンパイラのプリプロセサ宣言文を使用できるようにしています。

/E オプション（UNIX 版の場合は-E オプション）を指定した場合，プリプロセサでは次の機能を実行できます。

- #define 宣言文を使用した定数及びマクロの定義
- #include 文で引き込んだインクルードファイル内の定数及びマクロの定義※
- #ifdef 文，#if などの条件付きコンパイル
- 埋込み変数宣言時のマクロでの定数指定

注※

SQL 文，及び SQL TYPE IS 型の変数宣言は，インクルードファイルの中には記述できません（プリプロセサは，ヘッダのポストソースは生成しないため，コンパイル時にエラーとなります）。詳細は，「SQL の記述規則」の 23.を参照してください。

(2) 使用例

(a) 定数の使用

UAP ソースファイル中で，次の埋込み変数宣言が記述されているとします。

```
#include  “user.h”
EXEC SQL BEGIN DECLARE SECTION;
  char  xchar1[MAX_CHAR_LEN];
EXEC SQL END DECLARE SECTION;
```

UAP がインクルードしているヘッダファイル（user.h）の中で，次のような定数が定義されているとします。

```
#define  MAX_CHAR_LEN  256
```

プリプロセサは，インクルードファイルを読み込み，MAX_CHAR_LEN の定義値を使用して，埋込み変数宣言を「char xchar1[256];」に変換してから解析します。ただし，マクロ定数は，SQL 先頭子と SQL 終了子の間（SQL 文中） には使用できません。

インクルードファイルを検索するためのディレクトリパスは，オプションの引数として指定します。C コンパイラのデフォルトのディレクトリは指定する必要はありません。

(b) 条件付きコンパイル

#ifdef などを使用して，プリプロセスする SQL 文を選択できます。例を次に示します。

```
#ifdef  DEF_SWITCH
    EXEC SQL DECLARE CUR1 CURSOR FOR SELECT * FROM TABLE1;
#else
    EXEC SQL DECLARE CUR1 CURSOR FOR SELECT * FROM TABLE2;
#endif
```

ただし，C コンパイラのプリプロセサ宣言文は，SQL 先頭子と SQL 終了子の間（SQL 文中）には記述できません。

8.2.5　埋込み SQL 宣言節の不要化

(1)　概要

プリプロセサでは，/E オプション（UNIX 版の場合は-E オプション）を指定すると，UAP ソースファイル中の任意の箇所で宣言されている SQL のデータ型に対応した変数を，埋込み変数として使用できます。ただし，register 記憶クラスの変数は，埋込み変数として使用できません。

変数の有効範囲は，UAP ソースファイルを記述しているホスト言語の規則に従います。この機能を使用できるのは，C 言語又は COBOL 言語で UAP ソースファイルを記述している場合だけです。

この機能を使用すると，次のことができます。

- 変数宣言を，埋込み SQL 開始宣言（BEGIN DECLARE SECTION）及び埋込み SQL 終了宣言（END DECLARE SECTION）で囲まなくても，埋込み変数として使用できます。また，併用もできます。
- 大域変数，局所変数，及び関数の引数の有効範囲を，ホスト言語の文法に従って判定します。埋込み変数の有効範囲が異なれば，同じ名称の変数を宣言しても，異なる埋込み変数として区別されます。この場合，その変数を使用している SQL 文を含む最も内側の変数が指定されたとみなされます。

(2)　使用例

使用例を次に示します。

```
int fetchdata(long  xtanka){
  char    xscode[5];
  char    xsname[17];
  char    xcol[3];
  long    xgryo;
      :
  EXEC SQL
      DECLARE CR3 CURSOR FOR
         SELECT SCODE,SNAME,COL,ZSURYO
         FROM ZAIKO WHERE TANKA=:xtanka;
      :
  EXEC SQL OPEN CR3 ;
      :
  /*  見出し */
  printf("    *****  在庫表 リスト  *****¥n¥n");
  printf("    商品コード  商品名           色  単価      現在庫量¥n");
  printf("    ----        ---------------  --  --------  --------¥n");

  EXEC SQL WHENEVER SQLERROR GOTO OWARI;
  EXEC SQL WHENEVER NOT FOUND GOTO OWARI;

  /* FETCH */
  for(;;){
      EXEC SQL
          FETCH CR3 INTO :xscode,:xsname,:xcol,:xgryo;
          printf("    %4s      %-16s  %2s  %8d  %8d¥n",
                  xscode, xsname, xcol, xtanka, xgryo);
    }
```

```
  }
OWARI:
```

8.2.6　ポインタでの埋込み変数指定

(1)　概要

C 言語では，/E オプション（UNIX 版の場合は-E オプション）でポインタを埋込み変数として宣言して使用できます。この機能を使用すると，動的に確保した領域を SQL 文で直接指定できます。

オプションについては，「UNIX 環境でのプリプロセス」又は「Windows 環境でのプリプロセス」のオプションの説明を参照してください。また，ポインタを使用できる SQL については，「プリプロセサの/E2，/E3 オプションを指定した場合のポインタ，構造体，及び構造体修飾の使用可否」を参照してください。

ポインタ変数は，C 言語の文法に従って宣言します。例を次に示します。

```
long  *xtanka;
long  *xgryo;
char  *xsname;
:
xtanka  =  (long *)malloc(sizeof(long));
xgryo  =  (long *)malloc(sizeof(long));
xsname  =  (char *)malloc(MAX_CHAR_LEN+1);
memset(xsname,  ‘  ‘, MAX_CHAR_LEN);
xsname [MAX_CHAR_LEN] = ’¥0’;
EXEC SQL FETCH CUR1 INTO :xtanka,:xgryo,:xsname;
```

(2)　規則

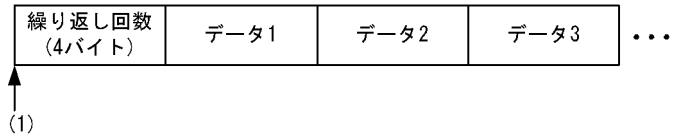
1. ポインタ変数は，SQL 文中では変数名の前にコロンを付けます。アスタリスクは付けません。
2. 参照される値のサイズは，宣言で指定した型のサイズになります。ただし，固定長文字列型（CHAR）を除きます。
3. 固定長文字列型のポインタのデータ長は，プリプロセス時ではなく，実行時に求められます。値のサイズは，ポインタが指している領域が格納している，文字列の終端（¥0）までの長さ（strlen（ポインタ変数））になります。1 行 SELECT 文や FETCH 文の検索結果を格納する場合も，SQL 文を実行する前に，あらかじめ領域全体を¥0 以外の文字でクリアして，末尾に¥0 を設定しておく必要があります。
4. ポインタが指す領域は，ユーザが確保しておく必要があります。固定長文字列型のポインタの場合は，¥0 を格納するために 1 バイト余分に領域を確保します。ポインタが不正な値の場合，及びデータを格納するのに十分な領域が確保できない場合は，動作が保証されません。
5. ポインタへのポインタは使用できません。
6. 構造体へのポインタは指定できます。

7. クラスへのポインタは使用できません。
8. 配列へのポインタは使用できません。配列へのポインタを使用する場合は，構造体を使用して次のように宣言します。

```
struct {
long  xtanka[50];
long  xgryo[50];
char  xsname[50][17];
} *xrec_ptr;
```

(3) RISC 型の CPU を使用しているマシンで繰り返し型のポインタを使用する場合の注意事項

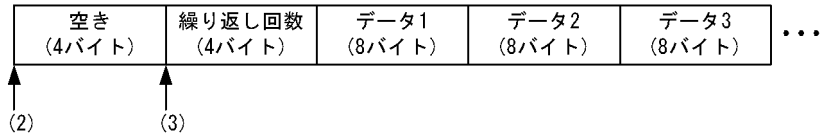
1. 繰り返し列型の変数は，次の構造となっているため，語境界に合わせたアドレス(1)をポインタに設定する必要があります。



   通常，malloc()などで確保した領域は，既に語境界に調整されているため，特に問題は発生しません。ただし，ユーザが独自にメモリアドレスを計算して割り当てる場合は，語境界に調整する必要があります。

   ポインタに設定するアドレスが語境界に設定されていないと，繰返し列の操作用のマクロを利用して，データを参照，設定したときにメモリアクセス例外が発生します。繰返し列の埋込み変数の構造については，「繰返し列の展開形式」を参照してください。

2. FLOAT 型の繰返し列の場合，繰り返し回数を格納する領域よりも繰返し要素の方がデータ長が大きくなります。そのため，繰返し要素の語長に合わせたアドレスに，境界調整をしておく必要があります。ポインタには，先頭の空き領域を含めたアドレス(2)を設定します。



   プリプロセサは，自動的に先頭から 4 バイト後ろのアドレス(3)を，繰返し列の先頭として使用するポストソースを生成します。FLOAT 型の繰返し列を操作するマクロも，アドレス(3)を先頭として使用しています。ただし，SQL 記述領域を使用して繰返し列のアドレスを直接設定する場合は，(3)のアドレスを設定してください。

3. 繰返し要素数の最大値は宣言時の値で決まるため，それより小さい領域を割り当てると，メモリアクセス例外が発生するおそれがあります。

   通常は，次のコーディングのようにメモリを確保すれば問題は発生しません。

```
PD_MV_SINT(32) *ptr;                /* 最大要素数32 */
ptr = malloc(sizeof(*ptr));
EXEC SQL FETCH CUR1 INTO :ptr;
```

8.2.7　構造体の参照

(1)　概要

オプションを指定すると，C 言語の構造体を使用して，複数個の埋込み変数を一度に指定できます。

構造体は，次の箇所に埋込み変数として使用できます。

- 1 行 SELECT 文又は FETCH 文の INTO 句
- INSERT 文の VALUES 句
- EXECUTE 文の USING 句又は INTO 句

オプションについては，「UNIX 環境でのプリプロセス」又は「Windows 環境でのプリプロセス」のオプションの説明を参照してください。また，構造体を使用できる SQL については，「プリプロセサの/E2，/E3 オプションを指定した場合のポインタ，構造体，及び構造体修飾の使用可否」を参照してください。

(2)　規則

1. 構造体を埋込み変数として指定すると，プリプロセサは構造体の各メンバが埋込み変数として記述されたとみなして，メンバが個別に記述された場合と同じポストソースを展開します。ポストソース中に展開されるメンバの順序は，構造体のメンバの宣言順序と同じです。構造体が記述された SQL 文の検索項目や列の順序と，メンバの順序は一致している必要があります。
2. 構造体のメンバを埋込み変数として個別に記述することもできます。
3. 共用体を含む構造体は使用できません。
4. 構造体を含む構造体は使用できません。ただし，可変長文字列型，及び BINARY 型に対応する構造体は使用できます。

(3)　使用例

- **構造体の使用例**

構造体の使用例を次に示します。

```
struct {
  char    xscode[5];
  char    xsname[17];
  char    xcol[3];
  long    xgryo;
  long    xtanka;
} xrec;
    :
EXEC SQL
  DECLARE CR3 CURSOR FOR
    SELECT SCODE,SNAME,COL,ZSURYO, TANKA FROM ZAIKO;
    :
EXEC SQL OPEN CR3 ;
```

```
  /*  見出し */
  printf("    *****  在庫表 リスト  *****¥n¥n");
  printf("    商品コード  商品名           色  単価      現在庫量¥n");
  printf("    ----        ---------------  --  --------  --------¥n");

  EXEC SQL WHENEVER SQLERROR GOTO OWARI;
  EXEC SQL WHENEVER NOT FOUND GOTO OWARI;

  /* FETCH */
  for(;;){
    EXEC SQL FETCH CR3 INTO :xrec;
    printf("    %4s      %-16s  %2s  %8d  %8d¥n",
        xrec.xscode, xrec.xsname, xrec.xcol, xrec.xtanka, xrec.xgryo);
    }
OWARI:
           :
```

- **標識変数を含む構造体の使用例**

埋込み変数として構造体を使用する場合に，標識変数も使用するときは，標識変数も構造体で宣言します。標識変数用の構造体の各メンバは，埋込み変数用の構造体の各メンバに，宣言順に対応させます。例を次に示します。

```
  struct {
      char    xscode[5];
      char    xsname[17];
      char    xcol[3];
      long    xgryo;
      long    xtanka;
  } xrec;
  struct {
      short    xscode_ind;
      short    xsname_ind;
      short    xcol_ind;
      short    xgryo_ind;
      short    xtanka_ind;
  } xrec_ind;
       :
  /* FETCH */
  for(;;){
    EXEC SQL FETCH CR3 INTO :xrec :xrec_ind;
      printf("    %4s      %-16s  %2s  %8d  %8d¥n",
         xrec.xscode, xrec.xsname, xrec.xcol, xrec.xtanka, xrec.xgryo);
  }
      :
```

- **構造体へのポインタで埋込み変数を指定した例**

構造体へのポインタで埋込み変数を指定することもできます。ポインタが指す領域は，あらかじめユーザが確保しておく必要があります。

```
struct  tag_xrec {
  char    xscode[5];
```

```
  char    xsname[17];
  char    xcol[3];
  long    xgryo;
  long    xtanka;
} *xrec_ptr;
struct tag_xrec_ind {
  short    xscode_ind;
  short    xsname_ind;
  short    xcol_ind;
  short    xgryo_ind;
  short    xtanka_ind;
} *xrec_ind_ptr;
      :
/* FETCH */
xrec_ptr = (struct tag_xrec *)malloc(sizeof(struct tag_xrec));
xrec_ind_ptr = (struct tag_xrec_ind *)
                malloc(sizeof(struct tag_xrec_ind));
  for(;;){
    EXEC SQL FETCH CR3 INTO :xrec_ptr :xrec_ind_ptr;
      printf("    %4s      %-16s  %2s  %8d  %8d¥n",
             xrec_ptr->xscode, xrec_ptr->xsname, xrec_ptr->xcol,
             xrec_ptr->xtanka, xrec_ptr->xgryo);
  }
       :
```

8.2.8 プリプロセサの/E2，/E3 オプションを指定した場合のポインタ，構造体，及び構造体修飾の使用可否

プリプロセサで/E2，/E3 オプション（UNIX 版の場合は-E2，-E3 オプション）を指定した場合の，ポインタ，構造体，及び構造体修飾の使用可否を次の表に示します。

なお，ポインタとは，(型名* 変数名)で宣言した変数を示します。構造体とは，(struct 構造体型名 変数名)で宣言した変数を示します（ただし，SQL 文を指定する構造体，並びに VARCHAR 型，及び BINARY 型に対応する構造体を除きます)。構造体修飾とは，(構造体.メンバ変数名)(構造体->メンバ変数名)の形式の変数を示します。

表 8–12　プリプロセサの/E2，/E3 オプションを指定した場合のポインタ，構造体，及び構造体修飾の使用可否

埋込み変数又は標識変数を指定する SQL		ポインタ	構造体	構造体修飾
操作系 SQL	CALL 文	○	×	○
	DECLARE CURSOR	○	×	○
	DELETE 文	○	×	○
	DESCRIBE TYPE 文	○	×	×
	EXECUTE 文 INTO 指定	○	○	○
	EXECUTE 文 USING 指定	○	○	○

埋込み変数又は標識変数を指定する SQL		ポインタ	構造体	構造体修飾
	EXECUTE 文 BY 指定	○	×	○
	EXECUTE IMMEDIATE 文の SQL 文字列の箇所	○	×	×
	EXECUTE IMMEDIATE 文 INTO 指定	○	○	○
	EXECUTE IMMEDIATE 文 USING 指定	○	○	○
	FETCH 文 INTO 指定	○	○	○
	FETCH 文 USING DESCRIPTOR 指定	○	×	○
	INSERT 文 VALUES 指定	○	○	○
	OPEN 文	○	×	○
	PREPARE 文	○	×	×
	SELECT 文 INTO 指定	○	○	○
	UPDATE 文	○	×	○
	FREE LOCATOR	○	○	○
	SET	○	×	○
	ALLOCATE CURSOR	○	×	×
制御系 SQL	CONNECT 文	○	×	○
	CONNECT 文 TO 指定	○	×	○
	SET SESSION AUTHORIZATION 文	○	×	○
埋込み言語	GET DIAGNOSTICS	×	×	×
	COMMAND EXECUTE	×	×	×
	INSTALL JAR	○	×	×
	REPLACE JAR	○	×	×
	REMOVE JAR	○	×	×
	ALLOCATE CONNECTION HANDLE	×	×	×
	FREE CONNECTION HANDLE	×	×	×
	DECLARE CONNECTION HANDLE SET	×	×	×
	GET CONNECTION HANDLE	×	×	×

（凡例）

○：指定できます。

×：指定できません。

8.2.9　COBOL 言語で複数接続機能を使用する場合の注意事項

COBOL 言語で複数接続機能を使用する場合，次の SQL 文の SQL 終了子（END-EXEC）の後ろに終止符（.）を付けても，ポストソース中に生成した COBOL の文の後ろに終止符が付かないことがあります。このとき，SQL は完結文になりません。

- DECLARE CONNECTION HANDLE SET
- ALLOCATE CONNECTION HANDLE
- FREE CONNECTION HANDLE
- GET CONNECTION HANDLE

この問題を回避するために，プリプロセスのオプションに/Xm オプション（UNIX 版の場合は-Xm オプション）を指定します。/Xm オプションを指定すると，プリプロセス後のポストソースから終止符が削除されなくなります。

/Xm オプション指定とポストソースの終止符の有無を次の表に示します。

SQL	UAP ソース中の終止符	/Xm オプション指定	ポストソース中の終止符
• DECLARE CONNECTION HANDLE SET • ALLOCATE CONNECTION HANDLE • FREE CONNECTION HANDLE • GET CONNECTION HANDLE	あり	あり	終止符を付けます。
		なし	終止符を付けません。
	なし	−	終止符を付けません。
• DECLARE CURSOR • BEGIN DECLARE SECTION • END DECLARE SECTION • WHENEVER	−	−	左記の SQL 文の位置には COBOL の文を生成しないため，終止符はありません。
COPY※1	−※2	あり	ポストソース中に生成する COBOL の文は，COPY で引き込んだ登録集原文の内容を UAP ソース中に直接記述した場合と同じ内容となります。
		なし	
上記以外の SQL	あり	−	終止符を付けます。
	なし	−	終止符を付けません。

（凡例）

−：終止符，又は/Xm オプション指定の有無はポストソースに影響しません。

注※1

COPY で引き込む登録集原文の中には COBOL の文と COPY 以外の SQL が記述されています。

注※2

COPY の SQL 終了子の後ろの終止符の有無はポストソースに影響しません。

(1)　/Xm オプション指定時の注意事項

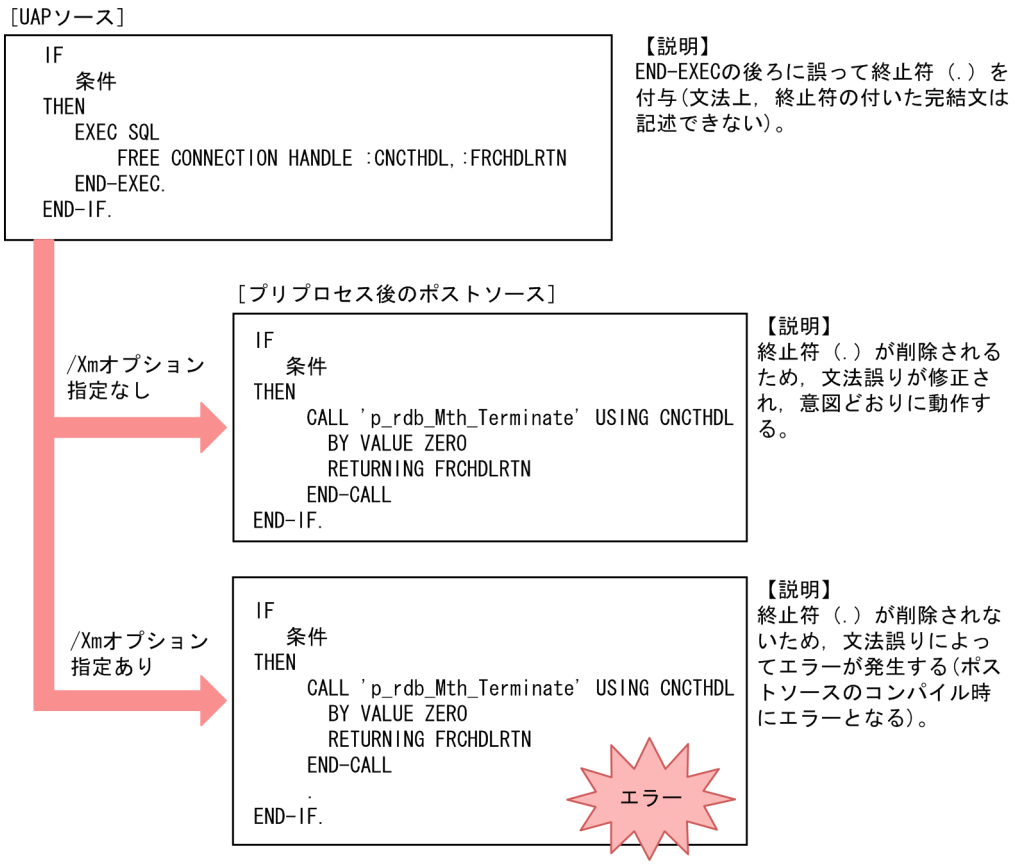
次に該当する場合，/Xm オプションを指定することで，それまで正常に動作していた UAP がエラーになることがあります。次のように SQL 文を記述しているとエラーになります。

- 文法上完結文を記述できない場所に終止符を付けた SQL 文を記述している場合
- 終止符の有無によってプログラムの動作が変わる場所に SQL 文を記述している場合

(a)　文法上完結文を記述できない場所に終止符を付けた SQL 文を記述している場合

例えば，END-IF，END-EVALUATE などの明示範囲終了子で終了させた範囲内には，文法上完結文を記述できません。この場合，SQL 文に誤って終止符を付けていても，/Xm オプションを指定しないでプリプロセスすると終止符が削除されるため，エラーにはなりません。しかし，/Xm オプションを指定すると，終止符が削除されないため，ポストソースのコンパイルでエラーとなります。

文法上完結文を記述できない場所に終止符を付けた SQL 文を記述している場合の例を次に示します。



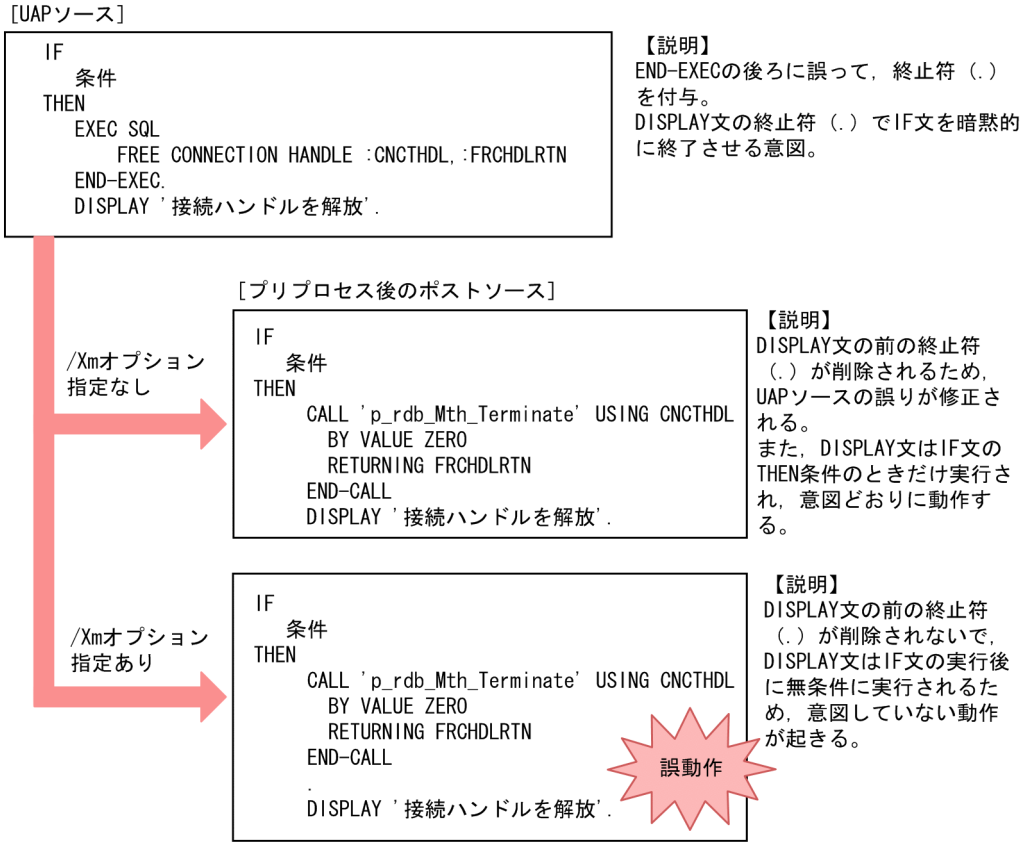
/Xm オプションを指定する場合は，文法上完結文を記述できない場所に終止符を付けた SQL 文を記述しないように UAP を修正してください。

(b)　終止符の有無によってプログラムの動作が変わる場所に SQL 文を記述している場合

例えば，END-IF，END-EVALUATE などの範囲終了子で終了させていない条件文でも，文末に終止符を付けることで暗黙的に終了します。そのため，終止符の有無によってプログラムの動作が変わることになりま

す。この場合，条件文の SQL 文に誤って終止符を付けていても，/Xm オプションを指定しないでプリプロセスすると終止符が削除され，意図していた動作をします。しかし，/Xm オプションを指定するとプログラムの内容が変わるため，意図していない動作をしてしまいます。

終止符の有無によってプログラムの動作が変わる場所に SQL 文を記述している場合の例を次に示します。



/Xm オプションを指定する場合は，条件文の SQL 文に終止符を付けないように UAP を修正してください。

8.2.10　バージョンによるプリプロセスオプションの変更点

SQL プリプロセサのオプションには，HiRDB のバージョンによってオプション省略時も適用するものがあります。オプション省略時も適用するオプションについて次の表に示します。

表 8–13　プリプロセスオプション省略時に適用するオプション値

HiRDB のバージョン	クライアント環境定義 PDDEFAULTOPTION	省略時も適用するプリプロセスオプション
09-50 より前	−	適用なし
09-50 以降	RECOM の場合	-Xs -Xm
	V0904 の場合	適用なし

（凡例）　−：該当しません。

なお，省略時に適用されるオプションによっては，生成するポストソースの内容が変わる場合があるため，バージョンアップ以前に動作実績のあるアプリケーションに対して再プリプロセスする必要がある場合は，PDDEFAULTOPTION に V0904 を指定してください。詳細については，「バージョン，リビジョンによるクライアント環境定義及びプリプロセスオプションの変更点」を参照してください。

8.2.11　COBOL のビッグエンディアンオプションへの対応

プリプロセスオプション-Xb（Windows 版の場合は/Xb）を指定してプリプロセスすることで，COBOL のビッグエンディアンオプションが利用できるようになります。

(1)　COBOL のビッグエンディアンオプション

COBOL のプログラムを AIX などのビッグエンディアンのプラットフォームから Linux などのリトルエンディアンのプラットフォームに移行する場合，2 進項目（2 進形式の数字項目）のバイトオーダが変わるために，ソースプログラムの変更が必要になることがあります。これを回避するために，COBOL2002 のコンパイラオプション-BigEndian,Bin があります。-BigEndian,Bin は，Linux 及び Windows 環境で，2 進項目のバイトオーダをビッグエンディアンにするオプションです。なお，HiRDB の埋込み型 UAP では COBOL2002 のコンパイラオプション-BigEndian,Float に対応していません。

COBOL2002 では，2 進項目のデータ記述で，USAGE 句に BINARY，COMP，COMP-4，COMP-5 のどれかを指定します。Linux 及び Windows 環境では，2 進項目のバイトオーダは USAGE 句の指定内容とコンパイラオプション-BigEndian,Bin の指定有無で決まります。詳細を次の表に示します。

表 8–14　Linux 及び Windows 環境の 2 進項目のバイトオーダ（COBOL2002 の仕様）

2 進項目のデータ記述の USAGE 句	コンパイラオプション-BigEndian,Bin の指定	2 進項目のバイトオーダ
BINARY，COMP 又は COMP-4	指定しない	リトルエンディアン
	指定する	ビッグエンディアン
COMP-5※	指定しない	リトルエンディアン
	指定する	

注※

COMP-5 を使用する場合は，コンパイラオプション-Comp5 を指定する必要があります。-Comp5 を指定しないで COMP-5 を使用するとコンパイルエラーになります。

(2)　クライアントライブラリ内で行うバイトオーダ変換

COBOL の埋込み型 UAP では，SQL 文で使用する 2 進項目のデータ記述の USAGE 句に，一部の例外を除いて※，COMP を指定することになっています。したがって，-BigEndian,Bin を指定してコンパイルすると，UAP の実行時に SQL 文で使用する 2 進項目のバイトオーダがビッグエンディアンになります。クライ

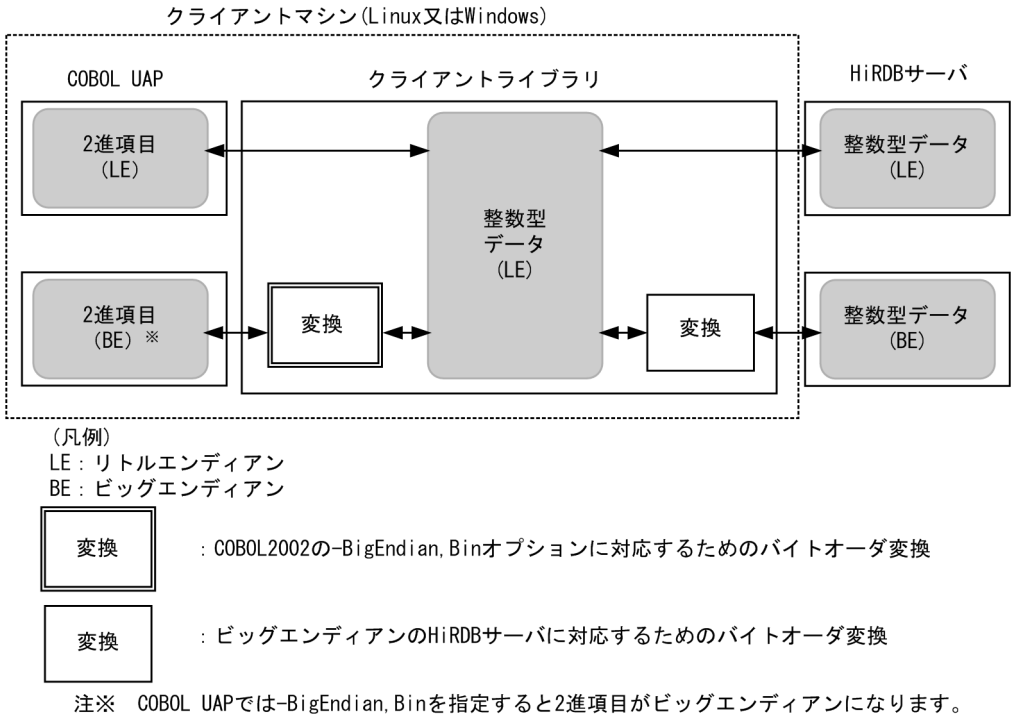
アントライブラリは SQL 文で使用する 2 進項目を C 言語の整数型データ（Linux 及び Windows 環境ではリトルエンディアン）として処理するので，クライアントライブラリ内では SQL 文で使用する 2 進項目をリトルエンディアンに変換して処理します。

注※

ROW 型変数に COMP-5 を使用できます。

クライアントライブラリ内で行うバイトオーダ変換を次の図に示します。

図 8–3　クライアントライブラリ内で行うバイトオーダ変換



バイトオーダ変換の対象となる「SQL 文で使用する 2 進項目」とは，次のデータ項目に含まれる 2 進項目です。

- 埋込み変数
- ？パラメタの値を格納するデータ項目
- 標識変数
- ？パラメタの標識を格納するデータ項目
- SQL 連絡領域
- SQL 記述領域
- 列名記述領域
- 型名記述領域
- 文字集合名記述領域

ただし，ROW 型変数※に格納する行データに含まれる 2 進項目については，前述のバイトオーダ変換を行いません。このため，Linux 及び Windows 環境で，ROW 型変数に含まれる 2 進項目のバイトオーダは，-BigEndian,Bin オプションの有無に関係なく，リトルエンディアンにする必要があります。したがって，ROW 型変数に含まれる 2 進項目はビッグエンディアンを前提にした処理で使用できません。

注※

ROW 型変数とは，SQL の行単位インタフェースで使用する ROW 型の埋込み変数，及び ROW 型の？パラメタの値を格納するデータ項目です。

(3) 埋込み型 UAP で COBOL のビッグエンディアンオプションを使用する方法

埋込み型 UAP のコンパイルで-BigEndian,Bin オプションを使用する場合は，プリプロセスオプション-Xb（Windows 環境では/Xb）を指定してプリプロセスします。プリプロセスオプション-Xb を指定すると，クライアントライブラリ内で，SQL 文で使用する 2 進項目のバイトオーダを変換します。コンパイル時には-BigEndian,Bin オプションと同時に-Comp5 オプションを指定します。Linux 及び Windows 環境で，SQL 文で使用する 2 進項目（ただし，ROW 型変数を除く）のバイトオーダと，データ記述，オプション指定の関係を次の表に示します。

表 8–15　SQL 文で使用する 2 進項目のバイトオーダと，データ記述，オプション指定の関係

項番	項目		各項目の指定内容	
			SQL 文で使用する 2 進項目（ROW 型変数を除く）のバイトオーダをリトルエンディアンにする場合	SQL 文で使用する 2 進項目（ROW 型変数を除く）のバイトオーダをビッグエンディアンにする場合
1	SQL 文で使用する 2 進項目（ROW 型変数を除く）のデータ記述の USAGE 句		COMP	COMP
2	SQL プリプロセサの-Xb オプション		指定しない	指定する
3	コンパイラオプション	-BigEndian,Bin	指定しない	指定する
4		-Comp5	任意	指定する※

注※

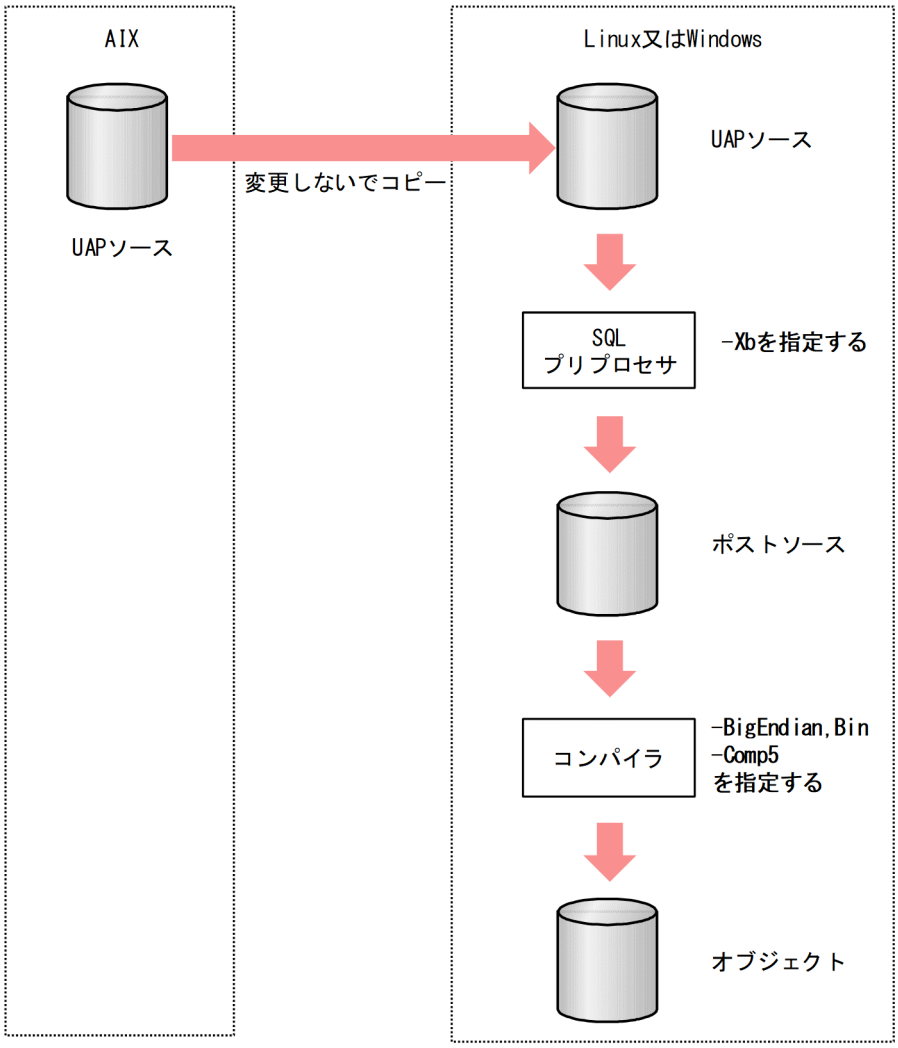
-Xb オプションを指定すると，SQL プリプロセサが展開するポストソース中で COMP-5 を使用するため，コンパイラオプションに-Comp5 を指定する必要があります。

プリプロセスオプション-Xb を使用する場合は，ROW 型変数に含まれる 2 進項目のデータ記述の USAGE 句を COMP-5 にする必要があります。なお，プリプロセスオプション-Xb を使用しない場合は，ROW 型変数に含まれる 2 進項目のデータ記述の USAGE 句に COMP と COMP-5 のどちらも指定できます。ROW 型変数に含まれる 2 進項目のデータ記述に COMP-5 を指定することによって，Linux 及び Windows 環境で，ROW 型変数に含まれる 2 進項目のバイトオーダは，-BigEndian,Bin オプションの有無に関係なく，リトルエンディアンになります。

埋込み型 UAP をビッグエンディアンのプラットフォームからリトルエンディアンのプラットフォームに移行する場合，コンパイラオプション-BigEndian,Bin とプリプロセスオプション-Xb を組み合わせて使用す

ることによって，ROW 型の変数を使用する場合を除き，2 進項目のバイトオーダの違いに対応するために UAP ソースを変更する必要がなくなります。COBOL の埋込み型 UAP をバイトオーダの異なるプラットフォームへ移行する手順の例を次の図に示します。

図 8–4　COBOL の埋込み型 UAP をバイトオーダの異なるプラットフォームへ移行する手順の例



(4)　注意事項

- コンパイラオプション-BigEndian,Bin とプリプロセスオプション-Xb は必ず組み合わせて使用してください。一方を指定して他方を指定しなかった場合，クライアントライブラリの動作を保証しません。
- コンパイラオプション-BigEndian,Bin とプリプロセスオプション-Xb を指定した場合，クライアントライブラリ内でバイトオーダ変換のためのオーバヘッドが発生します。このため，必要のないかぎりこのオプションは指定しないでください。コンパイラオプション-BigEndian,Bin が必要になる条件については COBOL2002 のマニュアルを参照してください。
- コンパイラオプション-BigEndian,Bin とプリプロセスオプション-Xb を指定し，かつインタフェース領域の内容の誤りによって SQL 文で使用する 2 進項目のバイトオーダ変換が失敗した場合は，クライアント環境定義の PDPRMTRC の指定に関係なく，SQL トレースにパラメタ情報を出力しません。

  インタフェース領域については，「インタフェース領域」を参照してください。

- コンパイラオプション-BigEndian,Bin とプリプロセスオプション-Xb を指定した場合，SQL 文で使用する入力用 2 進項目の内容を SQL 文の実行中だけ一時的にリトルエンディアンに変換します。このため，複数のプロセスやスレッドで同時に入力用 2 進項目のデータ領域を共用しないでください。共用すると，同時に実行するプログラムが，バイトオーダが逆転したデータを参照して，不正な動作をします。また，1 つの SQL 文中の複数個所に入力用 2 進項目を含む同じ埋込み変数を指定すると，SQL 文を正常に実行できないことがあるため，そのような指定をしないでください。

8.3　コンパイルとリンケージ

8.3.1　コンパイル，リンケージ時に指定するライブラリ

コンパイル，及びリンケージをするときには，HiRDB が提供するライブラリを指定します。コンパイル，及びリンケージをするときに指定するライブラリを次の表に示します。

- 表「コンパイル，及びリンケージをするときに指定するライブラリ（OLTP 下でない場合（UNIX 環境））」
- 表「コンパイル，及びリンケージをするときに指定するライブラリ（OLTP 下でない場合（Windows 環境））」
- 表「コンパイル，及びリンケージをするときに指定するライブラリ（OLTP 下の場合（UNIX 環境））」
- 表「コンパイル，及びリンケージをするときに指定するライブラリ（OLTP 下の場合（Windows 環境））」

表 8–16　コンパイル，及びリンケージをするときに指定するライブラリ（OLTP 下でない場合（UNIX 環境））

プラットフォーム	複数接続機能	スレッドモード	アドレッシングモード	ライブラリファイル名	
				共有ライブラリ	アーカイブライブラリ
AIX	使用する	シングルスレッド	32 ビット	libzclt6k.a libzclts.a※	libclts.a
		マルチスレッド（POSIX スレッド）	32 ビット	libzclt6k.a libzcltk.a※	libcltk.a
			64 ビット	libzclt6k64.a libzcltk64.a※	libcltk64.a
	使用しない	－	32 ビット	libzclt6k.a libzclt.a※	libclt.a
			64 ビット	libzclt6k64.a libzclt64.a※	libclt64.a
Linux	使用する	シングルスレッド	32 ビット	libzclts.so	libclts.a
		マルチスレッド（POSIX スレッド）	32 ビット	libzcltk.so	libcltk.a
	使用しない	－	32 ビット	libzclt.so	libclt.a
Linux(EM64T)	使用する	シングルスレッド	32 ビット	libzclts.so	－
		マルチスレッド（POSIX スレッド）	32 ビット	libzcltk.so	
			64 ビット	libzcltk64.so	
	使用しない	－	32 ビット	libzclt.so	

プラットフォーム	複数接続機能	スレッドモード	アドレッシングモード	ライブラリファイル名	
				共有ライブラリ	アーカイブライブラリ
			64 ビット	libzclt64.so	

（凡例）

－：該当しません。

注※

互換性のために提供しています。この共用ライブラリを使用すると，リンケージ時にシンボル重複の警告が発生するおそれがあるため，ほかの共用ライブラリの使用を検討してください。

表 8–17　コンパイル，及びリンケージをするときに指定するライブラリ（OLTP 下でない場合（Windows 環境））

プラットフォーム	複数接続機能	アドレッシングモード	共有ライブラリファイル名	推奨される UAP 開発環境※
Windows(x86)	使用する	32 ビット	PDCLTM32.LIB	Visual Studio 6.0 Visual Studio .NET 2002
			PDCLTM71.LIB	Visual Studio .NET 2003
			PDCLTM80S.LIB	Visual Studio 2005
			PDCLTM90.LIB	Visual Studio 2008
			PDCLTM100.LIB	Visual Studio 2010 Visual Studio 2012 Visual Studio 2013 Visual Studio 2015
			PDCLTM140.LIB	Visual Studio 2017
	使用しない	32 ビット	CLTDLL.LIB	Visual Studio 6.0 Visual Studio .NET 2002
			PDCLTM71.LIB	Visual Studio .NET 2003
			PDCLTM80S.LIB	Visual Studio 2005
			PDCLTM90.LIB	Visual Studio 2008
			PDCLTM100.LIB	Visual Studio 2010 Visual Studio 2012 Visual Studio 2013 Visual Studio 2015
			PDCLTM140.LIB	Visual Studio 2017
Windows(x64)	使用する	32 ビット	PDCLTM80S.LIB	Visual Studio 2005
			PDCLTM90.LIB	Visual Studio 2008
			PDCLTM100.LIB	Visual Studio 2010

プラットフォーム	複数接続機能	アドレッシングモード	共有ライブラリファイル名	推奨される UAP 開発環境※
				Visual Studio 2012 Visual Studio 2013 Visual Studio 2015
			PDCLTM140.LIB	Visual Studio 2017
		64 ビット	PDCLTM64.LIB	Visual Studio 2005
			PDCLTM90X.LIB	Visual Studio 2008
			PDCLTM100X.LIB	Visual Studio 2010 Visual Studio 2012 Visual Studio 2013 Visual Studio 2015
			PDCLTM140X.LIB	Visual Studio 2017
	使用しない	32 ビット	PDCLTM80S.LIB	Visual Studio 2005
			PDCLTM90.LIB	Visual Studio 2008
			PDCLTM100.LIB	Visual Studio 2010 Visual Studio 2012 Visual Studio 2013 Visual Studio 2015
			PDCLTM140.LIB	Visual Studio 2017
		64 ビット	PDCLTM64.LIB	Visual Studio 2005
			PDCLTM90X.LIB	Visual Studio 2008
			PDCLTM100X.LIB	Visual Studio 2010 Visual Studio 2012 Visual Studio 2013 Visual Studio 2015
			PDCLTM140X.LIB	Visual Studio 2017

注※

Windows 環境で動作する埋込み型 UAP の開発時は，HiRDB のライフサイクルだけでなく，Visual Studio のライフサイクルを意識する必要があります。「推奨される UAP 開発環境」列の Visual Studio のサポートが終了しているクライアントライブラリは，今後の HiRDB バージョンアップ時に，新機能をサポートしないことがあります。

このため，UAP を再作成する場合，HiRDB のバージョンアップ以外に，Visual Studio のバージョンアップ，及び指定するクライアントライブラリの変更についても検討する必要があります。

表 8–18　コンパイル，及びリンケージをするときに指定するライブラリ（OLTP 下の場合（UNIX 環境））

プラットフォーム	トランザクション登録方式	複数接続機能	スレッドモード	アドレッシングモード	ライブラリファイル名	
					共有ライブラリ	アーカイブライブラリ
AIX	動的登録	使用しない	シングルスレッド	32 ビット	libzclt6ys.a libzcltx.a※	libcltxa.a
		使用する	シングルスレッド	32 ビット	libzclt6ys.a libzcltxs.a※	libcltxas.a
			マルチスレッド（POSIX スレッド）	32 ビット	libzclt6yk.a libzcltxk.a※	libcltxak.a
	動的登録又は静的登録	使用しない	シングルスレッド	32 ビット	libzclt6ys.a libzclty.a※	libcltya.a
				64 ビット	libzclt6ys64.a libzclty64.a※	−
		使用する	シングルスレッド	32 ビット	libzclt6ys.a libzcltys.a※	libcltyas.a
				64 ビット	libzclt6ys64.a libzcltys64.a※	−
			マルチスレッド（POSIX スレッド）	32 ビット	libzclt6yk.a libzcltyk.a※	libcltyak.a
				64 ビット	libzclt6yk64.a	−
Linux	動的登録	使用しない	シングルスレッド	32 ビット	libzcltx.so	libcltxa.a
		使用する	シングルスレッド	32 ビット	libzcltxs.so	libcltxas.a
			マルチスレッド（POSIX スレッド）	32 ビット	libzcltxk.so	libcltxak.a
	動的登録又は静的登録	使用しない	シングルスレッド	32 ビット	libzclty.so	libcltya.a
		使用する	シングルスレッド	32 ビット	libzcltys.so	libcltyas.a
			マルチスレッド（POSIX スレッド）	32 ビット	libzcltyk.so	−
Linux(EM64T)	動的登録	使用しない	シングルスレッド	32 ビット	libzcltx.so	−
		使用する	シングルスレッド	32 ビット	libzcltxs.so	

プラットフォーム	トランザクション登録方式	複数接続機能	スレッドモード	アドレッシングモード	ライブラリファイル名	
					共有ライブラリ	アーカイブライブラリ
			マルチスレッド（POSIX スレッド）	32 ビット	libzcltxk.so	
	動的登録又は静的登録	使用しない	シングルスレッド	32 ビット	libzclty.so	
				64 ビット	libzclty64.so	
		使用する	シングルスレッド	32 ビット	libzcltys.so	
				64 ビット	libzcltys64.so	
			マルチスレッド（POSIX スレッド）	32 ビット	libzcltyk.so	
				64 ビット	libzcltyk64.so	

（凡例）

－：該当しません。

注※

互換性のために提供しています。この共用ライブラリを使用すると，リンケージ時にシンボル重複の警告が発生するおそれがあるため，ほかの共用ライブラリの使用を検討してください。

表 8–19　コンパイル，及びリンケージをするときに指定するライブラリ（OLTP 下の場合（Windows 環境））

プラットフォーム	トランザクション登録方式	複数接続機能	スレッドモード	アドレッシングモード	共有ライブラリファイル名	推奨される UAP 開発環境※
Windows(x86)	動的登録又は静的登録	使用しない	シングルスレッド	32 ビット	PDCLTX32.LIB	Visual Studio 6.0 Visual Studio .NET 2002 Visual Studio .NET 2003 Visual Studio 2005 Visual Studio 2008 Visual Studio 2010 Visual Studio 2012 Visual Studio 2013 Visual Studio 2015 Visual Studio 2017
		使用する	シングルスレッド	32 ビット	PDCLTXS.LIB	
			マルチスレッド	32 ビット	PDCLTXM.LIB	
Windows(x64)	動的登録又は静的登録	使用しない	シングルスレッド	32 ビット	PDCLTX32.LIB	Visual Studio 2005 Visual Studio 2008 Visual Studio 2010 Visual Studio 2012 Visual Studio 2013 Visual Studio 2015 Visual Studio 2017
		使用する	シングルスレッド	32 ビット	PDCLTXS.LIB（OTS 用）	
			マルチスレッド	32 ビット	PDCLTXM.LIB	

注

動的登録，及び静的登録については，マニュアル「HiRDB システム導入・設計ガイド」の HiRDB をトランザクションマネジャに登録する方法を参照してください。

注※

Windows 環境で動作する埋込み型 UAP の開発時は，HiRDB のライフサイクルだけでなく，Visual Studio のライフサイクルを意識する必要があります。「推奨される UAP 開発環境」列の Visual Studio のサポートが終了しているクライアントライブラリは，今後の HiRDB バージョンアップ時に，新機能をサポートしないことがあります。

このため，UAP を再作成する場合，HiRDB のバージョンアップ以外に，Visual Studio のバージョンアップ，及び指定するクライアントライブラリの変更についても検討する必要があります。

8.3.2　UNIX 環境でのコンパイルとリンケージ

SQL プリプロセサで生成したポストソースプログラムは，SQL を埋め込んだ UAP の言語に従ったコンパイラで，コンパイル，及びリンケージをします。

ここでは，コンパイル，及びリンケージを実行するときのコマンドの指定方法について，言語別に説明します。

(1)　C 言語の場合

C 言語のポストソースプログラムは，ANSI C に従ったコンパイラでコンパイルをし，C++言語のポストソースプログラムは，C++に従ったコンパイラでコンパイルをします。ANSI C に従ったコンパイラを起動するには，cc コマンドを使用し，C++に従ったコンパイラを起動するには，CC コマンドを使用します。cc コマンド，又は CC コマンドを実行すると，コンパイル，及びリンケージができます。

コンパイラを起動するコマンドの入力形式を次に示します。

```
cc ［オプション］ ファイル名称　ディレクトリ　提供ライブラリ
```

注　C++言語の場合，下線で示す部分を CC に置き換えてください。

ファイル名称：

ポストソースファイルの名称を指定します。

サフィックスは，.c（C 言語の場合），又は.C（C++言語の場合）にします。

ディレクトリ：

インクルードディレクトリ（HiRDB が提供するライブラリのヘッダファイルがあるディレクトリ）を指定します。

提供ライブラリ：

HiRDB が提供するライブラリを指定します。

HiRDB が提供するライブラリには，共用ライブラリとアーカイブライブラリがあります。通常時は，共用ライブラリを使用してください。使用するライブラリのバージョンを限定したい場合や，共用ライブラリが使用できないときだけ，アーカイブライブラリを使用してください。

UAP がスレッドを使用する場合，そのスレッドに対応した複数接続用ライブラリをリンクしてください。

オプション：

必要に応じて次に示すオプションを指定します。

-o：

出力する実行形式ファイルの名称を指定する場合に指定します。このオプションを省略すると，ファイル名称は a.out になります。

-I：

インクルードディレクトリを特定する場合に必ず指定します。このオプションを省略すると，コンパイルできません。

(a)　C 言語の場合のコマンド指定例

C 言語の場合の例を次に示します。なお，下線で示す部分は HiRDB のインストールディレクトリです。

●32 ビットモード対応の UAP の場合

＜例 1＞（共用ライブラリの場合）

- ポストソースファイルの名称が sample で，実行形式ファイルの名称を指定しない場合

```
cc -I/HiRDB/include  sample.c -L/HiRDB/client/lib -lzclt
```

＜例 2＞（アーカイブライブラリの場合）

- ポストソースファイルの名称が sample で，実行形式ファイルの名称を SAMPLE にする場合

```
aCC  +DD32 -I /HiRDB/include sample.c -L/HiRDB/client/lib -lzclt
```

＜例 3＞（Linux(EM64T)版の場合）

```
gcc -m32 -I /HiRDB/include sample.c -L/HiRDB/client/lib -lzclt
```

＜例 4＞（Linux(EM64T)版のマルチスレッドの場合）

```
gcc -m32 -D_REENTRANT -I /HiRDB/include sample.c -L/HiRDB/client/lib -lzcltk
```

●64 ビットモード対応の UAP の場合

＜例 1＞（共用ライブラリの場合）

- ポストソースファイルの名称が sample で，実行形式ファイルの名称を指定しない場合

AIX の場合

```
xlc -q64 -I/HiRDB/include  sample.c -Wl,-L/HiRDB/client/lib -lzclt64
```

Linux(EM64T)版の場合

```
gcc -I /HiRDB/include sample.c -L/HiRDB/client/lib -lzclt64
```

Linux(EM64T)版のマルチスレッドの場合

```
gcc -D_REENTRANT -I /HiRDB/include sample.c -L/HiRDB/client/lib -lzcltk64
```

注　シングルスレッドの UAP で複数接続機能を使用する場合も，libzcltk64.so を使用してください。

＜例 2＞（アーカイブライブラリの場合）

- ポストソースファイルの名称が sample で，実行形式ファイルの名称を指定しない場合

AIX の場合

```
xlc -q64 -I/HiRDB/include  sample.c -Wl,-L/HiRDB/client/lib -lclt64
```

(b)　C++言語の場合のコマンド指定例

C++言語の場合の例を次に示します。なお，下線で示す部分は HiRDB のインストールディレクトリです。

●32 ビットモード対応の UAP の場合

＜例 1＞（共用ライブラリの場合）

- ポストソースファイルの名称が sample で，実行形式ファイルの名称を指定しない場合

```
CC -I/HiRDB/include  sample.C -L/HiRDB/client/lib -lzclt
```

＜例 2＞（アーカイブライブラリの場合）

- ポストソースファイルの名称が sample で，実行形式ファイルの名称を SAMPLE にする場合

```
CC -o SAMPLE -I/HiRDB/include  sample.C /HiRDB/client/lib/libclt.a
```

●64 ビットモード対応の UAP の場合

＜例 1＞（共用ライブラリの場合）

- ポストソースファイルの名称が sample で，実行形式ファイルの名称を指定しない場合

AIX の場合

```
xlc -q64 -I/HiRDB/include  sample.C -Wl,-L/HiRDB/client/lib,-lzclt64
```

＜例 2＞（アーカイブライブラリの場合）

- ポストソースファイルの名称が sample で，実行形式ファイルの名称を指定しない場合

AIX の場合

```
xlc -q64 -I/HiRDB/include  sample.C -Wl,-L/HiRDB/client/lib,-lclt64
```

(2)　COBOL 言語の場合

COBOL 言語のポストソースプログラムは，COBOL85，又は COBOL2002 でコンパイルをし，OOCOBOL 言語のポストソースプログラムは，OOCOBOL に従ったコンパイラでコンパイルをします。

COBOL85 に従ったコンパイラを起動するには，ccbl コマンドを使用し，OOCOBOL に従ったコンパイラを起動するには，ocbl コマンドを使用します。ccbl コマンド，又は ocbl コマンドを実行すると，コンパイル，及びリンケージができます。

コンパイラを起動するときのコマンドの入力形式を次に示します。

```
ccbl　［オプション］　ファイル名称　ディレクトリ　提供ライブラリ
```

注　OOCOBOL 言語の場合，下線で示す部分を ocbl に置き換えてください。

ファイル名称：

ポストソースファイルの名称を指定します。

サフィックスは，.cbl（COBOL 言語の場合），又は.ocb（OOCOBOL 言語の場合）にします。

ディレクトリ：

インクルードディレクトリ（HiRDB が提供するライブラリのヘッダファイルがあるディレクトリ）を指定します。

提供ライブラリ：

HiRDB が提供する COBOL 言語，又は OOCOBOL 言語のライブラリを指定します。

オプション：

次に示すオプションを指定します。

また，注意が必要なオプションについては「注意事項」を参照してください。

-o：

出力する実行形式ファイルの名称を指定する場合に指定します。

このオプションを省略すると，ファイル名称は a.out になります。

オプションには，-Kl，又は-Xb オプションは指定しないでください。また，-Xc オプションと-Hf，-Hv，又は-V3 を同時に指定しないでください。

環境変数：

次に示す環境変数を指定します。

CBLLIB：

インクルードディレクトリを指定します。

(a)　COBOL 言語の場合のコマンド指定例

COBOL 言語の場合の例を次に示します。下線で示す部分は HiRDB のインストールディレクトリです。

●32 ビットモード対応の UAP の場合

＜例 1＞（共用ライブラリの場合）

- ポストソースファイルの名称が sample の場合

```
CBLLIB=/HiRDB/include
export CBLLIB
ccbl sample.cbl -L/HiRDB/client/lib -lzclt
```

＜例 2＞（アーカイブライブラリの場合）

- ポストソースファイルの名称が sample の場合

```
CBLLIB=/HiRDB/include
export CBLLIB
ccbl sample.cbl /HiRDB/client/lib/libclt.a
```

●64 ビットモード対応の UAP の場合

＜例 1＞（共用ライブラリの場合）

- ポストソースファイルの名称が sample の場合

```
CBLLIB=/HiRDB/include
export CBLLIB
ccbl2002 sample.cbl -L/HiRDB/client/lib -lzclt64
```

(b) OOCOBOL 言語の場合のコマンド指定例

OOCOBOL 言語の場合の例を次に示します。なお，下線で示す部分は HiRDB のインストールディレクトリです。

＜例 1＞（共用ライブラリの場合）

- ポストソースファイルの名称が sample の場合

```
CBLLIB=/HiRDB/include
export CBLLIB
ocbl sample.ocb -L/HiRDB/client/lib -lzclt
```

＜例 2＞（アーカイブライブラリの場合）

- ポストソースファイルの名称が sample の場合

```
CBLLIB=/HiRDB/include
export CBLLIB
ocbl sample.ocb /HiRDB/client/lib/libclt.a
```

(3) 注意事項

(a) COBOL2002 のコンパイラオプション-DynamicLink について

HiRDB クライアントライブラリを使用するアプリケーションのコンパイルでは，COBOL2002 のコンパイラオプション-DynamicLink の指定を次のどちらかにしてください。

- -DynamicLink オプションを指定しない
- -DynamicLink,IdentCall オプションを指定する（一意名指定の CALL 文だけを動的なリンクとする）

-DynamicLink,Call オプションを指定することもできますが，次の点に注意してください。

- 次のすべての条件が重なった場合，UAP が異常終了することがあります。
  (i) -MultiThread オプションを指定していない。
  (ii) マルチスレッドの HiRDB クライアントライブラリを指定している。
  (iii) 次のどちらかに該当し，定期的にパケットを送信する機能を使用している。
  - クライアント環境定義 PDDEFAULTOPTION に RECOM を指定又は PDDEFAULTOPTION を省略し，かつクライアント環境定義 PDKALVL の指定を省略している。
  - クライアント環境定義 PDKALVL に 0 以外を指定している。

  (iv) DISCONNECT 文を実行せずに，UAP を終了する。
  UAP の異常終了を回避するためには，上記の発生条件に該当しないように変更するか，又は COBOL2002 の環境変数 CBLNO_LIBFREE=EXIT を使用してください。
- AIX の場合，HiRDB クライアントライブラリは，アーカイブ化された共用ライブラリです。そのため，COBOL2002 の実行時環境変数 CBLLTAG に NOARMBR を指定しないでください。NOARMBR を指定すると，アーカイブ化された共用ライブラリは動的なリンク時の検索対象になりません。

COBOL2002 コンパイラオプション，実行時環境変数の詳細は，マニュアル「COBOL2002 使用の手引 手引編」を参照してください。

8.3.3 Windows 環境でのコンパイルとリンケージ

プリプロセサで生成したポストソースプログラムは，SQL を埋め込んだ UAP の言語に従ったコンパイラで，コンパイル，及びリンケージをします。

コンパイル，及びリンケージの方法については，各言語に従ったコンパイラのマニュアルを参照してください。ここでは，コンパイル，及びリンケージをするときのオプションについて，言語別に説明します。また，Windows(x64)版での指定を示します。

(1) C 言語の場合

C 言語のポストソースプログラムは，Microsoft Visual C++でコンパイルをします。

Microsoft Visual C++を使用してコンパイル，及びリンケージをするときにオプションを設定する場合，プロジェクトメニューから「設定」を選択します（Microsoft Visual C++のバージョンによっては，設定方法が異なります）。

「設定」で設定する項目を次の表に示します。

表 8–20 「設定」で設定する項目

項　目	カテゴリ	カテゴリの設定	設　定　値
コンパイラ	コード生成	構造体メンバのアライメント	8 バイト
		使用するランタイムライブラリ	マルチスレッド※
	プリプロセサ	インクルードファイルのパス	¥HiRDB¥include
リンカ	インプット	ライブラリ	¥HiRDB¥lib¥cltdll

注　下線で示す部分は，HiRDB のインストールディレクトリを指定してください。
注※　ライブラリが CLTDLL 以外の場合，マルチスレッドとなります。

64 ビットモードで UAP を作成する場合は，次の条件で作成してください。

- 構成メンバのアライメント：8 バイト
- 使用するランタイムライブラリ：マルチスレッド DLL
- インクルードファイルのパス：¥HiRDB¥INCLUDE
- リンケージライブラリ：¥HiRDB¥LIB¥PDCLTM64.LIB

(2)　COBOL 言語の場合

COBOL 言語のポストソースプログラムは，COBOL85 又は COBOL2002 に従ったコンパイラでコンパイルをし，OOCOBOL 言語のポストソースプログラムは，OOCOBOL に従ったコンパイラでコンパイルをします。

COBOL85（01-00 以降）を使用してコンパイル，及びリンケージをするときにオプションを設定する場合，編集メニューから「プロジェクト編集」を選択します。

Windows の場合は，オプションメニューから「コンパイラ」，及び「リンカ」メニューを選択します。

COBOL2002 の場合は，「プロジェクトの設定」メニューから「リンカ」タブを選択します。

COBOL85 の「プロジェクト編集」で設定する項目を次の表に示します。オプションには，-Kl，-Xb，-Bb，又は-Fb オプションは指定しないでください。また，-Xc オプションと-Hf，-Hv，又は-V3 を同時に指定しないでください。

表 8–21　COBOL85 の「プロジェクト編集」で設定する項目

項　　目	設定項目	設　定　値
リンケージオプション設定	インポートライブラリ	¥HIRDB¥lib¥cltdll.lib
	翻訳オプション	/NOI（ファイル識別子の大文字と小文字の区別）

注　下線で示す部分は，HiRDB のインストールディレクトリを指定してください。

COBOL2002 の「プロジェクトの設定」で設定する項目を次の表に示します。

表 8–22　COBOL2002 の「プロジェクトの設定」で設定する項目

項　　目	設定項目	設　定　値
リンク	ライブラリの指定	¥HIRDB¥lib¥cltdll.lib

注　下線で示す部分は，HiRDB のインストールディレクトリを指定してください。

COBOL85 の「翻訳環境」に対して設定するオプションがあります。COBOL85 の「翻訳環境」に対して設定する項目を次の表に示します。なお，COBOL2002 の場合は，環境変数に設定します。

表 8–23　COBOL85 の「翻訳環境」に対して設定する項目

項　　目	設定項目	設　定　値
環境変数設定※	CBLLIB 変数	¥HIRDB¥include

注　下線で示す部分は，HiRDB のインストールディレクトリを指定してください。

注※　COBOL2002 の場合，「環境変数」となります。

(3)　Windows(x64)版での指定

Windows(x64)版では，32 ビットモード用と 64 ビットモード用のクライアントライブラリをそれぞれ提供しています。32 ビットモードで UAP を作成する場合は，32 ビットモード用のコンパイルオプション及びライブラリを指定します。64 ビットモードで UAP を作成する場合は，64 ビットモード用のコンパイルオプション及びライブラリを指定します。

UAP の作成条件を次に示します。

UAP の作成	32 ビットモード	64 ビットモード
構造体メンバのアライメント	既定値（8 バイト）	既定値（8 バイト）
使用するランタイムライブラリ	マルチスレッド DLL	マルチスレッド DLL
インクルードファイルのディレクトリ	¥HiRDB¥INCLUDE	¥HiRDB¥INCLUDE
リンケージライブラリ※	¥HiRDB¥LIB¥PDCLTM80S.LIB	¥HiRDB¥LIB¥PDCLTM64.LIB

注

　下線で示す部分は，HiRDB のインストールディレクトリを指定してください。

注※

　複数接続機能を使用するかどうかに関係なく指定します。

8.3.4　複数接続機能を使用する場合のコンパイルとリンケージ

(1)　マルチスレッドの場合

OLTP 用でない通常の UAP で複数接続機能を使用する場合の，UAP のコンパイルとリンケージについて説明します。

(a)　UNIX 環境の場合

複数接続機能使用時にリンケージするライブラリについては，表「コンパイル，及びリンケージをするときに指定するライブラリ（OLTP 下でない場合（UNIX 環境））」及び表「コンパイル，及びリンケージをするときに指定するライブラリ（OLTP 下の場合（UNIX 環境））」を参照してください。なお，マルチスレッドを利用するためにリンケージしなければならないライブラリについては，各 OS のマニュアルを参照してください。

- **C 言語の例**

C 言語の場合の例を次に示します。なお，下線で示す部分は，HiRDB のインストールディレクトリです。

＜例 1＞ Linux で，UAP と共用ライブラリをリンクする場合

- ポストソースファイルの名称が sample で，実行形式ファイルの名称を指定しない場合

```
cc -I/HiRDB/include sample.c -D_REENTRANT -L/HiRDB/client/lib/ -lzcltk -lthread
```

＜例 2＞ AIX で，UAP と共用ライブラリをリンクする場合

- ポストソースファイルの名称が sample で，実行形式ファイルの名称を指定しない場合

```
xlc_r -I/HiRDB/include sample.c -L/HiRDB/client/lib/ -lzclt6k
```

＜例 3＞ AIX で，UAP と 64 ビットモード用の共用ライブラリをリンクする場合

- ポストソースファイルの名称が sample で，実行形式ファイルの名称を指定しない場合

```
xlc_r -I/HiRDB/include sample.c -q64 -L/HiRDB/client/lib/ -lzclt6k64
```

- **COBOL 言語の例**

COBOL 言語の場合，マルチスレッドに対応したバージョン（03-01 以降）の COBOL85 コンパイラが必要となります。

コンパイル時には，-Mt オプション（POSIX スレッドの場合は-Mp も必要）を指定します。-Mt オプションを指定してコンパイルしたオブジェクトと，指定しないでコンパイルしたオブジェクトをリンクすると，動作は保証されません。COBOL 言語の場合のコンパイルについては，マニュアル「COBOL85 使用の手引」を参照してください。

COBOL 言語の場合の例を次に示します。なお，下線で示す部分は，HiRDB のインストールディレクトリです。

＜例 1＞ Linux で，POSIX スレッド用の UAP と共用ライブラリをリンクする場合

- ポストソースファイルの名称が sample で，実行形式ファイルの名称を指定しない場合

```
setenv CBLLIB /HiRDB/include
ccbl -Mt -Mp sample.cbl -L/HiRDB/client/lib/ -lcltk -lpthread
```

＜例 2＞ AIX で，POSIX スレッド用の UAP と共用ライブラリをリンクする場合

- ポストソースファイルの名称が sample で，実行形式ファイルの名称を指定しない場合

```
setenv CBLLIB /HiRDB/include
ccbl -Mt -Mp sample.cbl -L/HiRDB/client/lib/ -lzclt6k -lpthread
```

(b) Windows 環境の場合

CLTDLL.LIB の代わりに，PDCLTM32.LIB をリンケージします。なお，OLTP 下で X/Open に従った API を使用した UAP の場合，PDCLTXM.LIB をリンケージします。

- **C 言語の場合**

ここでは，Microsoft Visual C++ Version 4.2 を前提として説明します。プロジェクトメニューから「設定」を選択して，各項目を設定します。「設定」で設定する項目を次の表に示します。マルチスレッドを利用するときに，リンケージする必要があるライブラリについては，各 OS のマニュアルを参照してください。

表 8–24 「設定」で設定する項目

項　目	カテゴリ	カテゴリの設定	設　定　値
コンパイラ	コード生成	構造体メンバのアライメント	8 バイト
		使用するランタイムライブラリ	マルチスレッド DLL
	プリプロセサ	インクルードファイルのパス	¥HIRDB¥INCLUDE
リンカ	インプット	ライブラリ	¥HIRDB¥LIB¥PDCLTM32.LIB

注　下線で示す部分は，HiRDB のインストールディレクトリを指定してください。

- **COBOL 言語の場合**

COBOL 言語の場合，マルチスレッドに対応したバージョンの COBOL85 コンパイラが必要となります。ここでは，COBOL85 Version 5.0 を前提に説明します。

コンパイル時には，コンパイラオプションダイアログボックスで-Mt オプションを指定します。-Mt オプションを指定してコンパイルしたオブジェクトと，指定しないでコンパイルしたオブジェクトをリンクすると，動作は保証されません。COBOL 言語の場合のコンパイルについては，マニュアル「COBOL85 操作ガイド」を参照してください。

「オプション」メニューで設定する項目を次の表に示します。

表 8–25 「オプション」メニューで設定する項目

サブメニュー	ダイアログ	設定項目	設定値
コンパイラ	COBOL85 コンパイラオプション	COBOL85 コンパイラオプション	-Mt の項目をチェック
		環境変数設定	CBLLIB=C:¥HIRDB¥INCLUDE
リンカ	リンカオプションの設定	インポート／ユーザ指定ライブラリ	C:¥HIRDB¥LIB¥PDCLTM32

注　下線で示す部分は，HiRDB のインストールディレクトリを指定してください。

(2)　シングルスレッドの場合

シングルスレッドの UAP で複数接続機能を使用する場合の，コンパイルとリンケージの方法について説明します。なお，ここでは HP-UX を例に説明します。

(a)　Linux 版でのコンパイルとリンケージ

- **C 言語の例**

C 言語の場合の例を次に示します。なお，下線で示す部分は，HiRDB のインストールディレクトリです。

＜例 1＞（共用ライブラリの場合）

- ポストソースファイルの名称が sample で，実行形式ファイルの名称を指定しない場合

```
gcc -I /HiRDB/include sample.c -L/HiRDB/client/lib -lzclts
```

8.4　注意事項

8.4.1　UAP 実行時の注意事項

UAP を実行する場合の注意事項を次に示します。

(1)　文字コード種別についての注意事項

UNIX 版の HiRDB クライアントを使って UAP を実行する場合は，HiRDB サーバとの接続時に，HiRDB サーバの文字コード種別（サーバが UNIX 版の場合は pdsetup コマンド，Windows 版の場合は pdntenv コマンドで指定した文字コード種別）をチェックできます。

HiRDB サーバの文字コード種別をチェックする場合は，HiRDB サーバの文字コード種別に合わせて，環境変数 LANG，又はクライアント環境定義 PDLANG を設定してください。LANG 又は PDLANG で指定した文字コード種別と HiRDB サーバの文字コード種別が異なると，サーバ接続時にエラーとなります。このとき出力するエラーメッセージ（KFPA11990-E）に HiRDB サーバの文字コード種別が表示されます。

HiRDB サーバの文字コード種別をチェックする場合の，LANG 又は PDLANG の設定値を次の表に示します。この表で PDLANG を設定するケースでは，PDLANG を省略して LANG で HiRDB サーバの文字コード種別を指定しても有効になりません。

表 8–26　HiRDB サーバの文字コード種別をチェックする場合の環境変数の設定値

HiRDB サーバの文字コード種別※	HiRDB クライアントのプラットフォームの環境変数			
	AIX		Linux	
	LANG	PDLANG	LANG	PDLANG
lang-c	C	–	C	–
sjis	Ja_JP	–	任意	SJIS
ujis	ja_JP	–	ja_JP.eucJP 又は ja_JP	–
chinese	任意	CHINESE	任意	CHINESE
utf-8 utf-8_ivs	任意	UTF-8	任意	UTF-8
chinese-gb18030	任意	CHINESE-GB18030	任意	CHINESE-GB18030

（凡例）

–：

PDLANG には値を設定しないでください。LANG にはこの表に記載されている値（HiRDB サーバの文字コード種別に対応するロケール名）を設定してください。各プラットフォームで，PDLANG にこの表に記載されていない値を設定した場合は，PDLANG の設定を無視します。例えば，この表の

AIX の列には PDLANG の値として SJIS が記載されてないため，AIX で PDLANG に SJIS を設定しても，PDLANG を省略しても，結果は同じになります。

任意：

PDLANG にこの表に記載されている値を設定すると，HiRDB クライアントは LANG を参照しないため，LANG にはプラットフォームがサポートしている任意のロケール名を設定できます。

注※

pdsetup コマンド又は pdntenv コマンドで指定した HiRDB サーバの文字コード種別を表します。

(2) SHLIB_PATH についての注意事項

UAP 実行時には，SHLIB_PATH に$PDDIR/client/lib を追加してください。なお，SHLIB_PATH は，それぞれのプラットフォームの環境変数に読み替えてください。

(3) HiRDB に回復不要 FES がある場合の注意事項

HiRDB に回復不要 FES がある場合，X/Open XA インタフェースを使用する UAP から回復不要 FES に接続したときは，その UAP からは SQL が実行できません。この場合，クライアント環境定義 PDFESHOST 及び PDSERVICEGRP を指定して，回復不要 FES でないフロントエンドサーバに接続してください。

(4) Windows での注意事項

Windows では，HiRDB クライアントが取得した通信用ソケットに対して他プログラムが既に使用している受信ポートが割り当てられることがあります。同一の受信ポートが割り当てられると，接続時に HiRDB サーバからの電文が他プログラムに送信され，HiRDB クライアントは電文を受信することができなくなり，KFPA11732-E エラー（受信タイムアウト）となります。このようなことを防ぐため，UAP を実行する場合は，受信ポートが重複しないように対策する必要があります。

受信ポートが重複する場合の条件と対策方法を次の表に示します。

表 8–27　受信ポートが重複する場合の条件と対策方法（1/2）

後発のクライアント		先発のクライアント					
		バージョン 08-03 より前のライブラリ		バージョン 08-03 以降のライブラリ			
				PDCLTBINDLOOPBACKADDR=YES		PDCLTBINDLOOPBACKADDR=NO	
		PDCLTRCVPORT 指定あり	PDCLTRCVPORT 指定なし	PDCLTRCVPORT 指定あり	PDCLTRCVPORT 指定なし	PDCLTRCVPORT 指定あり	PDCLTRCVPORT 指定なし
バージョン 08-03 より前のライブラリ	PDCLTRCVPORT 指定あり	−	−	−	−	−	−

後発のクライアント			先発のクライアント					
			バージョン 08-03 より前のライブラリ		バージョン 08-03 以降のライブラリ			
					PDCLTBINDLOOPBACKADDR=YES		PDCLTBINDLOOPBACKADDR=NO	
			PDCLTRCVPORT 指定あり	PDCLTRCVPORT 指定なし	PDCLTRCVPORT 指定あり	PDCLTRCVPORT 指定なし	PDCLTRCVPORT 指定あり	PDCLTRCVPORT 指定なし
		PDCLTRCVPORT 指定なし	−	−	−	−	−	×1
バージョン 08-03 以降のライブラリ	PDCLTBINDLOOPBACKADDR=YES	PDCLTRCVPORT 指定あり	−	−	−	−	−	−
		PDCLTRCVPORT 指定なし	−	−	−	−	−	−
	PDCLTBINDLOOPBACKADDR=NO	PDCLTRCVPORT 指定あり	−	−	−	−	−	−
		PDCLTRCVPORT 指定なし	−	×1	−	−	−	−
Type4 JDBC ドライバ	PDCLTBINDLOOPBACKADDR=YES	PDCLTRCVPORT 指定あり	−	−	−	−	−	−
		PDCLTRCVPORT 指定なし	−	−	−	−	−	×1
	PDCLTBINDLOOPBACKADDR=NO	PDCLTRCVPORT 指定あり	−	−	−	−	−	−
		PDCLTRCVPORT 指定なし	−	×1	−	×1	−	×1

表 8–28　受信ポートが重複する場合の条件と対策方法（2/2）

後発のクライアント			先発のクライアント，HiRDB サーバ，又はその他のプログラム				
			Type4 JDBC ドライバ				HiRDB サーバ又はその他プログラム
			PDCLTBINDLOOPBACKADDR=YES		PDCLTBINDLOOPBACKADDR=NO		
			PDCLTRCVPORT 指定あり	PDCLTRCVPORT 指定なし	PDCLTRCVPORT 指定あり	PDCLTRCVPORT 指定なし	
バージョン 08-03 より前のライブラリ		PDCLTRCVPORT 指定あり	−	−	−	−	−
		PDCLTRCVPORT 指定なし	−	−	−	×1	×2
バージョン 08-03 以降のライブラリ	PDCLTBINDLOOPBACKADDR=YES	PDCLTRCVPORT 指定あり	−	−	−	−	−
		PDCLTRCVPORT 指定なし	−	−	−	×1	×2
	PDCLTBINDLOOPBACKADDR=NO	PDCLTRCVPORT 指定あり	−	−	−	−	−
		PDCLTRCVPORT 指定なし	−	×1	−	×1	×2
Type4 JDBC ドライバ	PDCLTBINDLOOPBACKADDR=YES	PDCLTRCVPORT 指定あり	−	−	−	−	−
		PDCLTRCVPORT 指定なし	−	−	−	×1	×2
	PDCLTBINDLOOPBACKADDR=NO	PDCLTRCVPORT 指定あり	−	−	−	−	×2
		PDCLTRCVPORT 指定なし	−	×1	−	×1	×2

（凡例）

−：

受信ポートは重複しないため，対策の必要はありません。

×1：

受信ポートが重複します。先発のクライアント，及び後発のクライアントのクライアント環境変数 PDCLTRCVPORT でポート番号の範囲を指定し，それぞれのプログラムが使用するポート番号が重複しないようにしてください。

×2：

受信ポートが重複します。後発のクライアントのクライアント環境変数 PDCLTRCVPORT で，HiRDB サーバ又はその他のプログラムが使用しないポート番号を指定し，それぞれのプログラムが使用するポート番号が重複しないようにしてください。

8.4.2　X/Open に従った API（TX_関数）を使用した UAP の実行

X/Open に従った API（TX_関数）を使用した UAP は，専用のライブラリを使用します。TX_関数を使用した UAP をコンパイル，及びリンケージする場合，TX_関数専用のライブラリと HiRDB が提供するライブラリとを結合させる必要があります。

(1)　X/Open に従った API（TX_関数）を使用した UAP のプリプロセス

TP1/LiNK（トランザクション制御）と連携している HiRDB で UAP を実行する場合の注意事項について説明します。

TP1/LiNK と連携できるのは，HiRDB サーバ，HiRDB クライアントが，Windows 同士の場合です。

(a)　UAP のプリプロセス，リンケージ

TP1/LiNK 環境下で実行する UAP をプリプロセス，リンケージする場合，次のようにしてください。

- **プリプロセス**

  SQL プリプロセサ実行時に，次のオプションを指定してください。

  - /XAD：COBOL 言語で UAP を DLL として作成する場合に指定してください。
  - /XA：上記以外の場合はすべてこちらを指定してください。

  C 言語の場合のコマンド指定例：

```
PDCPP SAMPLE /XA
```

  COBOL 言語の場合のコマンド指定例：

```
PDCBL SAMPLE.ECB /XAD
```

- **リンケージ**

  UAP をリンケージする場合に，次のライブラリをリンクしてください。

  - %PDDIR%¥CLIENT¥LIB¥PDCLTX32.LIB

  CLTDLL.LIB はリンクしないでください。

(2) OpenTP1 を使用する場合

OpenTP1 を使用した場合の，UAP のコンパイル及びリンケージについて説明します。

なお，OpenTP1 でのコンパイル，及びリンケージの詳細については，マニュアル「OpenTP1 プログラム作成リファレンス C 言語編」，又はマニュアル「OpenTP1 プログラム作成リファレンス COBOL 言語編」を参照してください。

(a) C 言語の場合

- **トランザクションオブジェクトファイルの作成**

OpenTP1 で HiRDB をアクセスする UAP を作成するとき，OpenTP1 運用コマンドでトランザクション制御用オブジェクトファイルを作成する必要があります。

トランザクション制御用オブジェクトファイルは，trnmkobj コマンドを使用します。トランザクション制御用オブジェクトファイルを作成する場合，次のように指定します。

```
trnmkobj  -o  制御用オブジェクトファイル名称  -r  HiRDB_DB_SERVER
```

＜例＞

- トランザクション制御用オブジェクトファイル名称を seigyo とする場合

```
    trnmkobj -o seigyo -r HiRDB_DB_SERVER
```

- **コンパイル，及びリンケージ**

API を使用した UAP をコンパイル，及びリンケージする場合，次のように指定します。

共用ライブラリを使用する場合：

```
/usr/bin/cc  -c  -I$DCDIR/include  -I/HiRDB/include  ファイル名称.c

  /usr/bin/cc  -o  UAP実行形式ファイル名称  UAPファイル名称.o
  $DCDIR/spool/trnrmcmd/userobj/制御用オブジェクトファイル名称.o
  -L/HiRDB/client/lib  -lzclty  -L$DCDIR/lib  -Wl, -B, immediate
  -Wl, -a, default  -lbetran  -L/usr/lib  -ltactk  -lbsd  -lc
```

注 1

下線で示す部分は HiRDB のインストールディレクトリです。

注 2

HiRDB が X/Open に従った API を使用した UAP のために提供するライブラリは，OLTP へ登録する方法によって次に示す 4 種類があります。リンケージで指定するライブラリ名は，OLTP へ登録する方法に合わせた名称を指定してください。

- -lzcltx（動的登録）
- -lzclty（静的登録/動的登録）

- -lzcltxs（動的登録で，複数接続機能を使用する場合）
- -lzcltys（静的登録/動的登録で，複数接続機能を使用する場合）

静的登録/動的登録は，TM に登録するスイッチ名称で静的登録又は動的登録のどちらかに切り替えられます。なお，登録方法については，マニュアル「HiRDB システム導入・設計ガイド」の HiRDB をトランザクションマネジャに登録する方法を参照してください。

注 3

AIX で，X/Open に従った API を使用した UAP を作成する場合，リンケージオプションに-brtl を指定する必要があります。

＜例＞

ファイル名称（UAP 名）を sample，UAP 実行形式ファイル名称を SAMPLE，トランザクション制御用オブジェクトファイル名称を seigyo とする場合

```
/usr/bin/cc -c -I$DCDIR/include -I/HiRDB/include sample.c

  /usr/bin/cc -o SAMPLE sample.o $DCDIR/spool/trnrmcmd/userobj/
  seigyo.o -L/HiRDB/client/lib -lzclty -L$DCDIR/lib
  -Wl,-B,immediate -Wl,-a,default
  -lbetran -L/usr/lib -ltactk -lbsd -lc
```

アーカイブライブラリを使用する場合：

```
/usr/bin/cc  -c  -I$DCDIR/include  -I/HiRDB/include  ファイル名称.c

  /usr/bin/cc  -o  UAP実行形式ファイル名称  UAPファイル名称.o
  $DCDIR/spool/trnrmcmd/userobj/制御用オブジェクトファイル名称.o
  -L/HiRDB/client/lib  -lcltxa  -L$DCDIR/lib  -Wl, -B, immediate
  -Wl, -a, default  -lbetran  -L/usr/lib  -ltactk  -lbsd  -lc
```

注 1

下線で示す部分は，HiRDB のインストールディレクトリを指定してください。

注 2

HiRDB が提供するアーカイブライブラリ（-lcltxa）は，OpenTP1 が提供するライブラリ（-lbetran）より前に指定してください。

注 3

HiRDB が X/Open に従った API を使用した UAP のために提供するライブラリは，OLTP へ登録する方法によって次に示す 2 種類があります。リンケージで指定するライブラリ名は，OLTP へ登録する方法に合わせた名称を指定してください。

- -lcltxa（動的登録）
- -lcltya（静的登録/動的登録）

静的登録/動的登録は，TM に登録するスイッチ名称で静的登録又は動的登録のどちらかに切り替えられます。なお，登録方法については，マニュアル「HiRDB システム導入・設計ガイド」の HiRDB をトランザクションマネジャに登録する方法を参照してください。

＜例＞

ファイル名称（UAP 名）を sample，UAP 実行形式ファイル名称を SAMPLE，トランザクション制御用オブジェクトファイル名称を seigyo とする場合

```
/usr/bin/cc -c -I$DCDIR/include -I/HiRDB/include sample.c

  /usr/bin/cc -o SAMPLE sample.o $DCDIR/spool/trnrmcmd/userobj/
  seigyo.o -L/HiRDB/client/lib -lcltxa -L$DCDIR/lib
  -Wl,-B,immediate -Wl,-a,default
  -lbetran -L/usr/lib -ltactk -lbsd -lc
```

(b) COBOL 言語の場合

- **トランザクションオブジェクトファイルの作成**

OpenTP1 で HiRDB をアクセスする UAP を作成するとき，OpenTP1 運用コマンドでトランザクション制御用オブジェクトファイルを作成する必要があります。

トランザクション制御用オブジェクトファイルは，trnmkobj コマンドを使用します。トランザクション制御用オブジェクトファイルを作成する場合，次のように指定します。

```
  trnmkobj  -o  制御用オブジェクトファイル名称  -r  HiRDB_DB_SERVER
```

＜例＞

- トランザクション制御用オブジェクトファイル名称を seigyo とする場合

```
  trnmkobj -o seigyo -r HiRDB_DB_SERVER
```

- **コンパイル，及びリンケージ**

API を使用した UAP をコンパイル，及びリンケージする場合，次のように指定します。

共用ライブラリの場合：

```
ccbl  -o  UAP実行形式ファイル名称  -Mw  ファイル名称.cbl
  $DCDIR/spool/trnrmcmd/userobj/制御用オブジェクトファイル名称.o
  -L/HiRDB/client/lib  -lzclty  -L$DCDIR/lib  -Wl, -B, immediate
  -Wl, -a, default  -lbetran  -L/usr/lib  -ltactk  -lbsd  -lc
```

注 1

下線で示す部分は，HiRDB のインストールディレクトリを指定してください。

注 2

HiRDB が X/Open に従った API を使用した UAP のために提供するライブラリは，OLTP へ登録する方法によって次に示す 4 種類があります。リンケージで指定するライブラリ名は，OLTP へ登録する方法に合わせた名称を指定してください。

- -lzcltx（動的登録）
- -lzclty（静的登録/動的登録）

- -lzcltxs（動的登録で，複数接続機能を使用する場合）
- -lzcltys（静的登録/動的登録で，複数接続機能を使用する場合）

静的登録/動的登録は，TM に登録するスイッチ名称で静的登録又は動的登録のどちらかに切り替えられます。なお，登録方法については，マニュアル「HiRDB システム導入・設計ガイド」の HiRDB をトランザクションマネジャに登録する方法を参照してください。

注 3

AIX 版で，X/Open に従った API を使用した UAP を作成する場合，リンケージオプションに-brtl を指定する必要があります。

＜例＞

ファイル名称（UAP 名）を sample，UAP 実行形式ファイル名称を SAMPLE，トランザクション制御用オブジェクトファイル名称を seigyo とする場合

```
ccbl -o SAMPLE -Mw sample.cbl
  $DCDIR/spool/trnrmcmd/userobj/seigyo.o
  -L/HiRDB/client/lib -lzclty -L$DCDIR/lib -Wl,-B,immediate
  -Wl,-a,default -lbetran -L/usr/lib -ltactk -lbsd -lc
```

アーカイブライブラリの場合：

```
ccbl  -o  UAP実行形式ファイル名称  -Mw  ファイル名称.cbl
  $DCDIR/spool/trnrmcmd/userobj/制御用オブジェクトファイル名称.o
  -L/HiRDB/client/lib  -lcltxa  -L$DCDIR/lib  -Wl, -B, immediate
  -Wl, -a, default  -lbetran  -L/usr/lib  -ltactk  -lbsd  -lc
```

注 1

下線で示す部分は，HiRDB のインストールディレクトリを指定してください。

注 2

HiRDB が提供するアーカイブライブラリ（-lcltxa）は，OpenTP1 が提供するライブラリ（-lbetran）より前に指定してください。

注 3

HiRDB が X/Open に従った API を使用した UAP のために提供するライブラリは，OLTP へ登録する方法によって次に示す 2 種類があります。リンケージで指定するライブラリ名は，OLTP へ登録する方法に合わせた名称を指定してください。

- -lcltxa（動的登録）
- -lcltya（静的登録/動的登録）

静的登録/動的登録は，TM に登録するスイッチ名称で静的登録又は動的登録のどちらかに切り替えられます。なお，登録方法については，マニュアル「HiRDB システム導入・設計ガイド」の HiRDB をトランザクションマネジャに登録する方法を参照してください。

＜例＞

ファイル名称（UAP 名）を sample，UAP 実行形式ファイル名称を SAMPLE，トランザクション制御用オブジェクトファイル名称を seigyo とする場合

```
ccbl -o SAMPLE -Mw sample.cbl
  $DCDIR/spool/trnrmcmd/userobj/seigyo.o
  -L/HiRDB/client/lib -lcltxa -L$DCDIR/lib -Wl,-B,immediate
  -Wl,-a,default -lbetran -L/usr/lib -ltactk -lbsd -lc
```

(3) TPBroker for C++を使用する場合

TPBroker for C++を使用した場合の，UAP のコンパイル及びリンケージについて説明します。なお，UAP はマルチスレッド対応の XA インタフェースを使用しているものとします。

なお，TPBroker for C++でのコンパイル及びリンケージ方法については，マニュアル「TPBroker ユーザーズガイド」を参照してください。

また，マルチスレッド対応の XA インタフェース専用ライブラリは，C 言語及び C++言語にだけ対応しています。

(a) トランザクションオブジェクトファイルの作成

TPBroker for C++で HiRDB をアクセスする UAP を作成するとき，TPBroker for C++のコマンド（tsmkobj コマンド）でトランザクション制御用オブジェクトファイルを作成します。指定例を次に示します。

```
tsmkobj -o 制御用オブジェクトファイル名 -r HiRDB_DB_SERVER
```

(b) コンパイル及びリンケージ

UAP をコンパイル及びリンケージする場合の指定例を次に示します。

```
aCC +inst_implicit_include +DAportable -c -I$TPDIR/include
    -I$TPDIR/include/dispatch -I/HiRDB/include -D_REENTRANT
    -D_HP_UX_SOURCE -D_POSIX_C_SOURCE=199506Lファイル名.c
aCC +inst_implicit_include +Daportable -o UAP実行形式ファイル名 UAPファイル名.o
    $TPDIR/otsspool/XA/制御用オブジェクトファイル名称.o
    -L/HiRDB/client/lib -lzcltxk -L$TPDIR/lib -Wl,+s -lots_r -lorb_r
    -Wl,-B,immediate -Wl,-a,default -L/usr/lib -lpthread
```

注 1

下線で示す部分は，HiRDB クライアントのインストールディレクトリを指定してください。

注 2

マルチスレッド対応の XA インタフェースを使用した UAP のため，HiRDB が提供するライブラリは，OLTP へ登録する方法（動的登録又は静的登録）によって，二種類（-lzcltxk 又は-lzcltyk）あります。リンケージで指定するライブラリ名は，それに合わせた名称を指定してください。

(4) TUXEDO を使用する場合

TUXEDO を使用した場合の，UAP のコンパイル及びリンケージについて説明します。なお，XA インタフェース専用のライブラリは，C 言語又は C++言語にだけ対応しています。

(a) UNIX 版の場合

- **トランザクションマネジャサーバ（TMS）のロードモジュールの構築**

```
buildtms -r HiRDB_DB_SERVER -o TMSロードモジュールファイル名
```

- **TUXEDO システムのサーバのロードモジュールの構築**

```
buildserver -r HiRDB_DB_SERVER -s サービス名 -o サーバロードモジュールファイル名
          -f サーバファイル名.o
```

- **TUXEDO クライアントモジュールの作成**

```
buildclient -o クライアントロードモジュール名 -f クライアントファイル名.c
```

(b) Windows 版の場合

- **トランザクションマネジャサーバ（TMS）のロードモジュールの構築**

```
set LINK=/EXPORT:_imp_pdtxa_switch=pdtxa_switch
           /EXPORT:_inp_pdtxa_switch_y=pdtxa_switch_y
buildtms -r HiRDB_DB_SERVER -o TMSロードモジュールファイル名
```

- **TUXEDO システムのサーバのロードモジュールの構築**

```
set LINK=/EXPORT:_imp_pdtxa_switch=pdtxa_switch
           /EXPORT:_inp_pdtxa_switch_y=pdtxa_switch_y
buildserver -r HiRDB_DB_SERVER -s サービス名 -o サーバロードモジュールファイル名
          -f サーバファイル名.obj
```

- **TUXEDO クライアントモジュールの作成**

```
buildclient -o クライアントロードモジュール名 -f クライアントファイル名.c
```

(5) TP1/EE を使用する場合（UNIX 版限定）

TP1/EE を使用した場合の，UAP のコンパイル及びリンケージについて説明します。なお，TP1/EE のコマンドについては，マニュアル「TP1/Server Base Enterprise Option 使用の手引」を参照してください。

(a) C言語の場合

- **リソースマネジャ連携用オブジェクトファイルの作成**

  TP1/EE で HiRDB をアクセスする UAP を作成する場合，TP1/EE 運用コマンドでリソースマネジャ連携用オブジェクトファイルを作成する必要があります。リソースマネジャ連携用オブジェクトファイルは，eetrnmkobj コマンドで作成します。

  リソースマネジャ連携用オブジェクトファイルを作成する場合，次のように指定します。

```
eetrnmkobj -o リソースマネジャ連携用オブジェクトファイル名称 -r HiRDB_DB_SERVER  ¥
          -s RMスイッチ名称 -O RM関連オブジェクトファイル名称  ¥
          -i HiRDB提供ヘッダパス
```

＜例＞

リソースマネジャ連携用オブジェクトファイル名称を seigyo として，静的登録方式で作成する場合

```
eetrnmkobj -o seigyo -r HiRDB_DB_SERVER -s pdtxa_switch_y  ¥
           -O /HiRDB/client/lib/libzcltyk.sl -i /HiRDB/include
```

- **コンパイル，及びリンケージ**

マルチスレッド対応 XA インタフェースを使用した UAP をコンパイル，及びリンケージする場合，次のように指定します。

- **共用ライブラリを使用する場合**

```
/usr/vac/bin/xlc_r -o 実行形式ファイル名 $DCDIR/lib/ee_main.o
    リソースマネジャ連携用オブジェクト -brtl -bdynamic -L/HiRDB/lib -L/HiRDB/client/li
b
    -L$DCDIR/lib -lpthread -lisode -lc_r -ldl -lzcltyk -lee -lee_rm
    -lbetran2 -ltactk
```

注 1

下線で示す部分は，HiRDB クライアントのインストールディレクトリを指定してください。

注 2

マルチスレッド対応 XA インタフェースを使用した，TP1/EE の UAP のためのライブラリとして，-lzcltyk を使用します。リンケージで指定するライブラリ名は，それに合わせた名称を指定してください。なお，登録方法については，マニュアル「HiRDB システム導入・設計ガイド」を参照してください。

＜例＞

UAP 実行形式ファイル名称を SAMPLE，リソースマネジャ連携用オブジェクトファイル名称を seigyo とする場合

```
/usr/vac/bin/xlc_r -o SAMPLE $DCDIR/lib/ee_main.o seigyo.o
     -brtl -bdynamic -L/HiRDB/lib -L/HiRDB/client/lib -L$DCDIR/lib
     -lpthread -lisode -lc_r -ldl -lzcltyk -lee -lee_rm -lbetran2 -ltactk
```

(b) COBOL 言語の場合

- **リソースマネジャ連携用オブジェクトファイルの作成**

TP1/EE で HiRDB をアクセスする UAP を作成する場合，TP1/EE 運用コマンドでリソースマネジャ連携用オブジェクトファイルを作成する必要があります。リソースマネジャ連携用オブジェクトファイルは，eetrnmkobj コマンドで作成します。

リソースマネジャ連携用オブジェクトファイルを作成する場合，次のように指定します。

```
eetrnmkobj -o リソースマネジャ連携用オブジェクトファイル名称 -r HiRDB_DB_SERVER  ¥
           -s RMスイッチ名称 -O RM関連オブジェクトファイル名称  ¥
           -i HiRDB提供ヘッダパス
```

＜例＞

リソースマネジャ連携用オブジェクトファイル名称を seigyo として，静的登録方式で作成する場合

```
eetrnmkobj -o seigyo -r HiRDB_DB_SERVER -s pdtxa_switch_y  ¥
           -O /HiRDB/client/lib/libzcltyk.sl -i /HiRDB/include
```

- **コンパイル，及びリンケージ**
  マルチスレッド対応 XA インタフェースを使用した UAP をコンパイル，及びリンケージについては，マニュアル「TP1/Server Base Enterprise Option 使用の手引」を参照してください。

8.4.3　COBOL2002 の Unicode 機能を使用した UAP の実行

COBOL2002 の Unicode 機能を使用する UAP では，UTF-16 の文字データを格納する日本語項目を，埋込み変数として使用できます。

(1)　SQL に関係する文字データの文字コード

COBOL2002 の Unicode 機能を使用する UAP での，UAP 内の文字データと文字コードを次の表に示します。

表 8–29　COBOL2002 の Unicode 機能を使用する UAP での，UAP 内の文字データと文字コード

UAP 内の文字データ		文字コード
SQL 文		UTF-8
埋込み変数	英数字項目	UTF-8
	日本語項目	UTF-16LE 又は UTF-16BE

(2)　HiRDB サーバが行う文字コード変換

COBOL2002 の Unicode 機能を使用する UAP で SQL を実行する場合に，HiRDB サーバが行う文字コード変換を次の表に示します。

表 8–30　COBOL2002 の Unicode 機能を使用する UAP で SQL を実行する場合に，HiRDB サーバが行う文字コード変換

UAP 内の文字データ		SQL 実行時に HiRDB サーバが行う文字コード変換
SQL 文		クライアント側とサーバ側の文字コードは両方とも UTF-8 のため，変換しません。
埋込み変数	英数字項目	代入，比較処理の対象となるサーバ側の文字データの文字集合が UTF-16 の場合，文字コードを変換します。※
	日本語項目	代入，比較処理の対象となるサーバ側の文字データの文字集合が既定文字集合(UTF-8)の場合，文字コードを変換します。※ なお，変換前のデータに半角文字がある場合は，半角文字（UTF-16 では 2 バイト）を格納します。 また，埋込み変数内のデータは-XU16 オプションの指定に従って，文字集合名 UTF-16LE，又は UTF-16BE を指定した文字データ型（CHAR 又は VARCHAR）のデータとして扱います。

注※

変換（代入，比較）できるデータ型については，マニュアル「HiRDB SQL リファレンス」を参照してください。

(3)　SQL を実行できる条件

COBOL2002 の Unicode 機能を使用する UAP で SQL を実行するには，次に示す条件をすべて満たす必要があります。

- HiRDB サーバの既定文字集合が UTF-8，又は IVS 対応 UTF-8 である
- SQL プリプロセサ実行時に-XU16 オプションを指定する
- 埋込み SQL 文中に JIS X0213 の第 3・4 水準漢字コードの文字を含まない
- クライアント環境定義 PDCLTCNVMODE に NOUSE（省略値）を指定する

(a)　HiRDB サーバの既定文字集合

pdntenv コマンド（UNIX 版の場合は pdsetup コマンド）で，文字コード種別に utf-8，又は IVS 対応 UTF-8 を指定した場合に，HiRDB サーバの既定文字集合が UTF-8 になります。

(b)　プリプロセスオプション

-XU16 オプションについては，表「プリプロセスオプション（UNIX 環境の COBOL 言語の場合）」，又は表「プリプロセスオプション（Windows 環境の COBOL 言語の場合）」を参照してください。

(c)　埋込み SQL 文中の文字の制限

UAP のソース中に埋め込んだ SQL 文に JIS X0213 の第 3・4 水準漢字コードの文字が含まれていると，プリプロセス時にエラーになります。なお，埋込み変数に格納する文字データは，COBOL2002 の Unicode 機能がサポートする範囲の文字をすべて使用できます。

(d)　クライアント環境定義

HiRDB クライアントが文字コード変換を行わないように，クライアント環境定義 PDCLTCNVMODE に NOUSE を指定します。

8.4.4　XDM/RD と UNIFY2000 で作成した UAP の移行性

XDM/RD，又は UNIFY2000 で作成した UAP は，SQL の互換性があるものについては SQL プリプロセサを実行することで HiRDB で使用できます。しかし，互換性がない SQL については書き替えが必要です。HiRDB で使用する SQL については，「HiRDB で使用できる SQL 一覧」を参照してください。また，SQL の詳細については，マニュアル「HiRDB SQL リファレンス」を参照してください。

XDM/RD，又は UNIFY2000 からの UAP の移行性を次の表に示します。

表 8–31　XDM/RD，又は UNIFY2000 からの UAP の移行性

クライアント UAP 種別		移行性	移行の条件
作成したシステム	API		
XDM/RD	埋込み型	○	SQL プリプロセサの再実行
	モジュール型	×	−
	CALL 型	×	−
UNIFY2000	埋込み型 SQL/A	○	SQL プリプロセサの再実行
	RHLI	×	−

（凡例）

○：移行できます。

×：移行できません。

−：該当しません。

8.4.5　HiRDB をバージョンアップした場合に必要な作業

HiRDB が提供している API は上位互換性があります。そのため，基本的には HiRDB をバージョンアップしても UAP の修正は必要ありません。

ただし，HiRDB の新機能を使用する場合は，再度プリプロセス，コンパイル，リンケージする必要があります。

8.4.6　バージョン，リビジョンによるクライアント環境定義及びプリプロセスオプションの変更点

クライアント環境定義及びプリプロセスオプションの見直しを行い，省略値の変更及び指定が必要なクライアント環境定義の削減を実施しました。これによって，より安全なシステムを構築することができます。ここでは，次の区分で変更となるクライアント環境定義及びプリプロセスオプションの一覧を示します。

1. バージョン，リビジョンによって省略値が異なるクライアント環境定義及びプリプロセスオプション
2. 指定不要となったクライアント環境定義

バージョンアップ後は推奨モードの省略値を適用します。バージョンアップを行う場合は，省略値変更によるメリット及びデメリットを確認してください。確認の結果，旧バージョンとの互換性を重視する場合は，旧バージョンと同等の省略値になる互換モードを適用してください。ただし，この場合は推奨値を指定できるクライアント環境定義があれば，個別に指定することを検討してください。

(1) バージョン，リビジョンによって省略値が異なるクライアント環境定義及びプリプロセスオプション

バージョン，リビジョンによって省略値が異なるクライアント環境定義を次の表に示します。省略値変更によるメリット及びデメリットを確認してください。確認の結果，旧バージョンとの互換性を重視する場合は，旧バージョンと同等の省略値になる互換モードを適用してください。互換モードはクライアント環境定義 PDDEFAULTOPTION で指定できます。PDDEFAULTOPTION の詳細は，「クライアント環境定義の設定内容」の「PDDEFAULTOPTION」を参照してください。

表 8–32　バージョン，リビジョンによって省略値が異なるクライアント環境定義

クライアント環境定義名	バージョン 09-50 以降の省略値	0904 互換モードを適用している場合の省略値	バージョン 09-50 より前の省略値	省略値変更によるメリット	省略値変更によるデメリット
PDSTJTRNOUT	YES	→	通常環境：NO OLTP 環境：YES	コネクションプーリング機能を使用している場合など接続が物理的に切断されない場合でも，トランザクション単位で統計情報を取得するため，トラブルシュート情報が拡充します。	通常環境の場合，UAP に関する統計情報のログ出力量が増加します。サーバ側の統計情報ログファイル容量を確認してください。UAP に関する統計情報の出力量については，マニュアル「HiRDB システム定義」の「統計ログファイル（pd_stj_file_size）の見積もり式」を参照してください。
PDKALVL	2	→	0	ルータやファイアウォールなどのネットワーク管理アプリケーションが備えている無通信時間監視機能によって，HiRDB の接続を不当に切断されることを防ぎます。また，サーバで時間監視している PDSWAITTIME，及び PDSWATCHTIME の監視時間を	1 接続当たり 1 スレッド（KeepAlive 通信用スレッド）増加します。スレッド数に関する OS パラメタやプロセスサイズ増加による影響がないか確認してください。 詳細は，「クライアント環境定義の設定内容」の「PDKALVL」を参照してください。
PDKATIME	600	→	3000		

クライアント環境定義名	バージョン 09-50 以降の省略値	0904 互換モードを適用している場合の省略値	バージョン 09-50 より前の省略値	省略値変更によるメリット	省略値変更によるデメリット
				リセットするため，サーバでの時間監視による切断を防げます。 詳細は，「クライアント環境定義の設定内容」の「PDKALVL」を参照してください。	
PDBLKBUFFSIZE（Type4 JDBC ドライバ使用時以外）	PDBLKF 指定なし：10 PDBLKF 指定あり：0	→	0	サーバからクライアントに検索結果を転送する場合の通信オーバヘッドが減り，検索時間を短縮できます。	クライアント及びサーバでブロック転送用のメモリが増加します。クライアント側のメモリ所要量については，「クライアントライブラリのメモリ容量見積もり」，サーバ側のメモリ所要量については，マニュアル「HiRDB システム導入・設計ガイド」の「SQL 実行時に必要なメモリ所要量の計算式」を参照し，メモリ増加による影響がないか確認してください。 また，「ブロック転送機能」の「注意事項」を参照し，ブロック転送機能の適用によって問題が発生しないか確認してください。
PDFORUPDATEEXLOCK	YES	→	NO	更新を前提とした検索に対する排他モードのロックを自動設定するので，信頼性が向上します。	排他モードのロックを設定しないことを意図していた UAP がないか見直しが必要です。

クライアント環境定義名	バージョン 09-50 以降の省略値	0904 互換モードを適用している場合の省略値	バージョン 09-50 より前の省略値	省略値変更によるメリット	省略値変更によるデメリット
PDCLTAPNAME（Type4 JDBC ドライバ使用時）	**Cosminexus 使用時** JDBC4.3 版：JDBC43_XXXX※ JDBC4.0 版：JDBC40_XXXX JDBC2.0 版：JDBC20_XXXX XXXX：J2EE サーバ名	**Cosminexus 使用時** JDBC4.0 版：JDBC40_XXXX JDBC2.0 版：JDBC20_XXXX XXXX：J2EE サーバ名	JDBC4.0 版：HiRDB_Type4_JDBC40_Driver JDBC2.0 版：HiRDB_Type4_JDBC_Driver	pdls -d prc コマンドで接続中のアプリケーションを確認する際，J2EE サーバ名が出力されるため，接続元の判別が容易になります。	この環境変数は PDDEFAULTOPTION に V0904 を指定した場合もバージョン 09-50 以降の省略値を適用します。 バージョン 09-50 より前の省略値を適用したい場合は，UAP 名称に明示的にバージョン 09-50 より前の省略値を指定してください。UAP 名称の指定方法については，「接続情報の優先順位」を参照してください。
	Cosminexus 未使用時 →	**Cosminexus 未使用時** →			
PDSQLTRCFMT	2	1	−	SQL トレースの出力情報が詳細になるため，トラブルシュート情報が拡充します。	HiRDB SQL Tuning Advisor 08-03 以前を使用して SQL トレースの解析を行う場合，出力形式 2 には対応していないため，明示的に 1 を指定するか，PDDEFAULTOPTION に V0904 を指定してください。
PDSQLEXECTIME	YES	→	NO	SQL トレースに SQL 実行時間が出力されるため，トラブルシュート情報が拡充します。	なし
PDTCPCONOPT	1	1	0	サーバとの通信で消費する TCP ポートの数を削減できます。	なし
PDNODELAYACK	YES	YES	NO	AIX では，OS パラメタ（tcp_nodelaya	なし

クライアント環境定義名	バージョン 09-50 以降の省略値	0904 互換モードを適用している場合の省略値	バージョン 09-50 より前の省略値	省略値変更によるメリット	省略値変更によるデメリット
				ck）の指定で，パケットを受信して即時に ACK を送信できます。これによって，通信処理のレスポンスに遅延が発生する場合，遅延が改善されることがあります。 OS パラメタを指定した場合，システム全体に影響しますが，この機能によって HiRDB の通信にだけ即時 ACK を適用できます。	
PDARYERRPOS	YES	YES	NO	配列を使った更新でエラーとなった場合，エラーとなった配列要素を示す値を SQL 連絡領域に設定するため，トラブルシュート情報が拡充します。	なし
PDBLKFERRBREAK	YES	YES	NO	ブロック転送機能で HiRDB サーバから複数行を取得している間に暗黙的ロールバックが行われた場合，HiRDB クライアントが保持する複数行の検索結果のうち，UAP が最初の行を取得するときにエラーを返却するため，エラーを	なし

クライアント環境定義名	バージョン 09-50 以降の省略値	0904 互換モードを適用している場合の省略値	バージョン 09-50 より前の省略値	省略値変更によるメリット	省略値変更によるデメリット
				早く検知できます。	
PDCLTSIGPIPE	IGNORE	IGNORE	CATCH	UNIX 版マルチスレッド用クライアントライブラリで複数接続機能を使用時，UAP 実行プロセス中に HiRDB クライアントのシグナルハンドラを設定しません。 これによって，UAP 実行プロセスから HiRDB クライアントライブラリがアンロード後に SIGPIPE が発生した場合，プロセスが異常終了するのを回避できます。	なし

（凡例）

→：バージョン 09-50 より前の省略値と同じ省略値であることを示します。

−：バージョン 09-50 より前では，未サポートのクライアント環境定義です。

注※

V10-08 以降だけ

バージョン，リビジョンによって省略時の適用有無が異なるプリプロセスオプションを次の表に示します。オプション適用によるメリット及びデメリットを確認してください。確認の結果，旧バージョンとの互換性を重視する場合は，旧バージョンと同等の省略値になる互換モードを適用してください。互換モードはクライアント環境定義 PDDEFAULTOPTION で指定できます。詳細は，「UNIX 環境でのプリプロセス」又は「Windows 環境でのプリプロセス」を参照してください。

表 8–33　バージョン，リビジョンによって省略時の適用有無が異なるプリプロセスオプション

省略するプリプロセスオプション	バージョン 09-50 以降のオプション適用有無	0904 互換モードを適用している場合のオプション適用有無	バージョン 09-50 より前のオプション適用有無	省略値変更によるメリット	省略値変更によるデメリット
Xm	適用あり	→	適用なし	複数接続機能を使用した UAP をプリプロセスする場合に指定が必要なオプションのため，オプションの指定漏れを回避できます。	生成するポストソースの内容が変わります。 バージョン 09-50 より前の HiRDB で動作実績のあるアプリケーションに対して再度プリプロセスを実行する場合は，「COBOL 言語で複数接続機能を使用する場合の注意事項」を確認してください。Xm オプション指定により問題が発生する可能性がある場合は，PDDEFAULTOPTION に V0904 を指定してプリプロセスを実行してください。
Xs	適用あり	→	適用なし	単純注釈を使用した UAP をプリプロセスする場合に指定が必要なオプションのため，オプションの指定漏れを回避できます。	生成するポストソースの内容が変わります。 バージョン 09-50 より前の HiRDB で動作実績のあるアプリケーションに対して再度プリプロセスを実行する場合は，プログラムソース中の SQL 文の中に単純注釈を使用しているかどうか確認してください。単純注釈を使用していない場合は，PDDEFAULTOPTION に V0904 を指定してプリプロセスを実行してください。

（凡例）

→：バージョン 09-50 より前のオプション適用有無と同じであることを示します。

(2)　バージョン，リビジョンによって指定不要となったクライアント環境定義

バージョンアップによって次の表に示すクライアント環境定義を指定する必要がなくなりました。バージョンアップした場合にこれらのオペランドを指定したままでもエラーになりませんが，オペランドを省略することを検討してください。

表 8–34　指定不要となったクライアント環境定義

クライアント環境定義名	バージョン 09-50 の省略値	0904 互換モードを適用している場合の省略値
PDTCPCONOPT	→	1
PDNODELAYACK	→	YES
PDHJHASHINGMODE	→	pd_hashjoin_hashing_mode 指定値
PDBLKFUPD	→	YES
PDARYERRPOS	→	YES
PDBLKFERRBREAK	→	YES
PDCLTSIGPIPE	→	IGNORE

（凡例）

→：0904 互換モード適用時の省略値と同じ省略値であることを示します。

8.4.7　Red Hat Enterprise Linux 10，Oracle Linux 10 以降での UAP 作成

Red Hat Enterprise Linux 10，Oracle Linux 10 以降では，OS から 32 ビットアプリケーションを作成及び実行するためのパッケージ（例:glibc.i686,libgcc.i686）が提供されなくなります。そのため，32 ビットのクライアント機能を使用したアプリケーションの作成及び実行ができなくなります。アプリケーションを作成及び実行する場合は，アプリケーションを 64 ビット化する必要があります。

HiRDB 製品のクライアント機能の Red Hat Enterprise Linux 10，Oracle Linux 10 以降でのサポート状況を次の表に示します。埋込み型 UAP でのクライアントライブラリサポート状況の詳細については，「コンパイルとリンケージ」を参照してください。

表 8–35　HiRDB 製品のクライアント機能の Red Hat Enterprise Linux 10，Oracle Linux 10 以降でのサポート状況

クライアント機能の名称		HiRDB 製品でのサポート状況
埋込み型 UAP	共用ライブラリ	○
	アーカイブライブラリ	×
Type4 JDBC ドライバ	JDBC2.0 版	○
	JDBC4.0 版	○
	JDBC4.3 版	○
Type2 JDBC ドライバ		×
ODBC3.5 ドライバ		○
SQLJ		×

（凡例）

○：アプリケーションを作成及び実行できます。

×：アプリケーションを作成及び実行できません。

9 Java ストアドプロシジャ，Java ストアドファンクション

この章では，処理手続きを Java で記述する Java ストアドプロシジャ，Java ストアドファンクションの作成方法，実行方法について説明します。

9.1 概要

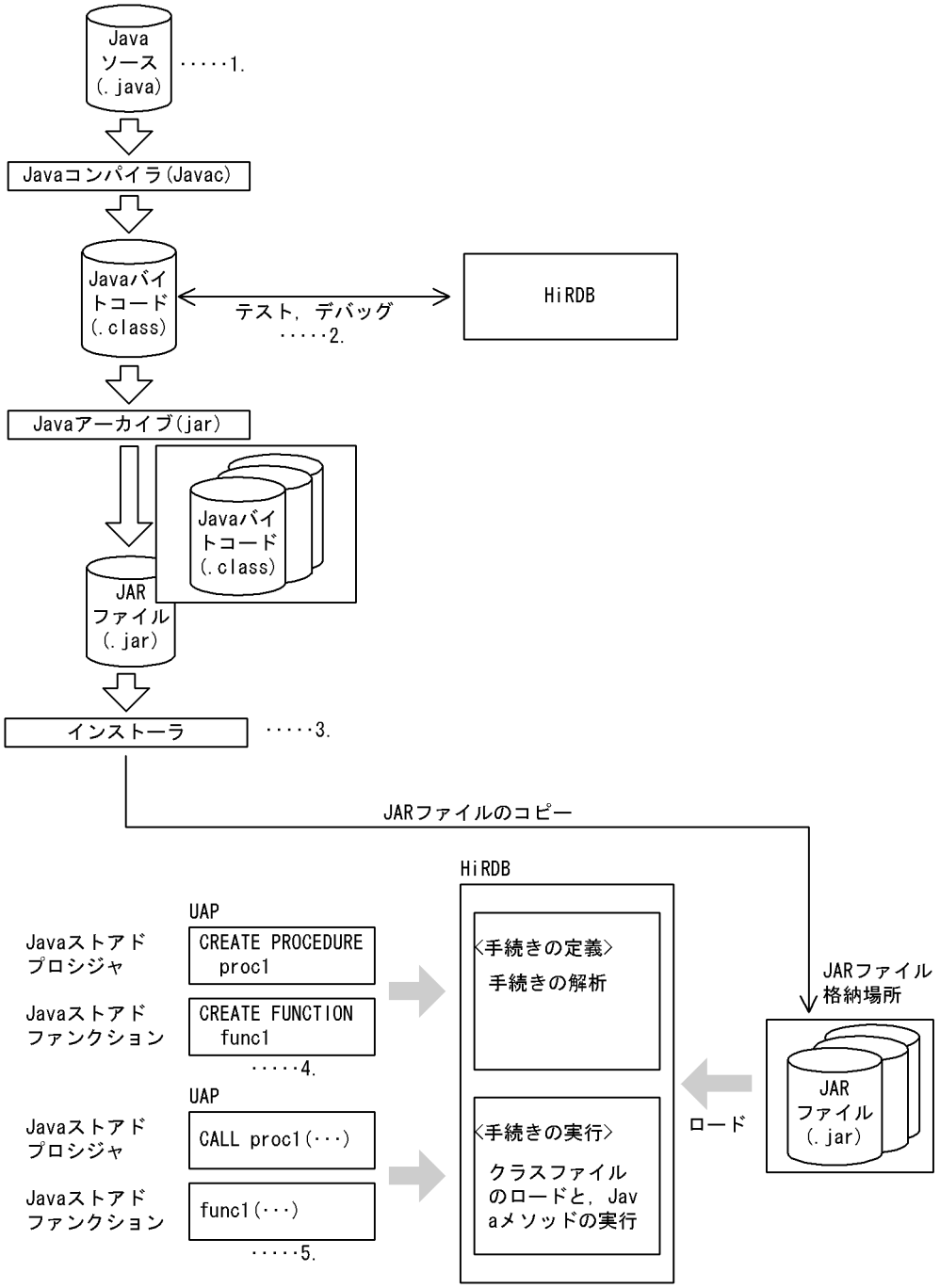
処理手続きを Java で記述したストアドプロシジャ，ストアドファンクションを，**Java ストアドプロシジャ**，**Java ストアドファンクション**といいます。

この章では，以降 Java ストアドプロシジャ，Java ストアドファンクションを総称して，**外部 Java ストアドルーチン**と呼びます。

なお，HiRDB の稼働プラットフォームによっては外部 Java ストアドルーチンを使用できません。詳細については，マニュアル「HiRDB システム運用ガイド」の「Java ストアドプロシジャ，Java ストアドファンクションを使用できる環境」を参照してください。

外部 Java ストアドルーチンの作成から実行までの流れを次の図に示します。

図 9–1　外部 Java ストアドルーチンの作成から実行までの流れ



[説明]

1. 外部 Java ストアドルーチンを作成します。詳細については，「外部 Java ストアドルーチンの作成」を参照してください。
2. クライアントの AP として，テスト，デバッグをします。詳細については，「外部 Java ストアドルーチンの作成」を参照してください。
3. HiRDB に JAR ファイルを登録します。詳細については，「JAR ファイルの新規登録」を参照してください。

4. 外部 Java ストアドルーチンを定義します。詳細については，「外部 Java ストアドルーチンの定義」を参照してください。
5. 外部 Java ストアドルーチンを実行します。詳細については，「外部 Java ストアドルーチンの実行」を参照してください。

- **外部 Java ストアドルーチンの特長**

  1. **サーバ，クライアント間の通信オーバヘッドがありません**

     外部 Java ストアドルーチンは，SQL ストアドプロシジャ，SQL ストアドファンクションと同様に，サーバ側で処理をします。したがって，サーバ，クライアント間での通信によるオーバヘッドはありません。

  2. **手続き本体，関数本体を Java で記述できます**

     記述言語が Java なので，SQL で記述するよりも高度な制御ができます。

  3. **異種 DBMS でも動作できます**

     Java はプラットフォームに依存しない言語です。したがって，Java で作成したプログラムは，外部 Java ストアドルーチンを使用できる異種 DBMS でも動作できます。

  4. **デバッグが簡単です**

     SQL ストアドプロシジャ，SQL ストアドファンクションのデバッグをする場合，実際にサーバ側で動作させる必要があります。これに対して，外部 Java ストアドルーチンのデバッグは，クライアント側に Java 言語のデバッガを用意することで，データベースアクセスを含めたデバッグができます。

- **外部 Java ストアドルーチン実行前の準備**

  外部 Java ストアドルーチンを実行する場合，事前に JDBC ドライバをインストールしておく必要があります。JDBC ドライバのインストールについては，「インストールと環境設定」を参照してください。

9.2　外部 Java ストアドルーチンの作成から実行までの各作業

外部 Java ストアドルーチンの作成から実行までの手順を次に示します。

1. 外部 Java ストアドルーチンの作成
2. JAR ファイルの新規登録
3. 外部 Java ストアドルーチンの定義
4. 外部 Java ストアドルーチンの実行
5. JAR ファイルの再登録・削除

9.2.1　外部 Java ストアドルーチンの作成

外部 Java ストアドルーチンを記述する場合，次の手順でします。

1. Java プログラムの記述（Java ファイルの作成）
2. コンパイル（Class ファイルの作成）
3. テスト，デバッグ
4. JAR 形式へのアーカイブ（JAR ファイルの作成）

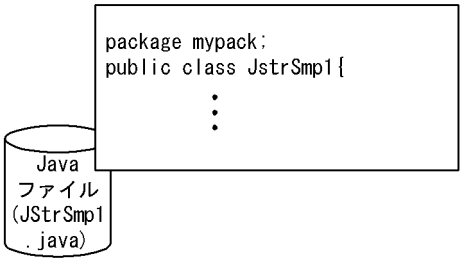
(1)　Java プログラムの記述（Java ファイルの作成）

外部 Java ストアドルーチンとして登録する Java プログラムを記述します。

Java プログラムを記述する場合の注意事項については，「Java プログラム作成時の注意事項」を参照してください。

Java プログラムの記述例を次の図に示します。

図 9–2　Java プログラムの記述例



(2)　コンパイル（Class ファイルの作成）

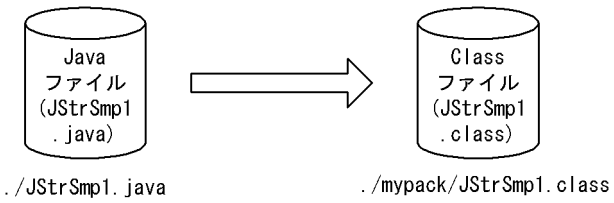
javac コマンドを使用して，Java ファイルから Class ファイルを作成します。

コンパイルの例を次の図に示します。

図 9–3　コンパイルの例

コマンド入力

```
% javac -d ./ JStrSmp1.java
```



[説明]

Java ファイルで package 指定をした場合は，コンパイル時に-d オプションを指定してください。

コンパイルすると，package 名のディレクトリが作成され，その下に Class ファイルが作成されます。

(3)　テスト，デバッグ

コンパイルしたファイルを，クライアント側の Java 仮想マシン上で実行し，テスト，デバッグをします。

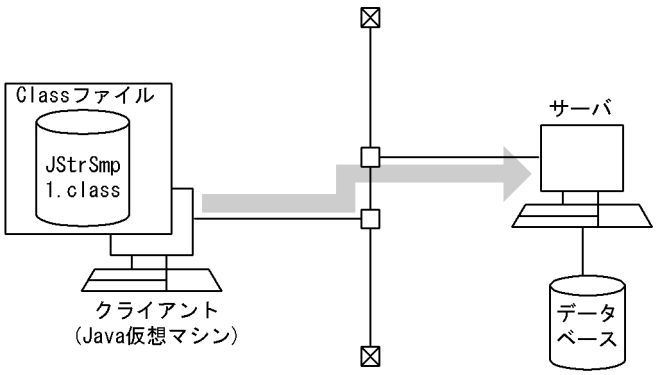
テスト，デバッグをするときの注意事項については，「テスト，デバッグ時の注意事項」を参照してください。

テスト，デバッグの概要を次の図に示します。

図 9–4　テスト，デバッグの概要

コマンド入力

```
% jdb mypack.JStrSmp1
```



(4)　JAR 形式へのアーカイブ（JAR ファイルの作成）

jar コマンドを使用して，複数の Class ファイルから JAR ファイルを作成します。

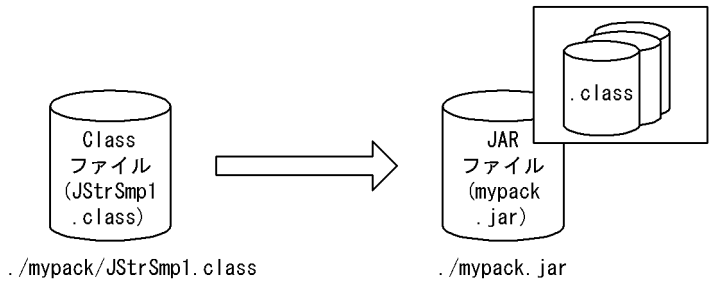
JAR ファイルを作成するときの注意事項については，「JAR ファイル作成時の注意事項」を参照してください。

JAR 形式へのアーカイブ例を次の図に示します。

図 9–5　JAR 形式へのアーカイブ例

コマンド入力

```
% jar cvf mypack.jar mypack/
```



9.2.2　JAR ファイルの新規登録

作成した JAR ファイルを，HiRDB サーバに新規登録（コピー）します。

登録するには，次の四つの方法があります。

- **SQL の INSTALL JAR の実行**

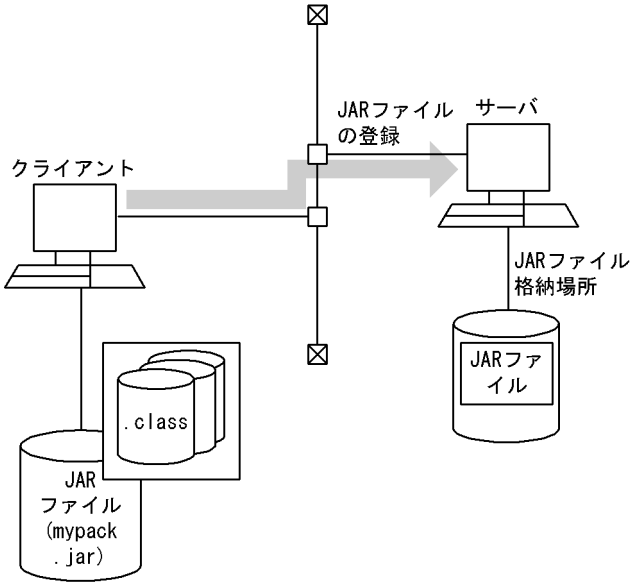
  UAP 中に INSTALL JAR を指定して実行するか，又はデータベース定義ユティリティで INSTALL JAR を指定して実行します。
- **pdjarsync コマンドの実行**

  pdjarsync コマンド（-I オプション指定）を実行します。pdjarsync コマンドは HiRDB 管理者だけが実行できます。
- **インストール用 Java メソッドの呼び出し**

  インストール用 Java メソッド「Jdbh_JARAccess クラスの Jdbh_JARReInstall」を呼び出すことで，JAR ファイルを登録できます。

JAR ファイルの登録の概要を次の図に示します。

図 9–6　JAR ファイルの登録の概要



参考

JAR ファイルを再登録する場合は，SQL の REPLACE JAR を実行します。JAR ファイルを削除する場合は，SQL の REMOVE JAR を実行します。また，HiRDB 管理者が JAR ファイルを再登録又は削除する場合は，pdjarsync コマンドを実行します。

なお，JAR ファイルの再登録及び削除は，JAR ファイルを新規登録したユーザ，又は HiRDB 管理者だけが実行できます。

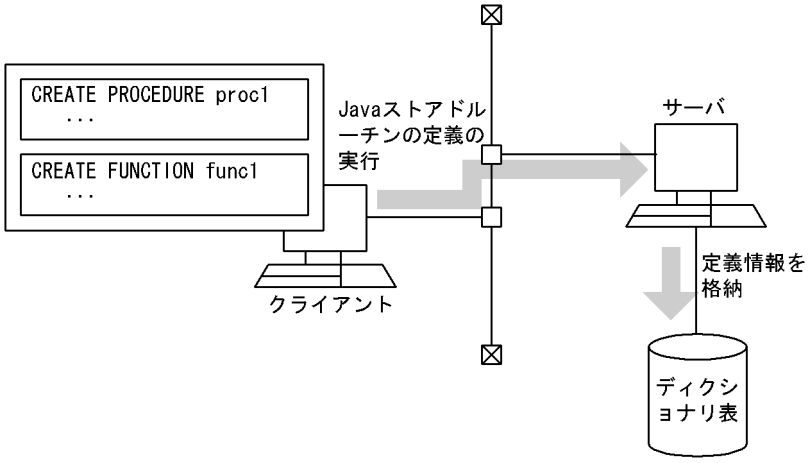
9.2.3　外部 Java ストアドルーチンの定義

外部 Java ストアドルーチンを定義する場合は，CREATE PROCEDURE 又は CREATE FUNCTION を使用します。CREATE PROCEDURE 又は CREATE FUNCTION で，Java メソッドと手続き名，又は Java メソッドと関数名との関連づけをします。

- **Java ストアドプロシジャの場合**
  CREATE PROCEDURE を使用して，Java メソッドを Java ストアドプロシジャとして登録します。
- **Java ストアドファンクションの場合**
  CREATE FUNCTION を使用して，Java メソッドを Java ストアドファンクションとして登録します。

外部 Java ストアドルーチンの定義例を次の図に示します。

図 9–7　外部 Java ストアドルーチンの定義例



パブリックルーチンの定義

他ユーザが定義した外部 Java ストアドルーチンを使用する場合は，UAP 中からストアドルーチンを呼び出すときに，所有者の認可識別子とルーチン識別子を指定する必要があります。

しかし，CREATE PUBLIC PROCEDURE 又は CREATE PUBLIC FUNCTION を実行して**パブリックルーチン**として定義すると，他ユーザが定義した外部 Java ストアドルーチンを使用する場合でも，UAP 中からストアドルーチンを呼び出すときに，所有者の認可識別子を指定する必要がなくなります（ルーチン識別子だけ指定します）。

外部 Java ストアドルーチンの再定義

java プログラムの修正などによって，一度定義した外部 Java ストアドルーチンを再度定義する場合は，ALTER PROCEDURE 又は ALTER ROUTINE を使用します。

外部 Java ストアドルーチンの削除

外部 Java ストアドルーチンを削除する場合は，DROP PROCEDURE 又は DROP FUNCTION を使用します。

また，パブリックルーチンを削除する場合は，DROP PUBLIC PROCEDURE 又は DROP PUBLIC FUNCTION を使用します。なお，パブリックルーチンを削除できるのは，パブリックルーチンを定義したユーザ，又は DBA 権限を持っているユーザだけです。

9.2.4　外部 Java ストアドルーチンの実行

外部 Java ストアドルーチンを実行する場合は，CALL 文又は関数呼出しを使用します。CALL 文又は関数呼出しを指定した SQL を実行することで，Java メソッドが外部 Java ストアドルーチンとして呼び出され，サーバ側の Java 仮想マシン上で実行されます。

- **Java ストアドプロシジャの場合**

  CALL 文を使用して，Java メソッドを Java ストアドプロシジャとして実行します。

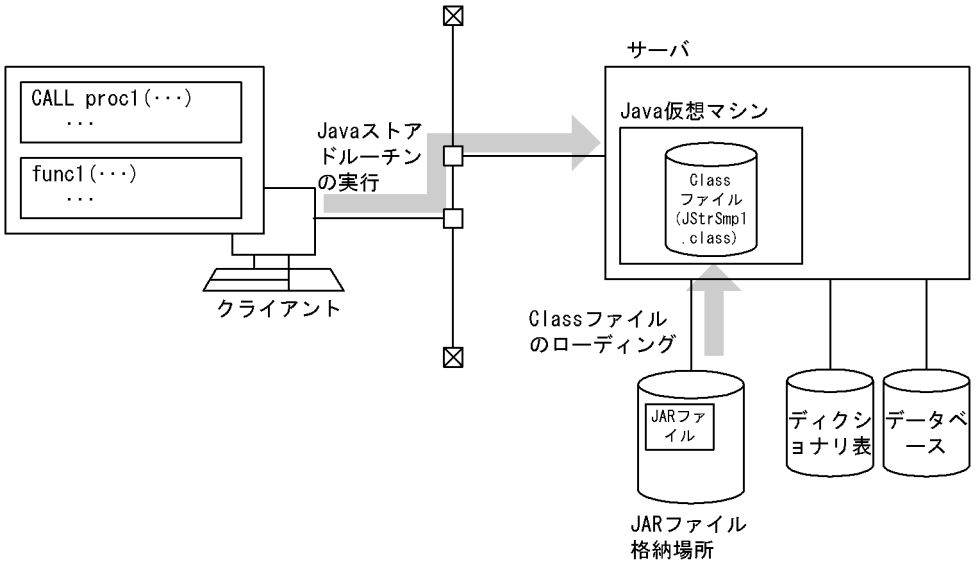
- **Java ストアドファンクションの場合**

関数呼出しを指定した SQL を使用して，Java メソッドを Java ストアドファンクションとして実行します。

CALL 文及び関数呼出しについては，マニュアル「HiRDB SQL リファレンス」を参照してください。

外部 Java ストアドルーチンの実行例を次の図に示します。

図 9–8　外部 Java ストアドルーチンの実行例



9.3 外部 Java ストアドルーチンのプログラム例

9.3.1 プログラム例

表 pics に BLOB 型として格納されているデータを SELECT 文で検索し，ZIP 圧縮してから呼び出し側に返却するストアドプロシジャの例を次に示します。

≪Java ストアドプロシジャの定義≫

```
CREATE PROCEDURE get_pic   .....................................1
  (IN pic_num INTEGER, OUT pic_data BLOB)   ....................1
  LANGUAGE JAVA   ..............................................2
  EXTERNAL NAME 'mypack.jar:JStrPics.getZippedPic(int, byte[][])'   ....3
  PARAMETER STYLE JAVA;   ......................................4
```

[説明]

1. プロシジャ名，パラメタの定義
2. LANGUAGE の設定
3. Java メソッドとの関連づけ
4. PARAMETER STYLE の設定

≪Java ストアドプロシジャの手続き本体≫

```
import java.sql.*;
import java.io.*;
import java.util.zip.*;

public class JStrPics{   .....................................1
  public static void getZippedPic(int jpic_num, byte[][] jpic_data)   ..2
    throws SQLException, IOException{   ........................3
    Connection con = DriverManager.getConnection(   ...........4
                     "jdbc:hitachi:hirdb");         ...........4

    PreparedStatement pstmt = con.prepareStatement   ...........5
        ("select p_name,p_data from pics where p_num = ?");  ...5
    pstmt.setInt(1, jpic_num);   ...............................5
    ResultSet rs = pstmt.executeQuery();   .....................6
    String name;   .............................................7
    byte[] srcPic;   ...........................................7

    while(rs.next()){
      name = rs.getString("p_name");   .........................8
      srcPic = rs.getBytes("p_data");   ........................9
    }
    ByteArrayOutputStream baos = new ByteArrayOutputStream();   ........10
    ZipOutputStream zos = new ZipOutputStream(baos);   .................10
    ByteArrayInputStream bais = new ByteArrayInputStream(srcPic);   ....10
    ZipEntry ze = new ZipEntry(name);   ........................10
    zos.putNextEntry(ze);   ....................................10

    int len = 0;   .............................................10
```

```
    byte[] buff = new byte[1024];   .....10
    while((len = bais.read(buff)) != -1){   .....10
    zos.write(buff, 0, len);   .....10
    }   .....10
    zos.closeEntry();   .....11
    bais.close();   .....11
    zos.close();   .....11

    jpic_data[0] = baos.toByteArray();   .....12
    baos.close();   .....12

    return;   .....13
  }
}
```

[説明]

1. クラス名の定義
2. メソッド名，パラメタ名の定義
3. 例外が発生した場合の定義
4. Connection オブジェクトの取得（ただし，HiRDB 接続ユーザ数が増えるのではなく，Java ストアドプロシジャはコール元の接続内で動作します）
5. SELECT 文の前処理
6. SELECT 文の実行，結果集合取得
7. 変数の宣言
8. 結果集合から p_name 列の値を取得
9. 結果集合から p_data 列の値を取得
10. srcPic 配列内のデータを ZIP 形式で圧縮し，zos ストリーム内に格納
11. 入力，出力ストリームのクローズ
12. baos ストリーム内の byte 列を，メソッドの OUT パラメタに設定
13. メソッド実行の終了

≪Java ストアドプロシジャの実行≫

```
import java.sql.*;
import java.io.* ;

public class Caller{   .....1
  public static void main(String[] args)   .....2
    throws SQLException, IOException{   .....3
    Connection con = DriverManager.getConnection(   .....4
                     "jdbc:hitachi:hirdb","USER1","PASS1");   .....4
    CallableStatement cstmt = con.prepareCall("{call get_pic(?,?)}");   ..5
    cstmt.setInt(1, 10);   .....5
    cstmt.registerOutParameter(2, java.sql.Types.LONGVARBINARY);   .....5

    cstmt.executeUpdate();   .....6
    byte[] getPic = cstmt.getBytes(2);   .....7
```

```
    ......

  }
}
```

[説明]

Java ストアドプロシジャを呼び出す Java アプリケーションのプログラム例です。

1. クラス名の定義
2. メソッド名，パラメタ名の定義
3. 例外が発生した場合の定義
4. Connection オブジェクトの取得（この Connection オブジェクトを取得することで，HiRDB に接続するため，ユーザ数が増えます）
5. CALL 文の前処理
6. CALL 文の実行
7. byte 配列型の OUT パラメタの取得

9.3.2 HiRDB が提供する外部 Java ストアドルーチンのサンプル

(1) サンプル 1

指定した年月のカレンダーを取得する Java ストアドプロシジャの例です。

- **外部 Java 手続き本体（ファイル名：sample1.java）**

```
/* ALL RIGHTS RESERVED,COPYRIGHT (C)2000,HITACHI,LTD. */
/* LICENSED MATERIAL OF HITACHI,LTD. */
/**************************************************************************/
/* name = HiRDB 06-00 Javaストアドプロシジャ サンプルプログラム 1      */
/**************************************************************************/
import java.lang.*;
import java.util.*;
/**************************************************************************/
/* name = サンプル１ クラス                                               */
/**************************************************************************/
public class sample1 {
    /*=====================================================================*/
    /* name = デバッグ用メインメソッド                                     */
    /*=====================================================================*/
    public static void main(java.lang.String[] args) {
        java.lang.Integer year = new Integer(args[0]);
        java.lang.Integer month = new Integer(args[1]);
        java.lang.String calendar[] = new String[1];
        calendar(year, month, calendar);
        System.out.println(calendar[0]);
    }
    /*=====================================================================*/
```

```
    /* name = サンプル１ メソッド                                              */
    /*=======================================================================*/
    public static void calendar(java.lang.Integer year,
                                java.lang.Integer month,
                                 java.lang.String[] calendar) {
        int DayOfWeek;          // 指定月の1日の曜日
        int week;               // 改行制御用
        int wyear = year.intValue();    // 年ワーク
        int wmonth = month.intValue();  // 月ワーク
        // カレンダーヘッダ作成
        calendar[0] = "         " +  wyear + " / " + wmonth + "¥n";
        calendar[0] += "Sun Mon Thu Wed Tue Fri Sat¥n";

        // カレンダーオブジェクト生成
        Calendar target_cal = new GregorianCalendar(wyear, wmonth - 1, 1);
        // 指定月の1日の曜日算出
        DayOfWeek = target_cal.get(Calendar.DAY_OF_WEEK);

        // 開始曜日まで空白設定
        for (week = 1; week < DayOfWeek; week++) {
            calendar[0] += "    ";
        }
        // 日付設定
        for (;
             target_cal.get(Calendar.MONTH) == wmonth - 1;
             target_cal.add(Calendar.DATE, 1), week++) {
            // 日付設定と桁数に応じたスペース調整
            if (target_cal.get(Calendar.DATE) < 10) {
                calendar[0] += "  " + target_cal.get(Calendar.DATE);
            } else {
                calendar[0] += " " + target_cal.get(Calendar.DATE);
            }
            // 日付間のパディング文字の設定
            if (week == 7) {
                calendar[0] += "¥n";
                week = 0;
            } else {
                calendar[0] += " ";
            }
        }

        return;
    }
}
```

上記の外部 Java 手続き本体を使用して，Java ストアドプロシジャを定義，実行する例を次に示します。

- **Java ファイルのコンパイル**

```
javac sample1.java
```

- **JAR ファイルの作成**

```
jar -cvf sample1.jar sample1.class
```

- **HiRDB への JAR ファイルの登録**（SQL の INSTALL JAR を使用した例です）

```
INSTALL JAR 'sample1.jar' ;
```

- **Java ストアドプロシジャの定義**

```
CREATE PROCEDURE calendar(IN pyear INT, IN pmonth INT, OUT calendar VARCHAR(255))
  LANGUAGE JAVA
  EXTERNAL NAME 'sample1.jar:sample1.calendar(java.lang.Integer, java.lang.Integer,
                 java.lang.String[]) returns void'
  PARAMETER STYLE JAVA
end_proc;
```

- **Java ストアドプロシジャの実行**

```
CALL calendar(?,?,?)
```

(2) サンプル 2

処理する範囲を年月日で指定し，その範囲の goods_no 列に対応した合計数を更新する例です。

なお，表は次のように定義されているものとします。

```
CREATE TABLE master_t1 (goods_no int,total_quantity dec(17,2))
CREATE TABLE tran_t1(goods_no int,quantity_1 dec(17,2),entrydate date)
```

- **外部 Java 手続き本体（ファイル名：sample2.java）**

```
/* ALL RIGHTS RESERVED,COPYRIGHT (C)2000,HITACHI,LTD. */
/* LICENSED MATERIAL OF HITACHI,LTD. */
/**************************************************************************/
/* name = HiRDB 06-00 Javaストアド サンプル２                             */
/**************************************************************************/
import java.lang.*;
import java.math.*;
import java.sql.*;
/**************************************************************************/
/* name = サンプル２クラス                                                */
/**************************************************************************/
public class sample2 {
    /*=====================================================================*/
    /* name = デバッグ用メインメソッド                                     */
    /*=====================================================================*/
    public static void main(String args[]) throws SQLException {
        java.sql.Date fromdate = Date.valueOf("1996-06-01");
        java.sql.Date todate = Date.valueOf("1996-06-30");
        try {
        // Driverクラスの登録
            Class.forName("JP.co.Hitachi.soft.HiRDB.JDBC.HiRDBDriver");
        } catch (ClassNotFoundException ex) {
                System.out.println("¥n*** ClassNotFoundException caught ***¥n");
                ex.printStackTrace();
                System.out.println ("");
                System.out.println("¥n************************************¥n");
                return;
        }
```

```
    jproc1(fromdate, todate);
}
/*====================================================================*/
/* name = サンプル２メソッド                                          */
/*====================================================================*/
public static void jproc1(java.sql.Date fromdate, java.sql.Date todate)
                                                    throws SQLException {

    java.lang.Integer x_goods_no;
    java.math.BigDecimal x_quantity_1, x_total_quantity;
    try {
        // コネクトオブジェクト生成(Java手続き内ではCONNECTは発行されない)
        java.sql.Connection con =
          DriverManager.getConnection("jdbc:hitachi:hirdb");

        con.setAutoCommit(false);  // 自動コミットの抑止
        // SELECT(stmt1)前処理
        java.sql.PreparedStatement stmt1 =
          con.prepareStatement("SELECT goods_no, quantity_1, entrydate FROM tran_t1
          WHERE entrydate BETWEEN ? AND ? ORDER BY entrydate");
        // SELECT(stmt2)前処理(ループの外で前処理しておく)
        java.sql.PreparedStatement stmt2 =
          con.prepareStatement("SELECT total_quantity FROM master_t1
          WHERE goods_no = ?");
        // INSERT(stmt3)前処理(ループの外で前処理しておく)
        java.sql.PreparedStatement stmt3 =
          con.prepareStatement("INSERT INTO master_t1 VALUES(?, ?)");

        // UPDATE(stmt4)前処理(ループの外で前処理しておく)
        java.sql.PreparedStatement stmt4 =
          con.prepareStatement("UPDATE master_t1 SET total_quantity = ?
          WHERE goods_no = ?");
         // SELECT(stmt1)入力パラメタ設定
        stmt1.setDate(1, fromdate);
        stmt1.setDate(2, todate);

        // SELECT(stmt1)実行
        java.sql.ResultSet rs1 = stmt1.executeQuery();
        while (rs1.next()) {
            // SELECT(stmt1)検索結果取得
            x_goods_no = (Integer)rs1.getObject("goods_no");
            x_quantity_1 = rs1.getBigDecimal("quantity_1");
            // SELECT(stmt2)入力パラメタ設定
            stmt2.setObject(1, x_goods_no);

            // SELECT(stmt2)実行
            java.sql.ResultSet rs2 = stmt2.executeQuery();
            // goods_noが登録済/未登録で処理分け
            if (!rs2.next()) {      // 未登録==>新規エントリ追加
                // 更新前にSELECT(stmt2)カーソルクローズ
                rs2.close();

                // INSERT(stmt3)入力パラメタ設定
                stmt3.setObject(1, x_goods_no);
                stmt3.setBigDecimal(2, x_quantity_1);
                // INSERT(stmt3)実行
                stmt3.executeUpdate();
```

```
                } else {                // 登録済==>既存エントリ更新
                    // 現在値取得
                    x_total_quantity = rs2.getBigDecimal("total_quantity");
                    // 増分
                    x_total_quantity = x_total_quantity.add(x_quantity_1);

                    // 更新前にSELECT(stmt2)カーソルクローズ
                    rs2.close();
                    // UPDATE(stmt4)入力パラメタ設定
                    stmt4.setBigDecimal(1, x_total_quantity);
                    stmt4.setObject(2, x_goods_no);
                    stmt4.executeUpdate() ;

                }
            }
            // SELECT(stmt1)カーソルクローズ
            rs1.close();

            // 各ステートメントオブジェクト解放
            stmt1.close();
            stmt2.close();
            stmt3.close();
            stmt3.close();
            // コネクション切断
            con.close();

        } catch (SQLException ex) { // SQLエラー処理
            SQLException fast_ex = ex;
            System.out.println("¥n***** SQLException caught *****¥n");
            while (ex != null) {
                System.out.println ("SQLState: " + ex.getSQLState ());
                System.out.println ("Message:  " + ex.getMessage ());
                System.out.println ("Vendor:   " + ex.getErrorCode ());
                ex.printStackTrace();
                ex = ex.getNextException ();
                System.out.println ("");
            }
            System.out.println("*********************************¥n");

            throw fast_ex;
        }
        return;
    }
}
```

上記の外部 Java 手続き本体を使用して，Java ストアドプロシジャを定義，実行する例を次に示します。

- **Java ファイルのコンパイル**

```
javac sample2.java
```

- **JAR ファイルの作成**

```
jar -cvf sample2.jar sample2.class
```

- **HiRDB への JAR ファイルの登録**（SQL の INSTALL JAR を使用した例です）

```
INSTALL JAR 'sample2.jar' ;
```

- **Java ストアドプロシジャの定義**

```
CREATE PROCEDURE jproc1(IN fromdate DATE, IN todate DATE)
  LANGUAGE JAVA
  EXTERNAL NAME 'sample2.jar:sample2.jproc1(java.sql.Date, java.sql.Date)
                 returns void'
  PARAMETER STYLE JAVA
end_proc;
```

- **Java ストアドプロシジャの実行**

```
CALL jproc1(IN ?,IN ?)
```

(3) サンプル 3

gzip，ungzip を使用して，BLOB 型データを圧縮，伸長する例です。

- **外部 Java 関数本体（ファイル名：sample3.java）**

```
/* ALL RIGHTS RESERVED,COPYRIGHT (C)2000,HITACHI,LTD. */
/* LICENSED MATERIAL OF HITACHI,LTD. */
/**************************************************************************/
/* name = HiRDB 06-00 Javaストアド サンプル３                             */
/**************************************************************************/
import java.util.zip.*;
import java.io.*;
public class sample3 {
    private final static int BUFF_SIZE = 4096;

    /*=======================================================================*/
    /* name = デバッグ用メインメソッド                                       */
    /*=======================================================================*/
    public static void main(String[] args) throws IOException {
        // 入力データ取得
        String sin = args[0];
        byte[] bin = args[0].getBytes();
        System.out.println("input data : " + sin);
        // GZIP(BLOB)
        byte[] bwork = gzip(bin);
        System.out.println("gzip(BLOB)  : " +
                           bin.length +  "=>" + bwork.length +
                           "(" + (bwork.length * 100 / bin.length) + "%): " +
                           "");
        // GUNZIP(BLOB)
        byte[] bout = gunzip(bwork);
        System.out.println("gunzip(BLOB): " +
                           bwork.length +  "=>" + bout.length +
                           "(" + (bout.length * 100 / bwork.length) + "%): " +
                           new String(bout));

        return;
    }
    /*=======================================================================*/
```

```
    /* name = サンプル３メソッド[gzip(BLOB)]                              */
    /*===================================================================*/
    public static byte[] gzip(byte indata[]) {
        // 圧縮データ出力用のストリーム生成
        ByteArrayOutputStream baos = new ByteArrayOutputStream();
        // 圧縮データ出力
        try {
            GZIPOutputStream zos = new GZIPOutputStream(baos);
            zos.write(indata, 0, indata.length);
            zos.close();
            baos.close();
        } catch (IOException ex) {
            System.out.println("gzip(BLOB): IOException: " + ex);
        ex.printStackTrace();
        }
        // 返却値の圧縮後バイト配列生成
        byte[] outdata = baos.toByteArray();
        return outdata;
    }
    /*===================================================================*/
    /* name = サンプル３メソッド[gunzip(BLOB)]                            */
    /*===================================================================*/
    public static byte[] gunzip(byte[] indata) {
        int rlen;                    // 入出力実長
        byte[] buff = new byte[BUFF_SIZE]; // 入出力用バッファ
        // 圧縮データ入力用のストリーム生成
        ByteArrayInputStream bais = new ByteArrayInputStream(indata);
        // 伸長データ出力用のストリーム生成
        ByteArrayOutputStream baos = new ByteArrayOutputStream();
        // 圧縮データ入力・伸長データ出力
        try {
            GZIPInputStream zis = new GZIPInputStream(bais);

            while ((rlen = zis.read(buff, 0, buff.length)) >= 0) {
              baos.write(buff, 0, rlen);
            }
            zis.close();
            bais.close();
            baos.close();
        } catch (IOException ex) {
            System.out.println("gunzip(BLOB): IOException: " + ex);
        ex.printStackTrace();
        }
        // 返却値の伸長後バイト配列生成
        byte[] outdata = baos.toByteArray();

        return outdata;
    }
}
```

上記の外部 Java 関数本体を使用して，Java ストアドファンクションを定義，実行する例を次に示します。

- **Java ファイルのコンパイル**

```
javac sample3.java
```

- **JAR ファイルの作成**

```
jar -cvf sample3.jar sample3.class
```

- **HiRDB への JAR ファイルの登録**（SQL の INSTALL JAR を使用した例です）

```
INSTALL JAR 'sample3.jar' ;
```

- **Java ストアドファンクションの定義**

```
CREATE FUNCTION gzip(indata BLOB(1M)) RETURNS BLOB(1M)
  LANGUAGE JAVA
  EXTERNAL NAME 'sample3.jar:sample3.gzip(byte[]) returns byte[]'
  PARAMETER STYLE JAVA
end_proc;
CREATE FUNCTION gunzip(indata BLOB(1M)) RETURNS BLOB(1M)
  LANGUAGE JAVA
  EXTERNAL NAME 'sample3.jar:sample3.gunzip(byte[]) returns byte[]'
  PARAMETER STYLE JAVA
end_proc;
```

- **Java ストアドファンクションの実行**

```
INSERT INTO t1 values(10, ?, gzip(? AS BLOB(1M)))
     :
SELECT c1, c2, gunzip(c3), length(c2), length(c3) from t1
```

(4)　サンプル 4

動的結果集合を使用して，2 表の検索した結果を返す例です。

- **外部 Java 手続き本体（ファイル名：sample4rs.java）**

```
/* ALL RIGHTS RESERVED,COPYRIGHT (C)2000,HITACHI,LTD. */
/* LICENSED MATERIAL OF HITACHI,LTD. */
/**************************************************************************/
/* name = HiRDB 06-00 Javaストアド Result Set 導通用ジョブ                */
/**************************************************************************/
import java.lang.*;
import java.math.*;
import java.sql.*;
/**************************************************************************/
/* name = Result Set 導通用クラス(プロシジャ側)                           */
/**************************************************************************/
public class sample4rs {
    /*=====================================================================*/
    /* name = デバッグ用メインメソッド                                     */
    /*=====================================================================*/
    public static void main(String args[]) throws SQLException {
        java.lang.Integer p1 = new Integer(10);
    int[]   cr_cnt = null;
    java.sql.ResultSet[] rs1 = null;
    java.sql.ResultSet[] rs2 = null;
    try {
        // Driverクラスの登録
```

```
        Class.forName("JP.co.Hitachi.soft.HiRDB.JDBC.HiRDBDriver");
    } catch (ClassNotFoundException ex) {
        System.out.println("¥n***** ClassNotFoundException caught *****¥n");
        ex.printStackTrace();
        System.out.println ("");
        System.out.println("*******************************¥n");
        return;
    }

    rs_proc(p1, cr_cnt, rs1, rs2);
    }
    /*====================================================================*/
    /* name = Result Set 導通用メソッド                                   */
    /*====================================================================*/
    public static void rs_proc(java.lang.Integer p1,int icnt_cr[],
                    java.sql.ResultSet[] rs1,
                    java.sql.ResultSet[] rs2) throws SQLException {
        java.lang.Integer x_goods_no;
        java.math.BigDecimal x_quantity_1, x_total_quantity;
        try {
            // コネクトオブジェクト生成(Java手続き内ではCONNECTは発行されない)
            java.sql.Connection con =
                DriverManager.getConnection("jdbc:hitachi:hirdb");

            con.setAutoCommit(false);  // 自動コミットの抑止

            // SELECT(stmt1)前処理
            java.sql.PreparedStatement stmt1 =
                con.prepareStatement("SELECT c1, c2 FROM rs_t1 WHERE c1 > ?");
            // SELECT(stmt1)入力パラメタ設定
            stmt1.setInt(1, p1.intValue());

            // SELECT(stmt2)前処理
            java.sql.PreparedStatement stmt2 =
                con.prepareStatement("SELECT c1, c2 FROM rs_t2 WHERE c1 > 10");
            // SELECT(stmt1)実行
            rs1[0] = stmt1.executeQuery();

            // SELECT(stmt2)実行
            rs2[0] = stmt2.executeQuery();

            // 動的結果集合の数
            icnt_cr[0] = 2;
            // SELECT(stmt2)実行(一行だけ取り出す)
            rs2[0].next();

         } catch (SQLException ex) { // SQLエラー処理
            SQLException fast_ex = ex;
            System.out.println("¥n***** SQLException caught *****¥n");
            while (ex != null) {
                System.out.println ("SQLState: " + ex.getSQLState ());
                System.out.println ("Message:  " + ex.getMessage ());
                System.out.println ("Vendor:   " + ex.getErrorCode ());
                ex.printStackTrace();
                ex = ex.getNextException ();
                System.out.println ("");
            }
            System.out.println("*******************************¥n");
```

```
            throw fast_ex;
        }

        return;
    }
}
```

- **UAP（sample4ap.java）**

```
/* ALL RIGHTS RESERVED,COPYRIGHT (C)2000,HITACHI,LTD. */
/* LICENSED MATERIAL OF HITACHI,LTD. */
/**************************************************************************/
/* name = HiRDB 06-00 Javaストアド Result Set 導通用ジョブ                */
/**************************************************************************/
import java.lang.*;
import java.math.*;
import java.sql.*;
/**************************************************************************/
/* name = Result Set 導通用クラス(CALL側)                                 */
/**************************************************************************/
public class sample4ap {
    /*=====================================================================*/
    /* name = デバッグ用メインメソッド                                     */
    /*=====================================================================*/
    public static void main(String args[]) throws SQLException {
        try {
        // Driverクラスの登録
                Class.forName("JP.co.Hitachi.soft.HiRDB.JDBC.HiRDBDriver");
        } catch (ClassNotFoundException ex) {
            System.out.println("¥n***** ClassNotFoundException caught *****¥n");
            ex.printStackTrace();
            System.out.println ("");
            System.out.println("*******************************¥n");
            return;
        }
        rs_call();
    }
    /*=====================================================================*/
    /* name = Result Set 導通用メソッド                                    */
    /*=====================================================================*/
    public static void rs_call() throws SQLException {
        java.lang.Integer xc1;
        java.lang.String xc2;
        int cr_cnt[] = new int[1];
        try {
            // コネクトオブジェクト生成(Java手続き内ではCONNECTは発行されない)
            java.sql.Connection con =
               DriverManager.getConnection
                            ("jdbc:hitachi:hirdb", "¥"USER1¥"", "¥"PASS1¥"");
            con.setAutoCommit(false);  // 自動コミットの抑止

            // CALL(stmt1)前処理
            java.sql.CallableStatement stmt1 =
                con.prepareCall("{CALL rs_proc(?,?)}");

            // CALL(stmt1)入力パラメタ設定
            stmt1.setInt(1, 10);
```

```
            stmt1.registerOutParameter(2, java.sql.Types.INTEGER);
            // CALL(stmt1)実行
            stmt1.execute();

            // CALL(stmt1)出力パラメタ取得
            cr_cnt[0] = stmt1.getInt(2);

            System.out.println("cr_cnt=" + cr_cnt[0] + "¥n");
            // 動的結果集合の取得
            java.sql.ResultSet rs = stmt1.getResultSet();

            while (rs.next()) {
                // SELECT(stmt1)検索結果取得
                xc1 = (Integer)rs.getObject("c1");
                xc2 = (String)rs.getObject("c2");
                System.out.println("xc1=" + xc1 + ",xc2=" + xc2 + "¥n");
            }
            // カーソルクローズ
            rs.close();

            if (stmt1.getMoreResults()) {
                rs = stmt1.getResultSet();
                while (rs.next()) {
                    // SELECT(stmt1)検索結果取得
                    xc1 = (Integer)rs.getObject("c1");
                    xc2 = (String)rs.getObject("c2");
                    System.out.println("xc1=" + xc1 + ",xc2=" + xc2 + "¥n");
                }
            }
            // カーソルクローズ
            rs.close();

            // 各ステートメントオブジェクト解放
            stmt1.close();

            // コネクション切断
            con.close();
        } catch (SQLException ex) { // SQLエラー処理
            SQLException fast_ex = ex;

            System.out.println("¥n***** SQLException caught *****¥n");
            while (ex != null) {
                System.out.println ("SQLState: " + ex.getSQLState ());
                System.out.println ("Message:  " + ex.getMessage ());
                System.out.println ("Vendor:   " + ex.getErrorCode ());
                ex.printStackTrace();
                ex = ex.getNextException ();
                System.out.println ("");
            }
            System.out.println("*********************************¥n");
            throw fast_ex;
        }

        return;
    }
}
```

- **Java ストアドプロシジャの定義**

```
CREATE PROCEDURE rs_proc(IN p1 INT,OUT cr_cnt INT)
  DYNAMIC RESULT SETS 2
  LANGUAGE JAVA
  EXTERNAL NAME 'sample4.jar:sample4rs.rs_proc(java.lang.Integer, int[], java.sql.
ResultSet[], java.sql.ResultSet[]) returns void'
  PARAMETER STYLE JAVA
end_proc;
```

9.4　Java プログラム作成時の注意事項

Java プログラムを作成する場合の注意事項について説明します。なお，Java で制御処理を記述する場合，次の制限があります。

- スレッドは作成できません。
- GUI は使用できません。
- ほかの DBMS へ接続できません。
- ファイルは操作できません。
- Java Runtime Environment のセキュリティポリシーは変更しないでください。

9.4.1　Type2 JDBC ドライバ又は Type4 JDBC ドライバの使用

Type2 JDBC ドライバ又は Type4 JDBC ドライバを使用して Java ストアドプロシジャを動作させる場合は，「Type2 JDBC ドライバからの移行」の設定内容を確認してください。ドライバ名称，HiRDB への接続時の URL で設定するプロトコル名称，サブプロトコル名称，及びサブネームを Java ストアドプロシジャで記述することで，その設定内容によって Type2 JDBC ドライバと Type4 JDBC ドライバのどちらを使用するかが決まります。

9.4.2　実行できないメソッド

Java 仮想マシン上では，セキュリティポリシーでのアクセス権の設定によって，実行できるメソッドを制限します。HiRDB 内の Java 仮想マシンでは，アクセス権が必要なメソッドはすべて実行できません。

セキュリティポリシーでのアクセス権の設定，及び実行できないメソッドの一覧については，JDK 付属のドキュメントを参照してください。

セキュリティポリシーでのメソッドの実行制限を次の図に示します。

図 9-9　セキュリティポリシーでのメソッドの実行制限



9.4.3　パッケージ，クラス，及びメソッドの定義

パッケージ，クラス，及びメソッドを定義する場合の注意事項について説明します。

パッケージ，クラス，及びメソッドについては，JDK 付属のドキュメントを参照してください。

(1)　パッケージ

1. パッケージ名は省略できます。
2. パッケージ名を指定する場合，「パッケージ名.クラス名」の文字列長を 255 文字以内にしてください。
3. 次のパッケージ名は使用できません。
   - JRE にあるパッケージの名称
   - HiRDB が提供するパッケージの名称

(2)　クラス

1. クラス名の文字列長は，255 文字以内にしてください。
2. クラスの定義は，「public class <クラス名>」の形式にしてください。

(3)　メソッド

1. メソッド名の文字列長は，255 文字以内にしてください。
2. メソッドの定義は次のようにしてください。

   Java ストアドプロシジャの場合：

   public static void <メソッド名>

   Java ストアドファンクションの場合：

   public static <返り値の型> <メソッド名>
3. メソッド内で例外が発生する可能性がある場合，発生する例外を throw 節で宣言するか，又は try.catch を記述する必要があります。Java ストアドプロシジャの場合，JDBC 内のほとんどのメソッドが，「SQLException」例外を発生する可能性があります。
4. メソッドから参照できるクラスは，Java プラットフォームコア API に含まれるクラス，及び実行中のメソッドがある JAR ファイルに含まれるクラスです。

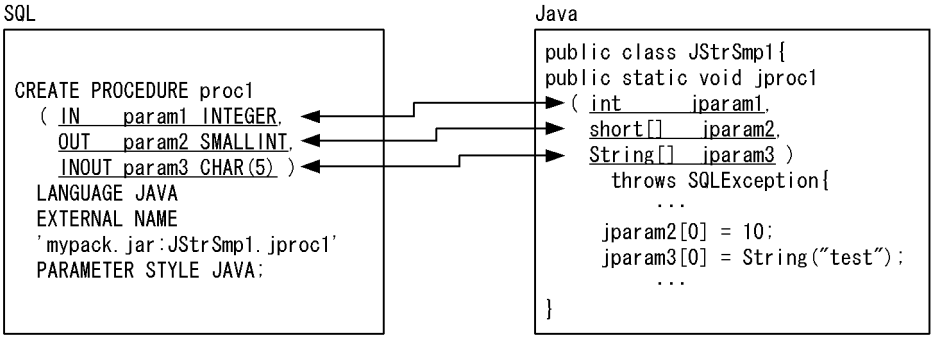
9.4.4　パラメタ入出力モードのマッピング（Java ストアドプロシジャ限定）

Java ストアドプロシジャでの，SQL のパラメタ入出力モード（IN，OUT，又は INOUT）のマッピングについて説明します。なお，Java ストアドファンクションについては，パラメタ入出力モードの指定はありません。

マッピングについては，マニュアル「HiRDB SQL リファレンス」の型マッピングを参照してください。

パラメタ入出力モードのマッピング例を次の図に示します。

図 9–10　パラメタ入出力モードのマッピング例



(1)　IN パラメタ

SQL で IN パラメタとして定義したパラメタの場合，Java プログラムでは，対応するデータ型をそのまま使用します。

例えば，CREATE PROCEDURE で，SQL の INTEGER 型として定義した IN パラメタは，Java プログラムでは対応する int 型又は java.lang.Integer 型で定義します（図「パラメタ入出力モードのマッピング例」の param1 と jparam1）。

(2)　OUT パラメタ又は INOUT パラメタ

SQL で OUT パラメタ又は INOUT パラメタとして定義したパラメタの場合，Java プログラムでは，対応するデータ型の配列型として定義します。Java 言語でのポインタ表現の手法が，「該当するデータ型の要素数 1 の配列としてパラメタを渡す」であるため，OUT 又は INOUT パラメタはこのように実現されます。

例えば，CREATE PROCEDURE で，SQL の SMALLINT 型として定義した OUT パラメタは，Java プログラムでは対応する short 型又は java.long.Short 型の配列型で定義します（図「パラメタ入出力モードのマッピング例」の param2 と jparam2，param3 と jparam3）。また，Java メソッド内で，OUT パラメタ又は INOUT パラメタに値を返却する場合，配列の先頭に値を設定します（図「パラメタ入出力モードのマッピング例」の jparam2，jparam3）。

9.4.5　結果集合返却機能（Java ストアドプロシジャ限定）

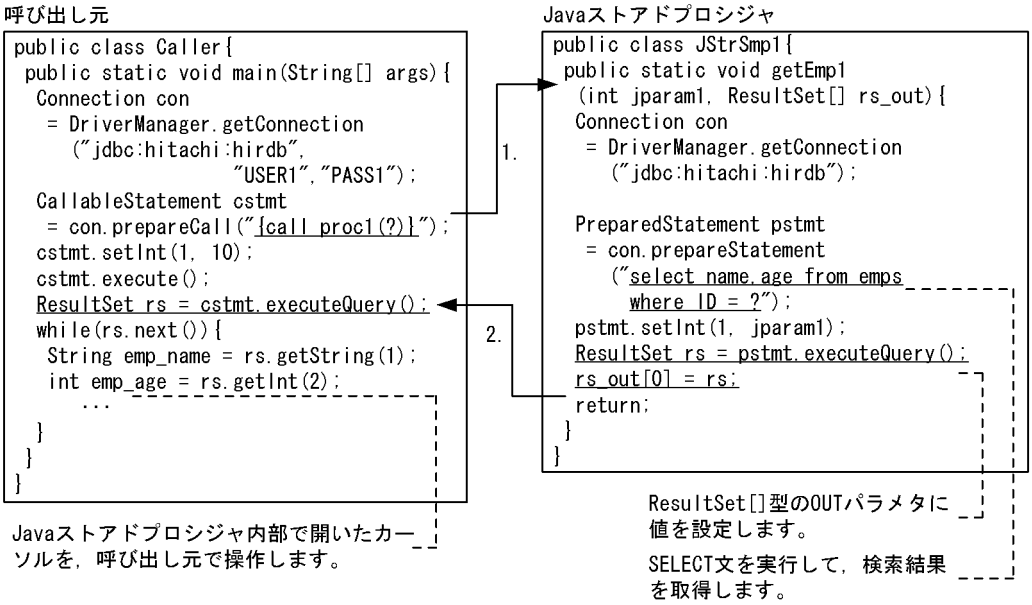
Java ストアドプロシジャ定義時に，CREATE PROCEDURE の DYNAMIC RESULT SETS 句に 1 以上の値を指定した場合，結果集合返却機能を使用できます。なお，Java ストアドファンクションについては，結果集合返却機能は使用できません。

(1)　結果集合返却機能とは

Java ストアドプロシジャ内での，SELECT 文の実行によって得られるカーソルを，Java ストアドプロシジャの呼び出し元で参照する機能を，**結果集合返却機能**といいます。

結果集合返却機能の概要を次の図に示します。

図 9–11　結果集合返却機能の概要（Java ストアドプロシジャの場合）



［説明］
1. CREATE PROCEDUREで関連付けられたメソッドを呼び出します。
2. 結果集合を返却します。

(2)　結果集合返却機能を使用できる呼び出し元の言語

結果集合返却機能を使用できる呼び出し元の言語を次に示します。

- Java
- C
- C++
- COBOL※
- OOCOBOL

注※

RDB ファイル入出力機能を使用していない場合，使用できます。

(3)　結果集合返却機能の使用例

Java ストアドプロシジャ内で，表 emps_1 及び表 emps_2 に対して，rank<10 の条件を満たす列 rank，name，及び age を取得します。呼び出し元で 2 個の結果集合を受け取り，これらを操作します。

≪Java ストアドプロシジャの定義≫

```
CREATE PROCEDURE proc2(IN param1 INTEGER)   .....................1
  DYNAMIC RESULT SETS 2   .....................................2
  LANGUAGE JAVA   .............................................3
  EXTERNAL NAME   .............................................4
```

```
    'mypack.jar:JStrSmp1.getEmp2(int, ResultSet[], ResultSet[])'   ..4
  PARAMETER STYLE JAVA;   ..........................................5
```

[説明]

1. プロシジャ名，パラメタの定義
2. 返却する検索結果情報数の指定
3. LANGUAGE の設定
4. Java メソッドとの関連づけ
5. PARAMETER STYLE の設定

≪Java ストアドプロシジャの手続き本体≫

```
import java.sql.*;   ..............................................1

public class JStrSmp1{   ..........................................2
  public static void getEmp2   ....................................3
    (int jparam1, ResultSet[] rs1_out, ResultSet[] rs2_out)   .....4
          throws SQLException {   ...................................4
    Connection con = DriverManager.getConnection (   ..............5
                    "jdbc:hitachi:hirdb");         ..............5
    con.setAutoCommit(false);    ..................................6

    PreparedStatement pstmt1 = con.prepareStatement   .............7
        ("select rank,name,age from emps_1 where rank < ?   .........7
         order by rank");   .......................................7
    pstmt1.setInt(1, jparam1);   .................................7
    ResultSet rs1 = pstmt1.executeQuery();   .....................8
    rs1_out[0] = rs1;   ..........................................9
    PreparedStatement pstmt2 = con.prepareStatement   .............10
        ("select rank,name,age from emps_2 where rank < ?   .........10
         order by rank");   .......................................10
    pstmt2.setInt(1, jparam1);   .................................10
    ResultSet rs2 = pstmt2.executeQuery();   .....................11
    rs2_out[0] = rs2;   ..........................................12
    return;   ....................................................13
  }
}
```

[説明]

1. java.sql パッケージのインポート
2. クラス名の定義
3. メソッド名の定義
4. パラメタ名の定義（第 2，第 3 引数が結果集合返却用）
5. Connection オブジェクトの取得
6. 自動コミットの抑止
7. SELECT 文の前処理
8. SELECT 文の実行

9. ResultSet[]型の第 2 引数に，取得した結果集合 rs1 を設定

10. SELECT 文の前処理

11. SELECT 文の実行

12. ResultSet[]型の第 3 引数に，取得した結果集合 rs1 を設定

13. 呼び出し終了，及び結果集合返却

≪Java ストアドプロシジャの実行（呼び出し元）≫

```
import java.sql.*;   ..................................................1

public class Caller{   ...............................................2
  public static void main(String[] args) throws SQLException {   ........3
    Connection con = DriverManager.getConnection(   ....................4
                  ”jdbc:hitachi:hirdb” ,” USER1” ,” PASS1”);   ...........4
    CallableStatement cstmt = con.prepareCall(”{call proc2(?)}”);   .....5
    cstmt.setInt(1, 10);   ..............................................5
    ResultSet rs;   .....................................................6
    int emp_rank;   .....................................................6
    String emp_name;   ..................................................6
    int emp_age;   ......................................................6

    if(cstmt.execute()){   ..............................................7
       rs = cstmt.getResultSet();   .....................................8
       System.out.println(”*** emps_1 ***”);   ..........................9
       while(rs.next()){   ..............................................9
         emp_rank = rs.getInt(1);   .....................................9
         emp_name = rs.getString(2);   ..................................9
         emp_age = rs.getInt(3);   ......................................9
         System.out.println(”RANK =” + emp_rank +   ......................9
           ”  NAME = ” + emp_name + ”  AGE = ” + emp_age);   ............9
         }
    }
    if(cstmt.getMoreResults()){   ......................................10
       rs= cstmt.getResultSet();   .....................................11
       System.out.println(”*** emps_2 ***”);   ..........................12
       while(rs.next()){   ..............................................12
         emp_rank = rs.getInt(1);   .....................................12
         emp_name = rs.getString(2);   ..................................12
         emp_age = rs.getInt(3);   ......................................12

         System.out.println(”RANK =” + emp_rank +   ......................12
           ”  NAME = ” + emp_name + ”  AGE = ” + emp_age);   ............12
       }
       rs.close();   ...................................................13
    }
  }
}
```

[説明]

1. java.sql パッケージのインポート

2. クラス名の定義

3. メソッド名の定義

4. Connection オブジェクトの取得
5. CALL 文の前処理
6. 変数の宣言
7. CALL 文の実行
8. 結果集合の取得
9. 1 個目の結果集合からの情報を出力
10. 次の結果集合があるかどうかの確認
11. 次の結果集合を取得
12. 2 個目の結果集合からの情報を出力
13. 結果集合のクローズ

(4)　結果集合返却機能を使用する場合の注意事項

(a)　CREATE PROCEDURE での Java ストアドプロシジャ定義時

1. DYNAMIC RESULT SETS 句に，Java ストアドプロシジャ内から返却する結果集合数の最大値を指定します。ここに 0 を指定した場合，結果集合返却機能は使用できません。
2. Java ストアドプロシジャのパラメタに設定する ResultSet[]型の OUT パラメタは，CREATE PROCEDURE のパラメタには設定しないでください。
3. EXTERNAL NAME で Java プログラムとの対応付けをする場合，ResultSet[]型の引数を含めてください。

(b)　呼び出し元のメソッド作成時

1. CALL 文のパラメタには，Java ストアドプロシジャ用メソッドの ResultSet[]型のパラメタを含めないでください。
2. 返却される結果集合が 2 個以上の場合，2 個目以降の結果集合を受け取るときは，getMoreResult（次の検索結果があるかどうかの確認），及び getResultSet（次の検索結果を受け取る）メソッドを使用してください。

(c)　Java ストアドプロシジャ用のメソッド作成時

検索結果（ResultSet）は，クローズしないで ResultSet[]型の OUT パラメタに設定してください。

9.4.6　Java ストアドプロシジャ中のコネクション

Java ストアドプロシジャ中では，アクティブなコネクションは 1 回だけ生成できます。Java ストアドプロシジャの終了前にガベージコレクターに任せたデータベースと JDBC リソースの解除，又は close()メソッドで明示的なデータベースと JDBC リソースの解除をした場合，コネクションオブジェクトを使用したデータベースの操作はできません。

9.4.7　結果集合の解放

結果集合のオブジェクトを解放する場合は，close()メソッドで明示的に解放してください。ガベージコレクターでの暗黙的な解放では，Java ストアドプロシジャ終了までリソースが解放されません。

9.5　テスト，デバッグ時の注意事項

外部 Java ストアドルーチンは，通常の Java プログラムを DBMS サーバで動作させるというアーキテクチャであるため，そのテスト，デバッグ方法も通常の Java アプリケーションのテスト，デバッグと同様に実行できます。

Java プログラムを作成した後，Java プログラムがストアドプロシジャ又はストアドファンクションとして正常に動作するかを，テスト，デバッグします。ここでは，テスト，デバッグ時の注意事項について説明します。

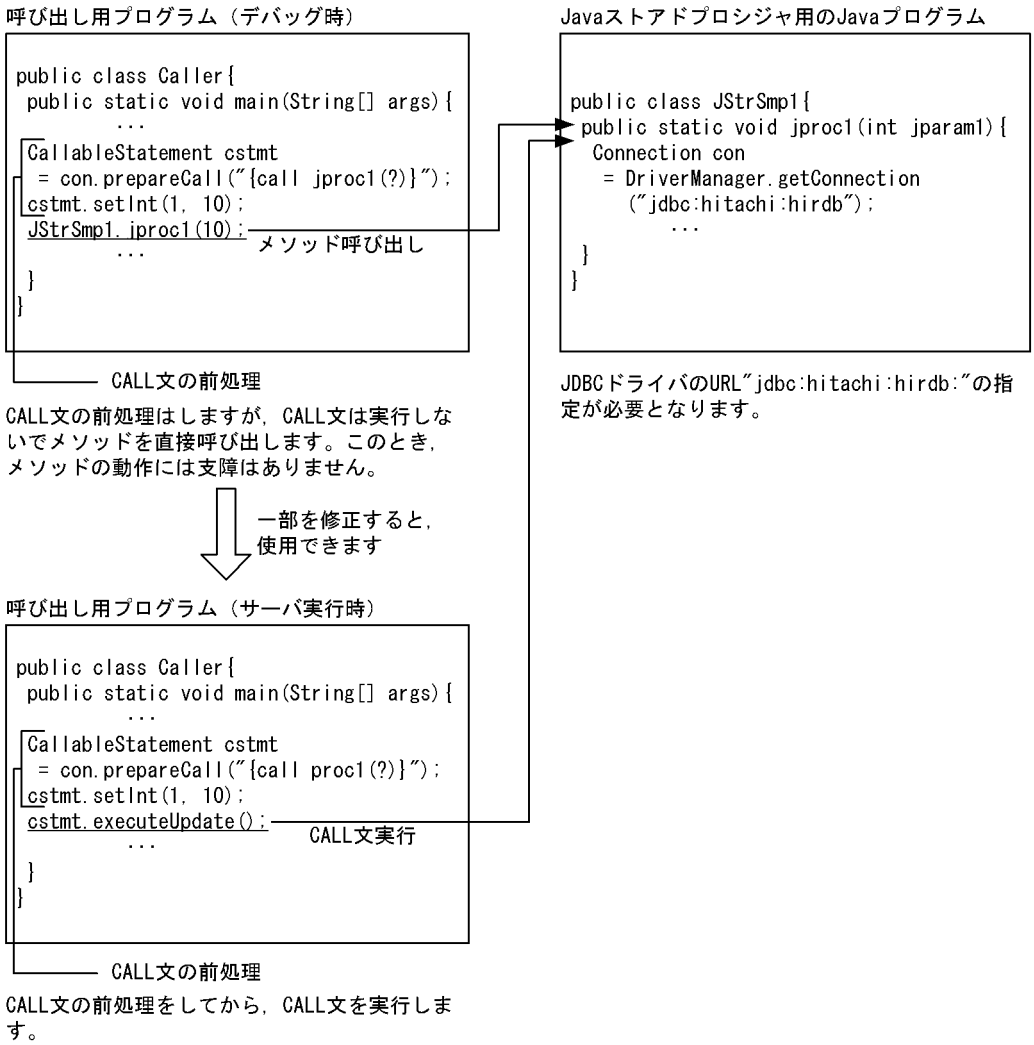
9.5.1　Java ストアドプロシジャ用の Java プログラムの場合

Java ストアドプロシジャ用の Java プログラムのテスト，デバッグをする場合，次の点を考慮してください。

1. Java ストアドプロシジャ用の Java プログラムは，サーバ実行時にデバッグ時のものを修正しなくても使用できます。
2. 呼び出し元の Java プログラムは，デバッグ時には Java ストアドプロシジャ用の Java プログラムのメソッドを直接コールします。サーバ実行時には，CALL 文として呼び出します。
3. デバッグ時とサーバ実行時とでは，Java 仮想マシンの環境が同じでないため，使用できるメソッドが異なる場合もあります。実行できないメソッドについては，「実行できないメソッド」を参照してください。

Java ストアドプロシジャ用の Java プログラムのテスト，デバッグ手順を次の図に示します。

図 9–12　Java ストアドプロシジャ用の Java プログラムのテスト，デバッグ手順



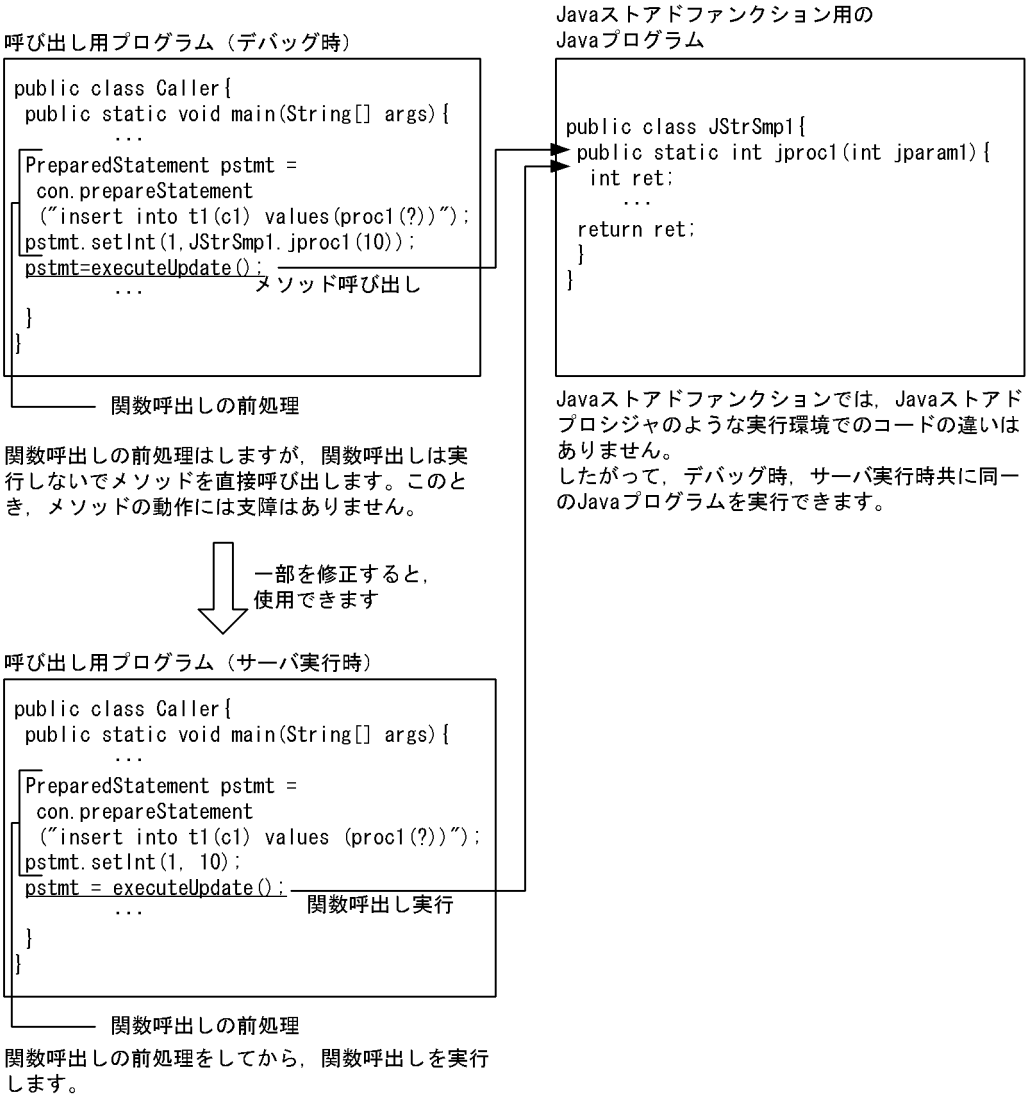
9.5.2　Java ストアドファンクション用の Java プログラムの場合

Java ストアドファンクション用の Java プログラムのテスト，デバッグをする場合，次の点を考慮してください。

1. Java ストアドファンクション用の Java プログラムは，サーバ実行時にデバッグ時のものを修正しなくても使用できます。
2. 呼び出し元の Java プログラムは，デバッグ時には Java ストアドファンクション用の Java プログラムのメソッドを直接コールします。サーバ実行時には，関数呼出しとして呼び出します。
3. デバッグ時とサーバ実行時とでは，Java 仮想マシンの環境が同じでないため，使用できるメソッドが異なる場合もあります。実行できないメソッドについては，「実行できないメソッド」を参照してください。

Java ストアドファンクション用の Java プログラムのテスト，デバッグ手順を次の図に示します。

図 9–13　Java ストアドファンクション用の Java プログラムのテスト，デバッグ手順



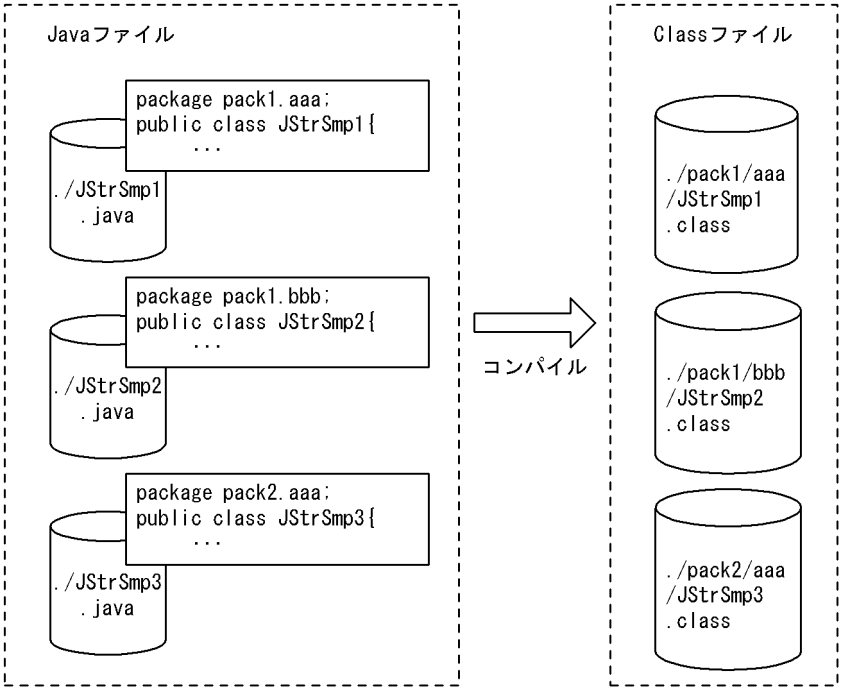
9.6　JAR ファイル作成時の注意事項

JAR ファイルを作成するときの注意事項について説明します。

Java には，プログラムを機能ごとに分割管理するための，「パッケージ」という概念があります。実際には，パッケージはディレクトリ構造として表現されるため，コンパイル後はパッケージ名のディレクトリ下に Class ファイルが作成されます。

Class ファイルが作成される場所を次の図に示します。

図 9–14　Class ファイルが作成される場所



Java ファイルを作成する場合，ディレクトリ構造を含めたファイルの統合，圧縮ができます。

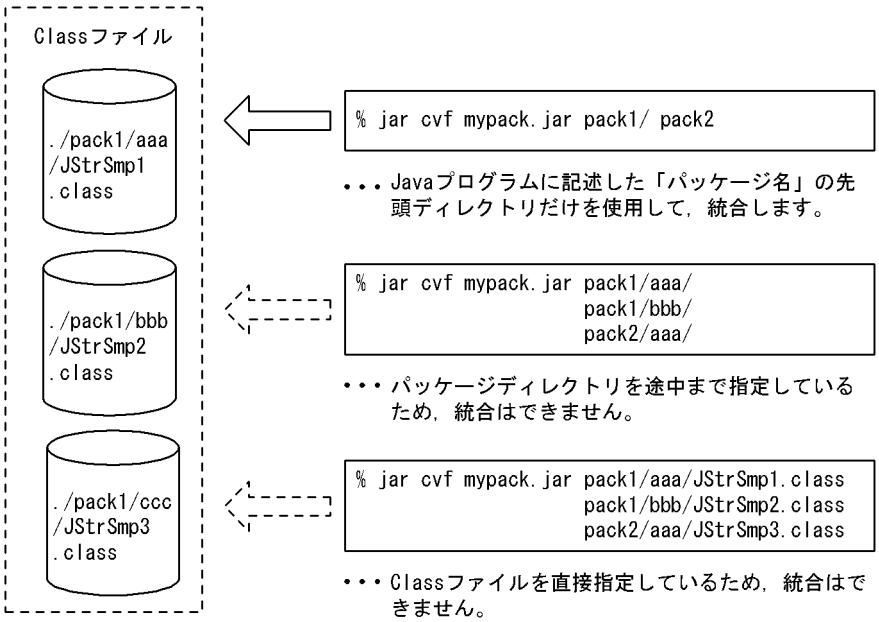
JAR ファイルには，Class ファイルだけでなく Java ファイルも同時に統合できます。HiRDB に登録した JAR ファイル内から，GET_JAVA_STORED_ROUTINE_SOURCE 指定の検索で特定の Class の Java プログラムソースを取得する場合には，Java ファイルも同時に統合する必要があります。
GET_JAVA_STORED_ROUTINE_SOURCE 指定については，マニュアル「HiRDB SQL リファレンス」を参照してください。

9.6.1　Class ファイルを統合する場合

Java プログラム作成時にパッケージ指定をした場合，JAR ファイルへの統合はパッケージディレクトリごとにします。なお，このとき Class ファイルだけ単体で指定しないでください。

Class ファイルを JAR ファイルへ統合する例を次の図に示します。

図 9–15　Class ファイルを JAR ファイルへ統合する例



同一の Class ファイル名の場合でも，パッケージが異なっていれば，同一の JAR ファイルへ統合できます。

9.6.2　Java ファイルを統合する場合

Java ファイルを統合する場合の注意事項を次に示します。

1. Class ファイルに対応する Java プログラムのソースを検索する場合，Java ファイルも Class ファイルと同時に統合する必要があります。
2. JAR ファイルに統合する Java ファイルは，任意のディレクトリに格納しておきます。
3. 同一の Class ファイル名が複数のパッケージにある場合，Java ファイルを異なるディレクトリに格納することで，それぞれの Java プログラムのソースを検索できるようになります。例を次に示します。

   （例）

   Class ファイルが次のようなパッケージで構成されている場合，pack1.aaa.JStrAAA と pack2.ccc.JStrAAA は，生成される Class ファイル名が同じになります。

   ./pack1/aaa/JstrAAA.class

   ./pack1/bbb/JstrBBB.class

   ./pack2/ccc/JstrAAA.class

   Java ファイルは，ディレクトリ構造下で管理する必要はありませんが，同一名のファイルがある場合は同一ディレクトリ下には格納できません。このような場合，次のように格納すれば，同一名のファイルでも格納できます。

   ./src1/JStrAAA.java

   ./src1/JStrBBB.java

   ./src2/JStrAAA.java

なお，この場合，指定した Class ファイルに対応する Java ファイルが特定できないため，同一名称のすべての Java ファイルが検索結果として取得されます。

例えば，JStrAAA.java を検索した場合，pack1.aaa.JStrAAA.java，及び pack2.ccc.JStrAAA.java の両方の情報が取得されます。JStrBBB.java を検索した場合は，pack1.bbb.JStrBBB.java の情報が取得されます。

10 C ストアドプロシジャ，C ストアドファンクション

この章では，処理手続きを C 言語で記述する C ストアドプロシジャ，C ストアドファンクションの作成方法，実行方法について説明します。

10.1 概要

処理手続きを C 言語で記述したストアドプロシジャ，ストアドファンクションを，**C ストアドプロシジャ**，**C ストアドファンクション**といいます。

この章では，以降 C ストアドプロシジャ，C ストアドファンクションを総称して，**外部 C ストアドルーチン**と呼びます。

外部 C ストアドルーチンの特長

- **サーバ，クライアント間の通信オーバヘッドがありません**
  外部 C ストアドルーチンは，SQL ストアドプロシジャ，SQL ストアドファンクションと同様に，サーバ側で処理をします。したがって，サーバ，クライアント間での通信によるオーバヘッドはありません。
- **手続き本体，関数本体を C 言語で記述できます**
  C 言語の命令を直接記述できるため，SQL 制御文よりも柔軟に処理を記述できます。
- **デバッグが簡単です**
  SQL ストアドプロシジャ，SQL ストアドファンクションのデバッグをする場合，実際にサーバ側で動作させる必要があります。これに対して，外部 C ストアドルーチンのデバッグは，クライアント側に C 言語のデバッガを用意することで，デバッグができます。

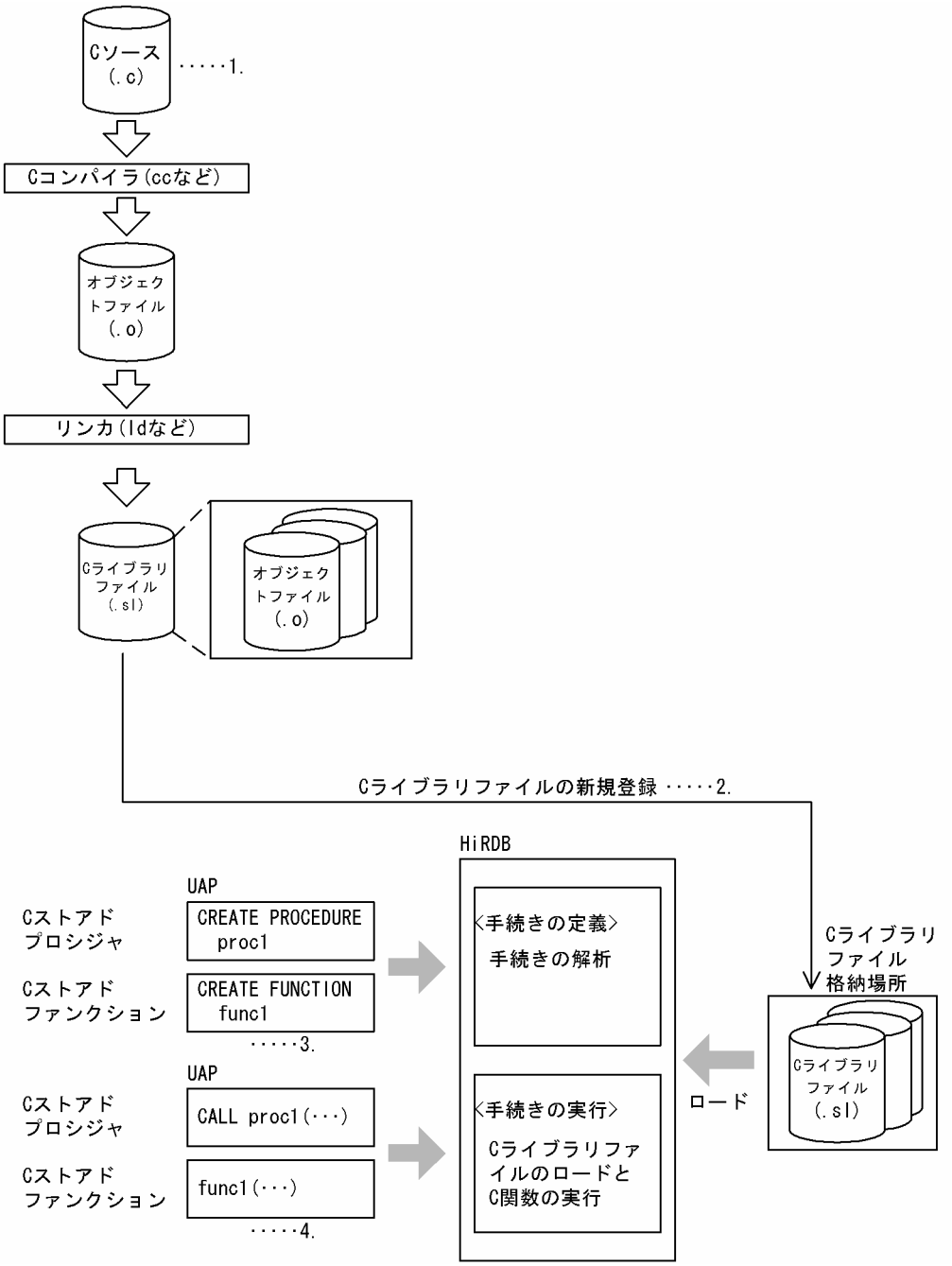
10.2　外部 C ストアドルーチンの作成から実行までの各作業

外部 C ストアドルーチンの作成から実行までの手順を次に示します。

1. 外部 C ストアドルーチンの作成
2. C ライブラリファイルの新規登録
3. 外部 C ストアドルーチンの定義
4. 外部 C ストアドルーチンの実行

外部 C ストアドルーチンの作成から実行までの流れを次の図に示します。

図 10–1　外部 C ストアドルーチンの作成から実行までの流れ



注

C ライブラリファイルの拡張子は OS によって異なります。

[説明]

1. C ストアドルーチンを作成します。詳細については，「外部 C ストアドルーチンの作成」を参照してください。
2. HiRDB に C ライブラリファイルを新規登録します。詳細については，「C ライブラリファイルの新規登録」を参照してください。

3. 外部 C ストアドルーチンを定義します。詳細については，「外部 C ストアドルーチンの定義」を参照してください。
4. 外部 C ストアドルーチンを実行します。詳細については，「外部 C ストアドルーチンの実行」を参照してください。

10.2.1　外部 C ストアドルーチンの作成

外部 C ストアドルーチンを作成する手順を次に示します。

1. C プログラムの記述（C ファイルの作成）
2. コンパイル（オブジェクトファイルの作成）
3. リンケージ（C ライブラリファイルの作成）

(1)　C プログラムの記述（C ファイルの作成）

外部 C ストアドルーチンとして登録する C プログラムを記述します。

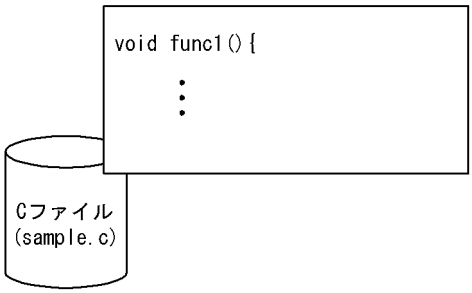
(a)　C プログラムを作成するときの規則

C プログラムを作成するときの規則を次に示します。なお，「C プログラム作成時の制限事項」も併せて参照してください。

- C 言語の既定の呼出し規約（__cdecl）を指定してください。
- 正常終了する場合は，SQLSTATE を表す出力パラメタに，00000 を設定してください。
  （例）memcpy(sqlstate, "00000", 5);
- 異常終了する場合は，SQLSTATE を表す出力パラメタに，38XYY を設定してください。X と Y は次の範囲で設定してください。
  X：I～Z
  Y：0～9 又は A～Z
  （例 1）memcpy(sqlstate, "38I01", 5);
  （例 2）memcpy(sqlstate, "38ZCB", 5);
- 異常終了する場合は，エラーの原因を意味するメッセージテキストを設定してください。

C プログラムの記述例を次の図に示します。

図 10–2　C プログラムの記述例



(b)　パラメタ入出力モードのマッピング

外部 C ストアドルーチンでの，SQL のパラメタ入出力モード（IN，OUT，又は INOUT）のマッピング例を次の図に示します。マッピングについては，マニュアル「HiRDB SQL リファレンス」の型マッピングを参照してください。

図 10–3　パラメタ入出力モードのマッピング例（外部 C ストアドルーチン）

SQL

```
CREATE PROCEDURE proc1
  ( IN    param1 INTEGER,
    OUT   param2 SMALLINT,
    INOUT param3 CHAR(5)
    INOUT param4 VARCHAR(255) )
  LANGUAGE C
  EXTERNAL NAME
 'sample.sl!cproc1'
  PARAMETER STYLE RDSQL;
```

下線部分が対応しています。

Cプログラム

```
…

/*-------------------------------------------------------*/
/* VARCHAR形式のデータを受け取るための構造体を定義する  */
/*-------------------------------------------------------*/
typedef struct varchar{
  short len;
  char str[80];
}VCH;

…

void cproc1
 ( int    *cparam1,     /* SQLパラメタparam1のデータ部 */
   short  *cparam2,     /* SQLパラメタparam2のデータ部 */
   char   *cparam3,     /* SQLパラメタparam3のデータ部 */
   char   *cparam4,     /* SQLパラメタparam4のデータ部 */
   short  *cind1,       /* SQLパラメタparam1の標識部   */
   short  *cind2,       /* SQLパラメタparam2の標識部   */
   short  *cind3,       /* SQLパラメタparam3の標識部   */
   short  *cind4,       /* SQLパラメタparam4の標識部   */
   char   *sqlstate,    /* SQLSTATE                    */
   VCH  *routine_name,  /* ルーチン名                  */
   VCH  *specific_name, /* 特定名                      */
   VCH  *message_text)  /* メッセージテキスト          */
{
        …
   /* 出力パラメタへ値を設定 */
   *cparam2 = 10;
   *cind2 = 0;
   strcpy(cparam3, “abcde”);
   *cind3 = 0;
   strcpy(cparam4, “abcde”);
   *cind4 = 0;

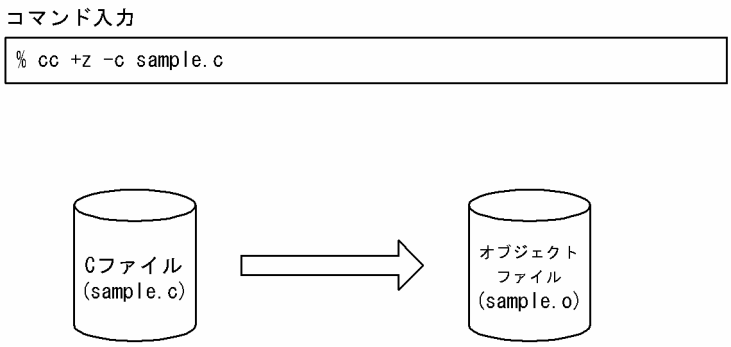
   /* SQLSTATEを設定（正常終了）*/
   memcpy(sqlstate, “00000”, 5);
   /* メッセージテキストに空文字列を設定 */
   message_text->len = sprintf(message_text->str, “”);
        …
}
```

(2)　コンパイル（オブジェクトファイルの作成）

cc コマンドなどを使用して，C ファイルからオブジェクトファイルを作成します。

コンパイルの例を次の図に示します。コンパイルオプションについては，OS のマニュアルを参照してください。

図 10–4　コンパイルの例（外部 C ストアドルーチン）



注

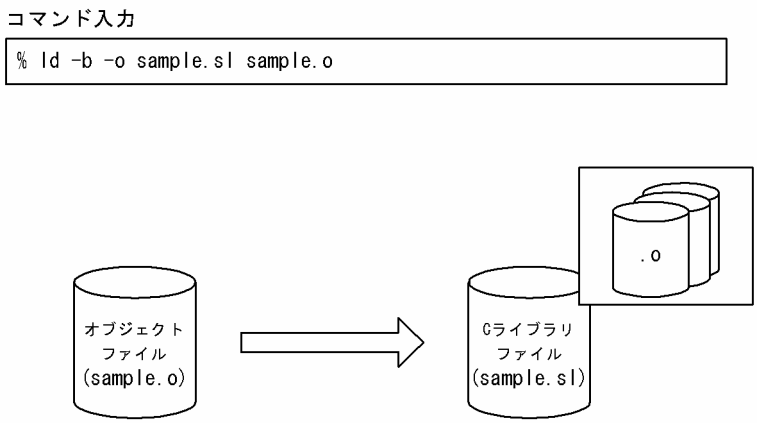
　コンパイルオプションについては，各 OS のマニュアルを参照してください。

(3)　リンケージ（C ライブラリファイルの作成）

ld コマンドなどを使用して，複数のオブジェクトファイルから C ライブラリファイルを作成します。ライブラリ関数を使用する場合は，-l オプションなどを使用して必要なライブラリをリンクしてください。

リンケージ例を次の図に示します。

図 10–5　リンケージの例



注 1

　C ライブラリファイルの拡張子は OS によって異なります。

注 2

　リンカオプションについては，各 OS のマニュアルを参照してください。

(4)　各 OS での C ライブラリファイルの作成例

C ファイル（sample.c）から C ライブラリファイルを作成する例を，OS ごとに説明します。C ライブラリファイルの作成方法は OS によって異なるため，詳細については，各 OS のマニュアルを参照してください。

なお，例ではコンパイラ及びリンカにパスが通っていることを前提とします。

(a)　Linux の場合

gcc コマンドに-shared オプションを指定して，C ライブラリファイルを作成します。sample.c から sample.so という名称の C ライブラリファイルを作成する例を次に示します。

```
$ gcc -shared -fPIC -o sample.so sample.c
```

(b)　AIX の場合

sample.c から sample.so という名称の C ライブラリファイルを作成する例を次に示します。

1. xlc コマンドでオブジェクトファイルを作成します。

   ```
   $ xlc -c -o sample.o sample.c
   ```

2. xlc コマンドに-G オプションを指定して，C ライブラリファイルを作成します。

   ```
   $ xlc -G -bexpall -o sample.so sample.o
   ```

64 ビットモードの場合：

64 ビットモードの HiRDB で使用する C ライブラリファイルは，64 ビットモードでコンパイル及びリンケージしてください。コンパイルオプションに-q64 を指定して，リンカオプションに-b64 を指定します。

POSIX ライブラリ版の場合：

64 ビットモードの HiRDB で使用する C ライブラリファイルは，マルチスレッドに対応する必要があるため，次に示す条件を満たしてください。

- コンパイル時に xlc_r コマンドを使用して C ファイルをコンパイルします。
- スレッドセーフな関数を使用します。

(c)　Windows の場合

sample.c から sample.dll という名称の C ライブラリファイル（DLL ファイル）を作成する例を次に示します。

1. cl コマンドでオブジェクトファイルを作成します。

   ```
   cl /MD /c sample.c
   ```

2. link コマンドで C ライブラリファイル（DLL ファイル）を作成します。また，モジュール定義ファイル（sample.def）を作成して，エクスポートする関数を指定します。

link /dll /def:sample.def sample.obj

注

- 作成する DLL のベースアドレス（デフォルト・ロード・アドレス）は指定しないでください。指定した場合，HiRDB やシステムの DLL とアドレスが競合して，DLL のローディング処理に負荷が掛かることがあります。
- 使用する Microsoft Visual C++ランタイムライブラリは，マルチスレッド DLL 版（/MD）でなければなりません。それ以外のライブラリを使用すると，領域管理の処理が不正となり，サーバプロセスが異常終了するおそれがあります。

10.2.2　C ライブラリファイルの新規登録

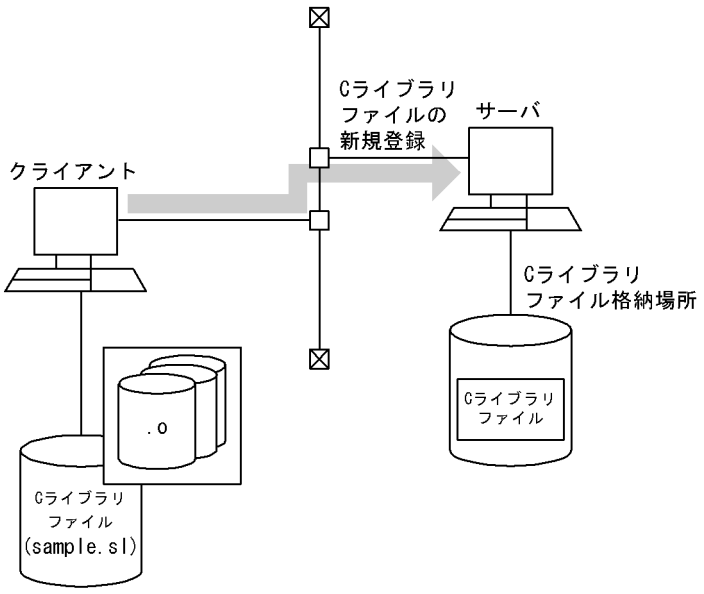
作成した C ライブラリファイルを，HiRDB サーバに新規登録（コピー）します。C ライブラリファイルの新規登録は，UAP 開発者又は HiRDB 管理者が実施します。

- **UAP 開発者が新規登録する場合**
  SQL の INSTALL CLIB を UAP に記述して実行します。
- **HiRDB 管理者が新規登録する場合**
  pdclibsync コマンド（-I オプション指定）を実行します。pdclibsync コマンドは HiRDB 管理者だけが実行できます。

なお，C ライブラリファイルは pd_c_library_directory オペランドに指定したディレクトリに格納されます。

C ライブラリファイルの登録の概要を次の図に示します。

図 10–6　C ライブラリファイルの登録の概要



参考

C ライブラリファイルを再登録する場合は，SQL の REPLACE CLIB を実行します。C ライブラリファイルを削除する場合は，SQL の REMOVE CLIB を実行します。また，HiRDB 管理者が C ライブラリファイルを再登録又は削除する場合は，pdclibsync コマンドを実行します。

なお，C ライブラリファイルの再登録及び削除は，C ライブラリファイルを新規登録したユーザ，又は HiRDB 管理者だけが実行できます。

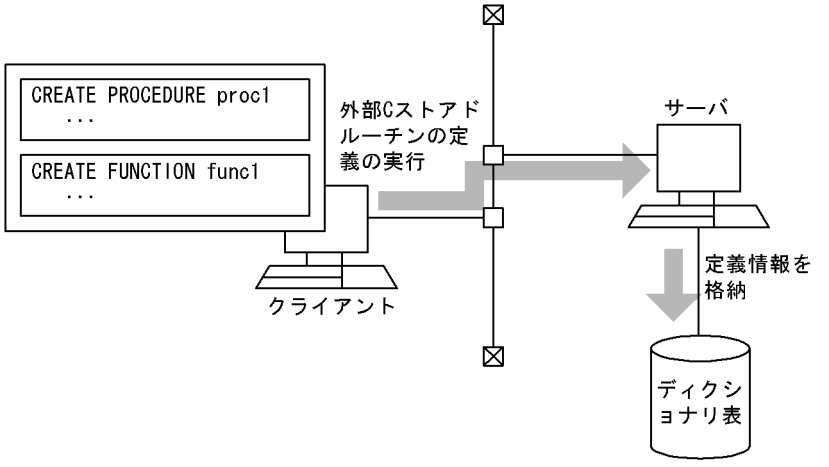
10.2.3　外部 C ストアドルーチンの定義

外部 C ストアドルーチンを定義する場合は，CREATE PROCEDURE 又は CREATE FUNCTION を使用します。CREATE PROCEDURE 又は CREATE FUNCTION で，C 関数と手続き名，又は C 関数と関数名との関連づけをします。

- **C ストアドプロシジャの場合**
  CREATE PROCEDURE を使用して，C 言語で記述した C 関数を C ストアドプロシジャとして登録します。
- **C ストアドファンクションの場合**
  CREATE FUNCTION を使用して，C 言語で記述した C 関数を C ストアドファンクションとして登録します。

外部 C ストアドルーチンの定義例を次の図に示します。

図 10–7　外部 C ストアドルーチンの定義例



パブリックルーチンの定義

他ユーザが定義した外部 C ストアドルーチンを使用する場合は，UAP 中からストアドルーチンを呼び出すときに，所有者の認可識別子とルーチン識別子を指定する必要があります。

しかし，CREATE PUBLIC PROCEDURE 又は CREATE PUBLIC FUNCTION を実行して**パブリックルーチン**として定義すると，他ユーザが定義した外部 C ストアドルーチンを使用する場合でも，UAP 中から

ストアドルーチンを呼び出すときに，所有者の認可識別子を指定する必要がなくなります（ルーチン識別子だけ指定します）。

外部 C ストアドルーチンの再定義

C プログラムの修正などによって，一度定義した外部 C ストアドルーチンを再度定義する場合は，ALTER PROCEDURE 又は ALTER ROUTINE を使用します。

外部 C ストアドルーチンの削除

外部 C ストアドルーチンを削除する場合は，DROP PROCEDURE 又は DROP FUNCTION を使用します。

また，パブリックルーチンを削除する場合は，DROP PUBLIC PROCEDURE 又は DROP PUBLIC FUNCTION を使用します。なお，パブリックルーチンを削除できるのは，パブリックルーチンを定義したユーザ，又は DBA 権限を持っているユーザだけです。

10.2.4　外部 C ストアドルーチンの実行

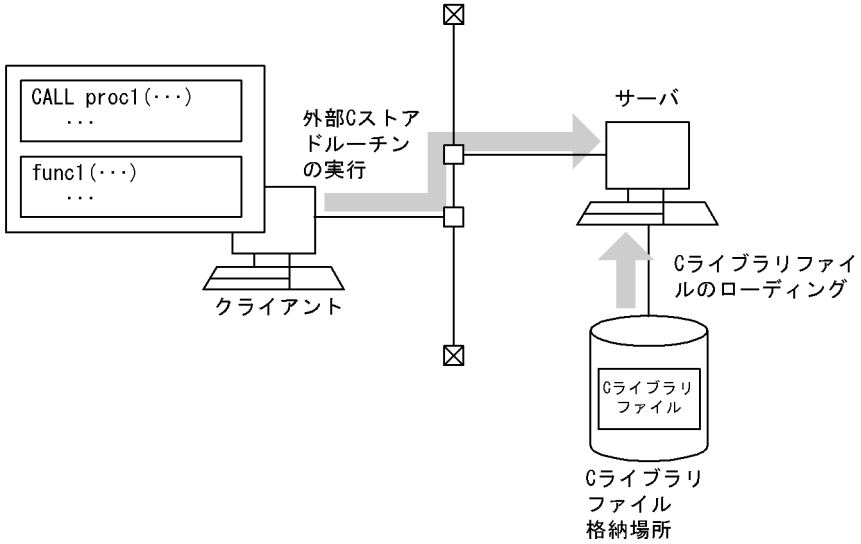
外部 C ストアドルーチンを実行する場合は，CALL 文又は関数呼出しを使用します。CALL 文又は関数呼出しを指定した SQL を実行することで，C 関数が外部 C ストアドルーチンとして呼び出され，サーバマシン上で実行されます。

- **C ストアドプロシジャの場合**
  CALL 文を使用して，C 関数を C ストアドプロシジャとして実行します。
- **C ストアドファンクションの場合**
  関数呼出しを指定した SQL を使用して，C 関数を C ストアドファンクションとして実行します。

CALL 文及び関数呼出しについては，マニュアル「HiRDB SQL リファレンス」を参照してください。

外部 C ストアドルーチンの実行例を次の図に示します。

図 10–8　外部 C ストアドルーチンの実行例



10.3　外部 C ストアドルーチンのプログラム例

実数から小数部を取得する外部 C 関数の例を次に示します。

- **C 関数本体（ファイル名：sample1.c）**

```
/* ALL RIGHTS RESERVED,COPYRIGHT (C)2007,HITACHI,LTD. */
/* LICENSED MATERIAL OF HITACHI,LTD. */
/**************************************************************************/
/* name = HiRDB 08-03 Cストアドファンクション サンプルプログラム 1     */
/**************************************************************************/
#include <math.h>
#include <stdio.h>

/*------------------------------------------------------------------------*/
/* VARCHAR形式のデータを受け取るための構造体を定義する                    */
/*------------------------------------------------------------------------*/
typedef struct varchar{
  short len;
  char str[80];
}VCH;

/**************************************************************************/
/* name = サンプル1                                                       */
/**************************************************************************/
 void func_modf(double *value, double *ret,
                short *ind1, short *ind_ret,
                char *sqlstate, VCH *routine_name,
                VCH *specific_name, VCH *message_text )
{
    double int_value;

    /* modf関数を呼出して得られた小数部を戻り値に設定 */
    *ret = modf(*value, &int_value);

    /* 戻り値の標識部を設定(非ナル) */
    *ind_ret = 0;

    /* SQLSTATEを設定（正常終了） */
    memcpy(sqlstate, “00000”, 5);

    /* メッセージテキストに空文字列を設定 */
    message_text->len = sprintf(message_text->str, “” );
}
```

上記の C 関数本体を使用して，C ストアドファンクションを定義，実行する例を次に示します。

- **C ファイルのコンパイル及び C ライブラリファイルの作成**

  Linux の場合の例です。そのほかの OS の例については，「各 OS での C ライブラリファイルの作成例」を参照してください。

```
gcc -shared -fPIC -o sample1.so sample1.c
```

- **C ライブラリファイルの新規登録**（SQL の INSTALL CLIB を使用した例です）

```
INSTALL CLIB 'sample1.sl' ;
```

- **C ストアドファンクションの定義**

```
CREATE function func_modf( parm1 FLOAT ) RETURNS FLOAT
  LANGUAGE C
  EXTERNAL NAME 'sample1.sl!func_modf'
  PARAMETER STYLE RDSQL;
```

- **C ストアドファンクションの実行**

```
select func_modf(double_value) from t1
```

10.4　C プログラム作成時の制限事項

C 言語で制御処理を記述する場合，次の制限があります。

- 次の関数は使用しないでください。**使用した場合，HiRDB の動作に深刻な影響を与えるおそれがあります。**
  - fork()，exit()，abort()，exec()などのプロセス操作関数
  - sleep()，select()，wait()
  - スタック操作関数（setjmp()，longjmp()など）
  - 共用メモリ操作関数
  - セマフォ操作関数
  - ソケット操作関数
  - システム資源操作関数(setrlimit など)
  - mmap()，munmap()
  - gethostent()，sethostent()，endhostent()，gethostbyname()，gethostbyaddr()，herror()
  - tempnam()，tmpnam()
  - pstat()
  - system()
- スレッドを作成しないでください。
- GUI は使用しないでください。
- SQL は記述できません。
- ファイルは操作しないでください。
- PIPE などのスペシャルファイルは使用しないでください。
- 標準入力，標準出力，標準エラー出力は使用しないでください。
- 関数を再起呼び出ししないでください。
- グローバル変数及び関数名には，次の名称を使用できません。
  - 大文字の「SQL」，大文字の「Y」，及び大文字の「Z」で始まる名称
  - 小文字の「p_」，小文字の「pd」，小文字の「yy」及び小文字の「z」で始まる名称
  - 小文字の「_p」で始まる名称
  - 小文字の「da」，小文字の「dbr」及び小文字の「dp」で始める名称
- 環境変数の設定及び変更はしないでください。
- シグナル操作はしないでください。
- システムの日付や時刻は変更しないでください。

- メモリを確保した場合は，ルーチンが終了するときに必ず解放してください。
- 使用するスタックサイズの最大値は 4,096 バイト以下にしてください。

11 UAP の障害対策

この章では，UAP 実行時の履歴やエラー情報を取得するトラブルシュート，及び UAP の障害の種別と回復方法について説明します。

11.1　トラブルシュート

UAP に障害が発生した場合に，トラブルシュート機能を利用して障害要因を調査できます。トラブルシュート機能には，次のものがあります。

- SQL トレース機能
- クライアントエラーログ機能
- 拡張 SQL エラー情報出力機能
- UAP 統計レポート機能
- コマンドトレース機能
- SQL トレース動的取得機能
- 再接続トレース機能
- HiRDB SQL Tuning Advisor 用アクセスパス情報ファイル

11.1.1　SQL トレース機能

実行した UAP の SQL トレース情報を SQL トレースファイルに取得します。

UAP 実行時に SQL エラーが発生した場合，SQL トレース情報を参照すると，エラーの原因となる SQL 文を特定できます。

SQL トレースファイルは，取得した情報で満杯になると，最も古い情報から順次新しい情報に書き替えられます。

(1)　SQL トレース情報の取得方法

SQL トレース情報に関するクライアント環境定義を次の表に示します。各クライアント環境定義については，「クライアント環境定義（環境変数の設定）」を参照してください。

表 11–1　SQL トレース情報に関するクライアント環境定義

種別	クライアント環境定義	説明
SQL トレース取得有無	PDSQLTRACE	SQL トレースファイルのサイズを指定する。省略した場合は SQL トレースを取得しない。
ファイル出力先	PDCLTPATH	SQL トレースファイルの格納先ディレクトリを指定する。
出力ファイル名	PDXATRCFILEMODE	X/Open に従った API を使用した接続形態での，トレースファイル名の形式を指定する。
ファイルオープン契機	PDSQLTRCOPENMODE	SQL トレースファイルのオープンモードを指定する。

種別	クライアント環境定義	説明
出力形式	PDSQLTRCFMT	SQL トレースの出力形式を指定する。
	PDPRMTRC	SQL トレースにパラメタ情報及び検索データを出力するかどうかを指定する。
	PDPRMTRCSIZE	SQL トレースに出力するパラメタ情報及び検索データの最大データ長を指定する。
	PDSQLTEXTSIZE	SQL トレースに出力する SQL 文のサイズを指定する。
	PDSQLEXECTIME	SQL トレースに SQL 実行時間を出力するかどうかを指定する。
出力契機	PDSQLTRCTYPE	SQL トレースを出力するタイミングを指定する。
	PDJDBSQLTRCMEMNUM	SQL トレースをエラー発生時に出力する場合，退避しておく SQL トレース情報の最大数を指定する。
	PDSQLTRCBUFFSIZE	SQL トレースをエラー発生時に出力する場合，SQL トレース情報を退避しておく領域のサイズを指定する。

SQL トレースの出力先及びファイル名称は，トレース取得機能及びクライアント環境定義の指定値によって異なります。SQL トレースの出力先及びファイル名称を次の表に示します。

注意事項

表 11-2，表 11-3，及び表 11-4 で，SQL トレースの出力ファイル名称が pdsql1.trc，pdsql2.trc や pd2sql1.trc，pd2sql2.trc となるクライアント環境定義を指定している場合，複数の接続から SQL を実行すると各接続の SQL トレース情報が混在します。また，複数の接続が同時に SQL を実行すると，SQL トレース情報が正しく出力されないことがあります。この場合，表 11-2，表 11-3，及び表 11-4 で SQL トレースの出力ファイル名称がコネクト要求時間やコネクト通番を含む名称となるようクライアント環境定義を指定してください。

表 11–2　SQL トレース取得機能及びクライアント環境定義の指定値による出力ファイル名称（1/3）

項番	取得機能	出力先（数字は出力先の優先順位を示す）	クライアント環境定義 PDXATRCFILEMODE	クライアント環境定義 PDSQLTRCOPENMODE※5	クライアント環境定義 PDSQLTRCFMT	クライアントライブラリ（通常ライブラリ※1）出力ファイル名称
1	クライアント環境定義 PDSQLTRACE 指定による取得	1. クライアント環境定義 PDCLTPATH で指定したディレクトリ	SEPARATE 又は指定なし	SQL 又は指定なし	1	pdsql1.trc pdsql2.trc
2					2 又は指定なし	pd2sql1.trc pd2sql2.trc

項番	取得機能	出力先（数字は出力先の優先順位を示す）	クライアント環境定義 PDXATRCFILEMODE	クライアント環境定義 PDSQLTRCOPENMODE※5	クライアント環境定義 PDSQLTRCFMT	クライアントライブラリ（通常ライブラリ※1）出力ファイル名称
3		2. PDCLTPATH 指定なしの場合はカレントディレクトリ		CNCT	1	PDSQLTRCOPENMODE 指定値に依存しないため，項番 1，2 と同様。
4					2 又は指定なし	サーバと接続できた場合： pd2sqlHHMMSSfff_XXXXXXX_YYYYYYYYY_1.trc pd2sqlHHMMSSfff_XXXXXXX_YYYYYYYYY_2.trc サーバと接続できなかった場合： pd2sql1.trc pd2sql2.trc
5			LUMP	−	1	PDXATRCFILEMODE 指定値に依存しないため，項番 1～4 と同様。
6					2 又は指定なし	
7	クライアント環境定義 PDUAPREPLVL 指定による UAP 統計レポートの取得※2	1. PDREPPATH で指定したディレクトリ 2. クライアント環境定義 PDCLTPATH で指定したディレクトリ 3. PDCLTPATH 指定なしの場合はカレントディレクトリ	−	−	1	pdHHMMSSfff_YYYYYYYY_1.trc pdHHMMSSfff_YYYYYYYY_2.trc
8					2 又は指定なし	サーバと接続できた場合： pd2sqlHHMMSSfff_XXXXXXX_YYYYYYYYY_1.trc pd2sqlHHMMSSfff_XXXXXXX_YYYYYYYYY_2.trc サーバと接続できなかった場合： pd2sql1.trc pd2sql2.trc
9	pdtrcmgr コマンド※3 による取得	クライアント環境定義 PDTRCPATH で指定したディレクトリ	−	−	1	pdHHMMSSfff_YYYYYYYY_1.trc pdHHMMSSfff_YYYYYYYY_2.trc

項番	取得機能	出力先（数字は出力先の優先順位を示す）	クライアント環境定義 PDXATRCFILEMODE	クライアント環境定義 PDSQLTRCOPENMODE※5	クライアント環境定義 PDSQLTRCFMT	クライアントライブラリ（通常ライブラリ※1）出力ファイル名称
10					2 又は指定なし	サーバと接続できた場合： pd2csqlHHMMSSfff_XXXXXXXX_YYYYYYYYYY_1.trc pd2csqlHHMMSSfff_XXXXXXXX_YYYYYYYYYY_2.trc サーバと接続できなかった場合： pd2csql1.trc pd2csql2.trc
11	pdclttrc コマンド※4 による取得	-l オプション指定なし： 1. クライアント環境定義 PDCLTPATH で指定したディレクトリ 2. PDCLTPATH 指定なしの場合はカレントディレクトリ -l オプション指定あり： 1. クライアント環境定義 PDREPPATH で指定したディレクトリ 2. PDREPPATH 指定なしの場合は PDCLTPATH で指定したディレクトリ 3. PDCLTPATH 指定なしの場合はカレントディレクトリ	−	−	1	pdXXXXXXXXyyyyyyyyyy_1.trc pdXXXXXXXXyyyyyyyyyy_2.trc
12					2 又は指定なし	pd2XXXXXXXXyyyyyyyyyy_1.trc pd2XXXXXXXXyyyyyyyyyy_2.trc

（凡例）

−：指定値に依存しない。

HHMMSSfff：コネクト要求時間（HH：時，MM：分，SS：秒，fff：ミリ秒）

XXXXXXXX：接続サーバ名

YYYYYYYYYY：コネクト通番

yyyyyyyyyy：サーバプロセス ID

注※1　ライブラリ種別の詳細は，「ライブラリ種別」を参照してください。

注※2　UAP 統計レポートの詳細は，「UAP 統計レポート機能」を参照してください。

注※3　pdtrcmgr コマンドの詳細は，「SQL トレース動的取得機能」を参照してください。

注※4　pdclttrc コマンドの詳細は，マニュアル「HiRDB コマンドリファレンス」の「pdclttrc（SQL トレースの動的取得）」を参照してください。

注※5　PDSQLTRCTYPE に ERROR を指定した場合は，競合しないようにするため，PDSQLTRCOPENMODE に CNCT を指定した場合と同様とします。

表 11–3　SQL トレース取得機能及びクライアント環境定義の指定値による出力ファイル名称（2/3）

項番	取得機能	出力先（数字は出力先の優先順位を示す）	クライアント環境定義 PDXATRCFILEMODE	クライアント環境定義 PDSQLTRCOPENMODE※5	クライアント環境定義 PDSQLTRCFMT	クライアントライブラリ（XA ライブラリ※1）出力ファイル名称
1	クライアント環境定義 PDSQLTRACE 指定による取得	1. クライアント環境定義 PDCLTPATH で指定したディレクトリ 2. PDCLTPATH 指定なしの場合はカレントディレクトリ	SEPARATE 又は指定なし	SQL 又は指定なし	1	TX_関数使用なしの場合： pdsql1.trc pdsql2.trc TX_関数使用ありの場合： pdsqlxxxxx-1.trc pdsqlxxxxx-2.trc
2					2 又は指定なし	サーバと接続できた場合： pd2sqlHHMMSSfff_XXXXXXXX_YYYYYYYYY_1.trc pd2sqlHHMMSSfff_XXXXXXXX_YYYYYYYYY_2.trc サーバと接続できなかった場合： pd2sql1.trc pd2sql2.trc
3				CNCT	1	PDSQLTRCOPENMODE 指定値に依存しないため，項番 1 と同様。
4					2 又は指定なし	サーバと接続できた場合： pd2sqlHHMMSSfff_XXXXXXXX_YYYYYYYYY_1.trc

項番	取得機能	出力先（数字は出力先の優先順位を示す）	クライアント環境定義 PDXATRCFILEMODE	クライアント環境定義 PDSQLTRCOPENMODE※5	クライアント環境定義 PDSQLTRCFMT	クライアントライブラリ（XA ライブラリ※1）出力ファイル名称
						pd2sqlHHMMSSfff_XXXXXXXX_YYYYYYYYYY_2.trc サーバと接続できなかった場合： pd2sql1.trc pd2sql2.trc
5			LUMP	−	1	pdsql1.trc pdsql2.trc
6					2 又は指定なし	pd2sql1.trc pd2sql2.trc
7	クライアント環境定義 PDUAPREPLVL 指定による UAP 統計レポートの取得※2	1. PDREPPATH で指定したディレクトリ 2. クライアント環境定義 PDCLTPATH で指定したディレクトリ 3. PDCLTPATH 指定なしの場合はカレントディレクトリ	−	−	1	pdHHMMSSfff_YYYYYYYY_1.trc pdHHMMSSfff_YYYYYYYY_2.trc
8					2 又は指定なし	サーバと接続できた場合： pd2sqlHHMMSSfff_XXXXXXXX_YYYYYYYYYY_1.trc pd2sqlHHMMSSfff_XXXXXXXX_YYYYYYYYYY_2.trc サーバと接続できなかった場合： pd2sql1.trc pd2sql2.trc
9	pdtrcmgr コマンド※3による取得	クライアント環境定義 PDTRCPATH で指定したディレクトリ	−	−	1	pdHHMMSSfff_YYYYYYYY_1.trc pdHHMMSSfff_YYYYYYYY_2.trc
10					2 又は指定なし	サーバと接続できた場合： pd2csqlHHMMSSfff_XXXXXXXX_YYYYYYYYYY_1.trc

項番	取得機能	出力先（数字は出力先の優先順位を示す）	クライアント環境定義 PDXATRCFILEMODE	クライアント環境定義 PDSQLTRCOPENMODE※5	クライアント環境定義 PDSQLTRCFMT	クライアントライブラリ（XA ライブラリ※1）出力ファイル名称
						pd2csqlHHMMSSfff_XXXXXXXX_YYYYYYYYY_2.trc サーバと接続できなかった場合： pd2csql1.trc pd2csql2.trc
11	pdclttrc コマンド※4 による取得	-l オプション指定なし： 1. クライアント環境定義 PDCLTPATH で指定したディレクトリ 2. PDCLTPATH 指定なしの場合はカレントディレクトリ -l オプション指定あり： 1. クライアント環境定義 PDREPPATH で指定したディレクトリ 2. PDREPPATH 指定なしの場合は PDCLTPATH で指定したディレクトリ 3. PDCLTPATH 指定なしの場合はカレントディレクトリ	−	−	1	pdXXXXXXXXyyyyyyyyyy_1.trc pdXXXXXXXXyyyyyyyyyy_2.trc
12					2 又は指定なし	pd2XXXXXXXXyyyyyyyyyy_1.trc pd2XXXXXXXXyyyyyyyyyy_2.trc

（凡例）

−：指定値に依存しない。

HHMMSSfff：コネクト要求時間（HH：時，MM：分，SS：秒，fff：ミリ秒）

XXXXXXXX：接続サーバ名

YYYYYYYYY：コネクト通番

xxxxx：クライアントプロセス ID

yyyyyyyyyy：サーバプロセス ID

注※1　ライブラリ種別の詳細は，「ライブラリ種別」を参照してください。

注※2　UAP 統計レポートの詳細は，「UAP 統計レポート機能」を参照してください。

注※3　pdtrcmgr コマンドの詳細は，「SQL トレース動的取得機能」を参照してください。

注※4　pdclttrc コマンドの詳細は，マニュアル「HiRDB コマンドリファレンス」の「pdclttrc（SQL トレースの動的取得）」を参照してください。

注※5　PDSQLTRCTYPE に ERROR を指定した場合は，競合しないようにするため，PDSQLTRCOPENMODE に CNCT を指定した場合と同様とします。

表 11–4　SQL トレース取得機能及びクライアント環境定義の指定値による出力ファイル名称（3/3）

項番	取得機能	出力先（数字は出力先の優先順位を示す）	クライアント環境定義 PDXATRCFILEMODE	クライアント環境定義 PDSQLTRCOPENMODE※4	クライアント環境定義 PDSQLTRCFMT	Type4 JDBC ドライバ出力ファイル名称
1	クライアント環境定義 PDSQLTRACE 指定による取得	1. クライアント環境定義 PDCLTPATH で指定したディレクトリ 2. PDCLTPATH 指定なしの場合はカレントディレクトリ	SEPARATE 又は指定なし	SQL 又は指定なし	1	PDSQLTRCOPENMODE 指定値に依存しないため，項番 3，4 と同様。
2					2 又は指定なし	
3				CNCT	1	サーバと接続できた場合： • クライアント環境定義 PDCONTYPE に PASSIVE を指定した場合 pdjsqlXXXXXXXX_ppppp_1.trc pdjsqlXXXXXXXX_ppppp_2.trc • クライアント環境定義 PDCONTYPE に ACTIVE を指定した場合 pdjsqlHHMMSSfff_XXXXXXXX_YYYYYYYYY_1.trc pdjsqlHHMMSSfff_XXXXXXXX_YYYYYYYYY_2.trc サーバと接続できなかった場合： pdjsql1.trc pdjsql2.trc
4					2 又は指定なし	サーバと接続できた場合：

項番	取得機能	出力先（数字は出力先の優先順位を示す）	クライアント環境定義 PDXATRCFILEMODE	クライアント環境定義 PDSQLTRCOPENMODE※4	クライアント環境定義 PDSQLTRCFMT	Type4 JDBC ドライバ出力ファイル名称
						pd2jsqlHHMMSSfff_XXXXXXXX_YYYYYYYYY_1.trc pd2jsqlHHMMSSfff_XXXXXXXX_YYYYYYYYY_2.trc サーバと接続できなかった場合： pd2jsql1.trc pd2jsql2.trc
5			LUMP	−	1	PDXATRCFILEMODE 指定値に依存しないため，項番 1～4 と同様。
6					2 又は指定なし	
7	クライアント環境定義 PDUAPREPLVL 指定による UAP 統計レポートの取得※1	1. PDREPPATH で指定したディレクトリ 2. クライアント環境定義 PDCLTPATH で指定したディレクトリ 3. PDCLTPATH 指定なしの場合はカレントディレクトリ	−	−	1	サーバと接続できた場合： • クライアント環境定義 PDCONTYPE に PASSIVE を指定した場合 pdjsqlXXXXXXXX_ppppp_1.trc pdjsqlXXXXXXXX_ppppp_2.trc • クライアント環境定義 PDCONTYPE に ACTIVE を指定した場合 pdjsqlHHMMSSfff_XXXXXXXX_YYYYYYYYY_1.trc pdjsqlHHMMSSfff_XXXXXXXX_YYYYYYYYY_2.trc サーバと接続できなかった場合： pdjsql1.trc

項番	取得機能	出力先（数字は出力先の優先順位を示す）	クライアント環境定義 PDXATRCFILEMODE	クライアント環境定義 PDSQLTRCOPENMODE※4	クライアント環境定義 PDSQLTRCFMT	Type4 JDBC ドライバ出力ファイル名称
						pdjsql2.trc
8					2 又は指定なし	サーバと接続できた場合： pd2jsqlHHMMSSfff_XXXXXXXX_YYYYYYYYY_1.trc pd2jsqlHHMMSSfff_XXXXXXXX_YYYYYYYYY_2.trc サーバと接続できなかった場合： pd2jsql1.trc pd2jsql2.trc
9	pdtrcmgr コマンド※2 による取得	クライアント環境定義 PDTRCPATH で指定したディレクトリ	−	−	1	サーバと接続できた場合： • クライアント環境定義 PDCONTYPE に PASSIVE を指定した場合 pdjsqlXXXXXXXX_ppppp_1.trc pdjsqlXXXXXXXX_ppppp_2.trc • クライアント環境定義 PDCONTYPE に ACTIVE を指定した場合 pdjsqlHHMMSSfff_XXXXXXXX_YYYYYYYYY_1.trc pdjsqlHHMMSSfff_XXXXXXXX_YYYYYYYYY_2.trc サーバと接続できなかった場合： pdjsql1.trc pdjsql2.trc

項番	取得機能	出力先（数字は出力先の優先順位を示す）	クライアント環境定義 PDXATRCFILEMODE	クライアント環境定義 PDSQLTRCOPENMODE※4	クライアント環境定義 PDSQLTRCFMT	Type4 JDBC ドライバ出力ファイル名称
10					2 又は指定なし	サーバと接続できた場合： pd2jcsqlHHMMSSfff_XXXXXXXX_YYYYYYYYY_1.trc pd2jcsqlHHMMSSfff_XXXXXXXX_YYYYYYYYY_2.trc サーバと接続できなかった場合： pd2jcsql1.trc pd2jcsql2.trc
11	pdclttrc コマンド※3 による取得	-l オプション指定なし： 1. クライアント環境定義 PDCLTPATH で指定したディレクトリ 2. PDCLTPATH 指定なしの場合はカレントディレクトリ -l オプション指定あり： 1. クライアント環境定義 PDREPPATH で指定したディレクトリ 2. PDREPPATH 指定なしの場合は PDCLTPATH で指定したディレクトリ 3. PDCLTPATH 指定なしの場合はカレントディレクトリ	−	−	1	クライアント環境定義 PDCONTYPE に PASSIVE を指定した場合： pdjsqlXXXXXXXX_ppppp_1.trc pdjsqlXXXXXXXX_ppppp_2.trc クライアント環境定義 PDCONTYPE に ACTIVE を指定した場合： pdjsqlHHMMSSfff_XXXXXXXX_YYYYYYYYY_1.trc pdjsqlHHMMSSfff_XXXXXXXX_YYYYYYYYY_2.trc
12					2 又は指定なし	pd2jXXXXXXXXyyyyyyyyyy_1.trc pd2jXXXXXXXXyyyyyyyyyy_2.trc

（凡例）

−：指定値に依存しない。

HHMMSSfff：コネクト要求時間（HH：時，MM：分，SS：秒，fff：ミリ秒）

XXXXXXXX：接続サーバ名

YYYYYYYYY：コネクト通番

yyyyyyyyyy：サーバプロセス ID

ppppp：クライアント受信ポート番号

注※1　UAP 統計レポートの詳細は，「UAP 統計レポート機能」を参照してください。

注※2　pdtrcmgr コマンドの詳細は，「SQL トレース動的取得機能」を参照してください。

注※3　pdclttrc コマンドの詳細は，マニュアル「HiRDB コマンドリファレンス」の「pdclttrc（SQL トレースの動的取得）」を参照してください。

注※4　PDSQLTRCTYPE に ERROR を指定した場合は，競合しないようにするため，PDSQLTRCOPENMODE に CNCT を指定した場合と同様とします。

ライブラリ種別

ライブラリ種別を次に示します。

ライブラリ種別	UNIX 環境のライブラリ名※1	Windows 環境のライブラリ名
通常ライブラリ	libzclt.sl，ibzclt64.sl，libzclts.sl，libzclts64.sl，libzcltk.sl，libzcltk64.sl，libzcltm.sl，libclt.a，libclt64.a，libclts.a，libclts64.a，libcltk.a，libcltk64.a，libcltm.a，libzclt6k.a，libzclt6k64.a	cltdll.dll，pdcltm32.dll，pdcltp32.dll，pdcltm50.dll，pdcltm71.dll，pdcltm80s.dll，pdcltm90s.dll，pdcltm90.dll，pdcltm100.dll，pdcltm64.dll，pdcltm90s64.dll，pdcltm90x.dll，pdcltm100x.dll，pdcltl32.dll，pdcltm140.dll，pdcltm140x.dll
シングルスレッド版 XA ライブラリ	libzcltx.sl，libzclty.sl，libzcltx64.sl，libzclty64.sl，libzcltxs.sl，libzcltys.sl，libcltxa.a，libcltya.a，libcltxas.a，libcltyas.a，libzclt6ys.a，libzclt6ys64.a	pdcltx32.dll，pdcltxs.dll，pdcltx64.dll，pdcltxs64.dll
マルチスレッド版 XA ライブラリ	libzcltxk.sl，libzcltyk.sl，libzcltxk64.sl，libzcltyk64.sl，libcltxak.a，libclttyak.a，libzclt6yk.a	pdcltxm5.dll，pdcltxm64.dll，pdclto32.dll，pdclto64.dll

注※1

共用ライブラリのサフィックスは，プラットフォームによって異なります。Linux の場合は「.so」，AIX の場合は「.a」となります。

(2)　SQL トレースの解析

HiRDB SQL Tuning Advisor の SQL トレース解析機能では，次のような解析処理ができるため性能上の問題点を特定するのに便利です。

- SQL トレースの SQL 開始時間と SQL 終了時間を基に実行時間を算出し，表示します。
- 実行時間順に表示できるため，時間の掛かっている SQL が容易に特定できます。
- 複数のトランザクションを同時に実行して SQL トレースを取得した場合でも，トランザクションごと，又は SQL ごとにトレース情報を表示できます。

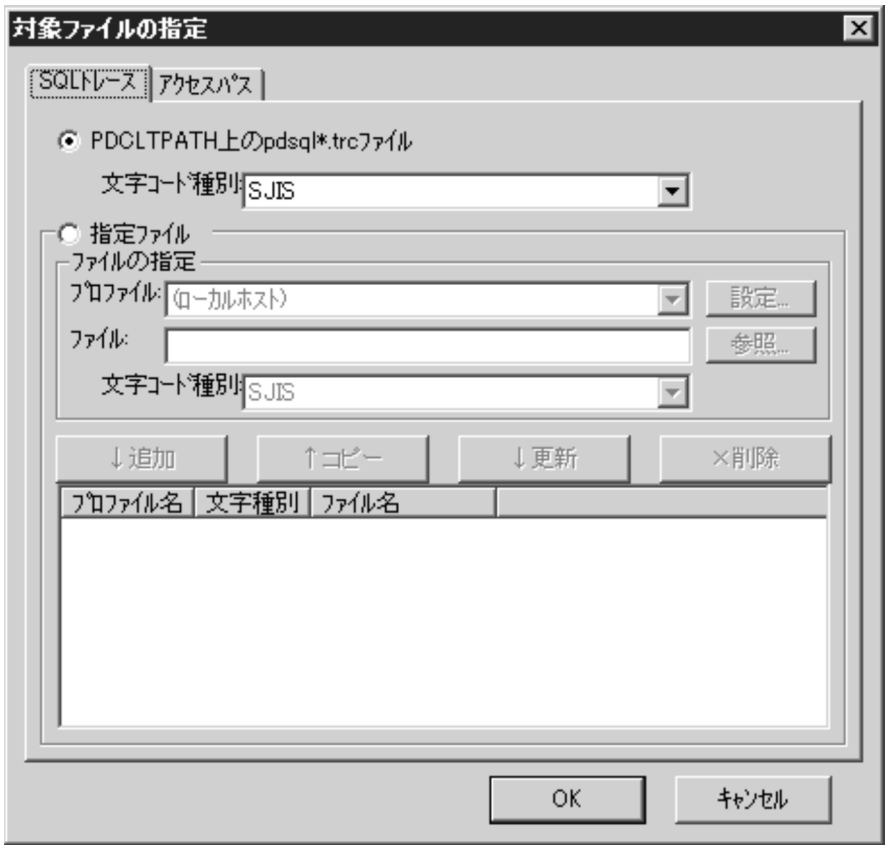
SQL トレースを解析する手順を次に示します。

[手順]

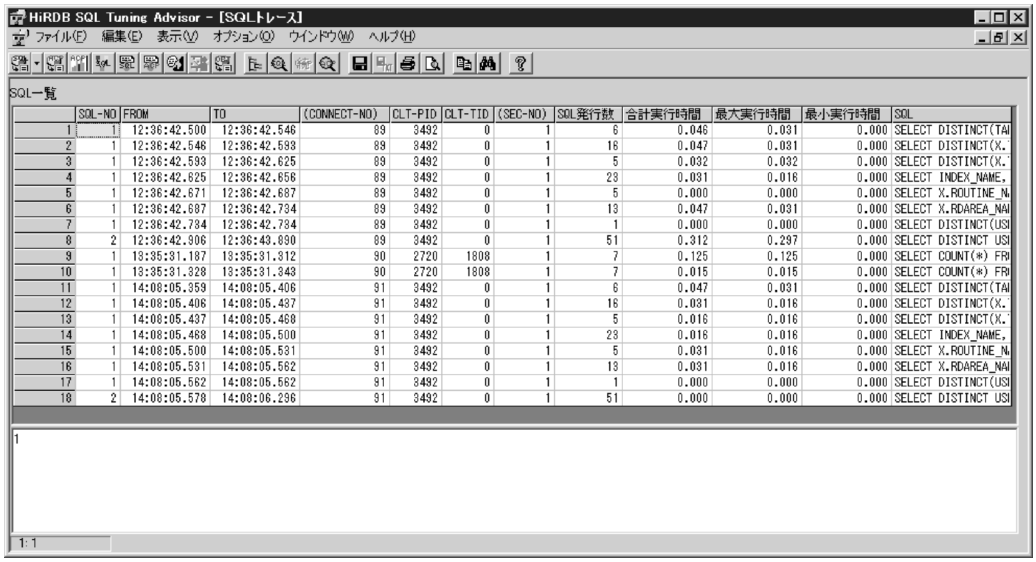
1. ［スタート］−［プログラム］−［HiRDB SQL Tuning Advisor］−［HiRDB SQL Tuning Advisor］を選択し，HiRDB SQL Tuning Advisor を起動します。

2. ［オプション］メニューから［対象ファイル指定］を選択します。
   ［対象ファイルの指定］画面が表示されます。

3. ［SQL トレース］タブで SQL トレースのファイルを指定します。クライアント環境変数 PDCLTPATH で指定したディレクトリ下のすべての SQL トレースを解析する場合は，［PDCLTPATH 上の pdsql*.trc ファイル］を選択します。それ以外の場合は，［指定ファイル］を選択し，SQL トレースファイル名を入力します。設定後，［OK］ボタンをクリックします。

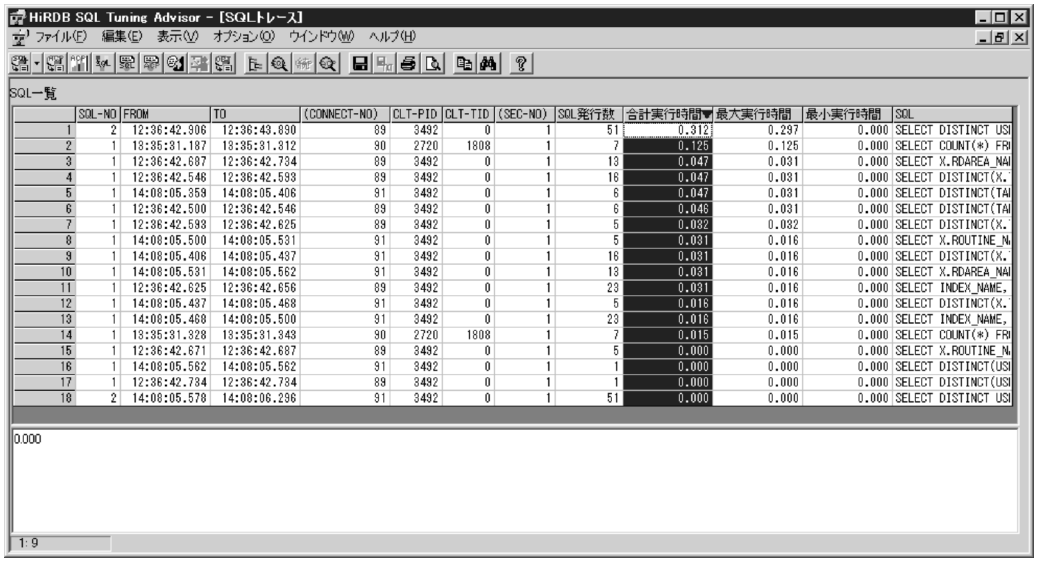


4. ［SQL トレース］ボタンから［SQL 一覧］をクリックします。
   各 SQL の情報が時系列順に表示されます。



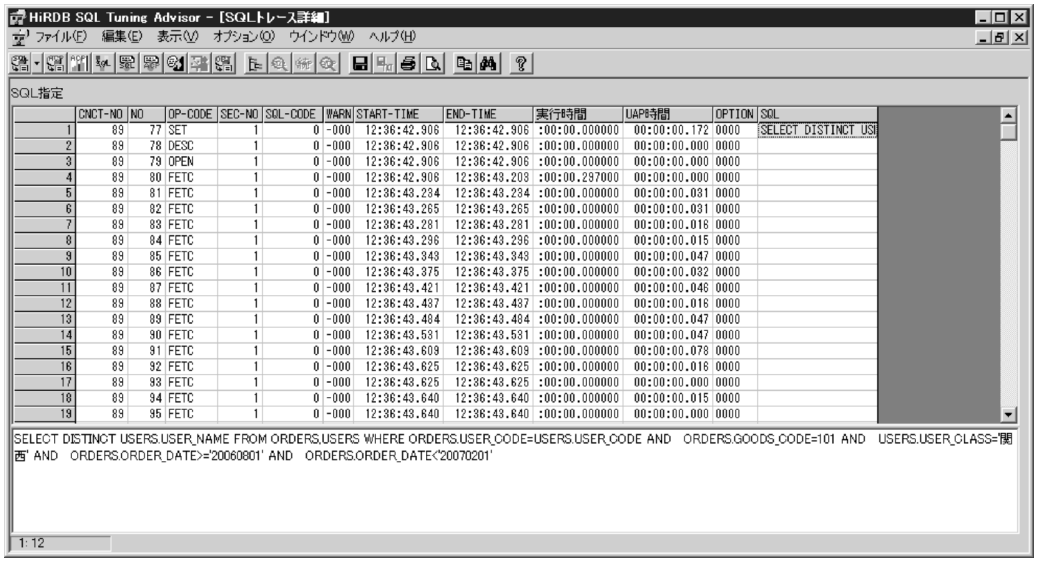
	SQL-NO	FROM	TO	(CONNECT-NO)	CLT-PID	CLT-TID	(SEC-NO)	SQL発行数	合計実行時間	最大実行時間	最小実行時間	SQL
1	1	12:36:42.500	12:36:42.546	89	3492	0	1	6	0.046	0.031	0.000	SELECT DISTINCT(TAI
2	1	12:36:42.546	12:36:42.593	89	3492	0	1	16	0.047	0.031	0.000	SELECT DISTINCT(X.
3	1	12:36:42.593	12:36:42.625	89	3492	0	1	5	0.032	0.032	0.000	SELECT DISTINCT(X.
4	1	12:36:42.625	12:36:42.656	89	3492	0	1	23	0.031	0.016	0.000	SELECT INDEX_NAME,
5	1	12:36:42.671	12:36:42.687	89	3492	0	1	5	0.000	0.000	0.000	SELECT X.ROUTINE_N
6	1	12:36:42.687	12:36:42.734	89	3492	0	1	13	0.047	0.031	0.000	SELECT X.RDAREA_NAI
7	1	12:36:42.734	12:36:42.734	89	3492	0	1	1	0.000	0.000	0.000	SELECT DISTINCT(USI
8	2	12:36:42.906	12:36:43.890	89	3492	0	1	51	0.312	0.297	0.000	SELECT DISTINCT USI
9	1	13:35:31.187	13:35:31.312	90	2720	1808	1	7	0.125	0.125	0.000	SELECT COUNT(*) FRO
10	1	13:35:31.328	13:35:31.343	90	2720	1808	1	7	0.015	0.015	0.000	SELECT COUNT(*) FRO
11	1	14:08:05.359	14:08:05.406	91	3492	0	1	6	0.047	0.031	0.000	SELECT DISTINCT(TAI
12	1	14:08:05.406	14:08:05.437	91	3492	0	1	16	0.031	0.016	0.000	SELECT DISTINCT(X.
13	1	14:08:05.437	14:08:05.468	91	3492	0	1	5	0.016	0.016	0.000	SELECT DISTINCT(X.
14	1	14:08:05.468	14:08:05.500	91	3492	0	1	23	0.016	0.016	0.000	SELECT INDEX_NAME,
15	1	14:08:05.500	14:08:05.531	91	3492	0	1	5	0.031	0.016	0.000	SELECT X.ROUTINE_N
16	1	14:08:05.531	14:08:05.562	91	3492	0	1	13	0.031	0.016	0.000	SELECT X.RDAREA_NAI
17	1	14:08:05.562	14:08:05.562	91	3492	0	1	1	0.000	0.000	0.000	SELECT DISTINCT(USI
18	2	14:08:05.578	14:08:06.296	91	3492	0	1	51	0.000	0.000	0.000	SELECT DISTINCT USI

- ［合計実行時間］の列をクリックすると，各 SQL の情報が実行時間の値順に表示されます。



	SQL-NO	FROM	TO	(CONNECT-NO)	CLT-PID	CLT-TID	(SEC-NO)	SQL発行数	合計実行時間▼	最大実行時間	最小実行時間	SQL
1	2	12:36:42.906	12:36:43.890	89	3492	0	1	51	0.312	0.297	0.000	SELECT DISTINCT USI
2	1	13:35:31.187	13:35:31.312	90	2720	1808	1	7	0.125	0.125	0.000	SELECT COUNT(*) FRO
3	1	12:36:42.687	12:36:42.734	89	3492	0	1	13	0.047	0.031	0.000	SELECT X.RDAREA_NAI
4	1	12:36:42.546	12:36:42.593	89	3492	0	1	16	0.047	0.031	0.000	SELECT DISTINCT(X.
5	1	14:08:05.359	14:08:05.406	91	3492	0	1	6	0.047	0.031	0.000	SELECT DISTINCT(TAI
6	1	12:36:42.500	12:36:42.546	89	3492	0	1	6	0.046	0.031	0.000	SELECT DISTINCT(TAI
7	1	12:36:42.593	12:36:42.625	89	3492	0	1	5	0.032	0.032	0.000	SELECT DISTINCT(X.
8	1	14:08:05.500	14:08:05.531	91	3492	0	1	5	0.031	0.016	0.000	SELECT X.ROUTINE_N
9	1	14:08:05.406	14:08:05.437	91	3492	0	1	16	0.031	0.016	0.000	SELECT DISTINCT(X.
10	1	14:08:05.531	14:08:05.562	91	3492	0	1	13	0.031	0.016	0.000	SELECT X.RDAREA_NAI
11	1	12:36:42.625	12:36:42.656	89	3492	0	1	23	0.031	0.016	0.000	SELECT INDEX_NAME,
12	1	14:08:05.437	14:08:05.468	91	3492	0	1	5	0.016	0.016	0.000	SELECT DISTINCT(X.
13	1	14:08:05.468	14:08:05.500	91	3492	0	1	23	0.016	0.016	0.000	SELECT INDEX_NAME,
14	1	13:35:31.328	13:35:31.343	90	2720	1808	1	7	0.015	0.015	0.000	SELECT COUNT(*) FRO
15	1	12:36:42.671	12:36:42.687	89	3492	0	1	5	0.000	0.000	0.000	SELECT X.ROUTINE_N
16	1	14:08:05.562	14:08:05.562	91	3492	0	1	1	0.000	0.000	0.000	SELECT DISTINCT(USI
17	1	12:36:42.734	12:36:42.734	89	3492	0	1	1	0.000	0.000	0.000	SELECT DISTINCT(USI
18	2	14:08:05.578	14:08:06.296	91	3492	0	1	51	0.000	0.000	0.000	SELECT DISTINCT USI

- 行をダブルクリックすると，［SQL トレース詳細］画面が表示されます。ここでは，各 SQL のオペレーションの情報が表示されます。



(3) SQL トレース情報の見方

SQL トレース情報は，SQL 文の実行終了時に出力されます。

出力される SQL トレース情報の例とその説明を次に示します。

[クライアント環境定義 PDSQLTRCFMT に 1 を指定した出力例（出力形式 1)]

```
                                [25]                    [19] [22]
** UAP TRACE (CLT:VV-RR(Mmm dd yyyy) SVR:VV-RR US) WIN32(WIN32) **

  USER APPLICATION PROGRAM FILE NAME : XXXXXXXX [1]
  USERID : YYYYYYYY [2]
  UAP START TIME : YYYY/MM/DD HH:MM:SS [3]
  UAP ENVIRONMENT : [4]
    LANG(ja_JP.SJIS)
    USER("hirdb")
    HOST(h9000vr5)
    NAMEPORT(20281)
    FESHOST()
    SVCGRP()  SVCPORT()  SRVTYPE()
    SWAIT(600)  CWAIT(0)  SWATCH(0)
    BLKF(1)  RDABLKF(-1)  LCKLMT(0)  ISLLVL(2)  DBLOG(ALL)  DFLNVAL(NOUSE)
    AGGR(1024)  DLKPRIO(64)  EXWARN(NO)  VWOPTMODE(0)
    LOCKSKIP(NO) CLTGRP(A)  DSQLOBJCACHE(YES)  PLGIXMK(NO)
    CLTRCVPORT(5000)  CLTRCVADDR(192.134.35.4)  PLGPFSZ(8192)
    PLGPFSZEXP(8192)  SPACELVL(-1)  STJTRNOUT()
    OPTLVL("SELECT_APSL","RAPID_GROPING")
    ADDITIONALOPTLVL("COST_BASE_2","APPLY_HASH_JOIN")
    UAPREPLVL()  REPPATH()
    TRCPATH()
    IPC(MEMORY)  SENDMEMSIZE(16)  RECVMEMSIZE(32)
    HASHTBLSIZE(128)  CMMTBFDDL(NO)  PRPCRCLS( )
    SQLTRCOPENMODE(SQL)  AUTOCONNECT(ON)  CWAITTIMEWRNPNT(-1)  TCPCONOPT(0)
    WRTLNFILSZ(-1) WRTLNCOMSZ(1024)
    WRTLNPATH( ) UAPENVFILE( )
    TP1SERVICE(NO) AUTORECONNECT(NO) RCCOUNT(0) RCINTERVAL(0)
    KALVL(0) KATIME(0) CLTCNVMODE(NOUSE)
```

```
     PRMTRC(YES) PRMTRCSIZE(256) BESCONHOLD() BESCONHTI(-1)
     BLKBUFFSIZE(0) BINARYBLKF(NO) FORUPDATEEXLOCK(NO)
     CNSTRNTNAME() SQLTEXTSIZE(4096) RCTRACE(-1)
     FESGRP()
     NBLOCKWAITTIME(0) CONNECTWAITTIME(300) DBBUFLRU(YES)
     UAPEXERLOGUSE() UAPEXERLOGPRMSZ() HJHASHINGMODE(TYPE1)
     DDLDEAPRP(NO) DELRSVWDFILE() HATRNQUEUING()
     ODBSPLITSIZE(100) NODELAYACK(NO) CURSORLVL(0)
     TAAPINFPATH() TAAPINFMODE(0) TAAPINFSIZE(409600)
     JETCOMPATIBLE(NO) SUBSTRLEN() BLKFUPD() ARYERRPOS()
     CALCMDWAITTIME(0) BLKFERRBREAK(NO) XAAUTORECONNECT(NO)
     CLTBINDLOOPBACKADDR(NO)
     STANDARDSQLSTATE() LCKWAITTIME(-1) DDLDEAPRPEXE(NO)
     ODBGINFOSUPPRESS(NO) ODBSTANDARDARGSIZE(YES)
     ODBSTANDARDSQLSTATE(YES) ODBSTANDARDDESCCOL(YES)
     TMPTBLRDAREA(RDTMP01,RDTMP02,RDTMP03)
     EXTDECCHECK(NO)
     DNDPCOMPATIBLE(NO)
     RCTIMING(ALL)
     ODBSTANDARDGTYPEINFO(NO)
     HSICOPTIONS()
     DEFAULTOPTION(RECOM)
     TCPREDUCETIMEWAITPORT(YES) TCPREDUCECONCLOSERCVTIME(1000)
     AUTHTYPE(PA)
     CLTCNVBYTERATIO() CLTCNVUOCFUNC()
     CLTCNVUOCLIB()
     TIMEDOUTRETRY(2) BINDRETRYCOUNT(10) BINDRETRYINTERVAL(0)
     ODBCWRNSKIP(NO) ODBLOCATOR(NO) ODBESCAPE(1)
     ODBSTATCACHE(0) GDATAOPT(NO) DNDPTRACE()
     CLTRDNODE() RDCLTCODE() UAPERLOG(65536)
     ERRSKIPCODE()
     TRCMODE(ERR) XATRCFILEMODE(SEPARATE)
     RETRYCOUNT(10) RETRYINTERVAL(700)
     CONREFRCOUNT(9) CONREFRINTERVAL(700)
     IPCFILEDIR()
     CONTYPE(PASSIVE)
     SOCKETCRYPTO(NO)
     ODBERRTRC(YES) ODBERRTRCSIZE(65535)
     SQLTRCTYPE(SQL) JDBSQLTRCMEMNUM() SQLTRCBUFFSIZE(50)
     ENVGROUP(/USERS/hirdb/envgroup/HiRDB.ini)
   CONNECTION STATUS : [5]
     CURHOST(dcm3500)  CURPORT(4439)  SRVNAME(fes1)
     CNCTNO(1)  SVRPID(8945)  CLTPID(9155)  CLTTID( ) CLTCNCTHDL(0x0)

 [6]  [7]   [8]  [9]     [10] [11] [12] [13]  [14]         [15]         [16] [23]
 CNCT CLPID CLTID NO     OP   SEC  SQL  SQL   START-TIME   END-TIME     OP   EXEC-TIME
 NO                      CODE NO   CODE WARN                          TION
 ---- ----- ----- -- ---- ---  ---- ----- ------------ ------------ ---- -----------
   1  9155      0      1 CNCT   0     0 WC040 16:03:55.720 16:03:58.080 0001     2356125
   1  9155      0      2 AUI2   1     0 -0000 16:03:58.630 16:03:59.400 M000      769651

*SQL* INSERT INTO ZAIKO(GNO,GNAME,KIKAKU,TANKA,SURYO,GENKA) VALUES(?,?,   .........17
?,?,?,?,?)   ..................................................................17
   1  9155      0      3 SET    2     0 -0000 16:04:00.820 16:04:01.540 M000      719825

*SQL* SELECT GNO,GNAME,KIKAKU,TANKA,SURYO,GENKA FROM ZAIKO   ......................17

   1  9155      0      4 OPEN   2     0 -0000 16:04:02.090 16:04:02.800 M000      709123
```

```
   1  9155      0      5 FETC   2  -204 -0000 16:04:03.080 16:04:03.790 M000      708902
   1  9155      0      6 SET    2     0 W8800 16:04:04.060 16:04:04.830 M000      765147

*SQL(AUTHID)* INSERT INTO TBL01 VALUES('12345',12345)   .........................17

   1  9155      0      7 SAUT   0     0 -0000 16:04:04.834 16:04:04.835 M000         912

*USER* hirdb01   ............................................................18

   1  9155      0      8 AUI2   3     0 -0000 16:05:05.110 16:05:05.121 M000        9456

*SQL* INSERT INTO TBL01 VALUES(?,100)   ..........................................21
*PARAM* NO=       1 COD=c5 XDIM=   1 SYS=     0 LEN=      15 IND=       0   .......21
        DATA=30 35 2d 30 35 00 00 00 00 00 00 00 00 00     *05-05.........*   ..21

   1  9155             9 DISC   0     0 -0000 16:05:55.110 16:05:56.660 M004     1547893
```

[クライアント環境定義 PDSQLTRCFMT に 2 を指定した出力例（出力形式 2）(1/2)]

```
              [24]       [25]    [20]                            [19]  [22]
 ** UAP TRACE TYPE2 (CLT:09-50-01(Nov  5 2013) SVR:09-04   US) WIN32(WIN_M32)**↓

 USER APPLICATION PROGRAM FILE NAME : TESTAP↓    [1]
 USERID : hirdb↓    [2]
 UAP START TIME : 2013/11/13 15:20:14↓    [3]
 UAP ENVIRONMENT :↓    [4]
   LANG(ja_JP.SJIS)↓
   USER("hirdb")↓
   HOST(eserver2)↓
   NAMEPORT(20280)↓
   FESHOST()↓
   SVCGRP()  SVCPORT()  SRVTYPE()↓
   SWAIT(600)  CWAIT(0)  SWATCH(0)↓
   BLKF(1)  RDABLKF(-1)  LCKLMT(0)  ISLLVL(2)  DBLOG(ALL)  DFLNVAL(NOUSE)↓
   AGGR(1024)  DLKPRIO(64)  EXWARN(NO)  VWOPTMODE(0)↓
   LOCKSKIP(NO) CLTGRP(A)  DSQLOBJCACHE(YES)  PLGIXMK(NO)↓
   CLTRCVPORT(5000)  CLTRCVADDR()  PLGPFSZ(8192)↓
   PLGPFSZEXP(8192)  SPACELVL(-1)  STJTRNOUT(YES)↓
   OPTLVL("SELECT_APSL","RAPID_GROPING")↓
   ADDITIONALOPTLVL("COST_BASE_2","APPLY_HASH_JOIN")↓
   UAPREPLVL()  REPPATH()↓
   TRCPATH()↓
   IPC(MEMORY)  SENDMEMSIZE(16)  RECVMEMSIZE(32)↓
   HASHTBLSIZE(128)  CMMTBFDDL(NO)  PRPCRCLS()  DBACCS()↓
   SQLTRCOPENMODE(SQL)  AUTOCONNECT(ON)  CWAITTIMEWRNPNT(-1)  TCPCONOPT(1)↓
   WRTLNFILSZ(-1) WRTLNCOMSZ(1024)↓
   WRTLNPATH() UAPENVFILE()↓
   TP1SERVICE(NO) AUTORECONNECT(NO) RCCOUNT(0) RCINTERVAL(0)↓
   KALVL(2) KATIME(600) CLTCNVMODE(NOUSE)↓
   PRMTRC(YES) PRMTRCSIZE(256) BESCONHOLD() BESCONHTI(-1)↓
   BLKBUFFSIZE(10) BINARYBLKF(NO) FORUPDATEEXLOCK(YES)  DBORGUAP(NO)↓
   CNSTRNTNAME() SQLTEXTSIZE(4096) RCTRACE(-1)↓
   FESGRP()↓
   NBLOCKWAITTIME(0) CONNECTWAITTIME(300) DBBUFLRU(YES)↓
   UAPEXERLOGUSE() UAPEXERLOGPRMSZ() HJHASHINGMODE(TYPE1)↓
   DDLDEAPRP(NO) DELRSVWDFILE() HATRNQUEUING()↓
   ODBSPLITSIZE(100) NODELAYACK(YES) CURSORLVL(0)↓
```

```
    TAAPINFPATH() TAAPINFMODE(0) TAAPINFSIZE(409600)↓
    JETCOMPATIBLE(NO) SUBSTRLEN() BLKFUPD(YES) ARYERRPOS(YES)↓
    CALCMDWAITTIME(0) BLKFERRBREAK(YES) XAAUTORECONNECT(NO)↓
    CLTBINDLOOPBACKADDR(NO)↓
    STANDARDSQLSTATE() LCKWAITTIME(-1) DDLDEAPRPEXE(NO)↓
    ODBGINFOSUPPRESS(NO) ODBSTANDARDARGSIZE(YES)↓
    ODBSTANDARDSQLSTATE(YES) ODBSTANDARDDESCCOL(YES)↓
    TMPTBLRDAREA(RDTMP01,RDTMP02,RDTMP03)↓
    EXTDECCHECK(NO)↓
    DNDPCOMPATIBLE(NO)↓
    RCTIMING(ALL)↓
    ODBSTANDARDGTYPEINFO(NO)↓
    HSICOPTIONS()↓
    DEFAULTOPTION(RECOM)↓
    TCPREDUCETIMEWAITPORT(YES) TCPREDUCECONCLOSERCVTIME(1000)↓
    AUTHTYPE(PA)↓
    CLTCNVBYTERATIO() CLTCNVUOCFUNC()↓
    CLTCNVUOCLIB()↓
    TIMEDOUTRETRY(2) BINDRETRYCOUNT(10) BINDRETRYINTERVAL(0)↓
    ODBCWRNSKIP(NO) ODBLOCATOR(NO) ODBESCAPE(1)↓
    ODBSTATCACHE(0) GDATAOPT(NO) DNDPTRACE()↓
    CLTRDNODE() RDCLTCODE() UAPERLOG(65536)↓
    ERRSKIPCODE()↓
    TRCMODE(ERR) XATRCFILEMODE(SEPARATE)↓
    RETRYCOUNT(10) RETRYINTERVAL(700)↓
    CONREFRCOUNT(9) CONREFRINTERVAL(700)↓
    IPCFILEDIR()↓
    CONTYPE(PASSIVE)↓
    SOCKETCRYPTO(NO)↓
    ODBERRTRC(YES) ODBERRTRCSIZE(65535)↓
    SQLTRCTYPE(SQL) JDBSQLTRCMEMNUM() SQLTRCBUFFSIZE(50)↓
    ENVGROUP(C:¥envgroup¥HiRDB.ini)↓
  HIRDBINIPATH : C:¥WINDOWS¥HiRDB.ini↓    [26]
  CONNECTION STATUS :↓    [5]
    CURHOST(eserver2)↓
    CURPORT(4439)  SRVNAME(fes1)↓
    CNCTNO(1)  SVRPID(8945)  CLTPID(9155)  CLTTID() CLTCNCTHDL(0x0)↓
  SOCKET STATUS :↓    [27]
    2013/11/13 15:20:14.540  CLT(10.196.120.171 : 1071) -> RDM(10.197.10.244  : 20280)↓
    2013/11/13 15:20:14.545  CLT()                       <- RDM()↓
    2013/11/13 15:20:14.545  CLT(10.196.120.171 : 1072) -> NDM(10.197.10.244  : 20280)↓
    2013/11/13 15:20:14.550  CLT()                       <- NDM()↓
    2013/11/13 15:20:14.550  CLT(10.196.120.171 : 1073) -> SCD(10.197.10.244  : 4439)↓
    2013/11/13 15:20:14.560  CLT(10.196.120.171 : 5000) <- SRV(10.197.10.244  : 4440)↓
```

[クライアント環境定義 PDSQLTRCFMT に 2 を指定した出力例（出力形式 2）(2/2)]

```
[ 6]  [ 7]       [ 8]      [ 9]   [10] [11] [12]     [13]  [14]                  [15]
                  [16]   [23]        [28]      [29]                              [30]
 [31]
CNCT  CLPID      CLTID      NO     OP   SEC  SQL      SQL   START-TIME             END-T
IME                OP     EXEC-TIME   TRACE     PREPROCESS FILE                PREPROCESS
  ROOT AP↓
NO                                 CODE NO   CODE     WARN
                    TION               OUTPUT                                    TIME
   ↓
```

```
                                        TIME
  ↓
----- ---------- ---------- ------ ---- ---- -------- ----- ---------------------- -----
----------------- ------ ---------- --------- ------------------------------ ----------
- ------------------------------------------- ↓
   10       2764       3968      1 CNCT    0        0 -0000 2013/11/13 15:20:14.538 2013/
11/13 15:20:14.569 000000       35502         * *
*↓
   10       2764       3968      2 SETS    1        0 -0000 2013/11/13 15:20:14.600 2013/
11/13 15:20:14.601 000000       1388        44 test.ec                          1384327820
 10.196.120.171/2764/0x0000000000000001↓
     *ROLE* ROLEA↓     [41]
   10       2764       3968      3 AUI2    1        0 -0000 2013/11/13 15:20:14.928 2013/
11/13 15:20:14.944 000000       23362         44 test.ec                          138432782
0 10.196.120.171/2764/0x0000000000000001↓
     *SQL* INSERT INTO ZAIKO VALUES(1,'AAA','C',100,5,80)↓     [17]
     *SQLLEN* 46↓                                              [32]
     *XID* 0x1234567890123456↓                                 [33]
   10       2764       3968      4 CMIT    0        0 -0000 2013/11/13 15:20:14.959 2013/
11/13 15:20:14.959 000000        2243         4 *
* 10.196.120.171/2764/0x0000000000000001↓
   11       2764       3968      5 SET     2        0 -0000 2013/11/13 15:20:14.959 2013/
11/13 15:20:14.959 000003         328         0 test.ec                          138432782
0 10.196.120.171/2764/0x0000000000000001↓
     *SQL* INSERT INTO T1 VALUES(?)↓
     *SQLLEN* 24↓
     *XID* 0x1234567890123456↓
     *EXSTMT* PRE1↓        [34]
     *EXWITHHOLD* USE↓     [36]
     *AUTORECONNECT INFORMATION*↓[38]
        2013/11/13 15:20:14.959  CLT(10.196.120.171 : 1071) -> RDM(10.197.10.244  : 20280
)↓
        2013/11/13 15:20:14.964  CLT()                       <- RDM()↓
        2013/11/13 15:20:14.964  CLT(10.196.120.171 : 1072) -> NDM(10.197.10.244  : 20280
)↓
        2013/11/13 15:20:14.969  CLT()                       <- NDM()↓
        2013/11/13 15:20:14.969  CLT(10.196.120.171 : 1073) -> SCD(10.197.10.244  : 4439)
↓
        2013/11/13 15:20:14.979  CLT(10.196.120.171 : 5000) <- SRV(10.197.10.244  : 4440)
↓
   11       2764       3968      6 SET     6        0 -0000 2013/11/13 15:20:14.979 2013/
11/13 15:20:14.979 000000         578         1 test.ec                          138432782
0 10.196.120.171/2764/0x0000000000000001↓
     *SQL* SELECT * FROM T1↓
     *SQLLEN* 16↓
     *EXSTMT* PRE2↓
   11       2764       3968     7 ALCR    9        0 -0000 2013/11/13 15:20:14.979 2013/1
1/13 15:20:14.979 000000         100         0 *                                    *
 10.196.120.171/2764/0x0000000000000001↓
     *EXSTMT* PRE2↓
     *EXCURSOL* CUR1↓     [35]
     *HLDCURSOL* USE↓     [37]
   11       2764       3968     8 OPEN   10        0 -0000 2013/11/13 15:20:14.979 2013/1
1/13 15:20:14.979 000000           8         0 test.ec                          1384327820
 10.196.120.171/2764/0x0000000000000001↓
     *EXCURSOL* CUR1↓
   11       2764       3968     9 FETC   11        0 -0000 2013/11/13 15:20:14.979 2013/1
1/13 15:20:14.995 000000       11287        21 *                                    *
```

```
 10.196.120.171/2764/0x0000000000000001↓
     *EXCURSOL* CUR1↓
     *OUTPM* NO=    1 COD=f0 XDIM=    1 SYS=    0 LEN=         4 IND=         0↓     [21
]
             DATA=64 00 00 00                                    *d...            *↓
   11       2764       3968     10 CLOS   12        0 -0000 2013/11/13 15:20:14.995 2013/
11/13 15:20:14.995 000000         269         0 test.ec                          138432782
0 10.196.120.171/2764/0x0000000000000001↓
     *EXCURSOL* CUR1↓
   11       2764       3968     11 CMIT    0        0 -0000 2013/11/13 15:20:15.011 2013/
11/13 15:20:15.011 000000         310         0 *
* 10.196.120.171/2764/0x0000000000000001↓
   12       8275          0     12 SET     2        0 W8800 2013/11/13 15:20:15.011 2013/
11/13 15:20:15.776 000000      765147       578 test.ec                          138432782
0 10.196.120.171/2764/0x0000000000000001↓
     *SQL(AUTHID)* INSERT INTO TBL01 VALUES('12345',12345)↓
     *SQLLEN* 39↓
   12       8275          0     13 SAUT    0        0 -0000 2013/11/13 15:20:15.776 2013/
11/13 15:20:15.776 000000         912       500 *
* 10.196.120.171/2764/0x0000000000000001↓
     *USER* hirdb01↓     [18]
   12       8275          0     14 SOPT    0        0 -0000 2013/11/13 15:20:15.776 2013/
11/13 15:20:15.776 000000         379        64 *
* 10.196.120.171/2764/0x0000000000000001
     *SCHEMA* hirdb02          [39]
   11       2764       3968     12 DISC    0        0 -0000 2013/11/13 15:20:15.776 2013/
11/13 15:20:15.776 000000         434       200 *
* 10.196.120.171/2764/0x0000000000000001↓
```

[クライアント環境定義 PDSQLTRCTYPE に ERROR を指定した出力例（1/2)]

```
              [24]       [25] [20]                            [19]  [22]
  ** UAP TRACE TYPE2 (CLT:10-09(Nov  5 2024) SVR:10-09   US) WIN32(WIN_M32)**↓
  USER APPLICATION PROGRAM FILE NAME : TESTAP↓     [1]
  USERID : hirdb↓     [2]
  UAP START TIME : 2024/11/13 15:20:14↓     [3]
  UAP ENVIRONMENT :↓     [4]
    LANG(ja_JP.SJIS)↓
    USER("hirdb")↓
    HOST(eserver2)↓
    NAMEPORT(20280)↓
    FESHOST()↓
    SVCGRP()  SVCPORT()  SRVTYPE()↓
    SWAIT(600)  CWAIT(0)  SWATCH(0)↓
    BLKF(1)  RDABLKF(-1)  LCKLMT(0)  ISLLVL(2)  DBLOG(ALL)  DFLNVAL(NOUSE)↓
    AGGR(1024)  DLKPRIO(64)  EXWARN(NO)  VWOPTMODE(0)↓
    LOCKSKIP(NO) CLTGRP(A)  DSQLOBJCACHE(YES)  PLGIXMK(NO)↓
    CLTRCVPORT(5000)  CLTRCVADDR()  PLGPFSZ(8192)↓
    PLGPFSZEXP(8192)  SPACELVL(-1)  STJTRNOUT(YES)↓
    OPTLVL("SELECT_APSL","RAPID_GROPING")↓
    ADDITIONALOPTLVL("COST_BASE_2","APPLY_HASH_JOIN")↓
    UAPREPLVL()  REPPATH()↓
    TRCPATH()↓
    IPC(MEMORY)  SENDMEMSIZE(16)  RECVMEMSIZE(32)↓
    HASHTBLSIZE(128)  CMMTBFDDL(NO)  PRPCRCLS()  DBACCS()↓
    SQLTRCOPENMODE(SQL)  AUTOCONNECT(ON)  CWAITTIMEWRNPNT(-1)  TCPCONOPT(1)↓
    WRTLNFILSZ(-1) WRTLNCOMSZ(1024)↓
```

```
  WRTLNPATH() UAPENVFILE()↓
  TP1SERVICE(NO) AUTORECONNECT(NO) RCCOUNT(0) RCINTERVAL(0)↓
  KALVL(2) KATIME(600) CLTCNVMODE(NOUSE)↓
  PRMTRC(NO) PRMTRCSIZE(256) BESCONHOLD() BESCONHTI(-1)↓
  BLKBUFFSIZE(10) BINARYBLKF(NO) FORUPDATEEXLOCK(YES)  DBORGUAP(NO)↓
  CNSTRNTNAME() SQLTEXTSIZE(4096) RCTRACE(-1)↓
  FESGRP()↓
  NBLOCKWAITTIME(0) CONNECTWAITTIME(300) DBBUFLRU(YES)↓
  UAPEXERLOGUSE() UAPEXERLOGPRMSZ() HJHASHINGMODE(TYPE1)↓
  DDLDEAPRP(NO) DELRSVWDFILE() HATRNQUEUING()↓
  ODBSPLITSIZE(100) NODELAYACK(YES) CURSORLVL(0)↓
  TAAPINFPATH() TAAPINFMODE(0) TAAPINFSIZE(409600)↓
  JETCOMPATIBLE(NO) SUBSTRLEN() BLKFUPD(YES) ARYERRPOS(YES)↓
  CALCMDWAITTIME(0) BLKFERRBREAK(YES) XAAUTORECONNECT(NO)↓
  CLTBINDLOOPBACKADDR(NO)↓
  STANDARDSQLSTATE() LCKWAITTIME(-1) DDLDEAPRPEXE(NO)↓
  ODBGINFOSUPPRESS(NO) ODBSTANDARDARGSIZE(YES)↓
  ODBSTANDARDSQLSTATE(YES) ODBSTANDARDDESCCOL(YES)↓
  TMPTBLRDAREA(RDTMP01,RDTMP02,RDTMP03)↓
  EXTDECCHECK(NO)↓
  DNDPCOMPATIBLE(NO)↓
  RCTIMING(ALL)↓
  ODBSTANDARDGTYPEINFO(NO)↓
  HSICOPTIONS()↓
  DEFAULTOPTION(RECOM)↓
  TCPREDUCETIMEWAITPORT(YES) TCPREDUCECONCLOSERCVTIME(1000)↓
  AUTHTYPE(PA)↓
  CLTCNVBYTERATIO() CLTCNVUOCFUNC()↓
  CLTCNVUOCLIB()↓
  TIMEDOUTRETRY(2) BINDRETRYCOUNT(10) BINDRETRYINTERVAL(0)↓
  ODBCWRNSKIP(NO) ODBLOCATOR(NO) ODBESCAPE(1)↓
  ODBSTATCACHE(0) GDATAOPT(NO) DNDPTRACE()↓
  CLTRDNODE() RDCLTCODE() UAPERLOG(65536)↓
  ERRSKIPCODE()↓
  TRCMODE(ERR) XATRCFILEMODE(SEPARATE)↓
  RETRYCOUNT(10) RETRYINTERVAL(700)↓
  CONREFRCOUNT(9) CONREFRINTERVAL(700)↓
  IPCFILEDIR()↓
  CONTYPE(PASSIVE)↓
  SOCKETCRYPTO(NO)↓
  ODBERRTRC(YES) ODBERRTRCSIZE(65535)↓
  SQLTRCTYPE(ERROR) JDBSQLTRCMEMNUM() SQLTRCBUFFSIZE(50)↓
  ENVGROUP(C:¥envgroup¥HiRDB.ini)↓
HIRDBINIPATH : C:¥WINDOWS¥HiRDB.ini↓    [26]
CONNECTION STATUS :↓    [5]
  CURHOST(eserver2)↓
  CURPORT(4439)  SRVNAME(fes1)↓
  CNCTNO(1)  SVRPID(8945)  CLTPID(9155)  CLTTID() CLTCNCTHDL(0x0)↓
SOCKET STATUS :↓    [27]
  2024/11/13 15:20:14.540  CLT(10.196.120.171 : 1071) -> RDM(10.197.10.244  : 20280)↓
  2024/11/13 15:20:14.545  CLT()                       <- RDM()↓
  2024/11/13 15:20:14.545  CLT(10.196.120.171 : 1072) -> NDM(10.197.10.244  : 20280)↓
  2024/11/13 15:20:14.550  CLT()                       <- NDM()↓
  2024/11/13 15:20:14.550  CLT(10.196.120.171 : 1073) -> SCD(10.197.10.244  : 4439)↓
  2024/11/13 15:20:14.560  CLT(10.196.120.171 : 5000) <- SRV(10.197.10.244  : 4440)↓
```

[クライアント環境定義 PDSQLTRCTYPE に ERROR を指定した出力例（2/2）]

```
[ 6]  [ 7]       [ 8]      [ 9]   [10] [11] [12]     [13]  [14]                    [15]
                  [16]   [23]        [28]      [29]                           [30]
CNCT  CLPID      CLTID      NO     OP   SEC  SQL      SQL   START-TIME              END-T
IME                OP     EXEC-TIME   TRACE     PREPROCESS FILE                  PREPROCESS
↓
NO                                  CODE NO   CODE     WARN
                    TION               OUTPUT                                    TIME
  ↓

                                      TIME
  ↓
----- ---------- ---------- ------ ---- ---- -------- ----- ---------------------- -----
----------------- ------ ----------- --------- ------------------------------ ----------
- ↓
   11       2764       3968     84 AUI2    2     -204 -0000 2024/11/13 15:20:14.959 2024/
11/13 15:20:14.959 000003           *         0 test.ec                          138432782
0↓
     *SQL* SELECT * FROM T2↓
     *SQLLEN* 16↓
<< [3] OUTPUT TRACE(NUM:4 (5%)) CLT(IP:10.196.120.171 PORT:1070) SRV(PID:8945 IP:10.197.1
0.244 PORT:4440) >>↓    [40]
   11       2764       3968     88 SET     6        0 -0000 2024/11/13 15:20:14.979 2024/
11/13 15:20:14.979 000000           *         1 test.ec                          138432782
0↓
     *SQL* SELECT * FROM T1↓
     *SQLLEN* 16↓
   11       2764       3968     89 OPEN   10        0 -0000 2024/11/13 15:20:14.979 2024/
11/13 15:20:14.979 000000           *         0 test.ec                          138432782
0↓
     *EXCURSOL* CUR1↓
   11       2764       3968     90 FETC   11        0 -0000 2024/11/13 15:20:14.979 2024/
11/13 15:20:14.995 000000           *        21 *
*↓
     *EXCURSOL* CUR1↓
   11       2764       3968     91 CLOS   12     -723 -0000 2024/11/13 15:20:14.995 2024/
11/13 15:20:14.995 000000           *         0 test.ec                          138432782
0↓
```

（凡例）

↓：出力例（出力形式 2）での 1 行の末尾を示します。実際に出力するファイルでは表示しません。

[説明]

1. UAP 名称

   環境変数 PDCLTAPNAME で指定した名称を表示します。

2. 認可識別子

   UAP を実行したユーザの認可識別子を表示します。OS ログインユーザの簡易認証機能を使用している場合は，簡易認証ユーザを表示します。

3. UAP 開始時刻

   UAP の実行を開始した時刻を表示します。

4. UAP 実行環境

UAP を実行したときの環境変数の値を表示します。

SWATCH※，RDABLKF，SPACELVL，HASHTBLSIZE，CWAITTIMEWRNPNT，WRTLNFILSZ，BESCONHTI，RCTRACE，UAPEXERLOGPRMSZ，又は LCKWAITTIME，ISLLVL については，クライアント環境定義の指定を省略している場合，-1 を表示します。

注※

HiRDB のバージョンが 09-00 より前の場合は，次の値が表示されます。

PDSWATCHTIME の指定を省略したとき：SWATCH(0)

PDSWATCHTIME に 0 を指定したとき：SWATCH(-1)

ENVGROUP は次の優先順位で環境変数グループを表示します（数字は優先順位を示します）。指定していない場合は表示しません。

●クライアントライブラリを使用している場合

X/Open に従った API 未使用時：

1. 環境変数グループ名又は環境変数グループファイルのファイルパス

X/Open に従った API 使用時：

1. XA オープン文字列に指定した環境変数グループ名又は環境変数グループファイルのファイルパス

XA オープン文字列に指定した環境変数グループ識別子は表示しません。

●Type4 JDBC ドライバを使用している場合

1. ユーザプロパティ HiRDB_for_Java_DBID に指定した HiRDB 環境変数グループファイルのファイルパス
2. URL 中の DBID に指定した HiRDB 環境変数グループファイルのファイルパス
3. DataSource 系インタフェースの setDescription メソッドに指定した環境変数グループ名のファイルパス，又は setXAOpenString に指定した XA オープン文字列中の環境変数グループ識別子及び環境変数グループファイル名のファイルパス

次に示す項目は，システムが使用する情報です。

項番	表示内容
1	TCPCONOPT()
2	HJHASHINGMODE()
3	NODELAYACK()
4	BLKFUPD()
5	ARYERRPOS()
6	BLKFERRBREAK()
7	TCPREDUCETIMEWAITPORT()
8	TCPREDUCECONCLOSERCVTIME()
9	RETRYCOUNT()

項番	表示内容
10	RETRYINTERVAL()

5. UAP 実行ステータス

   UAP を実行したときのサーバとの接続状態を表示します。

   - CURHOST：接続先ホスト名
   - CURPORT：接続ポート番号
   - SRVNAME：フロントエンドサーバ名，又はシングルサーバ名
   - CNCTNO：コネクト通番
   - SVRPID：接続サーバのプロセス番号
   - CLTPID：UAP のプロセス番号

     Type4 JDBC ドライバから接続している場合は，0 を表示します。
   - CLTTID：UAP のスレッド番号

     Type4 JDBC ドライバではネイティブスレッドの番号を表示します。

     ただし，次の場合は 0 を表示します。

     ・JDBC2.0 の Type4 JDBC ドライバを使用している。

     ・クライアント環境定義 PDSQLTRCFMT に 1 を指定し，JDBC4.0 又は JDBC4.3 の Type4 JDBC ドライバを使用している。
   - CLTCNCTHDL：コネクションハンドル

   なお，取得できない情報があるときは，不正な値となって表示されることがあります（Windows 版の場合）。

6. コネクト通番

   サーバが CONNECT を受け付けるごとに順次カウントするコネクト通番を表示します。

7. UAP のプロセス番号

   UAP のプロセス番号を表示します。

   Type4 JDBC ドライバから接続している場合は，0 を表示します。なお，取得できないプロセス番号がある場合，不正な値となって表示されることがあります（Windows 版の場合）。

8. UAP のスレッド番号

   UAP がマルチスレッドで動作している場合に，UAP のスレッド番号を表示します。スレッドで動作していない場合は，0 が表示されます。取得できないスレッド番号が不正な値となって表示されることがあります。

   Type4 JDBC ドライバではネイティブスレッドの番号を表示します。

   ただし，次の場合は 0 を表示します。

   ・JDBC2.0 の Type4 JDBC ドライバを使用している。

・クライアント環境定義 PDSQLTRCFMT に 1 を指定し，JDBC4.0 又は JDBC4.3 の Type4 JDBC ドライバを使用している。

9. SQL カウンタ

   SQL 文を受け付けるごとに順次カウントして表示します。

   1 から 999999 までカウントできます。

   なお，999999 を超えると 1 に戻ります。

10. オペレーションコード

    SQL に対応するオペレーションコードを表示します。

    表示されるオペレーションコードに対応する SQL 文を次に示します。

オペレーションコード	対応する SQL 文
ALCR	ALLOCATE CURSOR 文
AUI2	DELETE 文（静的 SQL），INSERT 文（静的 SQL），UPDATE 文（静的 SQL），LOCK 文（静的 SQL），PURGE TABLE 文（静的 SQL），1 行 SELECT 文（静的 SQL），FREE LOCATOR 文（静的 SQL）
AUI3	代入文（静的 SQL）
AUX	EXECUTE 文
AUXI	EXECUTE IMMEDIATE 文，すべての定義系 SQL
AUXO	EXECUTE 文（INTO 指定）
CALL	CALL 文
CHBK	システムが使用
CHCT	システムが使用
CLIN	INSTALL CLIB
CLOS	CLOSE 文
CLRM	REMOVE CLIB
CLRP	REPLACE CLIB
CMIT	COMMIT 文
CNCT	CONNECT 文
CPRP	コミットプリペア※
DESC	DESCRIBE 文（OUTPUT 指定）
DEST	DESCRIBE TYPE 文
DISC	DISCONNECT 文，COMMIT 文（RELEASE 指定）
DISR	ROLLBACK 文（RELEASE 指定）
DIST	Disconnect ＋ Tran Check※

オペレーションコード	対応する SQL 文
DSCM	システムが使用
DSET	DEALLOCATE PREPARE 文
DSPR	システムが使用
DSRL	システムが使用
FETC	FETCH 文
GETD	GET DIAGNOSTICS
HVAR	DESCRIBE 文（INPUT 指定）
JARI	INSTALL JAR
JARR	REPLACE JAR
JARU	REMOVE JAR
OPEN	OPEN 文（動的 SQL）
OPN2	OPEN 文（静的 SQL）
OPNR	OPEN 文（動的 SQL(複数カーソル)）
RENV	システムが使用
RNCN	CONNECT 文（TO 指定）
RNDS	DISCONNECT 文（TO 指定）
ROLL	ROLLBACK 文
RSDC	DESCRIBE 文（OUTPUT，RESULT SET 指定）
RSFT	FETCH 文（RESULT SET 指定）
RSCL	CLOSE 文（RESULT SET 指定）
SAUH	SET SESSION AUTHORIZATION 文
SET	PREPARE 文
SINF	システムが使用
SOPT	システムが使用
SVLS	システムが使用
THRE	システムが使用
THSU	システムが使用
TRCK	システムが使用
TRC2	システムが使用
TRST	システムが使用
TSCM	システムが使用

オペレーションコード	対応する SQL 文
TSRL	Transfer Rollback※
TSPR	Transfer Prepare※

注※　XA インタフェースを使用した場合だけ出力されます。

11. セクション番号

SQL の対応を確認するための番号を表示します。

番号は，SQL プリプロセサが自動的に付けます。

12. SQLCODE

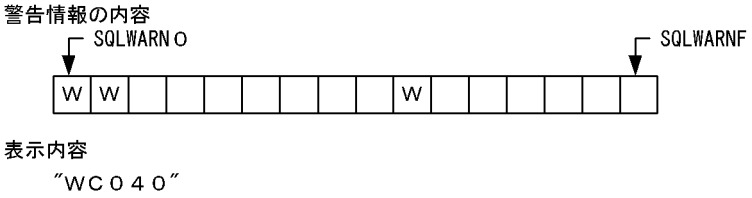
SQL 文を実行した結果，発生した SQLCODE を表示します。

13. SQLWARN

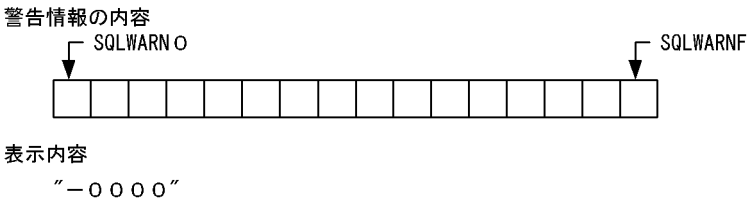
警告情報を 16 進表記で表示します。SQLWARN0 から SQLWARNF までの警告情報にそれぞれ左から 1 ビットを割り当て，警告フラグが設定されているものは 1，設定されていないものは 0 として 16 ビットの数値を求めます。これを 4 けたの 16 進数値として表示します。

一つ以上警告フラグが設定されている場合，先頭に'W'が，警告フラグが設定されていない場合は，'-'（ハイフン）が表示されます。

（例１）



（例２）



14. SQL 実行要求受付時刻

SQL の実行要求が受け付られた時刻を表示します。クライアント環境定義 PDSQLTRCFMT の指定値によって出力形式が異なります。

・PDSQLTRCFMT に 1 を指定した場合：hh:mm:ss.fff の形式

・PDSQLTRCFMT に 2 を指定した場合：YYYY/MM/DD hh:mm:ss.fff の形式

15. SQL 実行要求終了時刻

SQL の実行要求が終了した時刻を表示します。クライアント環境定義 PDSQLTRCFMT の指定値によって出力形式が異なります。

・PDSQLTRCFMT に 1 を指定した場合：hh:mm:ss.fff の形式

・PDSQLTRCFMT に 2 を指定した場合：YYYY/MM/DD hh:mm:ss.fff の形式

16. システムが使用する情報

システムが使用する情報を表示します。

1 バイト目にはプロセス間通信の種別と KeepAlive パケットの送信有無を示します。KeepAlive パケットとは，クライアント環境定義 PDKALVL に 1，2 又は 3 が指定された場合に使用されるパケットのことです。

1 バイト目の文字の意味を次に示します。

1 バイト目の文字	プロセス間通信の種別	KeepAlive パケットの送信有無
P	メモリ通信	KeepAlive パケットを送信した次の SQL 実行
K	メモリ通信以外	
M	メモリ通信	上記以外
0	メモリ通信以外	

Type4 JDBC ドライバを使用し，オペレーションコードが CMIT かつ 2 バイト目以降が次の場合，自動コミットによるコミットを意味します。

- PDSQLTRCFMT に 1 を指定した場合：004
- PDSQLTRCFMT に 2 を指定した場合：00004

ほかの部分は，HiRDB 開発者が使用する保守用の情報です。

17. SQL 文

オペレーションコードが，SET，AUXI，AUI2，又は OPN2 のときだけ，SQL 文を表示します。

なお，出力する SQL 文の長さは最大 4096 バイト（Type4 JDBC ドライバを使用している場合は最大 4096 文字）で，最大値を超えると超過分は切り捨てられます（最大値は環境変数 PDSQLTEXTSIZE で変更できます）。クライアントライブラリを使用している場合，切り捨てられた最後の文字に文字化けが発生することがあります。

また，プリプロセス時に-A オプション，又は/A オプションによって SQL 文中の認可識別子が省略されたときに仮定する認可識別子を指定している場合は，*SQL*が*SQL（仮定した認可識別子）*と表示されます。

18. ユーザ識別子を変更した場合の情報

一つのコネクションの中でユーザ識別子を変更した場合，変更後のユーザ識別子を表示します。ユーザ識別子の変更に失敗した場合も表示されます。

19. UAP が動作しているプラットフォーム

プラットフォーム	表示される文字
AIX	AIX
AIX（64 ビットモード）	AIX64

プラットフォーム	表示される文字
Linux	LINUX
Windows	WIN32
Linux (EM64T)	LINX64
Windows(x64)64 ビットモード	WINX64
Type4 JDBC ドライバ	Type4

20. HiRDB クライアントの作成日付

HiRDB クライアントの作成日付を表示します。日付の形式を次に示します。

Mmm：月（英語の先頭 3 文字（1 文字目は大文字）。例えば，June の場合は Jun）

dd：日

yyyy：西暦

21. パラメタトレース

クライアント環境定義で PDPRMTRC=YES，IN，OUT，又は INOUT を指定した場合，入力用パラメタ情報，出力用パラメタ情報，及び検索データを出力します。

なお，出力されるパラメタ情報のデータの長さは，PDPRMTRCSIZE の指定値（省略時は 256 バイト）を最大長として，最大長を超える部分は切り捨てられます。また，ASCII コードの範囲外の文字はドット（.）で出力されます。詳細については，「パラメタトレースの出力例」を参照してください。

パラメタ情報を次に示します。

NO：

パラメタ番号

COD：

データ型コード

XDIM：

配列要素数

SYS：

ギャップを含む 1 要素の領域長

LEN：

データの長さ

入力用パラメタ：

・HiRDB データプロバイダ for .NET Framework：

HiRDBParameter.Size の指定値

・上記以外：

パラメタの実長

出力用パラメタ：

パラメタの定義長

IND：

標識変数の値

ARRAY NUM：

繰返し列の要素数

ROW NUM：

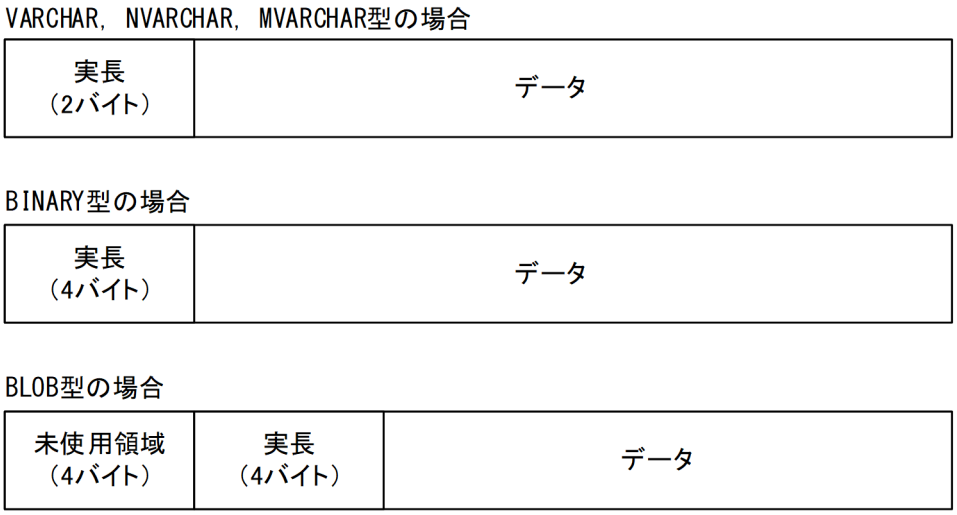
配列の埋込み変数を使用した SQL の実行行数

DATA：

データ（ダンプ形式）

可変長データのデータ型の形式を次に示します。

図 11–1　可変長データのデータ型の形式



22. リンクしたライブラリの名称

ライブラリ名称	表示される文字
libzclt.sl，libclt.a	UNIX
libzclts.sl，libclts.a	UNIX_S
libzcltm.sl，libcltm.a	UNIX_M
libzcltk.sl，libcltk.a，libzclt6k.a	UNIX_K
libzcltkt.so	UNIX_KT
libzcltx.sl，libcltxa.a	UNIX_XA
libzcltxs.sl，libcltxas.a，libzcltys.a，libzclt6ys.a	UNIX_XA_S
libzcltxm.sl，libcltxam.a	UNIX_XA_M
libzcltxk.sl，libcltxak.a，libzcltyk.a，libzclt6yk.a	UNIX_XA_K
libzclt64.sl，libclt64.a	UNIX_64

ライブラリ名称	表示される文字
libzcltk64.sl，libcltk64.a，libzclt6k64.a	UNIX_64K
libzclts64.sl	UNIX_64S
libzcltx64.sl，libzclty64.sl	UNIX_XA_64
libzcltxk64.sl，libzcltyk64.sl	UNIX_XA_64K
libzcltxs64.sl，libzcltys64.sl，libzclt6ys64.a	UNIX_XA_64S
CLTDLL.DLL	WIN_32
PDCLTM32.DLL	WIN_M32
PDCLTM50.DLL	WIN_M50
PDCLTM71.DLL	WIN_M71
PDCLTM80S.DLL	WIN_M80S
PDCLTM90S.DLL	WIN_M90S
PDCLTM90.DLL	WIN_M90
PDCLTM100.DLL	WIN_M100
PDCLTM140.DLL	WIN_M140
PDCLTM140T.DLL	WIN_M140T
PDCLTP32.DLL	WIN_P32
PDCLTX32.DLL	WIN_XA_32
PDCLTXM.DLL	WIN_XA_32M
PDCLTXS.DLL	WIN_XA_32S
PDCLTXM5.DLL	WIN_XA_50M
PDCLTM64.DLL	WIN_M64
PDCLTM90S64.DLL	WIN_M90S64
PDCLTM90X.DLL	WIN_M90X
PDCLTM100X.DLL	WIN_M100X
PDCLTM140X.DLL	WIN_M140X
PDCLTX64.DLL	WIN_XA_64
PDCLTXM64.DLL	WIN_XA_64M
PDCLTXS64.DLL	WIN_XA_64S
PDCLTL32.DLL	WIN_L32
PDCLTO32.DLL	WIN_O32
PDCLTO64.DLL	WIN_O64

ライブラリ名称	表示される文字
PDJDBC2.JAR	Type4
PDJDBC4.JAR	Type4_JDBC40
PDJDBC43.JAR	Type4_JDBC43

（凡例）

－：該当しません。

23. SQL 実行時間

クライアント環境定義に PDSQLEXECTIME=YES を指定した場合，SQL 実行時間をマイクロ秒単位で表示します。

NO を指定している場合，クライアント環境定義 PDSQLTRCFMT の指定値によって表示内容が異なります。

・PDSQLTRCFMT に 1 を指定した場合：この情報を表示しません。

・PDSQLTRCFMT に 2 を指定した場合：'*'を表示します。

JDBC2.0 の Type4 JDBC ドライバを使用している場合は，常に'*'を表示します。

24. SQL トレース出力形式

SQL トレースの出力形式を表示します。

25. HiRDB クライアント作成バージョン

HiRDB クライアントの作成バージョンを表示します。クライアント環境定義 PDSQLTRCFMT の指定値によって出力形式が異なります。

・PDSQLTRCFMT に 1 を指定した場合：VV-RR の形式

・PDSQLTRCFMT に 2 を指定した場合：VV-RR-XX の形式

26. 実際に使用している HiRDB.ini ファイルパス

次の優先順位で，どれか一つの HiRDB.ini のファイルパスを表示します（数字は優先順位を示します）。HiRDB.ini を使用していない場合，この情報は表示しません。

●クライアントライブラリを使用している場合

Windows 版：

1. Windows ディレクトリ下の HiRDB.ini

UNIX 版：

表示しません。

●Type4 JDBC ドライバを使用している場合

1. ユーザプロパティ HiRDB_for_Java_HiRDB_INI に指定したディレクトリ下の HiRDB.ini
2. URL 中の HIRDB_INI に指定したディレクトリ下の HiRDB.ini
3. DataSource 系インタフェースの setHiRDBINI メソッドの引数に指定したディレクトリ下の HiRDB.ini

●ODBC ドライバを使用している場合

Windows 版：

1. データソースで設定した HiRDB クライアント環境変数ファイル名
2. Windows ディレクトリ下の HiRDB.ini

UNIX 版：

1. データソースで設定した HiRDB クライアント環境変数ファイル名

27. サーバとの接続処理で使用したソケット情報

サーバとの接続処理で使用したソケット情報を表示します。接続形態によって表示内容が異なります。

- 日時：クライアント−サーバ間で通信した日時
  YYYY/MM/DD hh:mm:ss.fff の形式で表示します。
- CLT：クライアント側の IP アドレス，送受信ポート番号
- RDM，NDM，SCD，SRV：サーバ側の IP アドレス，送受信ポート番号
  接続形態ごとの表示内容を次に示します。

●通常接続の場合

```
クライアント環境定義PDCONTYPEにPASSIVEを指定した場合：
2013/11/13 15:20:14.540  CLT(10.196.120.171 : 1071) -> RDM(10.197.10.244  : 20280)
2013/11/13 15:20:14.545  CLT()                       <- RDM()
2013/11/13 15:20:14.545  CLT(10.196.120.171 : 1072) -> NDM(10.197.10.244  : 20280)
2013/11/13 15:20:14.550  CLT()                       <- NDM()
2013/11/13 15:20:14.550  CLT(10.196.120.171 : 1073) -> SCD(10.197.10.244  : 4439)
2013/11/13 15:20:14.560  CLT(10.196.120.171 : 5000) <- SRV(10.197.10.244  : 4440)
クライアント環境定義PDCONTYPEにACTIVEを指定した場合：
2023/07/13 15:20:14.540  CLT(10.196.120.171 : 1071) -> RDM(10.197.10.244  : 20280)
2023/07/13 15:20:14.545  CLT()                       <- RDM()
2023/07/13 15:20:14.545  CLT(10.196.120.171 : 1072) -> NDM(10.197.10.244  : 20280)
2023/07/13 15:20:14.550  CLT()                       <- NDM()
2023/07/13 15:20:14.550  CLT(10.196.120.171 : 1073) -> SCD(10.197.10.244  : 4439)
2023/07/13 15:20:14.560  CLT()                       <- SRV()
```

●FES ホストダイレクト接続の場合

```
クライアント環境定義PDCONTYPEにPASSIVEを指定した場合：
2013/11/13 15:20:14.540  CLT()                       -> RDM()
2013/11/13 15:20:14.545  CLT()                       <- RDM()
2013/11/13 15:20:14.545  CLT(10.196.120.171 : 1072) -> NDM(10.197.10.244  : 4437)
2013/11/13 15:20:14.550  CLT()                       <- NDM()
2013/11/13 15:20:14.550  CLT(10.196.120.171 : 1073) -> SCD(10.197.10.244  : 20280)
2013/11/13 15:20:14.560  CLT(10.196.120.171 : 5000) <- SRV(10.197.10.244  : 4440)
クライアント環境定義PDCONTYPEにACTIVEを指定した場合：
2023/07/13 15:20:14.540  CLT()                       -> RDM()
2023/07/13 15:20:14.545  CLT()                       <- RDM()
2023/07/13 15:20:14.545  CLT(10.196.120.171 : 1072) -> NDM(10.197.10.244  : 4437)
2023/07/13 15:20:14.550  CLT()                       <- NDM()
2023/07/13 15:20:14.550  CLT(10.196.120.171 : 1073) -> SCD(10.197.10.244  : 20280)
2023/07/13 15:20:14.560  CLT()                        <- SRV()
```

●高速接続の場合

```
クライアント環境定義PDCONTYPEにPASSIVEを指定した場合：
2013/11/13 15:20:14.540  CLT()                        -> RDM()
2013/11/13 15:20:14.545  CLT()                        <- RDM()
2013/11/13 15:20:14.545  CLT()                        -> NDM()
2013/11/13 15:20:14.550  CLT()                        <- NDM()
2013/11/13 15:20:14.550  CLT(10.196.120.171 : 1073) -> SCD(10.197.10.244  : 4439)
2013/11/13 15:20:14.560  CLT(10.196.120.171 : 5000) <- SRV(10.197.10.244  : 4440)
クライアント環境定義PDCONTYPEにACTIVEを指定した場合：
2023/07/13 15:20:14.540  CLT()                        -> RDM()
2023/07/13 15:20:14.545  CLT()                        <- RDM()
2023/07/13 15:20:14.545  CLT()                        -> NDM()
2023/07/13 15:20:14.550  CLT()                        <- NDM()
2023/07/13 15:20:14.550  CLT(10.196.120.171 : 1073) -> SCD(10.197.10.244  : 4439)
2023/07/13 15:20:14.560  CLT()                        <- SRV()
```

28. SQL トレース出力処理時間

    SQL トレースの出力に掛かった処理時間をマイクロ秒単位で表示します。前回のオペレーションコード実行時に掛かった処理時間を該当する行に表示します。最大は 999999999 マイクロ秒となり，超えた場合，999999999 マイクロ秒を表示します。初回の SQL トレース出力時は'*'を表示します。

    JDBC2.0 の Type4 JDBC ドライバを使用している場合，精度はミリ秒となります。

29. プリプロセスファイル名

    プリプロセスした UAP ソースファイル名を表示します。ファイル名を表示しない場合，'*'を表示します。Type4 JDBC ドライバを使用している場合，この情報を表示しません。

30. プリプロセス時間

    UAP をプリプロセスした時間を表示します。時間を表示しない場合，'*'を表示します。Type4 JDBC ドライバを使用している場合，この情報を表示しません。

31. ルートアプリケーション情報

    Cosminexus から Type4 JDBC ドライバを使用している場合，Cosminexus のルートアプリケーション情報を表示します。ルートアプリケーション情報については，Cosminexus のマニュアルを参照してください。

    クライアントライブラリを使用している場合，この情報を表示しません。

32. SQL 文長

    SQL 文の実長を表示します。単位を次に示します。

    - Type4 JDBC ドライバを使用している場合：Unicode コード単位の数
    - 上記以外：バイト数

    17.の SQL 文と併せて表示します。SQL 文を表示しない場合，この情報も表示しません。

    Type4 JDBC ドライバでの Unicode コード単位の扱いについては，「クライアント環境定義の設定内容」の「PDSQLTEXTSIZE」を参照してください。

33. トランザクション識別子

    X/Open に従った API を使用した接続形態の場合にトランザクション識別子を表示します。

次の場合に表示します。

- サーバと接続後の初回 SQL 実行
- トランザクション決着後の初回 SQL 実行
- xa_commit，xa_rollback，xa_prepare 発行時

上記以外の場合，この情報は表示しません。

34. 拡張文名

拡張文名を使用した SQL 実行時，使用した拡張文名を表示します。拡張文名を使用していない場合，この情報は表示しません。

35. 拡張カーソル名

拡張カーソル名を使用した SQL 実行時，使用した拡張カーソル名を表示します。拡張カーソル名を使用していない場合，この情報は表示しません。

36. 前処理結果保持機能指定情報

次に示すどれかの場合に表示します。

●Type4 JDBC ドライバを使用している場合

- システムプロパティの HiRDB_for_Java_DAB_STATEMENT_COMMIT_BEHAVIOR の有効値が TRUE となっている。
- ユーザプロパティの HiRDB_for_Java_STATEMENT_COMMIT_BEHAVIOR の有効値が TRUE となっている。
- URL の STATEMENT_COMMIT_BEHAVIOR の有効値が TRUE となっている。
- DataSource 系インタフェースの setStatementCommitBehavior メソッドの引数に true を指定する。

●Type2 JDBC ドライバを使用している場合

- ユーザプロパティの COMMIT_BEHAVIOR の有効値が CLOSE 又は PRESERVE となっている。
- URL 中の COMMIT_BEHAVIOR の有効値が CLOSE 又は PRESERVE となっている。
- DataSource 系インタフェースの setCommit_Behavior メソッドの引数に CLOSE 又は PRESERVE を指定する。

上記以外の場合，この情報は表示しません。

37. ホールダブルカーソル指定情報

ホールダブルカーソルを使用した SQL 実行時に表示します。ホールダブルカーソルを使用していない場合，この情報は表示しません。

38. 自動再接続時にサーバとの接続処理で使用したソケット情報

自動再接続時にサーバとの接続処理で使用したソケット情報を表示します。

表示内容については 27.を参照してください。

自動再接続の詳細は，「自動再接続機能」を参照してください。

39. デフォルト認可識別子設定情報

    デフォルト認可識別子変更メソッド（Connection インタフェースの setSchema メソッド）で設定した認可識別子を表示します。

40. SQL トレース出力開始情報

    クライアント環境定義 PDSQLTRCTYPE に ERROR を指定している場合に，SQL トレース情報を出力する開始位置を，次の形式で示します。

    << [出力の通番] OUTPUT TRACE(NUM:出力する SQL トレース情報数 (退避の上限に対する比率)) CLT(IP:クライアントの IP アドレス PORT:クライアントのポート番号) SRV(PID:サーバのプロセス ID IP:サーバの IP アドレス PORT:サーバのポート番号) >>

    退避の上限はクライアント環境定義 PDJDBSQLTRCMEMNUM 又は PDSQLTRCBUFFSIZE の指定値。CLT 及び SVR の項目は，ファイル出力時点での情報を表示します。また，未接続の場合，CLT() SRV()として表示します。

41. ロール名

    SET ROLE 文で設定したロール名を表示します。

    NONE を指定した場合は，空白文字を表示します。

(4) SQL トレースファイルのバックアップの取得

SQL トレース情報を出力中に SQL トレースファイルの容量が一杯になると，HiRDB はそのファイルへ出力しないで，もう一方の SQL トレースファイルに SQL トレース情報を出力します。このとき，切り替え先の SQL トレースファイルに格納されている古い SQL トレース情報から順に消去され，新しい SQL トレース情報に書き換えられます。このため，必要な情報は UAP 終了時に SQL トレースファイルの内容をコピーしてバックアップを取得しておいてください。

なお，現在使用している SQL トレースファイルを知りたい場合は，ファイルの最終更新日時を調べてください。最終更新日時の新しい方が現在使用している SQL トレースファイルになります。

HiRDB クライアントが Windows 版の場合は dir コマンド又はエクスプローラで，ファイルの最終更新日時を調べてください。

HiRDB クライアントが UNIX 版の場合は OS の ls -l コマンドで，ファイルの最終更新日時を調べてください。

(5) パラメタトレースの出力例

代表的なパラメタトレースの出力例を次に示します。

(a)　INSERT 文の場合（ナル値あり，繰返し列あり）

```
CNCT  CLPID CLTID NO     OP   SEC  SQL  SQL   START-TIME   END-TIME     OP
NO                       CODE NO   CODE WARN                          TION
----- ----- ----- ------ ---- ---- ---- ----- ------------ ------------ ----
    7  1088  2060      1 CNCT    0    0 -0000 18:47:21.435 18:47:21.755 0000
    7  1088  2060      2 AUI2    1    0 -0000 18:47:21.765 18:47:21.765 0000
     *SQL* INSERT INTO TBL01(C1,C3) VALUES(?,?)
  [1] *INPRM* NO=    1 COD=f0 XDIM=    1 SYS=    0 LEN=         4 IND=         0
             DATA=00 00 00 65                                        *...e            *
     *INPRM* NO=    2 COD=c1 XDIM=    5 SYS=  102 LEN=       100 IND=         0
           [2]ARRAY NUM=    5 [5]
        [3] 0 DATA( [4] 0)=00 01 61                                          *..a             *
            0 DATA(    1)=00 07 62 62 62 62 62 62 62                         *..bbbbbbb       *
            0 DATA(    2)=00 04 63 63 63 63                                  *..cccc          *
            0 DATA(    3)=00 09 64 64 64 64 64 64 64 64 64                   *..ddddddddd     *
            0 DATA(    4)=00 0a 65 65 65 65 65 65 65 65 65 65                *..eeeeeeeeee    *
    7  1088  2060      3 AUI2    2    0 -0000 18:47:21.785 18:47:21.795 0000
     *SQL* INSERT INTO TBL01(C1,C3) VALUES(?,?)
     *INPRM* NO=    1 COD=f0 XDIM=    1 SYS=    0 LEN=         4 IND=         0
             DATA=00 00 00 66                                        *...f            *
     *INPRM* NO=    2 COD=c1 XDIM=    5 SYS=  102 LEN=       100 IND=         0
             ARRAY NUM=    5
           0 DATA(    0)=00 01 61                                          *..a             *
         [6]-1 DATA(    1)=
            0 DATA(    2)=00 04 63 63 63 63                                  *..cccc          *
           -1 DATA(    3)=
            0 DATA(    4)=00 4f 65 65 65 65 65 65 65 65 65 65 65 65 65 65   *.Oeeeeeeeeeeeeee*
                          65 65 65 65 65 65 65 65 65 65 65 65 65 65 65 65   *eeeeeeeeeeeeeeee*
                     [7] --- SAME 3 LINES ---
                          65                                              *e               *
    7  1088  2060      4 AUI2    3    0 -0000 18:47:21.805 18:47:21.815 0000
     *SQL* INSERT INTO TBL01(C1,C3) VALUES(?,?)
     *INPRM* NO=    1 COD=f0 XDIM=    1 SYS=    0 LEN=         4 IND=         0
             DATA=00 00 00 67                                        *...g            *
     *INPRM* NO=    2 COD=c1 XDIM=    5 SYS=  102 LEN=       100 IND=        -1
          [6] DATA=
    7  1088  2060      5 DISC    0    0 -0000 18:47:21.825 18:47:21.825 0000
```

[説明]

INSERT 文で，INTEGER，及び VARCHAR(10)の繰返し列（要素数 5）を挿入する場合のパラメタトレース出力例です。？パラメタの指定順に出力されます。

1. 入力パラメタの場合，"*INPRM*"と表示されます。ただし，PDPRMTRC=YES の場合は，"*PARAM*"と表示されます。
2. 繰返し列の場合，ARRAY NUM に繰返し要素数が表示されます。
3. DATA 句の前の数値は，繰返し列の要素ごとの標識変数です。
4. DATA 句の括弧内の数値は，繰返し列の要素番号です。
5. VARCHAR 型の場合，DATA の先頭 2 バイトはデータ長領域です。PDPRMTRC が YES の場合は，定義長＋データ領域長のサイズ分出力されます。PDPRMTRC が IN，OUT，又は INOUT の場合は，実データ長＋データ領域長のサイズ分出力されます。
6. 標識変数がマイナス値の場合，"DATA="まで表示されます。
7. データが 2 行以上続く場合，"--- SAME x LINES ---"が出力されます（x は行数）。ただし，PDPRMTRC=YES の場合は，データがすべて出力されます。

(b)　1 行 SELECT 文の場合

```
CNCT  CLPID CLTID NO     OP   SEC  SQL  SQL   START-TIME   END-TIME     OP
NO                       CODE NO   CODE WARN                           TION
----- ----- ----- ------ ---- ---- ---- ----- ------------ ------------ ----
   12  1492  2260      1 CNCT    0    0 -0000 19:18:31.914 19:18:32.135 0000
   12  1492  2260      2 AUI2    1    0 -0000 19:18:32.135 19:18:32.145 0000
     *SQL* SELECT C2,C3 FROM TBL02 WHERE C1=? AND C4=?
  [1] *OUTPM* NO=    1 COD=c4 XDIM=    1 SYS=    0 LEN=        10 IND=         0
            DATA=41 41 41 41 41 41 41 41 41 41                      *AAAAAAAAAA      *
  [1] *OUTPM* NO=    2 COD=c0 XDIM=    1 SYS=    0 LEN=        10 IND=         0
            DATA=00 08 61 61 61 61 61 61 61 61                      *..aaaaaaaa      *
  [2] *INPRM* NO=    3 COD=f0 XDIM=    1 SYS=    0 LEN=         4 IND=         0
            DATA=00 00 00 65                                        *...e            *
  [2] *INPRM* NO=    4 COD=f4 XDIM=    1 SYS=    0 LEN=         2 IND=         0
            DATA=00 62                                              *.b              *
   12  1492  2260      3 DISC    0    0 -0000 19:18:32.155 19:18:32.155 0000
```

[説明]

PDPRMTRC=INOUT 指定時のパラメタトレース出力例です。検索項目の順に検索データ情報が先に出力され，入力パラメタ情報が指定順で後に出力されます。

1. 検索データ情報です。PDPRMTRC=IN の場合は出力されません。また，PDPRMTRC=YES の場合は，"*OUTPM*"が"*PARAM*"になります。
2. 入力パラメタ情報です。PDPRMTRC=OUT の場合は出力されません。また，PDPRMTRC=YES の場合は，"*INPRM*"が"*PARAM*"になります。

(c)　ストアドプロシジャの実行（CALL 文）の場合

```
CNCT  CLPID CLTID NO     OP   SEC  SQL  SQL   START-TIME   END-TIME     OP
NO                       CODE NO   CODE WARN                           TION
----- ----- ----- ------ ---- ---- ---- ----- ------------ ------------ ----
   16  1456  2188      1 CNCT    0    0 -0000 19:43:00.486 19:43:00.797 0000
   16  1456  2188      2 CALL    1    0 -0000 19:43:00.797 19:43:00.807 0000
     *SQL* CALL PROC1(IN?,OUT?,INOUT?)
  [1] *INPRM* NO=    1 COD=f0 XDIM=    1 SYS=    0 LEN=         4 IND=         0
            DATA=00 00 00 78                                        *...x            *
  [2] *INPRM* NO=    2 COD=c0 XDIM=    1 SYS=    0 LEN=        10 IND=         0
            DATA=00 09 63 63 63 63 63 63 63 63                      *..cccccccc      *
  [3] *OUTPM* NO=    1 COD=f0 XDIM=    1 SYS=    0 LEN=         4 IND=         0
            DATA=00 00 00 dc                                        *....            *
  [4] *OUTPM* NO=    2 COD=c0 XDIM=    1 SYS=    0 LEN=        10 IND=         0
            DATA=00 09 63 63 63 63 63 63 63 63                      *..cccccccc      *
   16  1456  2188      3 DISC    0    0 -0000 19:43:00.829 19:43:00.829 0000
```

[説明]

1. IN パラメタです。PDPRMTRC=OUT の場合は出力されません。
2. INOUT パラメタの入力パラメタです。ただし，DATA 句の内容は出力データとなります。
3. OUT パラメタです。PDPRMTRC=IN，又は YES の場合は出力されません。
4. INOUT パラメタの出力パラメタです。PDPRMTRC=IN，又は YES の場合は出力されません。

(d) 検索（FETCH 文）の場合

```
CNCT  CLPID CLTID NO     OP   SEC  SQL  SQL   START-TIME   END-TIME     OP
NO                       CODE NO   CODE WARN                          TION
----- ----- ----- ------ ---- ---- ---- ----- ------------ ------------ ----
    6   668  1664      1 CNCT    0    0 -0000 14:49:54.326 14:49:54.696 0000
    6   668  1664      2 OPN2    1    0 -0000 14:49:54.736 14:49:54.746 0000
     *SQL* SELECT*FROM TBL03
    6   668  1664      3 FETC    1    0 -0000 14:49:54.746 14:49:54.746 0000
     *OUTPM* NO=    1 COD=f1 XDIM=    1 SYS=    0 LEN=          4 IND=          0
             DATA=00 00 00 78                                      *...x           *
     *OUTPM* NO=    2 COD=c5 XDIM=    1 SYS=    0 LEN=         10 IND=          0
             DATA=41 41 41 41 41 41 41 41 41 41                    *AAAAAAAAAA     *
     *OUTPM* NO=    3 COD=c1 XDIM=    1 SYS=    0 LEN=         10 IND=          0
             DATA=00 08 61 61 61 61 61 61 61 61                    *..aaaaaaaa     *
    6   668  1664      4 FETC    1    0 -0000 14:49:54.756 14:49:54.756 0000
     *OUTPM* NO=    1 COD=f1 XDIM=    1 SYS=    0 LEN=          4 IND=          0
             DATA=00 00 00 96                                      *....           *
     *OUTPM* NO=    2 COD=c5 XDIM=    1 SYS=    0 LEN=         10 IND=          0
             DATA=43 43 43 43 43 43 43 43 43 43                    *CCCCCCCCCC     *
     *OUTPM* NO=    3 COD=c1 XDIM=    1 SYS=    0 LEN=         10 IND=          0
             DATA=00 06 63 63 63 63 63 63                          *..cccccc       *
    6   668  1664      5 FETC    1    0 -0000 14:49:54.756 14:49:54.766 0000
     *OUTPM* NO=    1 COD=f1 XDIM=    1 SYS=    0 LEN=          4 IND=          0
             DATA=00 00 00 b4                                      *....           *
     *OUTPM* NO=    2 COD=c5 XDIM=    1 SYS=    0 LEN=         10 IND=          0
             DATA=44 44 44 44 44 44 44 44 44 44                    *DDDDDDDDDD     *
     *OUTPM* NO=    3 COD=c1 XDIM=    1 SYS=    0 LEN=         10 IND=          0
             DATA=00 09 64 64 64 64 64 64 64 64                    *..dddddddd     *
 [1]6   668  1664      6 FETC    1  100 -0000 14:49:54.766 14:49:54.766 0000
    6   668  1664      7 CLOS    1    0 -0000 14:49:54.776 14:49:54.776 0000
    6   668  1664      8 CMIT    0    0 -0000 14:49:54.776 14:49:54.776 0000
```

[説明]

FETCH 文のパラメタトレース出力例です。PDPRMTRC=IN，又は YES の場合，パラメタトレースは出力されません。

1. FETCH 文の SQLCODE が 0 以外の場合，パラメタトレースを出力されません。

(e)　検索（配列を使用した FETCH 機能）の場合

```
CNCT  CLPID CLTID NO     OP   SEC  SQL  SQL   START-TIME   END-TIME     OP
NO                       CODE NO   CODE WARN                          TION
----- ----- ----- ------ ---- ---- ---- ----- ------------ ------------ ----
    6   668  1664      9 OPN2    2    0 -0000 14:49:54.786 14:49:54.786 0000
     *SQL* SELECT*FROM TBL03
    6   668  1664     10 FETC    2    0 -0000 14:49:54.786 14:49:54.796 0002
 [1] *ROW NUM = 2*
     *OUTPM* NO=    1 COD=f1 XDIM=    1 SYS=    4 LEN=         4 IND=         0
           0 DATA(    0)=00 00 00 78                                        *...x            *
           0 DATA(    1)=00 00 00 96                                        *....            *
     *OUTPM* NO=    2 COD=c5 XDIM=    1 SYS=   11 LEN=        10 IND=         0
        [2] 0 DATA( [3]0)=41 41 41 41 41 41 41 41 41 41                     *AAAAAAAAAA      *
           0 DATA(    1)=43 43 43 43 43 43 43 43 43 43                     *CCCCCCCCCC      *
     *OUTPM* NO=    3 COD=c1 XDIM=    1 SYS=   12 LEN=        10 IND=         0
           0 DATA(    0)=00 08 61 61 61 61 61 61 61 61                     *..aaaaaaaa      *
           0 DATA(    1)=00 06 63 63 63 63 63 63                           *..cccccc        *
    6   668  1664     11 FETC    2  100 -0000 14:49:54.796 14:49:54.806 0001
     *ROW NUM = 1*
 [4] *OUTPM* NO=    1 COD=f1 XDIM=    1 SYS=    4 LEN=         4 IND=         0
           0 DATA(    0)=00 00 00 b4                                        *....            *
     *OUTPM* NO=    2 COD=c5 XDIM=    1 SYS=   11 LEN=        10 IND=         0
           0 DATA(    0)=44 44 44 44 44 44 44 44 44 44                     *DDDDDDDDDD      *
     *OUTPM* NO=    3 COD=c1 XDIM=    1 SYS=   12 LEN=        10 IND=         0
           0 DATA(    0)=00 09 64 64 64 64 64 64 64 64                     *..ddddddddd     *
    6   668  1664     12 CLOS    2    0 -0000 14:49:54.806 14:49:54.816 0000
    6   668  1664     13 DISC    0    0 -0000 14:49:54.826 14:49:54.846 0000
```

[説明]

配列を使用した FETCH 機能のパラメタトレース出力例です。PDPRMTRC=IN，又は YES の場合，パラメタトレースは出力されません。

1. ROW NUM には配列要素数（検索行数）が表示されます。
2. DATA 句の前の数値は配列要素ごとの標識変数です。
3. DATA 句の括弧内の数値は配列要素番号です。
4. FETCH 文の SQLCODE が 0 以外の場合，サーバから返却された行数分のパラメタトレースが出力されます。

11.1.2　クライアントエラーログ機能

クライアントと HiRDB サーバ間の通信処理中，又は X/Open で規定した XA インタフェースでエラーが発生した場合，エラー情報をクライアントエラーログとしてクライアントエラーログファイルに取得します。

クライアントエラーログファイルは，取得した情報で満杯になると，最も古い情報から順次新しい情報に書き替えられます。

(1)　クライアントエラーログ情報の取得方法

クライアントエラーログは，クライアント環境定義の PDCLTPATH 及び PDUAPERLOG に値を設定することで取得できます。各クライアント環境定義については，「クライアント環境定義（環境変数の設定）」を参照してください。

クライアント環境定義で指定した出力先ディレクトリが存在しない，又は書き込み権限がない場合，カレントディレクトリに出力をします。カレントディレクトリにも書き込み権限がない場合は，出力しません。

情報を取得するクライアントエラーログファイルは，指定したディレクトリに二つ作成されます。作成されるファイルは，X/Open に従った API（TX_関数）の使用の有無によって異なります。

X/Open に従った API（TX_関数）の使用の有無と作成されるクライアントエラーログファイルの関係を次の表に示します。

表 11–5　X/Open に従った API（TX_関数）の使用の有無と作成されるクライアントエラーログファイルの関係

TX_関数の使用	作成されるクライアントエラーログファイル
なし	pderr1.trc，及び pderr2.trc
あり	pderr×××××-1.trc，及び pderr×××××-2.trc

（凡例）×××××：UAP 実行時のプロセス ID

(2)　クライアントエラーログ情報の見方

クライアントエラーログは，SQL 文実行時，通信処理時，又は X/Open で規定した XA インタフェース関数実行時でエラーが発生したときに出力されます。

出力されるクライアントエラーログの例とその説明を次に示します。

[出力例]

```
 1. 2.  3.  4.  5.         6.                    7.
>  672  0  5223  0  1997/12/18 22:07:46  KFPZ02444-E Communication error,fu ect,
                                         errno=2
>  672  0  5223  1  1997/12/18 22:07:46  KFPZ02444-E Communication error,fu ect,
                                         errno=2
                                                  8.      9.    10.      11.     12.
>> 672  0  5223  2  1997/12/18 22:07:46  SQLCODE:-723  5  (205622412)  cltsql.c:  AUI2
KFPA11723-E Communication error occurred, reason=NETWORK  7.
```

[説明]

1. クライアントエラーログ先頭識別子
   SQL 実行でエラーが発生した場合には'>>'を，それ以外のエラーの場合には'>'を表示します。

2. UAP のプロセス番号

エラーが発生した UAP のプロセス番号を表示します。

なお，正しいプロセス番号が取得できない場合，不正な数値が表示される場合があります（Windows 版の場合）。

3. UAP のスレッド番号

   エラーが発生した UAP がマルチスレッドで動作している場合に UAP のスレッド番号を表示します。マルチスレッドで動作していない場合は 0 を表示します。正しいスレッド番号が取得できず，不正な数値が表示される場合があります。

4. サーバのプロセス番号

   接続しているサーバのプロセス番号を表示します。

5. クライアントエラーログカウンタ

   クライアントエラーログを受け付けるごとに順次カウントして表示します。

   0 から 65535 までカウントできます。

6. エラー取得日時

   クライアントエラーログを取得した日時を YYYY/MM/DD HH：MM：SS の形式で表示します。

7. ログデータ

   障害情報（エラーメッセージ）を表示します。

8. SQLCODE

   クライアントエラーログが UAP に返す SQLCODE に対応している場合に，その SQLCODE を表示します。

9. SQL カウンタ

   エラーが発生した SQL 文の SQL カウンタを表示します。SQL カウンタが 6 けたの場合，上 5 けたを表示します。SQL カウンタの詳細については，「SQL トレース機能」の出力例の説明を参照してください。

10. エラー取得時間

    クライアントエラーログを取得した時間をミリ秒単位で表示します。

11. エラー検出箇所

    エラーを検出したソースファイルの名前と行番号を表示します。

12. オペレーションコード

    エラーが発生した SQL 文のオペレーションコードを表示します。

(3)　クライアントエラーログファイルのバックアップの取得

クライアントエラーログ情報を出力中にクライアントエラーログファイルの容量が一杯になると，HiRDB はそのファイルへ出力しないで，もう一方のクライアントエラーログファイルにクライアントエラーログ情報を出力します。このとき，切り替え先のクライアントエラーログファイルに格納されている古いクライアントエラーログ情報から順に消去され，新しいクライアントエラーログ情報に書き換えられます。こ

のため，必要な情報は UAP 終了時にクライアントエラーログファイルの内容をコピーしてバックアップを取得しておいてください。

なお，現在使用しているクライアントエラーログファイルを知りたい場合は，ファイルの最終更新日時を調べてください。最終更新日時の新しい方が現在使用しているクライアントエラーログファイルになります。

HiRDB クライアントが Windows 版の場合は dir コマンド又はエクスプローラで，ファイルの最終更新日時を調べてください。

HiRDB クライアントが UNIX 版の場合は OS の ls -l コマンドで，ファイルの最終更新日時を調べてください。

11.1.3　拡張 SQL エラー情報出力機能

(1)　拡張 SQL エラー情報出力機能とは

拡張 SQL エラー情報出力機能とは，次の機能のことをいいます。

- クライアントエラーログ機能の情報（JDBC4.0 又は JDBC4.3 の Type4 JDBC ドライバの場合は，Exception トレースログのログファイル）に，SQL 文，及びパラメタ情報を出力する（クライアントエラーログ機能の情報に，SQL 文，及びパラメタ情報を加えた情報を **SQL エラー情報**といいます）。
- サーバ側にも SQL エラー情報を出力する（SQL エラー情報を出力するファイルを **SQL エラーレポートファイル**といいます）。

(2)　利点

- **SQL エラー情報の一元管理**
  SQL エラーとなった場合，クライアント側だけではなくサーバ側にも SQL エラー情報を出力します。複数のクライアントの SQL エラー情報を，1 サーバの SQL エラーレポートファイルに出力するため，SQL エラー情報を一元管理できます。
- **SQL 文及びパラメタ情報の出力**
  エラーとなった SQL 文，及びパラメタ情報を出力します。これらの情報から，エラーとなった SQL を調査できます。

(3)　使用方法

拡張 SQL エラー情報出力機能を使用する場合，次のシステム定義，又はクライアント環境定義を設定します。

- **拡張 SQL エラー情報出力機能の使用有無**
  pd_uap_exerror_log_use オペランド，又は PDUAPEXERLOGUSE を設定します。HiRDB 全体に対して指定する場合は pd_uap_exerror_log_use オペランドを，アプリケーションごとに指定する場合は PDUAPEXERLOGUSE を指定します。

- **SQL エラーレポートファイルの出力先ディレクトリと最大容量**

  SQL エラーレポートファイルの出力先ディレクトリは，pd_uap_exerror_log_dir オペランドで設定します。また，SQL エラーレポートファイルの最大容量は pd_uap_exerror_log_size オペランドで設定します。

- **クライアントエラーログファイル及び SQL エラーレポートファイルに出力するパラメタ情報の最大データ長**

  pd_uap_exerror_log_param_size オペランド，又は PDUAPEXERLOGPRMSZ を設定します。HiRDB 全体に対して指定する場合は pd_uap_exerror_log_param_size オペランドを，アプリケーションごとに指定する場合は PDUAPEXERLOGPRMSZ を指定します。

(4) SQL エラー情報の見方

(a) SQL エラーレポートファイルの出力形式

SQL エラーレポートファイルの出力形式を次に示します。

[出力形式]

```
** UAP ERROR INFORMATION aa....aa bbbbbbbbbbbbbbbbbbbbbbbbb ** [1]

* UAP INFORMATION * [2]
  UAP_NAME(cc....cc) USERID(dd....dd)
  IPADDR(ee....ee) CLTPID(ff....ff) THRDID(gg....gg)
  START_TIME(hhhhhhhhhhhhhhhhhhh)

* SERVER INFORMATION * [3]
  HOST(ii....ii) PORT(jj....jj) PLATFORM(kk....kk)
  SVRNAME(ll....ll) SVRPID(mm....mm)

* SQL INFORMATION * [4]
  OPTIMIZE_LEVEL(nn....nn) ADDITIONAL_OPTIMIZE_LEVEL(oo....oo)
  ISOLATION_LEVEL(pp....pp)

CNCTNO     SQL-       OP   SEC  SQL   SQL   OP   ERROR
           COUNTER    CODE NO   CODE  WARN  TION COUNTER
---------- ---------- ---- ---- ----- ----- ---- -----
rrrrrrrrrr ssssssssss tttt uuuu vvvvv wwwww xxxx yyyyy

START-TIME      END-TIME        EXEC-TIME
--------------- --------------- -----------------
zzzzzzzzzzzzzzz AAAAAAAAAAAAAAA          BB....BB

* SQL MESSAGE * [5]
  "CC....CC" [DD....DD]

* SQL STATEMENT * [6]
  "EE....EE"

* PARAMETER * [7]
*ELM NO= FFFFF*
*GGGGG* NO=HHHHH COD=III XDIM=JJJJJ SYS=KKKKK LEN=LLLLLLLLLL IND=MMMMMMMMMM
```

```
        ARRAY NUM=NNNNN
      DATA=00....00
```

[説明]

1. SQL エラーレポートファイルのタイトル
2. UAP 情報
3. サーバ情報
4. SQL 情報
5. SQL メッセージ
6. SQL 文
7. パラメタ情報

aa....aa：

HiRDB のバージョンを次の形式で出力します（出力文字数は最大 8 バイト）。

"vv-rr-zz"

-zz がない場合，-zz は出力されません。

bbbbbbbbbbbbbbbbbbbbbbbbbb：

エラー情報を出力した日時を次の形式で出力します（出力文字数は最大 26 バイト）。

"YYYY/MM/DD hh:mm:ss.uuuuuu"

YYYY：年

MM：月

DD：日

hh：時

mm：分

ss：秒

uuuuuu：マイクロ秒※

cc....cc：

クライアント環境定義 PDCLTAPNAME で指定した UAP の名称を出力します（出力文字数は最大 30 バイト）。

dd....dd：

コネクトしたユーザの認可識別子を出力します（出力文字数は最大 30 バイト）。

ee....ee：

UAP の IP アドレスを出力します（出力文字数は最大 15 バイト）。

ff....ff：

UAP のプロセス番号を表示します（出力文字数は最大 10 バイト）。

正しいプロセス番号を取得できない場合，不正な数値が表示されることがあります（Windows 版の場合）。

gg....gg：

UAP がマルチスレッドで動作している場合，UAP のスレッド番号を表示します（出力文字数は最大 11 バイト）。マルチスレッドで動作していない場合は 0 を表示します。

正しいスレッド番号が取得できない場合，不正な数値が表示されることがあります。また，クライアントのバージョンが 07-01 以前の場合，"*"を出力します。

hhhhhhhhhhhhhhhhhhh：

UAP を実行した時刻を次の形式で出力します（出力文字数は最大 19 バイト）。

"YYYY/MM/DD hh:mm:ss"

YYYY：年

MM：月

DD：日

hh：時

mm：分

ss：秒

ii....ii：

サーバプロセスが動作しているホストの名称を出力します（出力文字数は最大 30 バイト）。

jj....jj：

サーバプロセスの通信ポート番号を出力します（出力文字数は最大 5 バイト）。

kk....kk：

クライアントライブラリが対応しているプラットフォームを出力します（出力文字数は最大 6 バイト）。

出力内容については，「SQL トレース情報の見方」の UAP が動作しているプラットフォームを参照してください。なお，クライアントのバージョンが 07-01 以前の場合，"*"を出力します。

ll....ll：

シングルサーバ又はフロントエンドサーバのサーバ名を出力します（出力文字数は最大 8 バイト）。

mm....mm：

サーバプロセスのプロセス番号を出力します（出力文字数は最大 10 バイト）。

nn....nn：

SQL 最適化オプションの値を 10 進数形式で出力します（出力文字数は最大 10 バイト）。

oo....oo：

SQL 拡張最適化オプションの値を 10 進数形式で出力します（出力文字数は最大 10 バイト）。

pp....pp：

データ保証レベルの値を出力します（出力文字数は最大 10 バイト）。

rrrrrrrrrr：

サーバが CONNECT を受け付けるごとに順次カウントするコネクト通番を出力します（出力文字数は最大 10 バイト）。コネクト通番は右詰めで出力し，余白には半角スペースが入ります。

ssssssssss：

SQL 文を受け付けるごとにカウントする SQL カウンタを出力します（出力文字数は最大 10 バイト）。SQL カウンタは右詰めで出力し，余白には半角スペースが入ります。

tttt：

SQL に対応するオペレーションコードを出力します（出力文字数は最大 4 バイト）。

uuuu：

SQL に対応するセクション番号を出力します（出力文字数は最大 4 バイト）。セクション番号は右詰めで出力し，余白には半角スペースが入ります。制御系 SQL 実行時にエラーが発生した場合，"****"を出力します。

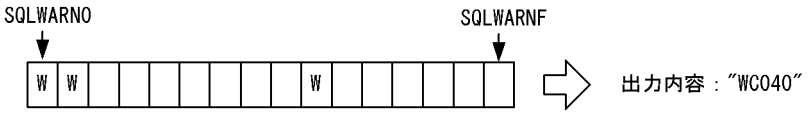
vvvvv：

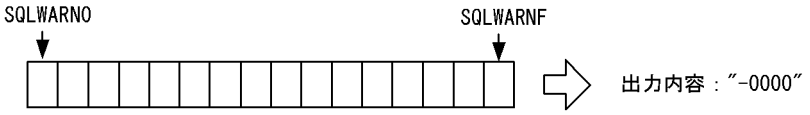
SQL を実行した結果の SQLCODE を出力します（出力文字数は最大 5 バイト）。SQLCODE は右詰めで出力し，余白には半角スペースが入ります。

wwwww：

警告情報を 16 進数形式で出力します（出力文字数は最大 5 バイト）。SQLWARN0～SQLWARNF の警告情報に，それぞれ左から 1 ビットを割り当て，警告フラグが設定されているものは 1，設定されていないものは 0 とします。これを 4 けたの 16 進数の値として出力します。一つ以上の警告フラグが設定されている場合は先頭に"W"を，設定された警告フラグがない場合は"-"を付けます。例を次に示します。

（例）





xxxx：

システムが使用する情報を出力します（出力文字数は最大 4 バイト）。

1 バイト目が"M"の場合，プロセス間メモリ通信機能を使用していることを示します。ほかの 3 バイトは，保守用の情報です。ただし，クライアントのバージョンが 07-01 以前の場合は，"****"を出力します。

yyyyy：

クライアントエラーログファイルに出力する同じ SQL 文に対応するクライアントエラーログカウンタの値を出力します（出力文字数は最大 5 バイト）。

この値は右詰めで出力し，余白には半角スペースが入ります。

ただし，クライアントのバージョンが 07-01 以前の場合は，"*****"を出力します。

zzzzzzzzzzzzzzz：

クライアントから SQL 実行要求を受けた時刻を，次の形式で出力します（出力文字数は最大 15 バイト）。

"hh:mm:ss.uuuuuu"

hh：時

mm：分

ss：秒

uuuuuu：マイクロ秒

AAAAAAAAAAAAAAA：

クライアントからの要求に対する処理が終了した時刻を，次の形式で出力します（出力文字数は最大 15 バイト）。

"hh:mm:ss.uuuuuu"

hh：時

mm：分

ss：秒

uuuuuu：マイクロ秒※

BB....BB：

クライアントからの要求に対する処理時間を，次の形式で出力します（出力文字数は最大 17 バイト）。秒は右詰めで出力し，余白には半角スペースが入ります。

"sssssssssss.uuuuuu"

sssssssssss：秒

uuuuuu：マイクロ秒※

CC....CC：

SQL 実行中に発生したメッセージを出力します（出力文字数は最大 254 バイト）。

DD....DD：

システムが使用する情報を出力します（出力文字数は最大 21 バイト）。

EE....EE：

SQL 文を出力します（出力文字数は最大 2000000 バイト）。

SQL 文中に注釈（コメント）や SQL 最適化指定を記述している場合，それらも含めて出力します。制御系 SQL 実行時にエラーとなった場合，"*"を出力します。注釈，及び SQL 最適化指定については，マニュアル「HiRDB SQL リファレンス」を参照してください。

FFFFF：

配列を使用した SQL でエラーが発生した場合，その要素の番号を出力します（出力文字数は最大 5 バイト）。

GGGGG：

入力用パラメタ情報の場合は INPRM，出力用パラメタ情報の場合は OUTRM を出力します。また，入出力用パラメタ情報の場合，入力情報は INPRM，出力情報は OUTRM を出力します（出力文字数は最大 5 バイト）。

HHHHH：

パラメタ番号を出力します（出力文字数は最大 5 バイト）。

III：

データ型コードを出力します（出力文字数は最大 3 バイト）。

JJJJJ：

配列要素数を出力します（出力文字数は最大 5 バイト）。

KKKKK：

ギャップを含む 1 要素の領域の長さを出力します（出力文字数は最大 5 バイト）。

LLLLLLLLLL：

データの長さを出力します（出力文字数は最大 11 バイト）。

MMMMMMMMMM：

標識変数の値を出力します（出力文字数は最大 11 バイト）。

NNNNN：

繰返し列を含む場合，繰返し列の要素数を出力します（出力文字数は最大 5 バイト）。繰返し列を含まない場合は出力されません。

OO....OO：

パラメタ情報を出力します（出力文字数は pd_uap_exerror_log_param_size オペランドの指定値）。パラメタ情報には入力用パラメタ情報，出力用パラメタ情報，及び入出力用パラメタ情報があります。パラメタ情報についての規則を次に示します。

- 入力用パラメタが BLOB 型の位置付け子，又は BINARY 型の位置付け子の場合，BLOB 型の位置付け子，又は BINARY 型の位置付け子の値を出力します。
- 標識変数がマイナスの値の場合，"DATA="まで出力します。
- パラメタ情報が複数ある場合，パラメタ情報の指定順に出力します。
- 同様のデータが 2 行以上続く場合，"--- SAME x LINES ---"（x が行数）を出力します。
- パラメタ情報は，実データ長＋データ領域長サイズ分出力します。
- 繰返し列の場合，ARRAY NUM に繰返し列の要素数を出力します。
- 繰返し列の場合，"DATA"の前に繰返し要素ごとの標識変数を出力します。
- 繰返し列の場合，"DATA"の後に括弧付きで繰返し列の要素番号を出力します。

注※

PDUAPEXERLOGUSE を省略し，pd_uap_exerror_log_use が AUTO の場合，「uuuuuu：マイクロ秒」を出力しません。

(b)　クライアントエラーログファイルの出力形式

拡張 SQL エラー情報出力機能を使用した場合の，クライアントエラーログファイルの出力形式を次に示します。

[出力形式]

```
>  8355     0  8393     9 2005/08/12 14:06:30 KFPZ03000-I Error information, type=CONNECT ST
ATUS,
 inf=CLT=07-02(Aug  4 2005):WS SVR=07-02   US:WS LIBTYPE=UNIX_32
>  8355     0  8393    10 2005/08/12 14:06:30 KFPZ03000-I Error information, type=SQL STREAM
,
 inf=insert into t1 values ( ? , ? ,? )
>> 8355     0  8393    11 2005/08/12 14:06:30 SQLCODE:-404    47(140630218) sqaexp0.c   :234
8 AUX
 KFPA11404-E Input data too long for column or assignment target in variable 3
 UAP userprog1,hiuser01 [1]
 SVR host03,1146,sds,hp [2]
 SQLINF 1034,1,2,7,17,-0000,0000,14:06:30.216463,14:06:30.217765,0.001302 [3]
 SQL INSERT INTO T1 VALUES(?,?,?) [4]
 PRM [5]
 INPRM 1,f1,1,0,4,0
         DATA=00 00 ff ff                                        *....            *
 INPRM 2,c1,10,258,255,9
         ARRAY NUM=    9
       0 DATA(    0)=00 01 61                                          *..a            *
       0 DATA(    1)=00 02 61 62                                       *..ab           *
       0 DATA(    2)=00 03 61 62 63                                    *..abc          *
       0 DATA(    3)=00 04 61 62 63 64                                 *..abcd         *
       0 DATA(    4)=00 05 61 62 63 64 65                              *..abcde        *
       0 DATA(    5)=00 06 61 62 63 64 65 66                           *..abcdef       *
       0 DATA(    6)=00 07 61 62 63 64 65 66 67                        *..abcdefg      *
       0 DATA(    7)=00 08 61 62 63 64 65 66 67 68                     *..abcdefgh     *
      -1 DATA(    8)=
 INPRM 3,93,1,0,32002,0
         DATA=00 00 00 00 00 00 7d 02 41 41 41 41 41 41 41 41   *......}.AAAAAAAA*
              41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41   *AAAAAAAAAAAAAAAA*
              --- SAME 14 LINES ---
```

[説明]

1.UAP 情報

UAP 名称：

クライアント環境定義 PDCLTAPNAME で指定した UAP の名称を出力します。

認可識別子：

コネクトしたユーザの認可識別子を出力します。

2.サーバ情報

ホスト名：

サーバプロセスが動作しているホストの名称を出力します。

ポート番号：

サーバプロセスの通信ポート番号を出力します。

サーバ名：

シングルサーバ又はフロントエンドサーバのサーバ名を出力します。

プラットフォーム：

クライアントライブラリが対応しているプラットフォームを出力します。

出力内容については，「SQL トレース情報の見方」の UAP が動作しているプラットフォームを参照してください。なお，クライアントのバージョンが 07-01 以前の場合，"*"を出力します。

3.SQL 情報

SQL 最適化オプション：

SQL 最適化オプションの値を 10 進数形式で出力します。

SQL 拡張最適化オプション：

SQL 拡張最適化オプションの値を 10 進数形式で出力します。

データ保証レベル：

データ保証レベルの値を出力します。

コネクト通番：

サーバが CONNECT を受け付けるごとに順次カウントするコネクト通番を出力します。

セクション番号：

SQL に対応するセクション番号を出力します。

SQLWARN：

警告情報を 16 進数形式で出力します。SQLWARN0～SQLWARNF の警告情報に，それぞれ左から 1 ビットを割り当て，警告フラグが設定されているものは 1，設定されていないものは 0 とします。これを 4 けたの 16 進数の値として出力します。一つ以上の警告フラグが設定されている場合は先頭に"W"を，設定された警告フラグがない場合は"-"を付けます（例については，「SQL エラーレポートファイルの出力形式」の wwwww を参照してください）。

システム情報：

システムが使用する情報を出力します。

1 バイト目が"M"の場合，プロセス間メモリ通信機能を使用していることを示します。ほかの 3 バイトは，保守用の情報です。ただし，クライアントのバージョンが 07-01 以前の場合は，"****"を出力します。

SQL 開始時刻：

クライアントから SQL 実行要求を受けた時刻を，次の形式で出力します。

"hh:mm:ss.uuuuuu"

hh：時

mm：分

ss：秒

uuuuuu：マイクロ秒※

SQL 終了時刻：

クライアントからの要求に対する処理が終了した時刻を，次の形式で出力します。

"hh:mm:ss.uuuuuu"

hh：時

mm：分

ss：秒

uuuuuu：マイクロ秒※

SQL 実行時間：

クライアントからの要求に対する処理時間を，次の形式で出力します。秒は右詰めで出力し，余白には半角スペースが入ります。

"ssssssssss.uuuuuu"

ssssssssss：秒

uuuuuu：マイクロ秒※

4.SQL 文

SQL 文：

SQL 文を出力します。

SQL 文中に注釈（コメント）や SQL 最適化指定を記述している場合，それらも含めて出力します。出力する SQL 文のサイズは，クライアント環境定義の PDSQLTEXTSIZE の指定値になります。

制御系 SQL 実行時にエラーとなった場合，SQL 文は取得できません。この場合，"*"を出力します。

注釈，及び SQL 最適化指定については，マニュアル「HiRDB SQL リファレンス」を参照してください。

5.パラメタ情報

ELM NO：

配列を使用した SQL でエラーが発生した場合，その要素の番号を出力します。

パラメタ情報種別：

入力用パラメタ情報の場合は INPRM，出力用パラメタ情報の場合は OUTRM を出力します。また，入出力用パラメタ情報の場合，入力情報は INPRM，出力情報は OUTRM を出力します。

NO：

パラメタ番号を出力します。

COD：

データ型コードを出力します。

XDIM：

配列要素数を出力します。

SYS：

ギャップを含む 1 要素の領域の長さを出力します。

LEN：

データの長さを出力します。

IND：

標識変数の値を出力します。

ARRAY NUM：

繰返し列を含む場合，繰返し列の要素数を出力します。繰返し列を含まない場合は出力されません。

DATA：

パラメタ情報を出力します。

パラメタ情報には入力用パラメタ情報，出力用パラメタ情報，及び入出力用パラメタ情報があります。

パラメタ情報についての規則を次に示します。

- 入力用パラメタが BLOB 型の位置付け子，又は BINARY 型の位置付け子の場合，BLOB 型の位置付け子，又は BINARY 型の位置付け子の値を出力します。
- 標識変数がマイナスの値の場合，"DATA="まで出力します。
- パラメタ情報が複数ある場合，パラメタ情報の指定順に出力します。
- 同様のデータが 2 行以上続く場合，"--- SAME x LINES ---"（x が行数）を出力します。
- パラメタ情報は，実データ長＋データ領域長サイズ分出力します。
- 繰返し列の場合，ARRAY NUM に繰返し列の要素数を出力します。
- 繰返し列の場合，"DATA"の前に繰返し要素ごとの標識変数を出力します。
- 繰返し列の場合，"DATA"の後に括弧付きで繰返し列の要素番号を出力します。

注※

PDUAPEXERLOGUSE を省略し，pd_uap_exerror_log_use が AUTO の場合，「uuuuuu：マイクロ秒」を出力しません。

(5) SQL エラーレポートファイルについての規則

SQL エラーレポートファイルについての規則を次に示します。なお，SQL エラーレポートファイルを参照する場合は，テキストエディタなどを使用してください。

1. SQL を実行し，エラーの発生を検知するたびに SQL エラーレポートファイルのオープン，SQL エラー情報の書き込み，クローズをします。SQL エラーレポートファイルの最終位置から追加書き込みをするため，SQL エラー情報はファイル内で時系列順に蓄積されます。
2. SQL エラーレポートファイルは二つ作成されます（pduaperrlog1 及び pduaperrlog2）。現在書き込んでいるファイルのサイズが，システム定義の pd_uap_exerror_log_size オペランドの指定値を超えた場合に，もう一方のファイルに出力先を切り替えます。切り替わったファイルでも，これを繰り返しな

がら二つのファイルを交互に使用します（切り替え先の古い内容は削除されます）。HiRDB 開始後は，ファイルの最終更新日付の新しい方が出力先となります。

3. SQL 終了後は，SQL エラーレポートファイルはクローズしているため，SQL が実行されていないときに OS のコマンドを利用してバックアップを取得したり，ファイルを参照したりできます。また，SQL 実行中であっても出力先でないもう一方のファイルはバックアップ取得やファイルの参照ができます。
4. 現在使用中の SQL エラーレポートファイルは，OS の dir コマンド（UNIX の場合は ls -l コマンド）でファイルの最終更新日付を調べることで知ることができます。このとき，最終更新日時の新しい方が現在使用中の SQL エラーレポートファイルとなります。

(6) 注意事項

1. PDUAPEXERLOGUSE，又は pd_uap_exerror_log_use に YES を指定して拡張 SQL エラー情報出力機能を使用すると，SQL エラー情報を出力しない場合でも，SQL の開始時間と実行時間を採取するためにシステムコールを実行する時間が必要になります。性能への影響を最小限にしたい場合は，PDUAPEXERLOGUSE を省略し，pd_uap_exerror_log_use を AUTO にしてください。
2. クライアントエラーログファイル又は SQL エラーレポートファイルへの出力時に OS がエラーを検知した場合（ファイルシステム障害，ファイルの書き込み権限がないなど），クライアントエラーログファイル又は SQL エラーレポートファイルに SQL エラー情報は出力されません。
3. 拡張 SQL エラー情報出力機能を使用すると，パラメタ情報を出力するためのメモリが必要になります。

11.1.4 UAP 統計レポート機能

UAP 実行時の UAP 統計レポートを UAP 統計レポートファイルに出力します。

(1) UAP 統計レポートの取得方法

UAP 統計レポートは，クライアント環境定義の PDCLTPATH，PDSQLTRACE，及び PDUAPREPLVL に値を設定すると取得できます。

UAP 統計レポートファイルの出力先及びファイル名について説明します。

- 出力先

  PDCLTPATH で指定したディレクトリ下に二つ出力されます。PDREPPATH を指定すると，PDCLTPATH で指定したディレクトリとは別のディレクトリに出力されます。
- ファイル名

  出力ファイル名は「SQL トレース機能」を参照してください。

取得する情報は，クライアント環境定義の PDUAPREPLVL で設定できます。複数のレベルを設定した場合，指定したレベルに対応する複数の情報が取得できます。PDUAPREPLVL の指定値と取得する情報の関係を次の表に示します。

表 11–6　PDUAPREPLVL の指定値と取得する情報の関係

PDUAPREPLVL の指定値	取得する情報				
	SQL 単位の情報	UAP 単位の情報※1		アクセスパス情報	SQL 実行時の中間結果情報
		右記以外の情報	スレッド間ロック待ち時間情報		
s※2	○	−	−	−	−
u	−	○	−	−	−
uo	−	○	○	−	−
p	−	−	−	○	−
r	−	−	−	−	○
a 又は supr	○	○	−	○	○
ao 又は suopr	○	○	○	○	○

（凡例）

○：情報が取得されます。

−：情報が取得されません。

注※1

t を指定した場合は，UAP 単位の情報をトランザクション単位に集計して出力します。

注※2

s を指定した場合は，SQL トレース情報も取得されます。

- **注意事項**

1. アクセスパス情報，又は SQL 実行時の中間結果情報を取得する場合，SQL オブジェクトがバッファ中にあっても SQL オブジェクトを再作成するため，サーバの負荷が増えることがあります。
2. スレッド間ロック待ち時間情報を取得する場合，システム全体の性能に影響を与えるおそれがあります。通常の運用ではスレッド間ロック待ち時間情報を取得しないでください。
3. 次の場合は，UAP 単位の情報は出力されません。
   ・OLTP 下の X/Open に従った API を使用するプログラムの場合
   ・DISCONNECT をしないで UAP が終了した場合
4. アクセスパス情報，及び SQL 実行時の中間結果情報は，1 ギガバイトを超えると出力されません。
5. 時間の表示（SQL の実行時間，排他待ち時間，CPU 時間など）は，OS のシステムコールで取得できない小さい値があると，0 が表示されます。
6. HiRDB/パラレルサーバの場合，CONNECT したディクショナリサーバでの権限チェック処理は，UAP 単位の情報には含まれません。

7. アクセスパス情報，又は SQL 実行時の中間結果情報の出力を指定して，プロセス間メモリ通信機能を指定した場合（クライアント環境定義 PDIPC=MEMORY を指定した場合），PDIPC の指定は無効となり PDIPC=DEFAULT となります。

- **SQL トレースファイルの容量**

SQL トレースファイルの容量は，次の計算式から求められます。なお，SQL 文長は環境変数 PDSQLTEXTSIZE で変更できます。

```
SQLトレースファイルの容量
  ＝3208＋A＋80×オペレーション数＋SQL文長(最大4096)の総和
    （単位：バイト）
```

A：

クライアント環境定義の PDHOST，PDFESHOST，PDSQLOPTLVL，PDADDITIONALOPTLVL，PDREPPATH，及び PDTRCPATH の指定文字列長の合計

また，SQL 単位情報，UAP 単位情報，アクセスパス情報，及び SQL 実行時の中間結果情報を出力する場合は，次のサイズ（単位：バイト）も加算してください。

SQL 単位情報：

83※×SQL 数

UAP 単位情報：

2740※×DISCONNECT 数

アクセスパス情報：

「アクセスパス情報」を参照してください。

SQL 実行時の中間結果情報：

「SQL 実行時の中間結果情報」を参照してください。

注※

最大値です。表示するけた数で値は変わります。

(2) UAP 統計レポートの見方

出力される UAP 統計レポートの例を次に示します。また，その説明を(a)～(d)に示します。

[クライアント環境定義 PDSQLTRCFMT に 1 を指定した出力例（出力形式 1）]

```
 CNCT  CLPID  CLTID  NO  OP    SEC  SQL   SQL    START-TIME    END-TIME      OP
 NO                      CODE  NO   CODE  WARN                             TION
 ----  -----  -----  --  ----  ---  ----  -----  ------------  ------------  ----
    1   9155      0   1  CNCT    0     0  WC040  16:03:55.720  16:03:58.080  0001
    1   9155      0   2  AUI2    1     0  -0000  16:03:58.630  16:03:59.400  0000

*SQL*  INSERT INTO T1(C1,C2,C3,C4,C5,C6) VALUES(?,?,?,?,?,?)
00:00:00.770    00:00:00.430000    340    1    0    0    0    0    0 ........(a)
    [1]              [2]          [3]   [4] [5] [6] [7] [8] [9]
```

```
    1   9155      0   3  SET     2     0  -0000  16:04:00.820  16:04:01.540  0000
*SQL*  SELECT * from T1, T2, T3 where ((T1.C1='a' and T1.C2='A')
 or (T1.C1='a' and T1.C2='B')) and T1.C1=T2.C1 and T1.C2=T2.C2 and T2.C3>=1995
 and T1.C1=T3.C1 and T1.C2=T3.C2 order by T1.C1

00:00:00.720    00:00:00.240000    480    1     0    0    0

Result of SQL Optimizer :   .....................................(b)
 Connect No     :   1
 ------------------------------------------------------------------------------
 Section No     : 2
 UAP Source     :XXXXXXXX.ec
 Optimize Mode  : COST_BASE_2
 SQL Opt Level  : 0x00000420(1056) = "PRIOR_NEST_JOIN"(32),"RAPID_GROUPING"(1024)
 Add Opt Level  : 0x00000003(3) = "COST_BASE_2"(1),"APPLY_HASH_JOIN"(2)
 Work Table     : 0
 Total Cost     : 12672.66944
 ----- QUERY EXPRESSION BODY ID : 1 -----
              :
 ----- QUERY ID : 1 -----
              :
 JOIN
   :
 SCAN
   :

 ------------------------------------------------------------------------------

    1   9155      0   4  OPEN    2     0  -0000  16:04:02.090  16:04:02.800  0000
Result of SQL Execution :   .....................................(c)
------------------------------------------------------------------------------
Connect No     :    1
UAP Source     : XXXXXXXX.ec
Section No     : 2
----- QUERY EXPRESSION BODY ID : 1 -----
   :
----- QUERY ID : 1 -----
   :
JOIN
   :
SCAN
   :
------------------------------------------------------------------------------

    1   9155      0   9  DISC    0     0  -0000  16:05:55.110  16:05:56.660  0004

UAP INFORMATION:   ..............................................(d)
 [1]UAPNAME()
 [2]SVHOST(dcm3500) [3]SVPORT(4439)  [4]SVNAME(fes1)   [5]CNCTNO(1)
 [6]SVPID(8945)     [7]CLPID(9155)     [8]CLTTID(0)
 [9]WAITT(0)       [10]CTIME(0)
 [11]ROREQ(0)      [12]ROHITS(0)
 [13]SOREQ(10)     [14]SOHITS(3)   [15]SOCRT(0)       [16]SOMAX(0)
 [17]COMT(0)       [18]ROLB(0)     [19]FROW(0)    [20]DROW(0)     [21]IROW(3)
 [22]UROW(0)       [23]SET(1)     [24]OPEN(2)     [25]FETC(1)     [26]CLOS(0)
 [27]DESC(0)       [28]SEL(1)     [29]INS(3)      [30]UPD(0)      [31]DEL(0)
 [32]LOCK(0)      [33]CRTT(0)     [34]DRPT(0)     [35]ALTT(0)     [36]CRTI(0)
```

```
 [37]DRPI(0)     [38]CMTT(0)      [39]CMTC(0)     [40]CRTS(0)     [41]DRPS(0)
 [42]GRTR(0)     [43]GRTS(0)      [44]GRTA(0)     [45]GRTC(0)     [46]GRTD(0)
 [47]RVKR(0)     [48]RVKS(0)      [49]RVKA(0)     [50]RVKC(0)     [51]RVKD(0)
 [52]CRTV(0)     [53]DRPV(0)      [54]PRGT(0)     [55]CRTP(0)     [56]DRPP(0)
 [57]ALTP(0)     [58]CALL(0)      [59]DESI(0)     [60]MISC(0)
 [61]MAXIO(0)    [62]MAXIOM(0)    [63]MINIO(0)    [64]MINIOM(0)
 [65]IOTIM(0)    [66]IOTIMM(0)
 [67]DIDRC(0)    [68]DIDUC(0)     [69]DIDHC(0)    [70]DIDRD(0)    [71]DIDWT(0)
 [72]LBRFC(0)    [73]LBUPC(0)     [74]LBRHC(0)    [75]LBUHC(0)    [76]LBRDC(0)
 [77]LBWTC(0)    [78]BFSHC(2320)  [79]BRDWC(0)    [80]BWTWC(50)
 [81]BLKWC(2)    [82]MWFN(0)      [83]MWFEC(0)    [84]MWFVL(0)
 [85]WFRDC(0)    [86]WFWTC(0)     [87]WBFOC(0)
 [88]MWHTS(0)    [89]MBSL1(0)     [90]MBSL2(0)    [91]MBSL3(0)
 [92]SCHSKD(0)   [93]SCHCHG(0)
 [94]CINSM(0)    [95]CAFLS(0)     [96]CAFWR(0)    [97]CFMAX(0)    [98]CFAVG(0)
 [99]LDIRC(0)   [100]LDIUC(0)    [101]LDIHC(0)   [102]LDIRD(0)
[103]LDIWT(0)   [104]LBFSHC(0)
[105]ARREQ(0)   [106]ARWC(0)     [107]ARWT(0)    [108]ARWTM(0)
[109]ARWTA(0)   [110]ARWTMA(0)   [111]ARSTA(0)   [112]ARSTMA(0)
[113]HJMAX(0)   [114]HJCMC(0)    [115]HJHTC(0)
[138]LCKCTC(0)  [139]LCKCWC(0)  [140]LCKCWT(0)  [141]LCKCTM(0)
[142]LCKCWQ(0)  [143]LCKCQM(0)  [144]JNLFTC(0)  [145]JNLFWC(0)
[146]JNLFWT(0)  [147]JNLFTM(0)  [148]JNLFWQ(0)  [149]JNLFQM(0)
[150]BUFHTC(0)  [151]BUFHWC(0)  [152]BUFHWT(0)  [153]BUFHTM(0)
[154]BUFHWQ(0)  [155]BUFHQM(0)
[156]TUSEG(0)
[157]COMPC(0)  [158]COMPT(0)  [159]COMPTM(0)
[160]CMPMT(0)  [161]CMPMTM(0)
[162]EXTEC(0)  [163]EXTET(0)  [164]EXTETM(0)
[165]EXTMT(0)  [166]EXTMTM(0)
```

[クライアント環境定義 PDSQLTRCFMT に 2 を指定した出力例（出力形式 2）(1/2)]

```
CNCT  CLPID      CLTID      NO     OP   SEC  SQL      SQL   START-TIME              END-T
IME                OP     EXEC-TIME   TRACE     PREPROCESS FILE                PREPROCESS
  SQL INFORMATION↓
NO                                  CODE NO   CODE     WARN
                    TION              OUTPUT                                    TIME
  SQL EXEC-TIME   SVR EXEC-TIME   DIFF        EXEC   CREATE DELETE SQLOBJ HJCMP  HJSCH↓

                                      TIME
                                             COUNT  COUNT  COUNT  SIZE   COUNT  COUNT↓
 [a]


    [1]            [2]            [3]         [4]    [5]    [6]    [7]    [8]    [9]
----- ---------- ---------- ------ ---- ---- -------- ----- ---------------------- -----
----------------- ------ ----------- --------- ------------------------------ ----------
- --------------- --------------- ----------- ------ ------ ------ ------ ------ ------↓
   10       2764       3968      1 CNCT    0        0 -0000 2013/11/13 15:20:14.538 2013/
11/13 15:20:14.569 000000       35502         * *
*↓
   10       2764       3968      5 AUI2    4        0 -0000 2013/11/13 15:20:14.928 2013/
11/13 15:20:14.944 000000       23362        44 test.ec                          138432782
0 00:00:00.023362 00:00:00.016000        7362      1      0      0   2624      0      0↓
     *SQL* INSERT INTO T1 VALUES(100)↓
     *SQLLEN* 26↓
```

```
   10       2764       3968      8 SET     6        0 -0000 2013/11/13 15:20:14.959 2013/
11/13 15:20:14.959 000000         578         1 test.ec                          138432782
0 00:00:00.000578 00:00:00.000000         578      0      0      0   3024      0      0↓
     *SQL* SELECT * FROM T1↓
     *SQLLEN* 16↓
Result of SQL Optimizer :↓  [b]
Connect No     :   10↓
--------------------------------------------------------------------------------↓
Section No     : 4097↓
UAP Source     : test.ec↓
Optimize Mode  : COST_BASE_2↓
SQL Opt Level  : 0x000a07a0(657312) = "PRIOR_NEST_JOIN"(32),"PRIOR_OR_INDEXES"(128),"SORT
_DATA_BES"(256),"DETER_AND_INDEXES"(512),"RAPID_GROUPING"(1024),"DETER_WORK_TABLE_FOR_UPD
ATE"(131072),"APPLY_ENHANCED_KEY_COND"(524288)↓
Add Opt Level  : 0x00000001(1) = "COST_BASE_2"(1)↓
Work Table     : 0↓
Total Cost     : 22023.06397924731↓
----- QUERY ID : 1 -----↓
Query Type     : QUERY↓
SCAN↓
 # Table Name  : T1 0x0002007a(131194)↓
   Cost        : N (10000000ROW)  {T-21865.88711410221}↓
   RDAREA      : NON DIVISION  (1RD/1BES) [0x07(7)] ALL↓
   Scan Type   : TABLE SCAN↓
--------------------------------------------------------------------------------↓
START-TIME : 2013/11/13 15:20:14.959↓
END-TIME   : 2013/11/13 15:20:14.959↓

   10       2764       3968     12 OPEN   10        0 -0000 2013/11/13 15:20:14.959 2013/
11/13 15:20:14.959 000000           8          0 test.ec                          13843278
20↓
     *EXCURSOL* CUR1↓
   10       2764       3968     13 FETC   11        0 -0000 2013/11/13 15:20:14.959 2013/
11/13 15:20:14.975 000000       11287         21 *
 * 00:00:00.011287 00:00:00.016000           0      1      0      0      0      0      0
↓
     *EXCURSOL* CUR1↓
   10       2764       3968     14 CLOS   12        0 -0000 2013/11/13 15:20:14.975 2013/
11/13 15:20:14.975 000000         269          0 test.ec                          13843278
20 00:00:00.000269 00:00:00.000000         269      0      0      0      0      0      0
↓
     *EXCURSOL* CUR1↓
Result of SQL Execution :↓  [c]
--------------------------------------------------------------------------------↓
Connect No     :   10↓
UAP Source     : test.ec↓
Section No     : 4097↓
----- QUERY ID : 1 -----↓
Query          : 1 ROWS↓
SCAN↓
 # Table Name  : T1 0x0002007a(131194) ↓
   Row Count   : 1 ROWS↓
--------------------------------------------------------------------------------↓
START-TIME : 2013/11/13 15:20:14.975↓
END-TIME   : 2013/11/13 15:20:14.975↓
```

[クライアント環境定義 PDSQLTRCFMT に 2 を指定した出力例（出力形式 2）（2/2）]

```
10       2764       3968     19 DISC    0        0 -0000 2013/11/13 15:20:14.991 2013/11/
13 15:20:14.991 000000         434          0 *                                          *
↓
UAP INFORMATION :↓  [d]
[1]UAPNAME(Unknown)↓
[2]SVHOST(k00b701) [3]SVPORT(22210) [4]SVNAME(fes1) [5]CNCTNO(10)↓
[6]SVPID(2836) [7]CLPID(2764) [8]CLTTID(3968)↓
[9]WAITT(0) [10]CTIME(0)↓
[11]ROREQ(0) [12]ROHITS(0 ↓
[13]SOREQ(0) [14]SOHITS(0) [15]SOCRT(3) [16]SOMAX(3024)↓
[17]COMT(2) [18]ROLB(0) [19]FROW(1) [20]DROW(0) [21]IROW(1)↓
[22]UROW(0) [23]SET(2) [24]OPEN(1) [25]FETC(1) [26]CLOS(1)↓
[27]DESC(1) [28]SEL(0) [29]INS(1) [30]UPD(0) [31]DEL(0)↓
[32]LOCK(0) [33]CRTT(1) [34]DRPT(1) [35]ALTT(0) [36]CRTI(0)↓
[37]DRPI(0) [38]CMTT(0) [39]CMTC(0) [40]CRTS(0) [41]DRPS(0)↓
[42]GRTR(0) [43]GRTS(0) [44]GRTA(0) [45]GRTC(0) [46]GRTD(0)↓
[47]RVKR(0) [48]RVKS(0) [49]RVKA(0) [50]RVKC(0) [51]RVKD(0)↓
[52]CRTV(0) [53]DRPV(0) [54]PRGT(0) [55]CRTP(0) [56]DRPP(0)↓
[57]ALTP(0) [58]CALL(0) [59]DESI(1) [60]MISC(0)↓
[61]MAXIO(0)  [62]MAXIOM(0)  [63]MINIO(0)  [64]MINIOM(0)↓
[65]IOTIM(0)  [66]IOTIMM(0)↓
[67]DIDRC(0)  [68]DIDUC(0)  [69]DIDHC(0)  [70]DIDRD(0)  [71]DIDWT(0)↓
[72]LBRFC(0)  [73]LBUPC(0)  [74]LBRHC(0)  [75]LBUHC(0)  [76]LBRDC(0)↓
[77]LBWTC(0)  [78]BFSHC(2320)  [79]BRDWC(0)  [80]BWTWC(50)↓
[81]BLKWC(2)  [82]MWFN(0)  [83]MWFEC(0)  [84]MWFVL(0)↓
[85]WFRDC(0)  [86]WFWTC(0)  [87]WBFOC(0)↓
[88]MWHTS(0)  [89]MBSL1(0)  [90]MBSL2(0)  [91]MBSL3(0)↓
[92]SCHSKD(0)  [93]SCHCHG(0)↓
[94]CINSM(0)  [95]CAFLS(0)  [96]CAFWR(0)  [97]CFMAX(0)  [98]CFAVG(0)↓
[99]LDIRC(0)  [100]LDIUC(0)  [101]LDIHC(0)  [102]LDIRD(0)↓
[103]LDIWT(0)  [104]LBFSHC(0)↓
[105]ARREQ(0)  [106]ARWC(0)  [107]ARWT(0)  [108]ARWTM(0)↓
[109]ARWTA (0)  [110]ARWTMA (0)  [111]ARSTA (0)  [112]ARSTMA (0)↓
[113]HJMAX(0)  [114]HJCMC(0)  [115]HJHTC(0)↓
[138]LCKCTC(0)  [139]LCKCWC(0)  [140]LCKCWT(0)  [141]LCKCTM(0)↓
[142]LCKCWQ(0)  [143]LCKCQM(0)  [144]JNLFTC(0)  [145]JNLFWC(0)↓
[146]JNLFWT(0)  [147]JNLFTM(0)  [148]JNLFWQ(0)  [149]JNLFQM(0)↓
[150]BUFHTC(0)  [151]BUFHWC(0)  [152]BUFHWT(0)  [153]BUFHTM(0)↓
[154]BUFHWQ(0)  [155]BUFHQM(0)↓
[156]TUSEG(0)↓
[157]COMPC(0)  [158]COMPT(0)  [159]COMPTM(0)↓
[160]CMPMT(0)  [161]CMPMTM(0)↓
[162]EXTEC(0)  [163]EXTET(0)  [164]EXTETM(0)↓
[165]EXTMT(0)  [166]EXTMTM(0)↓
START-TIME : 2013/11/13 15:20:14.991↓
END-TIME   : 2013/11/13 15:20:14.991↓
```

（凡例）

↓：出力例（出力形式 2）での 1 行の末尾を示します。実際に出力するファイルでは表示しません。

(a) SQL 単位の情報

1. SQL 実行時間（単位：マイクロ秒）

SQL の実行時間を，HH:MM:SS.mmmmmm の形式で表示します。クライアント環境定義 PDSQLEXECTIME に NO を指定した場合，単位はミリ秒となります。

2. サーバ側での SQL 実行時間（単位：マイクロ秒）
   サーバ側での SQL の実行時間を，HH:MM:SS.mmmmmm の形式で表示します。

3. 1-2 の差分（単位：マイクロ秒）
   通信処理の時間の目安となります。クライアント環境定義 PDSQLEXECTIME に NO を指定した場合，単位はミリ秒となります。

4. 処理行数
   この接続中に発行した SQL 文で処理した，行数を表示します。

5. 作業表作成回数
   この接続中に発行した SQL 文の内部処理で作成した，作業表の作成回数を表示します。

6. 作業表削除回数
   この接続中に発行した SQL 文の内部処理で削除した，作業表の削除回数を表示します。

7. SQL オブジェクトサイズ（単位：バイト）
   この接続中に発行した SQL 文で作成した，SQL オブジェクトの大きさを表示します。

8. ハッシュジョイン，副問合せのハッシュ実行でのハッシュ表探索時の総比較回数
   この接続中に発行した SQL 文での，ハッシュ表探索時の同一ハッシュ値を持つデータに対する比較回数の合計です。

9. ハッシュジョイン，副問合せのハッシュ実行での総ハッシュ表探索回数
   この接続中に発行した SQL 文で，ハッシュ表を探索した回数です。

(b) アクセスパス情報

アクセスパス情報を表示します。Connect No には，コネクト通番を表示します。コネクト通番を基に上方向にサーチすることで，SQL トレース情報中に表示されている SQL 文を特定できます。動的 SQL の場合はコネクト通番を基に下方向に，静的 SQL の場合はコネクト通番を基に上方向にサーチすることで，SQL トレース情報中に表示されている SQL の実行要求開始時間，及び終了時間が分かります。また，SQL 単位の情報を取得すると，SQL 実行時間が表示されます。SQL 実行時間が長い SQL がある場合は，チューニングを実施してください。

UAP 統計レポートでアクセスパス情報を表示する場合，HiRDB のバージョン，バックエンドサーバ数，UAP 名称，認可識別子，SQL 最適化処理をした時間，及び SQL 文は表示しません。ただし，ルーチン中に操作系 SQL を指定した場合，SQL 文に操作系 SQL を表示します。

また，HiRDB/シングルサーバの場合に，アクセスパスが SELECT-APSL（複数のアクセスパスから，境界値によって実行時にアクセスパスを選択する）となるときは，最初に境界値の情報を表示し，後ろに Section No で区切られた候補を複数表示します。

アクセスパス情報については，マニュアル「HiRDB コマンドリファレンス」のアクセスパス表示ユティリティを参照してください。

●アクセスパス情報出力時の SQL 実行開始，終了時刻

［出力形式］

```
START-TIME : YYYY/MM/DD hh:mm:ss.fff....1
END-TIME   : YYYY/MM/DD hh:mm:ss.fff....2
```

［説明］

1. アクセスパス情報出力時に実行していた SQL の実行開始時刻を表示します。
2. アクセスパス情報出力時に実行していた SQL の実行終了時刻を表示します。

●注意事項

1. 外部 Java ストアドルーチンの実行では，アクセスパス情報は表示されません。
2. SQL ルーチンの実行では，ルーチン内で使用している表に対して，インデクスの追加又は削除で SQL オブジェクトのインデクス情報が無効になった場合，アクセスパス情報を表示します。
3. アクセスパス情報を表示すると，SQL トレースファイルの容量が増えます。増加するアクセスパス情報の容量は，次の計算式から求められます。ただし，表定義，インデクス定義，及び SQL でアクセスパス情報の容量は変わるため，計算式の結果は一応の目安としてください。

$$\text{アクセスパス情報の容量} = 1 + 0.1 \times \text{集合演算数} + 4 \times \sum_{i=1}^{n} (Si) \quad \text{（単位：キロバイト）}$$

n ：SQL文中の問合せ指定数
Si：i番目の問合せ指定中の表数
注　ルーチン中の問合せの場合は，計算式の結果にSQL文長を加算してください。

(c)　SQL 実行時の中間結果情報

SQL 実行時の中間結果情報を表示します。

SQL 実行時の中間結果情報を表示した場合，次の情報を確認できます（ここで表示される行数は，各中間段階で実際に HiRDB が処理した行数となります）。

- 一時表の格納先 RD エリア名，及び一時インデクスの格納先 RD エリア名
- 表から取り出した行数
- インデクスで絞り込まれた行数
- 結合ごとの結果の行数
- 問合せに指定した重複排除，GROUP BY，ORDER BY，及び LIMIT の入出力の行数，並びに問合せの結果の行数
- 集合演算ごとの結果の行数

この SQL 実行時の中間結果情報とアクセスパス情報を使用して，SQL のチューニングを実施してください。なお，アクセスパス情報を使用した SQL のチューニングについては，マニュアル「HiRDB コマンドリファレンス」のアクセスパス表示ユティリティを参照してください。

[出力形式]

```
 ------------------------------------------------------
Connect No     : aa...a
UAP Source     : bb...b
Section No     : cc...c
 ----- INSERT : ... -----   ..............................1
  :
 ----- QUERY EXPRESSION BODY ID : ... -----   ............2
  :
 ----- QUERY ID : ... -----   ..............................3
  :
 JOIN   ..................................................4
  :
 SCAN   ..................................................5
  :
```

[説明]

1. 一時表行挿入情報

   一時表行挿入情報については，「●一時表行挿入情報」を参照してください。

2. 集合演算情報

   集合演算情報については，「●集合演算処理情報」を参照してください。

3. 問合せ処理情報

   問合せ処理情報については，「●問合せ処理情報」を参照してください。

4. 結合処理情報

   結合処理情報については，「●結合処理情報」を参照してください。

5. 実表検索処理情報

   実表検索処理情報については，「●実表検索処理情報」を参照してください。

aa...a：

コネクト通番が表示されます。

bb...b：

UAP ソースファイル名が表示されます。

cc...c：

セクション番号（SQL の対応を確認するための番号）が表示されます。

Connect No 以降の情報は，SQL の個数分繰り返して表示されます。コネクト通番とセクション番号でサーチすることで，SQL トレース情報中に表示されている SQL 文やアクセスパス情報と対応付けることができます。

●一時表行挿入情報

一時表挿入情報を次に示します。

```
 ----- INSERT : ----------
# Table Name  : aa...a 0xbbbbbbbb(bb...b) IN cc...c
  Index Name  : dd...d 0xeeeeeeee(ee...e)IN ff...f
  Index Name  : dd...d 0xeeeeeeee(ee...e)IN ff...f
  Index Name  : dd...d 0xeeeeeeee(ee...e)IN ff...f
                       :
```

[説明]

aa...a：

挿入対象となる一時表の表名が表示されます。

0xbbbbbbbb(bb...b)：

挿入対象となる一時表の表 ID が 16 進数（10 進数）で表示されます。

cc...c：

一時表を格納した RD エリア名が表示されます。

dd...d：

一時インデクス名が表示されます。

0xeeeeeeee(ee...e)：

一時インデクスのインデクス ID が 16 進数（10 進数）で表示されます。

ff...f：

一時インデクスを格納した RD エリア名が表示されます。

dd...d，ee...e，ff...f は定義したすべてのインデクスの情報が表示されます。

●集合演算処理情報

集合演算処理情報を次に示します。

```
----- QUERY EXPRESSION BODY ID : aa...a -----
Query          : bb...b ROWS
Limit          : cc...c ROWS <-- dd...d ROWS
Order by       : ee...e ROWS
SetOpe Process : ff...f = gg...g ROWS <-- hh...h ii...i hh...h
                       :
```

[説明]

aa...a：

問合せ式本体 ID が表示されます。

集合演算を含む問合せ式本体単位に番号を付けます。SQL が複数の問合せ式本体で構成される場合，この行で区切って情報が表示されます。

「アクセスパス情報」を表示している場合，アクセスパス情報で表示されている問合せ式本体 ID に対応しています。

bb...b：

問合せ式の結果の行数が表示されます。

cc...c ROWS <-- dd...d ROWS：

最終的に，リミット行数分の検索結果を取得する処理（LIMIT 処理）の行数が表示されます。

LIMIT 句を指定していない場合，この行は表示されません。

cc...c：

LIMIT 処理の出力行数が表示されます。

dd...d：

LIMIT 処理の入力行数が表示されます。

ee...e：

ソート処理（ORDER BY 処理）の行数が表示されます。

次のどれかに該当する場合，この行は表示されません。

- ORDER BY 句を指定していない。
- ORDER BY 句で指定したソート処理が省略される。
- LIMIT 句を指定している。

ff...f = gg...g ROWS <-- hh...h ii...i hh...h：

集合演算の結果の行数が表示されます。

集合演算を複数指定した場合，複数行に分けて表示されます。

UNION ALL の分割スキャンをする機能（作業表を作成しないで各問合せの検索結果を連続して返す）を適用した場合，この行は表示されません。

ff...f：

集合演算結果の集合演算番号が，"LID（集合演算番号）"で表示されます。

アクセスパス情報を表示している場合，アクセスパス情報で表示されている集合演算番号に対応しています。

gg...g：

集合演算結果の行数が表示されます。

hh...h：

演算する問合せ式本体が問合せ指定の場合，"QID(問合せ ID)"が表示されます。演算する問合せ式本体が複数の問合せ指定の結合結果の場合，"LID(集合演算番号)"が表示されます。

ii...i：

集合演算の種別（"UNION"，"UNION ALL"，"EXCEPT"，又は"EXCEPT ALL"）が表示されます。前後の hh...h は，演算する問合せ式本体となります。

●問合せ処理情報

問合せ処理情報を次に示します。

```
----- QUERY ID : aa...a -----
Query          : bb...b ROWS
Limit          : cc...c ROWS <-- dd...d ROWS
```

```
Order by       : ee...e ROWS
Distinct       : ff...f ROWS <-- gg...g ROWS
Having         : hh...h ROWS
Group by       : ii...i ROWS <-- jj...j ROWS
```

[説明]

aa...a：

問合せ ID が表示されます。

問合せ指定に番号を付けます。SQL が複数の問合せ指定で構成される場合，この行で区切って情報が表示されます。

アクセスパス情報を表示している場合，アクセスパス情報で表示されている問合せ ID に対応しています。

bb...b：

問合せの結果の行数が表示されます。

cc...c ROWS <-- dd...d ROWS：

最終的に，リミット行数分の検索結果を取得する処理（LIMIT 処理）の行数が表示されます。

LIMIT 句を指定していない場合，この行は表示されません。

cc...c：

LIMIT 処理の出力行数が表示されます。

dd...d：

LIMIT 処理の入力行数が表示されます。

ee...e：

ソート処理（ORDER BY 処理）の行数が表示されます。ORDER BY 句を指定していない場合でも，暗黙的に ORDER BY 処理をすることがあります。

次のどれかに該当する場合，この行は表示されません。

- ORDER BY 句を指定していない。
- ORDER BY 句で指定したソート処理が省略される。
- 暗黙的に ORDER BY 処理をしない。
- LIMIT 句を指定している。

ff...f ROWS <-- gg...g ROWS：

重複排除処理の処理行数が表示されます。重複排除を指定していない場合でも，暗黙的に重複排除処理をすることがあります。

次のどれかに該当する場合，この行は表示されません。

- 重複排除を指定していない。
- 暗黙的に重複排除処理をしない。
- LIMIT 句を指定している。

ff...f：

重複排除処理の出力行数が表示されます。

gg...g：

重複排除処理の入力行数が表示されます。

hh...h：

HAVING 句を評価した後の行数が表示されます。

HAVING 句を指定していない場合，この行は表示されません。

ii...i ROWS <-- jj...j ROWS：

グループ分け処理（暗黙的グループ分け処理を含む）の処理行数が表示されます。

グループ分け処理をしない場合，この行は表示されません。

ii...i：

グループ分け処理の出力行数が表示されます。

jj...j：

グループ分け処理の入力行数が表示されます。

●結合処理情報

結合処理情報を次に示します。

```
JOIN
 # Join ID     : aa...a
   Row Count   : bb...b ROWS
   Left        : cc...c ROWS
   Right       : dd...d ROWS
   Join Type   : ee...e(ff...f)
```

[説明]

aa...a：

結合処理 ID が表示されます。

結合処理単位で番号を付け，結合処理が複数ある場合にはこの行で区切られます。

アクセスパス情報を表示している場合，アクセスパス情報で表示されている結合処理 ID に対応しています。

bb...b：

結合処理の結果の行数が表示されます。

cc...c：

左側の結合相手から取り出した行数が表示されます。

dd...d：

右側の結合相手から取り出した行数が表示されます。

ee...e：

- HiRDB/シングルサーバの場合，又は HiRDB/パラレルサーバで SQL 実行時に結合方式を動的に決定しない場合

結合処理の種別（"MERGE JOIN"，"NESTED LOOPS JOIN"，"CROSS JOIN"，又は"HASH JOIN"）が表示されます。

- HiRDB/パラレルサーバで SQL 実行時に結合方式を動的に決定する場合

  結合処理の種別"SELECT-APSL"が表示されます。

ff...f：

結合処理の実行種別（"INNER"，"LEFT OUTER"，"EXIST"，"NOT EXIST"，"ALL"，又は"VALUE"）が表示されます。

●実表検索処理情報

実表検索処理情報を次に示します。

- **インデクスを使用しない，又は一つだけ使用して検索する場合**

```
SCAN
 # Table Name  : aa...a(aa...a) 0xbbbbbbbbb(bb...b)
   Row Count   : cc...c ROWS
   Index Name  : dd...d 0xeeeeeeeee(ee...e)
                      Search    : ff...f gg...g
                      Key       : hh...h gg...g
```

[説明]

aa...a(aa...a)：

検索対象となる表名（相関名）が表示されます。

相関名を使用していない場合，（相関名）は表示されません。検索処理が複数ある場合，この行で区切って情報が表示されます。

0xbbbbbbbbb(bb...b)：

検索対象となる表 ID が 16 進数（10 進数）で表示されます。

cc...c：

実表から取り出した行数が表示されます。

dd...d：

検索で使用するインデクス名が表示されます。

次の場合，この行は表示されません。

- インデクスを使用しないで検索する。
- HiRDB/パラレルサーバで SQL 実行時に検索方法を動的に決定する。

0xeeeeeeeee(ee...e)：

検索で使用するインデクス ID が 16 進数（10 進数）で表示されます。

ff...f：

サーチ条件で絞り込まれた結果の行数が表示されます。

サーチ条件がない場合でも，インデクスを使用した検索のときは，インデクスを構成している行数が表示されます。

なお，プラグインインデクスのサロゲート機能を使用して集合関数の結果を求めている合，この行は表示されません。

gg...g：

繰返し列を含むインデクスの場合，"ELEMENTS"が表示されます。それ以外の場合，"ROWS"が表示されます。

hh...h：

キー条件で絞り込まれた結果の行数が表示されます。

キー条件がない場合，この行は表示されません。

- **インデクスを二つ以上使用して検索する場合**

```
SCAN
 # Table Name  : aa...a(aa...a) 0xbbbbbbbbb(bb...b)
   Row Count   : cc...c ROWS
   Index Name  : dd...d = ee...e 0xffffffff(ff...f)
                        Search    : gg...g hh...h
                        Key       : ii...i hh...h
                        Row Count : jj...j ROWS
                 dd...d = ee...e 0xffffffff(ff...f)
                        Search    : gg...g hh...h
                        Key       : ii...i hh...h
                        Row Count : jj...j ROWS
                 dd...d = kk...k ROWS <-- ll...l mm...m ll...l
```

[説明]

aa...a(aa...a)：

検索対象となる表名（相関名）が表示されます。

相関名を使用していない場合，（相関名）は表示されません。検索処理が複数ある場合，この行で区切って情報が表示されます。

0xbbbbbbbbb(bb...b)：

検索対象となる表 ID が 16 進数（10 進数）で表示されます。

cc...c：

実表から取り出した行数が表示されます。

dd...d：

AND PLURAL INDEXES SCAN※のときに作成する作業表番号が，"LID(作業表番号)"で表示されます。

アクセスパス情報を表示している場合，アクセスパス情報で表示されている作業表番号に対応しています。

ee...e：

AND PLURAL INDEXES SCAN※，又は OR PLURAL INDEXES SCAN※の場合，作業表を作成するために使用するインデクス名が複数行表示されます。ただし，インデクスを使用しないで作成する作業表については，インデクス名に"(NO USE)"が表示されます。

0xffffffff(ff...f)：

検索で使用するインデクス ID が 16 進数（10 進数）で表示されます。

gg...g：

サーチ条件で絞り込まれた結果の行数が表示されます。

サーチ条件がない場合でも，インデクスを使用した検索のときは，インデクスを構成している行数が表示されます。

hh...h：

繰返し列を含むインデクスの場合，"ELEMENTS"が表示されます。それ以外の場合，"ROWS"が表示されます。

ii...i：

キー条件で絞り込まれた結果の行数が表示されます。

キー条件がない場合，この行は表示されません。

jj...j：

実表から取り出した行数が表示されます。

dd...d = kk...k ROWS <-- ll...l mm...m ll...l：

AND PLURAL INDEXES SCAN※時の作業表の作成順序が表示されます。インデクスを三つ以上使用して検索する場合，複数行で表示されます。

kk...k：

演算結果の行数が表示されます。

ll...l：

演算をするための入力となる作業表が"LID(作業表番号)"で表示されます。

mm...m：

作業表間の演算の種別（"AND"，"OR"，又は"ANDNOT"）が表示されます。

注※

AND PLURAL INDEXES SCAN，及び OR PLURAL INDEXES SCAN については，マニュアル「HiRDB コマンドリファレンス」のアクセスパス表示ユティリティを参照してください。

- **ビュー表の結果を検索するために作業表を作成する場合**

```
SCAN
 # Table Name  : aa...a(aa...a) 0xbbbbbbbb(bb...b)
   Row Count   : cc...c ROWS
```

[説明]

aa...a(aa...a)：

ビュー名（相関名）が表示されます。

相関名を使用していない場合，（相関名）は表示されません。

0xbbbbbbbb(bb...b)：

ビュー ID が 16 進数（10 進数）で表示されます。

cc...c：

表から取り出した行数が表示されます。

- **WITH 句のために作業表を作成する場合**

```
SCAN
 # Table Name  : aa...a(aa...a)
   Row Count   : bb...b ROWS
```

[説明]

aa...a(aa...a)：

WITH 句問合せ名（相関名）が表示されます。

相関名を使用していない場合，（相関名）は表示されません。

bb...b：

表から取り出した行数が表示されます。

- **FROM 句に指定した導出表のために作業表を作成する場合**

```
SCAN
 # Table Name  : aa...a(aa...a)
   Row Count   : bb...b ROWS
```

[説明]

aa...a(aa...a)：

"(NO NAME)"，又は"(NO NAME)(相関名)"が表示されます。

bb...b：

表から取り出した行数が表示されます。

- **HiRDB が内部的に作成する作業表を検索する場合**

```
SCAN
 # Table Name  : aa...a
   Row Count   : bb...b ROWS
```

[説明]

aa...a：

HiRDB が内部的に作成する作業表名が表示されます。

HiRDB が内部的に作成する作業表名は，"(DUMMY 作業表番号)"となります。

作業表番号は 3 けたの整数です。

bb...b：

HiRDB が内部的に作成する，作業表から取り出した行数が表示されます。

- **中間結果情報出力時の SQL 実行開始，終了時刻**

[出力形式]

```
START-TIME : YYYY/MM/DD hh:mm:ss.fff....1
END-TIME   : YYYY/MM/DD hh:mm:ss.fff....2
```

[説明]

1. 中間結果情報出力時に実行していた SQL の実行開始時刻を表示します。
2. 中間結果情報出力時に実行していた SQL の実行終了時刻を表示します。

注意事項：

1. 次の SQL を実行した場合，SQL 実行時の中間結果情報が表示されます。
   - 定義系 SQL※1
   - ASSIGN LIST 文※5
   - CLOSE 文
   - DELETE 文
   - EXECUTE 文※1
   - EXECUTE IMMEDIATE 文※2
   - INSERT 文※3
   - PREPARE 文※4
   - PURGE TABLE 文※1
   - 1 行 SELECT 文
   - UPDATE 文
   - COMMIT 文※1
   - DISCONNECT 文※1
   - ROLLBACK 文※1
   - 暗黙的ロールバックありのエラーが発生した場合※1

   注※1

   閉じていないカーソルがある場合，SQL 実行時の中間結果情報が表示されます。

   注※2

   次の SQL の場合，SQL 実行時の中間結果情報が表示されます。

   ・ASSIGN LIST 文

   ・DELETE 文

   ・INSERT 文

・UPDATE 文

注※3

VALUES 句にスカラ副問合せ，又は問合せ指定を指定した場合，SQL 実行時の中間結果情報が表示されます。

注※4

クライアント環境定義 PDPRPCRCLS に YES を指定していて，かつ開いているカーソルで使用している SQL 識別子を再度 PREPRARE 文で使用する場合，開いていたカーソルの SQL 実行時の中間結果情報が表示されます。

注※5

FOR ALTER LIST を指定した場合，SQL 実行時の中間結果は表示されません。

2. ストアドプロシジャに記述した SQL については，CALL 文を実行しても SQL 実行時の中間結果情報は表示されません。
3. トリガに記述したトリガ SQL については，トリガが実行されても SQL 実行時の中間結果情報は表示されません。
4. HiRDB/パラレルサーバの場合，各サーバの合計行数が表示されます。
5. 表示される行数は正確な値でないことがあります。
6. SQL 実行時の中間結果情報を表示すると，SQL トレースファイルの容量が次の式で示す容量分増えます。SQL トレースファイルの見積もり時には注意してください。ただし，表定義，インデクス定義，及び SQL によって，中間結果情報の容量は大きく変わります。次の式で見積もった値は，ある程度の目安にしてください。

SQL実行時の中間結果情報容量

$$= 0.8 + 0.1 \times \text{集合演算数} + 0.9 \times \sum_{i=1}^{n} (Si) \quad \text{（単位：キロバイト）}$$

n：SQL 文中の問合せ指定数

Si：i 番目の問合せ指定中の表数

(d) UAP 単位の情報及び TRANSACTION 単位の情報

出力する情報の単位は，次の情報から判断してください。

- UAP INFORMATION：UAP 単位（コネクション単位）の情報
- TRANSACTION INFORMATION：TRANSACTION 単位の情報

1. UAP 名
   統計情報を編集した UAP の名称です。
2. ホスト名
   接続したサーバのホスト名です。

3. ポート番号
   接続したサーバのポート番号です。

4. 接続サーバ名
   接続したフロントエンドサーバ又はシングルサーバのサーバ名です。

5. コネクト通番
   サーバが CONNECT を受け付けるごとに，順次カウントする通番です。

6. サーバプロセス番号
   接続したサーバのプロセス番号です。

7. クライアントプロセス番号
   UAP のプロセス番号です。Type4 JDBC ドライバから接続している場合は，0 を表示します。

8. クライアントスレッド番号
   マルチスレッドで動作している UAP のスレッド番号です。Type4 JDBC ドライバから接続している場合は，0 を表示します。

9. 排他待ち時間（単位：ミリ秒）※1
   サーバ内で発生した排他取得要求に対して，ほかのユーザが排他を取得しているため排他取得待ちとなった時間です。

10. CPU 時間（単位：ミリ秒）※1
    UAP 実行時のトランザクション中に使用した，サーバ側の CPU 稼働時間です。

11. ストアドプロシジャの SQL オブジェクト取得要求回数
    シングルサーバ又はフロントエンドサーバで，SQL オブジェクト用バッファに対して，ストアドプロシジャの SQL オブジェクトの取得要求をした回数です。

12. ストアドプロシジャオブジェクトバッファヒット回数
    シングルサーバ又はフロントエンドサーバで，SQL オブジェクト用バッファから情報が見つかった回数です。

13. SQL オブジェクト取得要求回数
    発行した SQL 文に対して，SQL オブジェクトの取得要求をした回数です。

14. SQL オブジェクトバッファヒット回数
    発行した SQL 文に対して，取得要求をした SQL オブジェクト用バッファで情報が見つかった回数です。

15. SQL オブジェクト作成回数
    発行した SQL 文に対して，SQL オブジェクトを作成した回数です。

16. 作成した SQL オブジェクトサイズの最大値（単位：バイト）

発行した SQL 文で作成した，SQL オブジェクトサイズの最大値です。

17. COMMIT 文の実行回数

18. ROLLBACK 文の実行回数

19. FETCH 文，SELECT 文の実行で，UAP に返した検索行数

20. DELETE 文の実行で，削除した行数

21. INSERT 文の実行で，挿入した行数

22. UPDATE 文の実行で，更新した行数

23. 前処理実行回数

24. OPEN 文実行回数

25. FETCH 文実行回数

26. CLOSE 文実行回数

27. DESCRIBE 文（OUTPUT）実行回数

28. SELECT 文実行回数

29. INSERT 文実行回数

30. UPDATE 文実行回数

31. DELETE 文実行回数

32. LOCK 文実行回数

33. CREATE TABLE 実行回数

34. DROP TABLE 実行回数

35. ALTER TABLE 実行回数

36. CREATE INDEX 実行回数

37. DROP INDEX 実行回数

38. COMMENT（TABLE）実行回数

39. COMMENT（COLUMN）実行回数

40. CREATE SCHEMA 実行回数

41. DROP SCHEMA 実行回数

42. GRANT RDAREA 実行回数

43. GRANT SCHEMA 実行回数

44. GRANT アクセス権限実行回数

45. GRANT CONNECT 実行回数

46. GRANT DBA 実行回数

47. REVOKE RDAREA 実行回数

48. REVOKE SCHEMA 実行回数

49. REVOKE アクセス権限実行回数

50. REVOKE CONNECT 実行回数

51. REVOKE DBA 実行回数

52. CREATE VIEW 実行回数

53. DROP VIEW 実行回数

54. PURGE TABLE 文実行回数

55. CREATE PROCEDURE 実行回数

56. DROP PROCEDURE 実行回数

57. ALTER PROCEDURE 実行回数

58. CALL 文実行回数

59. DESCRIBE 文（INPUT）実行回数

60. そのほかの SQL の実行回数

61. 最大入出力時間（単位：秒）

62. 最大入出力時間（単位：マイクロ秒）（秒値は含まれません）

63. 最小入出力時間（単位：秒）

64. 最小入出力時間（単位：マイクロ秒）（秒値は含まれません）

    入出力時間が妥当かどうか検証してください。必要以上に時間が掛かっている場合はハード障害の可能性があるのでハードログを取得して検証してください。

    非同期 READ 機能を使用した場合は，非同期 READ プロセスでの一括先読みの入出力時間は含まれません。

65. データベースに対する入出力時間の累計（単位：秒）

66. データベースに対する入出力時間の累計（単位：マイクロ秒）（秒値は含まれません）
    原因が入出力なのか，又は CPU なのかの判断に使用してください。
    非同期 READ 機能を使用した場合は，非同期 READ プロセスでの一括先読みの入出力時間は含まれません。

67. データ，インデクス，及びディレクトリページを参照した回数
    データ，インデクス，及びディレクトリページを参照した回数が分かります。

68. データ，インデクス，及びディレクトリページを更新した回数
    データ，インデクス，及びディレクトリページを更新した回数が分かります。

69. データ，インデクス，及びディレクトリページのバッファヒット回数
    データ，インデクス，及びディレクトリページのバッファヒット回数が分かります。ヒット率（項番 69÷項番 67）×100 が低い場合は，グローバルバッファの統計を取得して，ヒット率の悪いグローバルバッファのチューニングをしてください。チューニング対象となるのは，LOB 用以外のグローバルバッファです。

70. データ，インデクス，及びディレクトリページの実 READ 回数
    データ，インデクス，及びディレクトリページを実 READ した回数が分かります。
    プリフェッチ機能を使用している場合は，プリフェッチで先読みした READ 回数も含まれます。また，非同期 READ 機能を使用した場合は，非同期 READ プロセスで先読みした READ 回数も含まれます。
    バッファヒット率が悪いと，READ 回数も多くなります。

71. データ，インデクス，及びディレクトリページの実 WRITE 回数
    データ，インデクス，及びディレクトリページを実 WRITE した回数が分かります。コミット出力機能を使用している場合は，コミット時にデータベースへ反映するために出力した回数もカウントされます。

72. LOB ページを参照した回数
    LOB ページを参照した回数が分かります。LOB データ，及びプラグインの検索はここにカウントされます。

73. LOB ページを更新した回数
    LOB ページを更新した回数が分かります。LOB データ，及びプラグインの更新はここにカウントされます。

74. LOB ページの参照バッファヒット回数
    LOB 用のグローバルバッファを使用している場合，参照バッファヒット回数が分かります。ヒット率（項番 74÷項番 72）×100 が低い場合は，グローバルバッファの統計を取得してグローバルバッファのチューニングをしてください。チューニング対象となるのは，LOB 用のグローバルバッファです。LOB 用のグローバルバッファを使用していない場合は，ヒット率は 0 になります。

75. LOB ページの更新バッファヒット回数

LOB 用のグローバルバッファを使用している場合，更新バッファヒット回数が分かります。LOB データの更新，又はプラグインインデクスの更新でヒット率（項番 75÷項番 73）×100 が低い場合は，グローバルバッファの統計を取得してグローバルバッファのチューニングをしてください。チューニング対象となるのは，LOB 用のグローバルバッファです。LOB 用のグローバルバッファを使用していない場合は，ヒット率は 0 になります。また，LOB データの新規追加では更新バッファヒットはしません。

76. LOB ページの実 READ 回数

LOB ページを実 READ した回数が分かります。LOB 用のグローバルバッファを使用している場合は，READ バッファヒット率が低いと READ 回数も多くなります。

77. LOB ページの実 WRITE 回数

LOB ページを実 WRITE した回数が分かります。プラグインインデクスを更新する場合は，LOB 用のグローバルバッファを使用することで実 WRITE 回数を削減できます。

78. グローバルバッファフラッシュ回数

新たなページを入力する空きバッファを作成するために，バッファを無効にした回数です。バッファ満杯によってページをメモリから追い出した回数が分かります。※2

79. グローバルバッファの READ 待ち発生回数

バッファ上のページが，ほかのユーザによって HiRDB ファイルからの入力中であったため，待ち状態になった回数です。他 UAP が READ 中のページに対してページ参照要求があり，READ 完了まで待ちになった回数が分かります。※2

80. グローバルバッファの WRITE 待ち発生回数

バッファ上のページが，ほかのユーザによって HiRDB ファイルへの出力中であったため，待ち状態になった回数です。WRITE 中のページに対してページ更新要求があり，WRITE 完了まで待ちになった回数が分かります。※2

81. グローバルバッファの排他待ち発生回数

バッファ上のページが，ほかのユーザによって使用中であったため，待ち状態になった回数です。他 UAP が更新中のページに対してページの参照又は更新要求があり，他 UAP の更新完了まで待ちになった回数が分かります。※2

82. 最大作業表用ファイル数

作業表用ファイルの最大使用数です。※3

作業表用ファイル用の HiRDB ファイルシステム領域を作成する，pdfmkfs コマンドの-l オプション（最大ファイル数）の指定値の妥当性をチェックできます。-l オプション指定値は次の式を満たしている必要があります。※4

```
-lオプション指定値
≧同時に実行するUAPのすべての作業表用ファイル数の合計値＋20
```

83. 最大作業表用ファイル増分回数

作業表用ファイルの最大増分回数です。※3

作業表用ファイル用の HiRDB ファイルシステム領域を作成する，pdfmkfs コマンドの-e オプション（最大増分回数）の指定値の妥当性をチェックできます。-e オプション指定値は次の式を満たしている必要があります。※4

```
-eオプション指定値
≧同時に実行するUAPのすべての作業表用ファイル増分回数の合計値
```

84. 作業表用ファイルの最大容量（単位：メガバイト）

作業表用ファイルの最大容量です。作業表用ファイル用の HiRDB ファイルシステム領域を作成する，pdfmkfs コマンドの-n オプション（最大増分回数）の指定値の妥当性をチェックできます。-n オプション指定値は次の式を満たしている必要があります。※4

```
-nオプション指定値
  ≧同時に実行するUAPのすべての作業表用ファイル最大容量の合計値
  ＋HiRDBファイルシステム領域の管理領域サイズ
```

85. 作業表用ファイルの READ 回数

作業表のデータを，ファイルからバッファへ入力した回数です。※1

86. 作業表用ファイルの WRITE 回数

作業表のデータを，バッファからファイルに出力した回数です。※1

87. 作業表用バッファの強制出力回数

作業表用バッファが不足したため，使用中バッファを強制的にファイル出力した回数です。※1

この値が 0 でない場合，システム定義の pd_work_buff_size オペランド（作業表用バッファ長）を大きくしてください。

88. ハッシュ表を一括して展開するための推定値（単位：キロバイト）

ハッシュジョイン，副問合せのハッシュ実行で，処理したハッシュデータを一括して展開するために必要なハッシュ表サイズの推定値です。※3

ハッシュ表サイズがこの値以上であれば，すべてバケット分割をしない一括ハッシュジョインになります※5。また，この値がハッシュ表サイズの指定範囲を超える場合は，一括ハッシュジョインにはできません。この値が 0 の場合は，ハッシュジョイン，副問合せのハッシュ実行が行われていません。

89. 1 レベル最大バケットサイズ（単位：キロバイト）

ハッシュジョイン，副問合せのハッシュ実行での，1 レベルバケット分割後の最大バケットサイズです。※3

ハッシュ表サイズがこの値以上であれば，バケット分割が 1 レベルで完了しています。また，バケット分割が 2 レベル以上の場合は，ハッシュ表サイズにこの値を設定することで，バケット分割が 1 レベルで完了するようになります※6。すべてバケット分割をしない一括ハッシュジョインになった場合，この値は 0 となります。

90. 2 レベル最大バケットサイズ（単位：キロバイト）

ハッシュジョイン，副問合せのハッシュ実行での，2 レベルバケット分割後の最大バケットサイズです。※3

ハッシュ表サイズがこの値以上であれば，バケット分割が 2 レベルで完了しています。また，バケット分割が 3 レベル以上の場合は，ハッシュ表サイズにこの値を設定することで，バケット分割が 2 レベルで完了するようになります※6。2 レベルバケット分割がされなかった場合，この値は 0 となります。

91. 3 レベル最大バケットサイズ（単位：キロバイト）

ハッシュジョイン，副問合せのハッシュ実行での，3 レベルバケット分割後の最大バケットサイズです。※3

ハッシュ表サイズがこの値以上であれば，最大 3 レベルのバケット分割で，バケット単位に処理されています。ハッシュ表サイズがこの値以下の場合，1 バケットを部分的にハッシュ表展開する処理となり，処理効率が悪くなります。この場合，ハッシュ表サイズをこの値以上に設定してください※6。又は，ハッシュジョイン，副問合せのハッシュ実行を適用しないようにした方が性能が良くなる場合があります。3 レベルバケット分割がされなかった場合，この値は 0 となります。

92. 空き領域の再利用のページサーチ空回り回数

新規ページ追加モードから空きページ再利用モードに切り替えたときに，再利用できる空き領域がなく，新規ページ追加モードに戻した回数です。この値が 0 以外の場合，UAP が実行した更新，挿入処理で効率が悪いページサーチ処理が発生していることが考えられます。空き領域の再利用機能については，マニュアル「HiRDB システム導入・設計ガイド」を参照してください。

93. 新規ページ追加モードから空きページ再利用モードへのモード切り替え回数

空き領域の再利用機能実行時に，新規ページ追加モードから空きページ再利用モードへ切り替わった回数です。この値が UAP で実行した更新，挿入処理の回数に近い場合，効率が悪いページサーチ処理が発生していることが考えられます。

94. キャッシュバッファ領域不足発生回数

システムが使用する内部情報

95. キャッシュバッファ領域割り当てフラッシュ回数

システムが使用する内部情報

96. キャッシュバッファ領域割り当てフラッシュ時の write 回数

システムが使用する内部情報

97. 最大キャッシュバッファ領域割り当てフラッシュ回数

システムが使用する内部情報

98. 平均キャッシュバッファ領域割り当てフラッシュ回数

システムが使用する内部情報

99. ローカルバッファを使用してデータページ，及びインデクスページを参照した回数

データページ，及びインデクスページを参照した回数です。

100. ローカルバッファを使用してデータページ，及びインデクスページを更新した回数

データページ，及びインデクスページを更新した回数です。

101. ローカルバッファでのデータページ，及びインデクスページのバッファヒット回数

データページ，及びインデクスページのバッファヒット回数です。

ランダムアクセスする UAP で，バッファヒット率（[101]÷[99]×100）が低い場合は，バッファヒット率の悪いバッファのチューニングをしてください。

102. ローカルバッファ使用時のデータページ，及びインデクスページの実 READ 回数

データページ，及びインデクスページを実 READ した回数です。

プリフェッチ機能を使用している場合は，プリフェッチで先読みした READ 回数も含まれます。バッファヒット率が悪いと READ 回数も多くなります。

103. ローカルバッファ使用時のデータページ，及びインデクスページの実 WRITE 回数

データページ，及びインデクスページを実 WRITE した回数です。

104. ローカルバッファフラッシュ回数

新たなページを入力する空きバッファを作成するために，バッファを無効にした回数（バッファ満杯によって，ページをメモリから追い出した回数）です。

105. 非同期 READ 要求回数

非同期 READ 機能使用時，非同期 READ プロセスで一括先読み処理を要求した回数です。

106. 非同期 READ 時の同期待ち回数

非同期 READ 機能使用時，非同期 READ プロセスで一括先読み処理での同期待ちが発生した回数です。

107. 非同期 READ 時の同期待ち時間の累計（単位：秒）

非同期 READ 機能使用時，非同期 READ プロセスで一括先読み処理での同期待ちが発生した時間の累計です。

108. 非同期 READ 時の同期待ち時間の累計（単位：マイクロ秒）（秒値は含まれません）

非同期 READ 機能使用時，非同期 READ プロセスで一括先読み処理での同期待ちが発生した時間の累計です。

109. 非同期 READ 時の平均同期待ち時間（単位：秒）

非同期 READ 機能使用時，非同期 READ プロセスで一括先読み処理での同期待ちが発生した時間の平均です。

110. 非同期 READ 時の平均同期待ち時間（単位：マイクロ秒）（秒値は含まれません）

非同期 READ 機能使用時，非同期 READ プロセスで一括先読み処理での同期待ちが発生した時間の平均です。

111. 非同期 READ 時の平均同期入出力時間（単位：秒）

非同期 READ 機能使用時，初回の先頭ページ一括読み込みでの同期 READ 時間の平均です。

112. 非同期 READ 時の平均同期入出力時間（単位：マイクロ秒）（秒値は含まれません）
非同期 READ 機能使用時，初回の先頭ページ一括読み込みでの同期 READ 時間の平均です。

113. ハッシュジョイン，副問合せのハッシュ実行でのハッシュ表探索時の最大比較回数※3
1 回のハッシュ表探索時の，同一ハッシュ値を持つデータに対する比較回数の最大値です。

114. ハッシュジョイン，副問合せのハッシュ実行でのハッシュ表探索時の総比較回数※1
ハッシュ表探索時の，同一ハッシュ値を持つデータに対する比較回数の合計です。

115. ハッシュジョイン，副問合せのハッシュ実行での総ハッシュ表探索回数※1
ハッシュ表を探索した回数です。

116. 欠番

117. 欠番

118. 欠番

119. 欠番

120. 欠番

121. 欠番

122. 欠番

123. 欠番

124. 欠番

125. 欠番

126. 欠番

127. 欠番

128. 欠番

129. 欠番

130. 欠番

131. 欠番

132. 欠番

133. 欠番

134. 欠番

135. 欠番

136. 欠番

137. 欠番

138. 排他制御用プールパーティションのスレッド間ロック要求回数
排他制御用プールパーティションのスレッド間ロック要求回数の総和です。

139. 排他制御用プールパーティションのスレッド間ロック待ち回数
排他制御用プールパーティションのスレッド間ロック待ち回数の総和です。

140. 排他制御用プールパーティションのスレッド間ロック待ち時間の合計値（単位：マイクロ秒）※7※8
排他制御用プールパーティションのスレッド間ロック待ち時間の総和です。

141. 排他制御用プールパーティションのスレッド間ロック待ち時間の最大値（単位：マイクロ秒）※7※8
排他制御用プールパーティションのスレッド間ロック待ち時間の最大値です。

142. 排他制御用プールパーティションのスレッド間ロック待ち行列数の合計値
排他制御用プールパーティションのスレッド間ロック時の待ち行列数の総和です。

143. 排他制御用プールパーティションのスレッド間ロック待ち行列数の最大値
排他制御用プールパーティションのスレッド間ロック待ち行列数の最大値です。

144. ログフラッシュ要求回数
ログフラッシュ要求回数の総和です。

145. ログフラッシュ待ち回数
ログフラッシュ待ち回数の総和です。

146. ログフラッシュ待ち時間の合計値（単位：マイクロ秒）※7※8
のログフラッシュ待ち時間の総和です。

147. ログフラッシュ待ち時間の最大値（単位：マイクロ秒）※7※8
ログフラッシュ待ち時間の最大値です。

148. ログフラッシュ待ち行列数の合計値
ログフラッシュ待ち行列数の総和です。

149. ログフラッシュ待ち行列数の最大値
ログフラッシュ待ち行列数の最大値です。

150. グローバルバッファプールのスレッド間ロック要求回数
グローバルバッファプールのスレッド間ロック要求回数の総和です。

151. グローバルバッファプールのスレッド間ロック待ち回数
グローバルバッファプールのスレッド間ロック待ち回数の総和です。

152. グローバルバッファプールのスレッド間ロック待ち時間の合計値（単位：マイクロ秒）
グローバルバッファプールのスレッド間ロック待ち時間の総和です。※7※8

153. グローバルバッファプールのスレッド間ロック待ち時間の最大値（単位：マイクロ秒）
グローバルバッファプールのスレッド間ロック待ち時間の最大値です。※7※8

154. グローバルバッファプールのスレッド間ロック待ち行列数の合計値
グローバルバッファプールのスレッド間ロック待ち行列数の総和です。

155. グローバルバッファプールのスレッド間ロック待ち行列数の最大値
グローバルバッファプールのスレッド間ロック待ち行列数の最大値です。

156. 一時表を使う SQL を実行した時の使用セグメント数の合計

157. データ圧縮処理の総実行回数※9
圧縮列へのデータ挿入及び圧縮列のデータを更新する際に発生した，データ圧縮処理の総実行回数です。

158. データ圧縮処理の総実行時間（単位：秒）※9

159. データ圧縮処理の総実行時間（単位：マイクロ秒）※9
圧縮列へのデータ挿入及び圧縮列のデータを更新する際に発生した，すべてのデータ圧縮処理に掛かった時間です。

160. データ圧縮処理の最大実行時間（単位：秒）※9

161. データ圧縮処理の最大実行時間（単位：マイクロ秒）※9
圧縮列へのデータ挿入及び圧縮列のデータを更新する際に発生した，データ圧縮処理 1 件ごとに掛かった時間の最大値です。

162. データ伸張処理の総実行回数※9
圧縮列のデータを検索又は更新する際に発生した，データ伸張処理の実行回数です。

163. データ伸張処理の総実行時間（単位：秒）※9

164. データ伸張処理の総実行時間（単位：マイクロ秒）※9
圧縮列のデータを検索又は更新する際に発生した，すべてのデータ伸張処理に掛かった時間です。

165. データ伸張処理の最大実行時間（単位：秒）※9

166. データ伸張処理の最大実行時間（単位：マイクロ秒）※9

圧縮列のデータを検索又は更新する際に発生した，データ伸張処理 1 件ごとに掛かった時間の最大値です。

注※1

HiRDB/パラレルサーバの場合は，各サーバの合計となります。

注※2

すべてのグローバルバッファの合計となります。

注※3

HiRDB/パラレルサーバの場合は，各バックエンドサーバの最大値となります。

注※4

一時的なフラグメンテーションによって，見積もり式以上の資源が必要になる場合があるため，実際の指定値には余裕を持たせてください。

注※5

ハッシュ表サイズを大きくすると，1 回のバケット分割数が増加することがあるため，チューニング情報取得時よりも大きなハッシュ表サイズが必要になることがあります。このチューニング情報を基にハッシュ表サイズを大きくした場合は，再度チューニング情報を取得してください。そこで意図した結果になっていない場合，チューニング情報を基に再度ハッシュ表サイズを大きくする必要があります。

注※6

ハッシュ表サイズを大きくすると，1 回のバケット分割数が増加することがあるため，チューニング情報取得時より小さいハッシュ表サイズでも，意図したレベルでバケット分割が完了することもあります。これに対して，ハッシュ表サイズを小さくすると，1 回のバケット分割数が減少することがあるため，チューニング情報取得時と同じレベルでバケット分割が完了しなくなることがあります。したがって，このチューニング情報は，ハッシュ表サイズを大きくしていく場合に使用するようにしてください。

注※7

次の場合，UAP 統計レポートにスレッド間ロック待ち時間が表示されます。

- クライアント環境定義 PDUAPREPLVL に o を指定した場合
  （「PDUAPREPLVL」を参照）
- pdclttrc コマンドの-l オプションに o を指定した場合
  （マニュアル「HiRDB コマンドリファレンス」を参照）
- pdtrcmgr コマンドの-k オプションに o を指定した場合
  （「SQL トレース動的取得機能」を参照）

スレッド間ロック待ち時間を取得していない場合は，0 が表示されます。

注※8

CPU 時間の精度は，使用している OS，及びハードウェアに依存します。したがって，CPU 時間の精度が 1 ミリ秒より大きい環境の場合，実際の値より小さい値が表示されることがあります。

注※9

この項目は BINARY 型のデータに対する情報です。

- **UAP 単位情報及びトランザクション単位情報出力時の SQL 実行開始，終了時刻**

[出力形式]

```
START-TIME : YYYY/MM/DD hh:mm:ss.fff....1
END-TIME   : YYYY/MM/DD hh:mm:ss.fff....2
```

[説明]

1. UAP 単位情報/トランザクション単位情報出力時に実行していた SQL の実行開始時刻を表示します。
2. UAP 単位情報/トランザクション単位情報出力時に実行していた SQL の実行終了時刻を表示します。

11.1.5　コマンドトレース機能

UAP からのコマンド実行時（SQL の COMMAND EXECUTE 実行時）に，クライアントのトレース情報をコマンドトレースファイルに出力します。

コマンドトレースファイルが取得情報で満杯になると，最も古い情報から順次新しい情報に書き替えられます。

(1)　コマンドトレース情報の取得方法

コマンドトレースは，クライアント環境定義の PDCLTPATH 及び PDCMDTRACE に値を設定することで取得できます。各クライアント環境定義については，「クライアント環境定義（環境変数の設定）」を参照してください。

pdccmd1.trc と pdccmd2.trc という名称の二つのコマンドトレースファイルが，指定したディレクトリ下に作成されます。

(2)　コマンドトレース情報の見方

コマンドトレースは，UAP からのコマンド実行時に出力されます。

出力されるコマンドトレースの例とその説明を次に示します。

[出力例]

```
** COMMAND TRACE (CLT:10-06   :Aug 24 2022) LINUX**  [1]

USER APPLICATION PROGRAM FILE NAME : TESTAP  [2]
COMMAND START TIME : 2022/08/24 10:55:27  [3]
COMMAND EXECUTE ENVIRONMENT & STATUS :  [4]
  PDASTHOST(dcm3500)
  PDASTPORT(20266)
  PDSYSTEMID("HRD1")
  PDUSER("hirdb")
  PDASTUSER("hirdb ")
  PDCMDWAITTIME(0)
```

```
  ENVGROUP("")
  CLTPID(9155) CLTTID(0)
[5]  [6]          [7]          [8]        [9]
9155   0   2022/08/24 10:55:27    0   pdhold -r RDDATA01
9155   0   2022/08/24 10:55:27    1   KFPZ02444-E Communication error,
                                       func=connect, errno=2
```

[説明]

1. コマンドトレースヘッダ

   ヘッダには次の情報が表示されます。

   - リンクしたライブラリのバージョン
   - ライブラリの作成日付（Mmm dd yyyy の形式）
   - 実行プラットフォーム（プラットフォームとして表示される文字については，「SQL トレース情報の見方」の[説明]を参照してください）

2. UAP 名称

   クライアント環境定義 PDCLTAPNAME での指定値が表示されます。

3. コマンド開始日時

   コマンド実行を開始した日時が表示されます。

4. コマンド実行環境及びステータス

   コマンド実行時のクライアント環境定義の値，及びステータスが表示されます。

5. UAP のプロセス番号

   UAP のプロセス番号が表示されます。

   なお，正しいプロセス番号が取得できない場合，不正な数値が表示される場合があります（Windows 版の場合）。

6. UAP のスレッド番号

   UAP がマルチスレッドで動作している場合，UAP のスレッド番号が表示されます。マルチスレッドで動作していない場合，0 が表示されます。なお，正しいスレッド番号を取得できないで，不正な数値を表示する場合があります。

7. コマンドトレース取得日時

   コマンドトレースを取得した日時が表示されます。

8. コマンドトレースカウンタ

   コマンドトレースを受け付けるごとに，順次カウントした番号が表示されます。0～65535 までカウントされます。

9. トレースデータ

   トレースデータが表示されます。

(3)　コマンドトレースファイルのバックアップの取得

コマンドトレース情報を出力中にコマンドトレースファイルの容量が一杯になると，HiRDB はそのファイルへ出力しないで，もう一方のコマンドトレースファイルに情報を出力します。このとき，切り替え先のコマンドトレースファイルに格納されている古いコマンドトレース情報から順に消去され，新しいコマンドトレース情報に書き換えられます。このため，必要な情報は UAP 終了時にコマンドトレースファイルの内容をコピーしてバックアップを取得しておいてください。

なお，現在使用しているコマンドトレースファイルを知りたい場合は，ファイルの最終更新日時を調べてください。最終更新日時の新しい方が現在使用しているコマンドトレースファイルになります。

HiRDB クライアントが Windows 版の場合は dir コマンド又はエクスプローラで，ファイルの最終更新日時を調べてください。

HiRDB クライアントが UNIX 版の場合は OS の ls -l コマンドで，ファイルの最終更新日時を調べてください。

11.1.6　SQL トレース動的取得機能

UAP 実行時に，コマンドで動的に SQL トレースを取得できます。SQL トレースは，次回の CONNECT から取得されます。

(1)　SQL トレースファイルの出力先及びファイル名

SQL トレースファイルの出力先及びファイル名については，「SQL トレース機能」を参照してください。

(2)　トレース取得コマンド（pdtrcmgr）

pdtrcmgr は，-d オプションで指定したディレクトリと，UAP 実行時に設定されていたクライアント環境定義 PDTRCPATH の指定ディレクトリが同じ場合，その UAP に対してトレース取得の開始又は停止を要求します。

(a)　形式

```
pdtrcmgr  -d PDTRCPATHで指定したディレクトリ名
          [ { -b | -e } ]
          [ -k PDUAPREPLVLで指定可能な値 ]
          [ -n PDCLTAPNAME]
          [ -s SQLトレースファイルのサイズ ]
          [ -o]
```

(b)　オプション

- **-d PDTRCPATH で指定したディレクトリ名**

  ～<パス名>

トレース取得の開始又は停止をしたい UAP の，実行時に設定されているクライアント環境定義 PDTRCPATH の指定値（ディレクトリ名）を，絶対パス名で指定します。

指定したディレクトリと PDTRCPATH のディレクトリが同じすべての UAP に対して，トレース取得の開始又は停止の指示をします。

- **{ -b ｜ -e }**

SQL トレースの取得を開始するか，又は停止するかを指定します。

-b：SQL トレースの取得を開始します。

-e：SQL トレースの取得を停止します。

- **-k PDUAPREPLVL で指定可能な値**

クライアント環境定義 PDUAPREPLVL の値を指定できます。PDUAPREPLVL の詳細は，「クライアント環境定義（環境変数の設定）」を参照してください。

省略した場合，SQL トレース情報だけ出力されます。-e オプションを指定している場合は，このオプションを指定しても無視されます。

UAP 統計レポートについては，「UAP 統計レポート機能」を参照してください。

クライアント環境定義 PDHSICOPTIONS に UAPREP 又は ALL を指定している場合，このオプションには ut を含む値を指定する必要があります。PDHSICOPTIONS の詳細は「クライアント環境定義（環境変数の設定）」を参照してください。

- **-n PDCLTAPNAME**

クライアント環境定義 PDCLTAPNAME で指定している UAP だけ対象にする場合に指定します。-e オプションを指定している場合は，このオプションを指定しても無視されます。

- **-s SQL トレースファイルのサイズ**

　　～＜符号なし整数＞((0，又は 32768～2000000000))《32768》

SQL トレースファイルのサイズをバイト単位で指定します。

0 を指定した場合は，ファイルの最大サイズとなります。32,768～2,000,000,000 を指定した場合は，指定値のファイルサイズとなります。

-e オプションを指定している場合は，このオプションを指定しても無視されます。

- **-o**

CONNECT，DISCONNECT 単位で SQL トレースファイルをオープン，クローズする場合に指定します。なお，このオプションは，-e オプションを指定している場合には無視されます。

CONNECT，DISCONNECT 単位で SQL トレースファイルをオープン，クローズすると，オペレーション単位（SQL 単位）でするよりオーバヘッドが削減されるため，SQL トレースを出力するときの時間が短縮できます。

このオプションを省略した場合，オペレーション単位で SQL トレースファイルをオープン，クローズします。

なお，このオプションを指定した場合，SQL トレースファイルをオープンしたままで情報を書き込むため，正常に DISCONNECT できなかったときには，SQL トレース情報が欠落することがあります。

(3) PDSQLTRACE を指定している場合の留意事項

クライアント環境定義 PDSQLTRACE を指定している環境で SQL トレース動的取得機能を使用した場合，PDSQLTRACE で指定した SQL トレースと，動的に取得する SQL トレースの両方を出力します。

ただし，Type4 JDBC ドライバを使用している場合には，動的に取得する SQL トレースだけを出力します。

11.1.7 再接続トレース機能

再接続トレースは，自動再接続機能で再接続が行われた場合，HiRDB が内部的に管理している接続ハンドルの値，再接続前の接続情報，再接続後の接続情報，及び再接続時刻を，再接続トレースファイルに出力します。この情報は，Cosminexus の PRF トレース機能で出力されるトレース中の接続情報を追跡するために使用します。

(1) 再接続トレースの取得方法

再接続トレースは，クライアント環境定義 PDRCTRACE に値を設定すると取得できます。

再接続トレースファイルは，クライアント環境定義 PDCLTPATH に指定したディレクトリに二つ作成されます。作成されるファイル名称を次に示します。

クライアント種別	ファイル名称
埋込み型 UAP ODBC ドライバ Type2 JDBC ドライバ SQLJ OLE DB プロバイダ HiRDB データプロバイダ for .NET Framework	pdrcnct1.trc pdrcnct2.trc
Type4 JDBC ドライバ	クライアント環境定義 PDCONTYPE に PASSIVE を指定した場合： pdrcnct_ppppp_1.trc pdrcnct_ppppp_2.trc

クライアント種別	ファイル名称
	クライアント環境定義 PDCONTYPE に ACTIVE を指定した場合： pdrcnct_HHMMSSfff_XXXXXXXX_YYYYYYYYY_1.trc pdrcnct_HHMMSSfff_XXXXXXXX_YYYYYYYYY_2.trc

（凡例）

ppppp：クライアント受信ポート番号

HHMMSSfff：コネクト要求時間（HH：時，MM：分，SS：秒，fff：ミリ秒）

XXXXXXXX：接続サーバ名

YYYYYYYYY：コネクト通番

(2) 再接続トレースの見方

再接続トレースは，自動再接続機能で自動的に接続が行われた場合に出力されます。

再接続トレースの出力例を次に示します。

```
   [1] [2]                          [3]                                  [4]
40004250 S 2004/04/12 11:10:36.766 - 2004/04/12 11:10:41.846 sds:9:23763 => sds:10:23750
40004250 S 2004/04/12 11:11:07.491 - 2004/04/12 11:11:12.547 sds:10:23750 => sds:11:23765
40004850 F 2004/04/12 11:17:58.285 - 2004/04/12 11:18:23.395 sds:14:23751 =>
40005050 S 2004/04/12 11:27:35.098 - 2004/04/12 11:27:40.152 sds:1:24414 => sds:2:24418
```

[説明]

1. 接続ハンドルの値

   HiRDB が内部的に管理している接続ハンドル値が 16 進数で出力されます。

   クライアントが 32 ビットモードの場合は 8 けた，64 ビットモードの場合は 16 けたとなります。接続ハンドルの値が同一のトレースは，UAP からすると同じ接続であることを意味します。

   上記出力例の場合，接続ハンドルの値として 40004250 が 2 回出力されています。これは，この接続ハンドルを使用している UAP からすると，再接続が 2 回行われたことを意味します。

2. 再接続結果

   再接続結果が出力されます。

   S：成功

   F：失敗

3. 再接続開始日時，及び再接続完了日時

   切断を検知してから再接続を開始した日時と，正常に再接続が完了した日時が，ミリ秒単位で出力されます。再接続に失敗した場合は，UAP に制御が戻る直前の日時を出力します。

4. 再接続前，及び再接続後の接続情報

   再接続前の接続情報，再接続後の接続情報が出力されます。接続情報は，接続サーバ名称，コネクト通番，及び接続サーバのプロセス ID をコロンで区切って出力します。

   再接続に失敗した場合，再接続後の接続情報は出力されません（空白となります）。

(3)　Cosminexus の PRF トレース情報との突き合わせ方法

Cosminexus の PRF トレースには，出力例の 4 で示した接続情報が出力されます。その後，自動再接続機能で再接続が行われた場合は，次の手順で付き合わせを行います。

＜手順＞

1. PRF トレース中の HiRDB の接続情報を取得してください。
2. 再接続トレースファイルの 4 の中から，手順 1 で取得した接続情報を探し，その接続ハンドル値を取得してください。
3. 再接続トレースファイルの 1 の中から，手順 2 で取得した接続ハンドルと同じ値のトレースを追跡します。同じ値の接続ハンドルが見付かった場合，再接続前の接続情報が一つ前の同じ接続ハンドルの，再接続後の接続情報と同じときは，追跡対象となります。異なる場合は，この接続ハンドルで新たに接続されているため（DISCONNECT−CONNECT），追跡対象とはなりません。

(4)　再接続トレースのバックアップの取得

再接続トレース出力中に再接続ログファイルの容量が一杯になると，そのファイルへは出力しないで，もう一方の再接続トレースファイルに再接続ログを出力します。この場合，切り替え先の再接続トレースファイルに格納されている古い再接続トレースは消去され，新しい再接続トレースに書き換えられます。このため，長時間運用をする場合は，必要に応じて再接続トレースファイルの内容をコピーして，バックアップを取得しておいてください。

なお，現在使用している再接続トレースファイルを知りたい場合は，ファイルの最終更新日時を調べてください。最終更新日時の新しい方が現在使用している再接続トレースファイルになります。

HiRDB クライアントが Windows 版の場合は dir コマンド又はエクスプローラで，ファイルの最終更新日時を調べてください。

HiRDB クライアントが UNIX 版の場合は OS の ls -l コマンドで，ファイルの最終更新日時を調べてください。

11.1.8　HiRDB SQL Tuning Advisor 用アクセスパス情報ファイル

HiRDB SQL Tuning Advisor が使用するアクセスパス情報ファイルを，HiRDB クライアント側に出力します。HiRDB SQL Tuning Advisor を使用して，このアクセスパス情報ファイルと，SQL トレース情報とを突き合わせて解析できます。これによって，性能上問題になる SQL を特定しやすくなります。HiRDB SQL Tuning Advisor の機能の詳細については，HiRDB SQL Tuning Advisor のヘルプを参照してください。

(1)　設定方法

HiRDB SQL Tuning Advisor 用アクセスパス情報ファイルを出力する場合，次に示すクライアント環境定義を設定します。

- PDTAAPINFPATH

アクセスパス情報ファイル出力ディレクトリを指定します。出力先ディレクトリがなかったり，書き込み権限がなかったりして，出力処理でエラーが発生した場合，アクセスパス情報を出力しません。なお，出力処理でエラーが発生しても，実行中の SQL はエラーにはなりません。ただし，JDBC4.0 又は JDBC4.3 の Type4 JDBC ドライバでは例外を投入します。

- PDTAAPINFMODE

  アクセスパス情報ファイルのファイル名の形式を指定します。

- PDTAAPINFSIZE

  アクセスパス情報ファイルのファイルサイズを指定します。アクセスパス情報は 2 個作成しますが，ここで指定したファイルサイズを超えると，出力先をもう一方のファイルに切り替えます。

(2)　アクセスパスの解析

HiRDB SQL Tuning Advisor 用アクセスパス情報ファイルは，次の手順で解析します。

［手順］

1. ［スタート］－［プログラム］－［HiRDB SQL Tuning Advisor］－［HiRDB SQL Tuning Advisor］を選択し，HiRDB SQL Tuning Advisor を起動します。

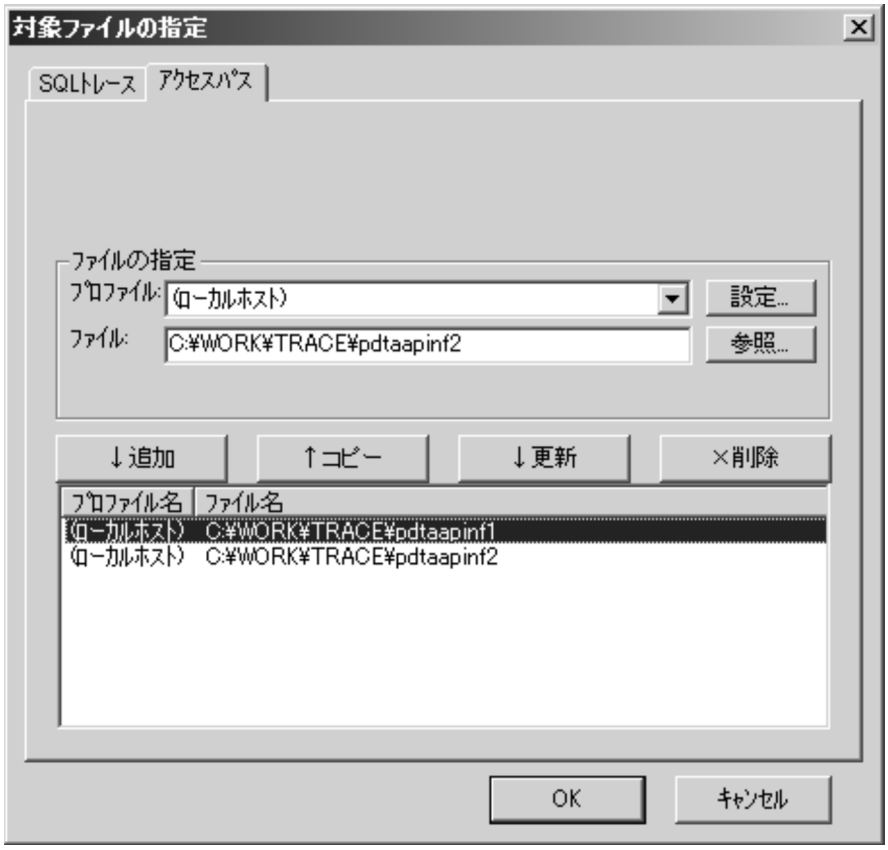
2. 接続の設定を行います。

   接続の設定方法については，「HiRDB SQL Tuning Advisor の環境設定」を参照してください。設定済みの場合，この手順は必要ありません。

3. ［オプション］メニューから［対象ファイル指定］を選択します。

   ［対象ファイルの指定］画面が表示されます。

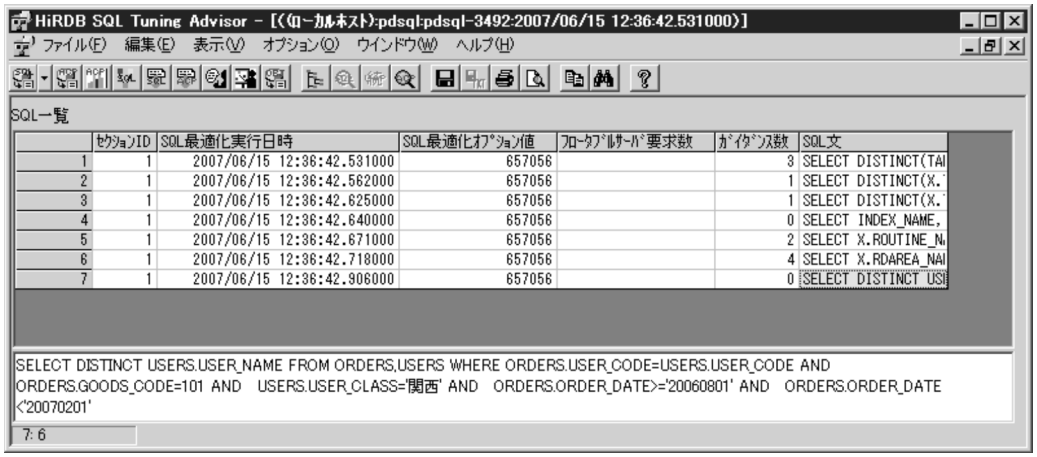
4. ［アクセスパス］タブで，アクセスパスファイル名を指定し，［追加］ボタンをクリックします。すべての対象ファイルを追加したら，［OK］ボタンをクリックします。



5. [アクセスパス解析] ボタンをクリックします。
   UAP 一覧が表示されます。

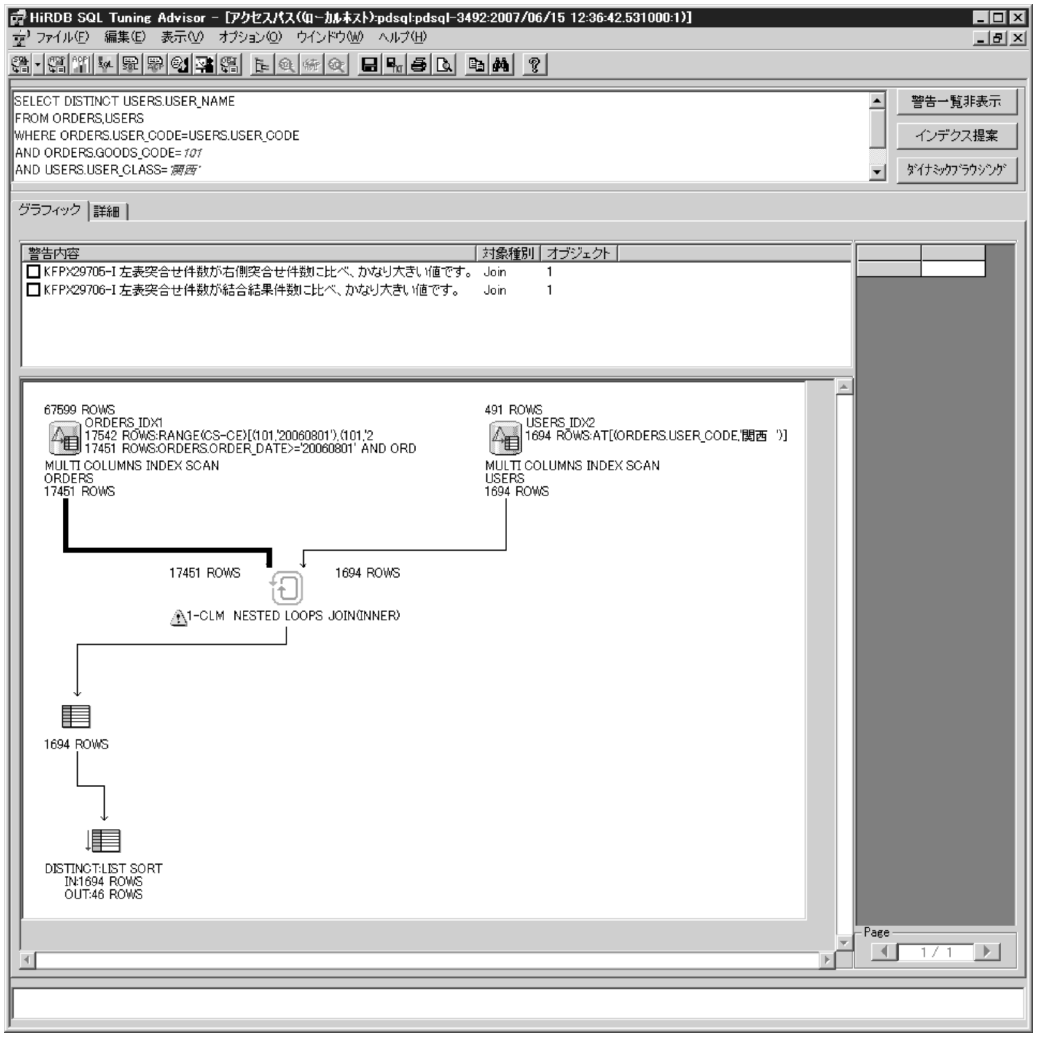


6. 該当する UAP の行をダブルクリックします。
   SQL 一覧が表示されます。



7. 該当する SQL の行をダブルクリックします。

   アクセスパスの解析結果が表示されます。［警告内容］に表示されるガイダンスを参照してください。



(3) 留意事項

- HiRDB サーバが，バージョン 06-00 以降であれば，この機能を使用できます。

- SQL オブジェクトがバッファ中にある SQL についても，SQL オブジェクトを再作成するため，HiRDB サーバ側の負荷が増えることがあります。
- この機能を使用している場合，プロセス間メモリ通信機能は使用できません。クライアント環境定義の PDIPC オペランドに MEMORY を指定していても，DEFAULT を指定した場合の動作となります。
- HiRDB SQL Tuning Advisor のダイナミックブラウジング機能を使用する場合は，クライアント環境定義 PDTAAPINFPATH の指定は無視されます。
- HiRDB サーバがバージョン 07-03 より前の場合にこの機能を使用するときは，UAP 統計レポート機能でアクセスパス情報を取得する設定（クライアント環境定義 PDUAPREPLVL に p 又は a を指定）をしていても，UAP 統計レポートのアクセスパス情報は出力しません。

11.2　UAP 障害の回復

11.2.1　UAP 障害時の回復方法

UAP に障害が発生した場合，HiRDB システム全体が停止しないように対処する必要があります。

ここでは，UAP 障害時の回復方法について説明します。

UAP 障害時の回復方法としては，次に示す三つに分けられます。

- HiRDB による UAP トランザクションのロールバック
- UAP 指示によるトランザクションのロールバック
- メモリ容量の再検討

UAP 障害の種別と回復方法を次の表に示します。

表 11–7　UAP 障害の種別と回復方法

障害種別	検　出　方　法	システム側の処置	回　復　方　法
UAP 異常終了	UAP 処理時間監視機能	UAP の切り離し	UAP トランザクションのロールバック
UAP 無限ループ			
トランザクション未終了			
UAP 処理エラー	サーバ※1 内の各種エラー検出機能	UAP へエラー応答	UAP 指示によるトランザクションのロールバック
UAP によるエラー検出とロールバック要求	UAP によるエラー検出	UAP の指示に従う	
デッドロック	HiRDB のデッドロック検出機能	UAP へエラー応答（暗黙的ロールバック）	UAP トランザクションの終了
メモリ不足	メモリ確保時のエラー	UAP 起動不可	共用メモリ，プロセス固有メモリの見直し※2

注※1

フロントエンドサーバ，及びバックエンドサーバを示します。

注※2

共用メモリ，プロセス固有メモリの見直しについては，システム管理者に依頼してください。

(1)　UAP 処理時間監視機能

UAP を実行すると，HiRDB の UAP 処理時間監視機能によってタイマ監視されます。これは，UAP に異常が発生して HiRDB の処理が長時間止まった状態になることを防止するためです。

タイマ監視は，クライアント環境定義で環境変数 PDSWAITTIME によって監視する時間を指定します。時間を指定しないと，HiRDB の仮定値によって監視します。

クライアント環境定義の詳細については，「クライアント環境定義（環境変数の設定）」を参照してください。

(2)　サーバ内の各種エラー検出機能

HiRDB/パラレルサーバの場合，SQL の実行中にフロントエンドサーバ内，又はバックエンドサーバ内で，データベース処理のプロセス異常などのエラーを検出すると，プロセスの切り離しなどが必要なため，UAP 側にエラーステータスを返します。エラーステータスに対して UAP がロールバック要求を発行すると，HiRDB としての回復処理がされます。

(3)　UAP によるエラー検出

UAP 内で障害を検出した場合，ロールバック要求を発行することで回復処理がされます。

なお，UAP が正常に処理された場合，UAP からの DISCONNECT 指示によってプロセスが切り離されます。

(4)　メモリ容量の再検討

共用メモリ，及びプロセス固有メモリが不足すると，メモリ，又はディスク容量の不足を表すメッセージが出力されます。メッセージが出力された場合，UAP を起動するために必要なメモリを確保した後，UAP を再実行します。

なお，共用メモリ，プロセス固有メモリの見直しについては，マニュアル「HiRDB システム導入・設計ガイド」を参照するか，又は HiRDB 管理者に連絡してください。

12 UAP からのコマンド実行

この章では，UAP からコマンドを実行する方法について説明します。

12.1　概要

UAP 中にコマンドを指定して実行できます。指定したコマンドは，HiRDB サーバ側で実行されます。

UAP からコマンドを実行する場合，次のどちらかの SQL を使用します。

- CALL COMMAND 文

  HiRDB の運用コマンド，及びユティリティを実行します。CALL COMMAND 文を使用する場合，コマンド実行のための準備は必要ありません。

- COMMAND EXECUTE

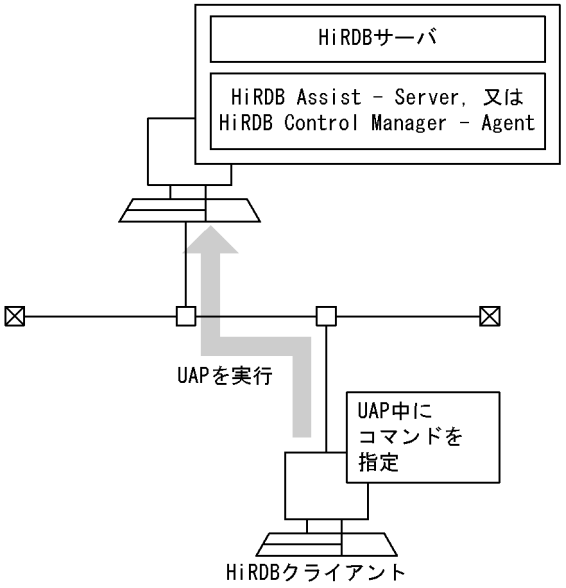
  HiRDB の運用コマンド，ユティリティ，及び OS のコマンドを実行します。

  COMMAND EXECUTE からのコマンド実行は，HiRDB クライアントと HiRDB Control Manager - Agent が連携することで実現しているため，HiRDB サーバ側に HiRDB Control Manager - Agent をインストールする必要があります。HiRDB Control Manager - Agent については，それぞれのリリースノートを参照してください。

  COMMAND EXECUTE からのコマンド実行は，C 言語の場合にだけ使用できます。

  COMMAND EXECUTE からのコマンド実行の概要を次の図に示します。

  図 12–1　COMMAND EXECUTE からのコマンド実行の概要



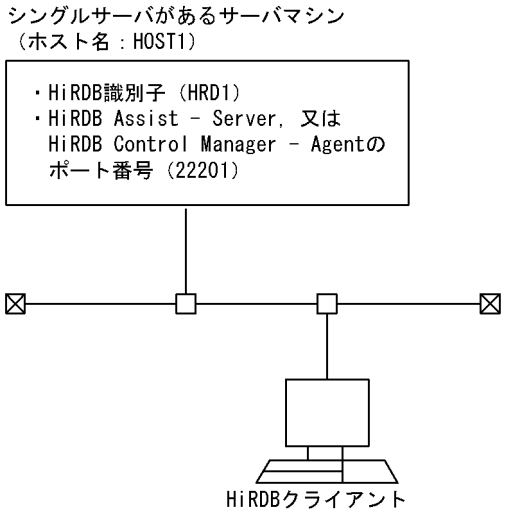
12.2　COMMAND EXECUTE からコマンドを実行するための準備

12.2.1　HiRDB/シングルサーバの場合

データロード（データベース作成ユティリティ）を実行する UAP を例にして説明します。

HiRDB/シングルサーバの場合のサーバ，クライアント構成例を次の図に示します。

図 12–2　HiRDB/シングルサーバの場合のサーバ，クライアント構成例



図「HiRDB/シングルサーバの場合のサーバ，クライアント構成例」のようなサーバ，クライアント構成で，データロードをする UAP を実行する場合，事前に次の設定をしておきます。

1. 次のクライアント環境定義を設定します。

   PDSYSTEMID：

   HiRDB サーバの HiRDB 識別子（HRD1）を設定します。

   PDASTHOST：

   HiRDB Control Manager - Agent のホスト名（HOST1）を設定します。

   PDASTPORT：

   HiRDB Control Manager - Agent のポート番号（22201）を設定します。

2. HiRDB サーバ側に，データロードで必要となる制御情報ファイル，及び入力データファイルを用意します。

3. HiRDB 管理者が USERA（パスワード USERA）で，データロードをする表の所有者が USERB（パスワード USERB）であったとします。この場合，次のクライアント環境定義を設定します。

```
PDASTUSER=USERA/USERA
PDUSER=USERB/USERB
```

これで，データロードをする UAP を実行できるようになります。なお，各クライアント環境定義については，「クライアント環境定義の設定内容」を参照してください。

データロードを実行する UAP の例を次に示します。

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

EXEC SQL BEGIN DECLARE SECTION;
char CmdLine[30000];                    /* CmdLine変数 */
long ReturnCode;                        /* リターンコード受取変数 */
long OutBufLen;                         /* 実行結果受取領域長 */
long CmdRetCode;                        /* 実行コマンドリターンコード
                                           受取変数 */
long OutDataLen;                        /* 実行結果長受取変数 */
PDOUTBUF OutBuf;                        /* 実行結果受取領域 */
char EnvGroup[256];                     /* 環境変数グループ名変数 */
EXEC SQL END DECLARE SECTION;

void main()
{
strcpy(CmdLine,"pdhold -r RDDATA10");  /* 実行コマンドライン
                                          (RDエリア閉塞)設定 */
OutBuf = malloc(30000);                 /* 実行結果受取領域確保 */
if (OutBuf == NULL){                    /* メモリ確保失敗 */
printf("メモリ確保失敗¥n");
return ;
}
OutBufLen = 30000 ;                     /* 実行結果受取領域長設定 */
EnvGroup[0] = '¥0' ;                    /* 環境変数グループ設定なし */

/* コマンド実行 */
EXEC  SQL  COMMAND  EXECUTE  :CmdLine, :ReturnCode, :OutBufLen,
:OutDataLen, :OutBuf, :CmdRetCode, :EnvGroup ;
if (ReturnCode == p_rdb_RC_NORM) {      /* COMMAND  EXECUTE正常終了 */
if (CmdRetCode==0) {                    /* 実行コマンド正常 */

/* 実行コマンドライン(データロード実行)設定 */
strcpy(CmdLine,"pdload  -i c  -be ZAIKO c:¥HiRDB_S¥conf¥LOAD");
EXEC  SQL  COMMAND  EXECUTE  :CmdLine, :ReturnCode, :OutBufLen,
:OutDataLen, :OutBuf, :CmdRetCode, :EnvGroup ;
if (ReturnCode == p_rdb_RC_NORM) {     /* COMMAND  EXECUTE正常終了 */
if (CmdRetCode==0) {                    /* 実行コマンド正常 */
printf("pdload command successfully¥n");
printf("%s¥n", OutBuf);
} else {                                /* 実行コマンドエラー */
printf("pdload command Error,Code = %d¥n", CmdRetCode);
printf("%s¥n", OutBuf);
}
} else {                                /* COMMAND  EXECUTEエラー */
printf("COMMAND EXECUTE Error,Code = %d¥n", ReturnCode);
printf("%s¥n", OutBuf);
}
} else {                                /* 実行コマンドエラー */
printf("pdhold command Error,Code = %d¥n", CmdRetCode);
printf("%s¥n", OutBuf);
```

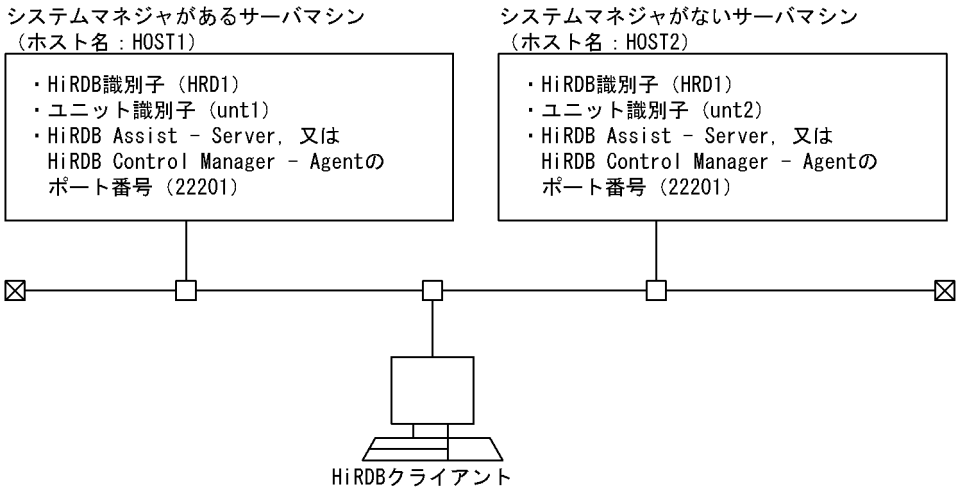
```
}
strcpy(CmdLine,"pdrels -r RDDATA10");  /* 実行コマンドライン
                                          (RDエリア閉塞解除)設定 */
EXEC  SQL  COMMAND  EXECUTE  :CmdLine, :ReturnCode, :OutBufLen,
:OutDataLen, :OutBuf, :CmdRetCode, :EnvGroup ;
if (ReturnCode == p_rdb_RC_NORM) {     /* COMMAND  EXECUTE正常終了 */
if (CmdRetCode!=0) {                   /* 実行コマンドエラー */
printf("pdrels command Error,Code = %d¥n", CmdRetCode);
printf("%s¥n", OutBuf);
}
} else {                               /* COMMAND  EXECUTEエラー */
printf("COMMAND EXECUTE Error,Code = %d¥n", ReturnCode);
printf("%s¥n", OutBuf);
}
} else {                               /* COMMAND  EXECUTEエラー */
printf("COMMAND EXECUTE Error,Code = %d¥n", ReturnCode);
printf("%s¥n", OutBuf);
}
return ;
}
```

12.2.2　HiRDB/パラレルサーバの場合

データロード（データベース作成ユティリティ）を実行する UAP を例にして説明します。

HiRDB/パラレルサーバの場合のサーバ，クライアント構成例を次の図に示します。

図 12–3　HiRDB/パラレルサーバの場合のサーバ，クライアント構成例



図「HiRDB/パラレルサーバの場合のサーバ，クライアント構成例」のようなサーバ，クライアント構成で，データロードをする UAP を実行する場合，事前に次の設定をします。

1. 次のクライアント環境定義を設定します。

   PDSYSTEMID：

   HiRDB サーバの HiRDB 識別子（HRD1）を設定します。

PDASTHOST：

HiRDB Control Manager - Agent のホスト名（HOST1）を設定します。HiRDB/パラレルサーバの場合は，システムマネジャがあるサーバマシンのホスト名を指定します。

PDASTPORT：

HiRDB Control Manager - Agent のポート番号（22201）を設定します。

2. HiRDB サーバ側に，データロードで必要となる制御情報ファイル，及び入力データファイルを用意します。
3. HiRDB 管理者が USERA（パスワード USERA）で，データロードをする表の所有者が USERB（パスワード USERB）であったとします。この場合，次のクライアント環境定義を設定します。

```
PDASTUSER=USERA/USERA
PDUSER=USERB/USERB
```

これで，データロードをする UAP を実行できるようになります。なお，各クライアント環境定義については，「クライアント環境定義の設定内容」を参照してください。

データロードを実行する UAP の例を次に示します。

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

EXEC SQL BEGIN DECLARE SECTION;
char CmdLine[30000];                    /* CmdLine変数 */
long ReturnCode;                        /* リターンコード受取変数 */
long OutBufLen;                         /* 実行結果受取領域長 */
long CmdRetCode;                        /* 実行コマンドリターンコード
                                           受取変数 */
long OutDataLen;                        /* 実行結果長受取変数 */
PDOUTBUF OutBuf;                        /* 実行結果受取領域 */
char EnvGroup[256];                     /* 環境変数グループ名変数 */
EXEC SQL END DECLARE SECTION;

void main()
{
strcpy(CmdLine,"pdhold -r RDDATA10"); /* 実行コマンドライン
                                           (RDエリア閉塞)設定 */
OutBuf = malloc(30000);                 /* 実行結果受取領域確保 */
if (OutBuf == NULL){                    /* メモリ確保失敗 */
printf("メモリ確保失敗¥n");
return ;
}
OutBufLen = 30000 ;                     /* 実行結果受取領域長設定 */
EnvGroup[0] = '¥0' ;                    /* 環境変数グループ設定なし */

/* コマンド実行 */
EXEC  SQL  COMMAND  EXECUTE  :CmdLine, :ReturnCode, :OutBufLen,
:OutDataLen, :OutBuf, :CmdRetCode, :EnvGroup ;
if (ReturnCode == p_rdb_RC_NORM) {      /* COMMAND  EXECUTE正常終了 */
if (CmdRetCode==0) {                    /* 実行コマンド正常 */
```

```
/* 実行コマンドライン(データロード実行)設定 */
strcpy(CmdLine,"pdload  -i c  -be ZAIKO c:¥HiRDB_P¥conf¥LOAD");
EXEC  SQL  COMMAND  EXECUTE  :CmdLine, :ReturnCode, :OutBufLen,
:OutDataLen, :OutBuf, :CmdRetCode, :EnvGroup ;
if (ReturnCode == p_rdb_RC_NORM) {    /* COMMAND  EXECUTE正常終了 */
if (CmdRetCode==0) {                  /* 実行コマンド正常 */
printf("pdload command successfully¥n");
printf("%s¥n", OutBuf);
} else {                              /* 実行コマンドエラー */
printf("pdload command Error,Code = %d¥n", CmdRetCode);
printf("%s¥n", OutBuf);
}
} else {                              /* COMMAND  EXECUTEエラー */
printf("COMMAND EXECUTE Error,Code = %d¥n", ReturnCode);
printf("%s¥n", OutBuf);
}
} else {                              /* 実行コマンドエラー */
printf("pdhold command Error,Code = %d¥n", CmdRetCode);
printf("%s¥n", OutBuf);
}
strcpy(CmdLine,"pdrels -r RDDATA10"); /* 実行コマンドライン
                                         (RDエリア閉塞解除)設定 */
EXEC  SQL  COMMAND  EXECUTE  :CmdLine, :ReturnCode, :OutBufLen,
:OutDataLen, :OutBuf, :CmdRetCode, :EnvGroup ;
if (ReturnCode == p_rdb_RC_NORM) {   /* COMMAND  EXECUTE正常終了 */
if (CmdRetCode!=0) {                 /* 実行コマンドエラー */
printf("pdrels command Error,Code = %d¥n", CmdRetCode);
printf("%s¥n", OutBuf);
}
} else{                              /* COMMAND  EXECUTEエラー */
printf("COMMAND EXECUTE Error,Code = %d¥n", ReturnCode);
printf("%s¥n", OutBuf);
}
} else {                             /* COMMAND  EXECUTEエラー */
printf("COMMAND EXECUTE Error,Code = %d¥n", ReturnCode);
printf("%s¥n", OutBuf);
}
return ;
}
```

12.3　コマンドの実行可否

HiRDB のコマンドには，UAP から実行できるものとできないものとがあります。UAP からのコマンドの実行可否を次の表に示します。

表 12–1　UAP からのコマンドの実行可否

種別	コマンド	内容	COMMAND EXECUTE からの実行可否	CALL COMMAND からの実行可否
システムの運用	pdadmvr	HiRDB バージョン情報の取得	○	○
	pdcat	ファイルの内容表示	○	○
	pdchgconf	システム構成変更コマンド	×	○
	pdclibsync	C ライブラリファイルの操作	○	○
	pdconfchk	システム定義のチェック	×	○
	pdcspool	トラブルシュート情報の削除	○	○
	pddivinfgt	表の分割条件の取得及び出力	×	×
	pdgeter	障害情報の取得	○	○
	pdinfoget	障害情報の取得と容量見積もり	×	○
	pdinfocoreget	core ファイルの取得と容量見積もり	×	×
	pdosinfoget	OS 情報の取得	×	×
	pditvtrc	HiRDB の状態の定期取得	○	○
	pditvstop	HiRDB の状態の定期取得の停止	○	○
	pdjarsync	JAR ファイルの操作	○	○
	pdlistls	リスト定義情報の表示	○	○
	pdls	HiRDB システムの状態表示	○	○
	pdmemsv	メモリの削減	×	×
	pdntenv	HiRDB の動作環境の設定	×	×
	pdobjconv	64 ビットモードの HiRDB への SQL オブジェクトの移行	○	×
	pdopsetup	HiRDB 付加 PP の組み込み	×	×
	pdseset	SELinux セキュリティコンテキストの設定及び解除	×	×
	pdsetenv	HiRDB ユニットの環境設定	×	×
	pdsetup	HiRDB システムの OS への登録・削除	×	×

種別	コマンド	内容	COMMAND EXECUTE からの実行可否	CALL COMMAND からの実行可否
	pdsvhostname	サーバのホスト名表示	×	○
	pdvrup	HiRDB バージョンアップ	×	○
HiRDB ファイルシステム	pdfbkup	HiRDB ファイルシステムのバックアップ	○	○
	pdfchfs	HiRDB ファイルシステム領域の管理情報変更	×	○
	pdfls	HiRDB ファイルシステムの内容表示	○	○
	pdfmkfs	HiRDB ファイルシステム領域の初期設定	○	○
	pdfrm	HiRDB ファイルの削除	○	○
	pdfrstr	HiRDB ファイルシステムのリストア	○	○
	pdfstatfs	HiRDB ファイルシステム領域の状態表示	○	○
	pdffsck	HiRDB ファイルシステム領域の整合性の検証及び修復	○	○
	pdfzeroinit	HiRDB ファイルシステム領域内の使用済み領域の初期化	×	○
ログ関係のファイル	pdlogadpf	ログ関係のファイルの割り当て	○	○
	pdlogatul	自動ログアンロード機能の制御	×	○
	pdlogchg	ログ関係のファイルのステータス変更	○	○
	pdlogcls	ログ関係のファイルのクローズ	○	○
	pdloginit	ログ関係のファイルの初期設定	○	○
	pdlogls	ログ関係のファイルの情報表示	○	○
	pdlogopen	ログ関係のファイルのオープン	○	○
	pdlogrm	ログ関係のファイルの削除	○	○
	pdlogswap	ログ関係のファイルのスワップ	○	○
	pdlogsync	シンクポイントダンプの取得	○	○
	pdlogucat	アンロードログファイルの情報表示	○	○
	pdlogunld	ログ関係のファイルのアンロード	○	○
ステータスファイル	pdstscls	ステータスファイルのクローズ	○	○
	pdstsinit	ステータスファイルの初期設定	○	○
	pdstsopen	ステータスファイルのオープン	○	○
	pdstsrm	ステータスファイルの削除	○	○
	pdstsswap	ステータスファイルのスワップ	○	○

種別	コマンド	内容	COMMAND EXECUTE からの実行可否	CALL COMMAND からの実行可否
HiRDB の開始・終了	pdstart	HiRDB システム・ユニット・サーバの開始	○	○
	pdstop	HiRDB システム・ユニット・サーバの終了	○	○
統計ログ	pdstbegin	統計情報の出力開始	○	○
	pdstend	統計情報の出力停止	○	○
	pdstjswap	統計ログファイルの切り替え	○	○
	pdstjsync	統計ログファイルへの統計ログバッファの反映	○	○
RD エリア	pdclose	RD エリアのクローズ	○	○
	pddbls	RD エリアの状態表示	○	○
	pdhold	RD エリアの閉塞	○	○
	pdopen	RD エリアのオープン	○	○
	pdrels	RD エリアの閉塞解除	○	○
	pddbfrz	ユーザ LOB 用 RD エリアの満杯 HiRDB ファイルの更新凍結	○	○
	pdrdrefls	関連する RD エリアの情報の表示	○	○
グローバルバッファ	pdbufls	グローバルバッファ情報の表示	○	○
	pdbufmod	グローバルバッファの動的変更	○	○
トランザクションの制御	pdcmt	トランザクションのコミット	○	○
	pdfgt	トランザクションの強制終了	○	○
	pdrbk	トランザクションのロールバック	○	○
	pdtrndec	未決着トランザクションの強制自動決着	×	○
プロセスの制御	pdcancel	UAP，ユティリティ処理の強制終了	○	○
	pdchprc	サーバプロセスの起動本数変更	○	○
	pdkill	プロセスの強制停止	×	○
	pdpfresh	サーバプロセスのリフレッシュ	○	○
	pdrpause	プロセスサーバプロセスの再起動	×	○
修正版 HiRDB の入れ替え	pdprgcopy	修正版 HiRDB のコピー	×	○
	pdprgrenew	修正版 HiRDB の入れ替え	×	○
HiRDB Datareplicator 連携	pdrplstart	HiRDB Datareplicator 連携の開始	×	○
	pdrplstop	HiRDB Datareplicator 連携の終了	×	○

種別	コマンド	内容	COMMAND EXECUTE からの実行可否	CALL COMMAND からの実行可否
インナレプリカ機能	pddbchg	レプリカ RD エリアのレプリカステータスの切り替え	○	○
更新可能なオンライン再編成	pdorbegin	オンライン再編成のデータベース静止化	○	○
	pdorcheck	オンライン再編成の適用条件チェック	○	○
	pdorchg	オンライン再編成のカレント RD エリアの切り替え	○	○
	pdorcreate	オンライン再編成の追い付き反映環境の作成	○	○
	pdorend	オンライン再編成の追い付き反映	○	○
セキュリティ監査	pdaudbegin	監査証跡の取得開始	○	○
	pdaudend	監査証跡の取得停止	○	○
	pdaudrm	閉塞中の監査証跡ファイルの削除	○	○
	pdaudswap	現用の監査証跡ファイルのスワップ	○	○
	pdaudatld	監査証跡表への自動データロード機能の制御	○	○
	pdaudput	JP1/NETM/Audit 用監査ログ出力	○	×
CONNECT 関連セキュリティ機能	pdacunlck	連続認証失敗アカウントロック状態の解除	×	○
リアルタイム SAN レプリケーション	pdrisechk	リアルタイム SAN レプリケーションの構成確認	×	○
	pdrisedbto	リアルタイム SAN レプリケーションのデータベース引き継ぎ	×	○
	pdriseset	リアルタイム SAN レプリケーションのサイト状態の設定	×	○
インメモリデータ処理	pdmemdb	インメモリデータ処理に関する操作	○	○
SQL トレースの取得	pdclttrc	SQL トレースの動的取得	○	○
PRF トレース情報	pdprfed	PRF トレース情報の編集出力	×	×
	pdprflevel	PRF トレース取得レベルの表示及び変更	×	×
SQL オブジェクトの情報表示	pdobils	SQL オブジェクトの統計情報表示	×	○
SQL の仕様関連	pdextfunc	拡張システム定義スカラ関数の定義及び削除	○	○
Windows ファイアウォールの例外登録	pddelfw.bat	Windows ファイアウォールの例外リストから HiRDB を削除	×	×
	pdsetfw.bat	Windows ファイアウォールの例外リストに HiRDB を登録	×	×
ファイルセキュリティ強化機能	pdsetacl	ファイルセキュリティ強化機能の設定	×	×

種別	コマンド	内容	COMMAND EXECUTE からの実行可否	CALL COMMAND からの実行可否
SQL の翻訳	pdcbl	COBOL プリプロセサ	×	×
	pdcpp	C プリプロセサ	×	×
	pdocb	OOCOBOL プリプロセサ	×	×
	pdocc	C++プリプロセサ	×	×
データベースの生成	pdinit	データベース初期設定ユティリティ	×	○
	pddef	データベース定義ユティリティ	×	○
	pdload	データベース作成ユティリティ	○	○
	pdparaload	パラレルローディング	×	×
	pdsql※	会話型 SQL 実行ユティリティ	×	×
	pddefrev	定義系 SQL の生成	×	○
データベースの運用	pdmod	データベース構成変更ユティリティ	○	○
	pdrorg	データベース再編成ユティリティ	○	○
	pdexp	ディクショナリ搬出入ユティリティ	×	○
	pdrbal	リバランスユティリティ	×	○
	pdreclaim	空きページ解放ユティリティ	○	○
	pdpgbfon	グローバルバッファ常駐化ユティリティ	○	○
	pdconstck	整合性チェックユティリティ	×	○
	pdchpathf	ファイルパス変更オフラインコマンド	×	×
	pdchpathn	ファイルパス変更オンラインコマンド	×	×
チューニング	pdstedit	統計解析ユティリティ	×	○
	pddbst	データベース状態解析ユティリティ	○	○
	pdgetcst	最適化情報収集ユティリティ	×	○
	pdvwopt	アクセスパス表示ユティリティ	×	○
データベースの障害対策	pdcopy	データベース複写ユティリティ	○	○
	pdbkupls	バックアップファイルの情報表示	○	○
	pdrstr	データベース回復ユティリティ	○	○
	pdmstchk	マスタディレクトリ用 RD エリアの整合性チェック	×	×
プラグイン関連	pdplgrgst	プラグインの登録	×	×
	pdplgset	プラグインのセットアップ	×	×

種別	コマンド	内容	COMMAND EXECUTE からの実行可否	CALL COMMAND からの実行可否
	pdreginit	レジストリ機能初期設定ユティリティ	○	×
コマンドの代行	pdcmdact	HiRDB 管理者以外のユーザによるコマンド実行	×	×
	pdcmdls	コマンドの実行権限表示	×	×
	pdcmdset	コマンドの実行権限変更	×	×
リードレプリカ機能	pdrrpref	参照 DB の更新	×	×

（凡例）

○：UAP から実行できます。

×：UAP から実行できません。

注※

Windows 版の場合，コマンドはありません。HiRDB SQL Executer で実行します。

13 ODBC 対応アプリケーションプログラムからの HiRDB アクセス

この章では，ODBC 対応アプリケーションプログラムから HiRDB をアクセスする場合に必要な，ODBC ドライバのインストール，ODBC 関数，チューニング，トラブルシュートなどについて説明します。

なお，ODBC2.0 ドライバは廃止になりました。現在はアプリケーションの互換性を保つために使用できますが，将来は削除されます。代わりに ODBC3.5 ドライバを使用してください。

ODBC2.0 ドライバを使用する場合は，HiRDB サーバとの接続時にパスワードを指定するユーザインタフェースで，29 バイト以上のパスワードを引用符で囲んだ指定はできません。

13.1　ODBC 対応アプリケーションプログラム

ODBC 対応アプリケーションプログラムには，Microsoft Access や Microsoft Excel などがあります。これらのアプリケーションプログラムから HiRDB をアクセスする場合，ODBC ドライバをインストールする必要があります。ODBC ドライバのインストールについては，「ODBC2.0 ドライバのインストール」を参照してください。また，HiRDB が提供している ODBC 関数を使用した UAP から，ODBC ドライバを経由して HiRDB をアクセスすることもできます。HiRDB が提供する ODBC 関数については，「HiRDB が提供する ODBC 関数」を参照してください。

なお，ODBC3.5 ドライバを経由すると，ODBC3.X のインタフェースを使用している UAP から HiRDB にアクセスできます。

13.2 ODBC2.0 ドライバのインストール

ODBC 対応のアプリケーションプログラム，又は ODBC 関数を使用した UAP から HiRDB をアクセスする場合，HiRDB クライアントに ODBC ドライバをインストールする必要があります。

また，HiRDB サーバ上で ODBC 経由の UAP を実行する場合は，HiRDB サーバにも ODBC ドライバをインストールする必要があります。

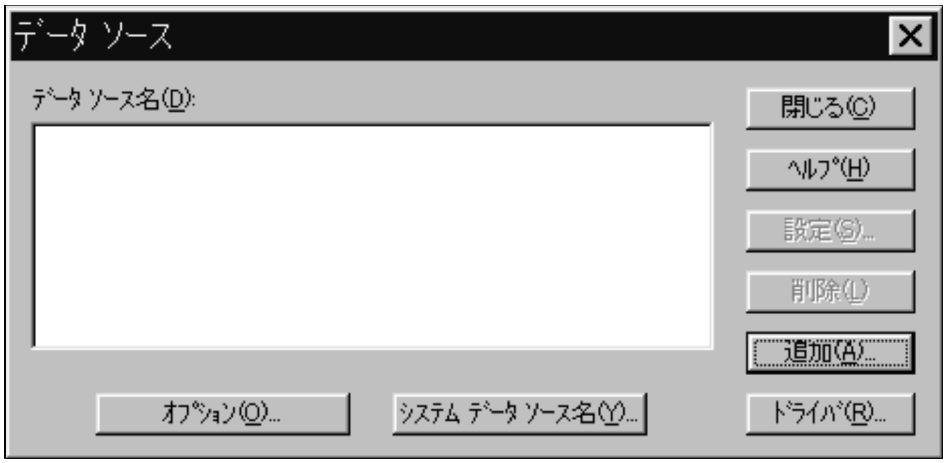
ODBC ドライバのインストール手順を次に示します。なお，インストールを実行する前に，すべての Windows アプリケーションを終了させてください。

1. HiRDB クライアントのインストール後，HiRDB クライアントのインストールディレクトリ¥utl ディレクトリ下に展開される ODBC20INST.exe ファイルを実行してください。日立自己展開型プログラムが起動します。

2. 「日立自己展開型プログラム」画面で「インストール実行(R)」ボタンをクリックすると，ODBC ドライバのインストーラが起動します。

3. 表示された HiRDB のドライバを選択して［OK］ボタンをクリックしてください。選択しないで OK した場合，インストールされないので注意してください。

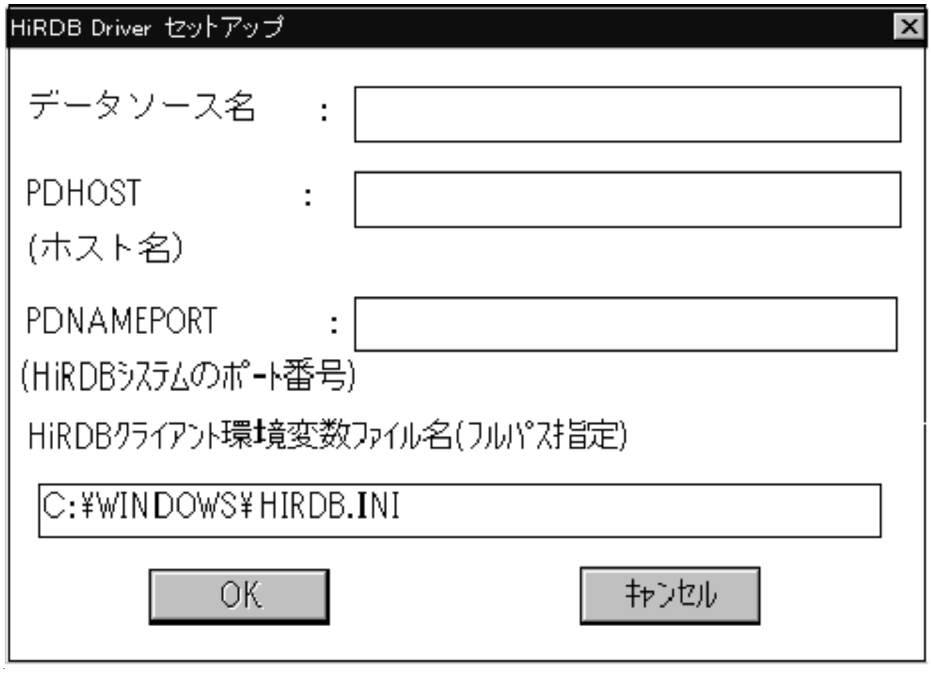
4. 今までに定義したデータソースが表示されます。データソースを何も定義していない場合は，何も表示されません。［追加］ボタンをクリックします。



5. データソースを追加する対象として HiRDB のドライバを選択します。



6. データソースのセットアップ画面が表示されます。



[説明]

データソース名

データソースを識別するための，任意の名称を指定します。名称は，すべて半角文字の場合は 32 文字，すべて全角文字の場合は 16 文字で指定できます。なお，半角文字，全角文字は混在できます。

PDHOST(ホスト名)

サーバマシンのホスト名を指定します。指定する内容は，クライアント環境定義と同じです。PDHOST の内容については，「クライアント環境定義の設定内容」を参照してください。この項目を省略した場合，クライアント環境定義の値が仮定されます。

PDNAMEPORT(HiRDB システムのポート番号)

サーバマシンのポート番号を指定します。指定する内容は，クライアント環境定義と同じです。PDNAMEPORT の内容については，「クライアント環境定義の設定内容」を参照してください。この項目を省略した場合，クライアント環境定義の値が仮定されます。

HiRDB クライアント環境変数ファイル名(フルパス指定)※

HiRDB クライアント環境定義ファイルの名称を，絶対パス名で指定します。データソースごとに HiRDB クライアント環境変数の指定値を変更したい場合に指定してください。例えば，高速接続機能（PDSERVICEPORT）を使用し複数の HiRDB に接続する場合に，HiRDB クライアント環境定義ファイルのファイル名を指定し，データソースごとに接続先を変更するときなどに指定します。

省略した場合は，HIRDB.INI が仮定されます。PDHOST，PDNAMEPORT 以外のクライアント環境変数は，ここで指定した HiRDB クライアント環境定義ファイルの設定値が使用されます。

ここで HIRDB.INI 以外のファイルを指定した場合，HIRDB.INI 内の指定は無視されます。

7. すべての項目を設定した後，［OK］ボタンをクリックしてください。すると，設定したデータソースが表示されます。設定を変更する場合は，［設定］ボタンをクリックすると一つ前の画面に戻ります。



注※

HiRDB クライアント環境定義ファイルは，HiRDB クライアントをインストールすると，自動的にシステムディレクトリに HIRDB.INI というファイル名で作成されます。

HiRDB クライアントをインストールする前に ODBC ドライバをインストールするときには，この HIRDB.INI ファイルはないため，あらかじめユーザが作成しておく必要があります。

クライアント環境定義ファイルを作成する場合は，インストール CD-ROM 中の odb32¥Disk1¥Sampleap ディレクトリにある HIRDB.INI ファイルを適当なディレクトリにコピーし，編集してください。なお，各クライアント環境変数の内容については，「クライアント環境定義の設定内容」を参照してください。

13.3　ODBC3.5 ドライバのインストールと環境変数の設定

13.3.1　インストール

(1)　インストールディレクトリ

ODBC3.5 ドライバのインストールディレクトリを次の表に示します。

表 13–1　ODBC3.5 ドライバ（32 ビットモード）のインストールディレクトリ

プラットフォーム	インストールディレクトリ
Windows	Windows ディレクトリ¥System32，又は Windows ディレクトリ¥SysWOW64
Linux	/HiRDB/client/lib/

注 1

Windows ディレクトリは，デフォルトでは，C:¥WINDOWS となります。

注 2

下線で示す部分は，HiRDB クライアントのインストールディレクトリとなります。

表 13–2　ODBC3.5 ドライバ（64 ビットモード）のインストールディレクトリ

プラットフォーム	インストールディレクトリ
Windows	Windows ディレクトリ¥System32
Linux	/HiRDB/client/lib/

注 1

Windows ディレクトリは，デフォルトでは，C:¥WINDOWS となります。

注 2

下線で示す部分は，HiRDB クライアントのインストールディレクトリとなります。

(2)　インストールの流れ

ODBC3.5 ドライバのインストールの流れを次に示します。

1. ODBC3.5 ドライバのインストール
   提供媒体をセットし，インストールをします。
2. ODBC ドライバマネージャのインストール
   Windows 版の場合，ODBC ドライバマネージャのバージョンが古いときは，ODBC ドライバマネージャをインストールします。

UNIX 版の場合，HiRDB では ODBC ドライバマネージャを提供していません。ODBC3.5 API をサポートし UNIX 上で動作する ODBC ドライバマネージャを別途インストールしてください。

3. データソースの設定
   データソースを設定します。

(3) インストール手順（Windows 版の場合）

(a) ODBC3.5 ドライバのインストール

1. 統合 CD-ROM 中の hcd_inst.exe を実行して，日立総合インストーラを起動してください。
2. 「日立総合インストーラ」画面で次のどちらかを選択して，［インストール実行］ボタンをクリックしてください。HiRDB のセットアッププログラムが起動します。

   Windows 版クライアント製品の場合：

   - HiRDB/Run Time の場合は［HiRDB/Run Time］
   - HiRDB/Developer's Kit の場合は［HiRDB/Developer's Kit］

   Windows 版サーバ製品の場合：

   - HiRDB/シングルサーバの場合は［HiRDB/Single Server］
   - HiRDB/パラレルサーバの場合は［HiRDB/Parallel Server］
3. 次の操作をしてください。選択したプログラムプロダクトのセットアッププログラムが起動します。

   Windows 版クライアント製品の場合：

   HiRDB のセットアッププログラムの「プログラムプロダクトの選択」画面で次のどちらかを選択して，［次へ］ボタンをクリックしてください。

   - HiRDB/Run Time の場合は［HiRDB/Run Time］
   - HiRDB/Developer's Kit の場合は［HiRDB/Developer's Kit］

   Windows 版サーバ製品の場合：

   HiRDB のセットアッププログラムの「プログラムプロダクトの選択」画面で［HiRDB/Run Time］を選択して，［次へ］ボタンをクリックしてください。
4. 「インストール先の選択」ダイアログボックスが表示されるので，必要に応じてインストール先を変更して［次へ］ボタンをクリックします。
5. 「セットアップ方法の選択」ダイアログボックスで，「標準」又は「カスタム」を選択し，［次へ］ボタンをクリックします。
6. 5.で「カスタム」を選択した場合，「コンポーネントの選択」ダイアログボックスで，「ODBC3.5 ドライバ」を選択し，［次へ］ボタンをクリックします。
7. ODBC3.5 ドライバが Windows ディレクトリ¥System32 下にコピーされます。
8. これでインストールは完了となります。

(b)　ODBC ドライバマネージャ（MDAC2.6RTM に含まれる）のインストール

インストールされている ODBC ドライバマネージャのバージョンが古い場合，Microsoft のホームページから最新の MDAC を入手してインストールする必要があります。なお，ODBC ドライバマネージャのバージョンは，「ODBC アドミニストレータ」を起動して，「バージョン情報」タブをクリックすると確認できます。「ドライバマネージャ」のバージョンが 3.520.6526.0 未満の場合，古いバージョンとなります。

(c)　データソースの設定

1. 「ODBC データソースアドミニストレータ」を起動します。
2. タブの項目が「ユーザ DSN」又は「システム DSN」であることを確認し，［追加］ボタンをクリックします。
3. 「データソースの新規作成」ダイアログボックスが表示されるので，「HiRDB ODBC3.5 Driver」を選択して，［完了］ボタンをクリックします。
4. 「HiRDB ODBC3.5 Driver セットアップ」ダイアログボックスが表示されるので，各項目を設定します。

データソース名

データソース名を識別するための，任意の名称を指定します。名称は，すべて半角文字の場合は 32 文字，すべて全角文字の場合は 16 文字で指定できます。なお，半角文字，全角文字は混在できます。

PDHOST(ホスト名)

HiRDB/シングルサーバの場合，シングルサーバのあるサーバマシンのホスト名を指定します。HiRDB/パラレルサーバの場合，システムマネジャがあるサーバマシンのホスト名を指定します。

この指定はクライアント環境定義 PDHOST の指定と同等です。PDHOST については，「クライアント環境定義の設定内容」を参照してください。

この項目を省略した場合の各指定方法での優先順位については，「接続情報の優先順位」を参照してください。

PDNAMEPORT(HiRDB システムのポート番号)

アクセスする HiRDB サーバのポート番号（システム定義の pd_name_port オペランドの指定値）を指定します。

この指定はクライアント環境定義 PDNAMEPORT の指定と同等です。PDNAMEPORT については，「クライアント環境定義の設定内容」を参照してください。

この項目を省略した場合の各指定方法での優先順位については，「接続情報の優先順位」を参照してください。

HiRDB クライアント環境変数ファイル名

HiRDB クライアント環境定義ファイルの名称を，絶対パス名で指定します。データソースごとに HiRDB クライアント環境変数の指定値を変更したい場合に指定してください。例えば，高速接続機能（PDSERVICEPORT）を使用し複数の HiRDB に接続する場合に，HiRDB クライアント環境定義ファイルのファイル名を指定し，データソースごとに接続先を変更するときなどに指定します。

HiRDB クライアント環境定義ファイルについては，「Windows 環境の場合（ファイル登録）」を参照してください。

省略した場合は，HIRDB.INI が仮定されます。

クライアント環境定義の各指定方法での優先順位については，「接続情報の優先順位」を参照してください。

5. ［OK］ボタンをクリックすると，「ユーザ DSN」タブ又は「システム DSN」タブに戻り，登録したデータソースが表示されます。

- **データソースのセットアップの中止**

データソースのセットアップを中止する場合は，「HiRDB ODBC3.5 Driver セットアップ」ダイアログボックスの［キャンセル］ボタンをクリックしてください。［キャンセル］ボタンをクリックすると，データソースは登録されません。

- **データソースの削除**

データソースを削除する場合の手順を次に示します。

1.「データソース」ダイアログボックスの中の，削除するデータソース名を選択します。
2. ［削除］ボタンをクリックすると，データソースが削除されます。

(4)　インストール手順（UNIX 版の場合）

(a)　ODBC3.5 ドライバのインストール

日立 PP インストーラを起動して ODBC3.5 ドライバをインストールしてください。

(b)　ODBC ドライバマネージャのインストール

UNIX 版の場合，HiRDB では ODBC ドライバマネージャを提供していません。別途インストールしてください。

(c)　ODBC ドライバ情報の登録

ODBC ドライバの情報を登録するために，odbcinst.ini ファイルを編集してください。

ここでは，ドライバマネージャに unixODBC を使用した場合を例にして説明します。

odbcinst.ini ファイルは/usr/local/etc 下にあります。32 ビットモードの Linux 版 HiRDB ODBC3.5 ドライバを使用する場合の odbcinst.ini ファイルの編集例を次に示します。

（例）

```
[HiRDBOdbcDriver] ..............................................1
Driver = /HiRDB/client/lib/libodbcdrv.so  ......................2
```

1. **ドライバ名称**

   [ ]内の記述は，データソースと対応するドライバ名称です。任意の名称を指定できます。

2. **Driver**

UNIX 版 HiRDB ODBC3.5 ドライバを絶対パスで指定します。64 ビットモードの場合は libodbcdrv64.so となります。

(d) データソースの設定

odbc.ini ファイルを編集してください。

ここでは，ドライバマネージャに unixODBC を使用した場合を例にして説明します。

odbc.ini ファイルは，/usr/local/etc 下にあります。また，ホームディレクトリ下の.odbc.ini という隠しファイルとしても存在します。この二つのファイルは，Windows でのシステム DSN 及びユーザ DSN に相当します。ユーザ DSN として編集する場合は，.odbc.ini ファイルをアプリケーション実行ユーザのホームディレクトリに配置してください。

odbc.ini ファイルの編集例を次に示します。

（例）

```
[HiRDB_LIN30] ..................................................1
Driver = HiRDBOdbcDriver  ......................................2
PDHOST = 10.209.34.223  ........................................3
PDNAMEPORT = 22200  ............................................4
INIFLNAME = /usr/local/etc/HiRDB.ini  ..........................5
ODBCHARSET = CP932  ............................................6
```

1. **データソース名**

   データソース名を識別するための，任意の名称を指定します。使用する ODBC ドライバマネージャの規則に従って指定してください。

2. **Driver**

   ドライバの登録で odbcinst.ini ファイルに設定したドライバ名称を指定します。

3. **PDHOST**

   HiRDB/シングルサーバの場合，シングルサーバのあるサーバマシンのホスト名を指定します。HiRDB/パラレルサーバの場合，システムマネジャがあるサーバマシンのホスト名を指定します。

   この指定はクライアント環境定義 PDHOST の指定と同等です。PDHOST については，「クライアント環境定義の設定内容」を参照してください。

   この項目を省略した場合の各指定方法での優先順位については，「接続情報の優先順位」を参照してください。

4. **PDNAMEPORT**

   アクセスする HiRDB サーバのポート番号（システム定義の pd_name_port オペランドの指定値）を指定します。

   この指定はクライアント環境定義 PDNAMEPORT の指定と同等です。PDNAMEPORT については，「クライアント環境定義の設定内容」を参照してください。

   この項目を省略した場合の各指定方法での優先順位については，「接続情報の優先順位」を参照してください。

5. **INIFLNAME**

   HiRDB クライアント環境定義ファイルの名称を，絶対パス名で指定します。データソースごとに HiRDB クライアント環境定義の指定値を変更したい場合に指定してください。例えば，高速接続機能（PDSERVICEPORT）を使用し複数の HiRDB に接続する場合に，HiRDB クライアント環境定義ファイルのファイル名を指定し，データソースごとに接続先を変更するときなどに指定します。

   HiRDB クライアント環境定義ファイルについては，「UNIX 環境の場合」を参照してください。

   なお，UNIX 版では HiRDB クライアント環境定義ファイル（HiRDB.ini）は作成されないため，ファイルを用意してから指定してください。

   クライアント環境定義の各設定方法での優先順位については，「接続情報の優先順位」を参照してください。

6. **ODBCHARSET**

   UNIX 版 HiRDB ODBC3.5 ドライバ固有の ODBC 環境定義です。文字コード変換で使用する文字コード種別を指定します。詳細は，「UNIX 版 HiRDB ODBC3.5 ドライバ固有の ODBC 環境定義」を参照してください。

13.3.2　UNIX 版 HiRDB ODBC3.5 ドライバ固有の ODBC 環境定義

ここでは，odbc.ini ファイルに指定できる UNIX 版 HiRDB ODBC3.5 ドライバ固有の ODBC 環境定義について説明します。なお，odbc.ini ファイルの編集については，「データソースの設定」を参照してください。

ODBCHARSET = iconv 関数に指定できる文字セット

UTF-16※の文字データを文字コード変換する場合に，iconv 関数で使用する文字セットを指定します。この指定は，HiRDB サーバに格納された文字データを UTF-16※にコード変換する場合にも有効となります。

指定できる文字セットは，UNIX 版 HiRDB ODBC3.5 ドライバの実行環境で iconv --list を実行することで確認できます。

使用できない文字セットを指定した場合は，KFPZ20004-E メッセージが出力されます。

この値を省略した場合は，UTF-16※の文字データを次の表に示す文字セットに従った文字コードに変換します。

注※

Linux 版の場合は UTF-16LE と読み替えてください。

表 13–3　ODBCHARSET 省略時に iconv 関数で使用される文字セット

項番	接続先 HiRDB サーバの文字コード種別	iconv 関数で使用される文字セット
1	SJIS	SHIFT-JIS
2	UJIS	EUC-JISX0213（EUC-JISX0213 を使用できない環境の場合は EUC-JP）
3	UTF-8	UTF8

項番	接続先 HiRDB サーバの文字コード種別	iconv 関数で使用される文字セット
4	上記以外	未サポート

なお，この ODBC 環境定義とクライアント環境定義の PDCLTCNVMODE オペランドを同時に指定した場合，PDCLTCNVMODE オペランドの指定値が有効になります。

13.3.3　環境変数の設定（Windows 版の場合）

次の環境変数を設定してください。

```
PATH=Windowsディレクトリ;Windowsディレクトリ¥System32
```

注 1

Windows ディレクトリは，デフォルトの場合は C:¥WINDOWS となります。

注 2

システム環境変数に設定してください。

13.3.4　ODBC3.5 ドライバのバージョン情報の確認方法

ODBC ドライバのバージョン情報は，「ODBC データソースアドミニストレータ」を起動し，「ドライバ」タブを選択すると確認できます。「ODBC データソースアドミニストレータ」の格納先については，「インストールディレクトリ」を参照してください。

注意事項

ODBC データソースアドミニストレータに登録される HiRDB ODBC3.5 ドライバの名称は，「HiRDB ODBC3.0 Driver」となりますが，機能は ODBC3.5 に準拠しています。

13.3.5　接続情報の優先順位

ODBC3.5 ドライバでは，意味が同じ接続情報を複数の設定方法で指定できます。このような複数の設定方法を持つ接続情報と，同時に複数の設定方法で設定された場合の優先順位を次の表に示します。

表 13–4　接続情報の優先順位（Windows 版の場合）

接続情報の意味	設定方法	優先順位	
		A	B
HiRDB のホスト名， HiRDB のポート番号	データソースで設定した PDHOST，PDNAMEPORT	1	−

接続情報の意味	設定方法	優先順位	
		A	B
	ユーザ環境変数 PDHOST，PDNAMEPORT	2	2
	システム環境変数 PDHOST，PDNAMEPORT	3	3
	データソースで設定したクライアント環境定義ファイル内の PDHOST，PDNAMEPORT	4	−
	Windows ディレクトリ下の HiRDB.INI ファイル※内の PDHOST，PDNAMEPORT	5	5
	SQLDriverConnect 関数の引数で設定した接続文字列のキーワード PDHOST，PDNAMEPORT	−	1
	SQLDriverConnect 関数の引数で設定した接続文字列のキーワード INIFLNAME で設定したクライアント環境定義ファイル内の PDHOST，PDNAMEPORT	−	4
PDHOST,PDNAMEPORT 以外のクライアント環境定義	ユーザ環境変数	1	1
	システム環境変数	2	2
	データソースで設定したクライアント環境定義ファイル	3	−
	Windows ディレクトリ下の HiRDB.INI ファイル※	4	4
	SQLDriverConnect 関数の引数で設定した接続文字列のキーワード INIFLNAME で設定したクライアント環境定義ファイル	−	3

(凡例)

A：SQLConnect，SQLBrowseConnect，SQLDriverConnect（DSN 指定）で接続する場合

B：SQLDriverConnect（DRIVER 指定）で接続する場合

−：接続方法によって指定できない

注※

データソース又は SQLDriverConnect 関数で環境変数グループファイルを設定している場合，HiRDB.INI ファイルでのすべての設定は無効になります。環境変数グループファイルで設定していないクライアント環境定義についても，HiRDB.INI ファイルでの設定は無効になります。

表 13–5　接続情報の優先順位（Linux 版の場合）

接続情報の意味	設定方法	優先順位	
		A	B
HiRDB のホスト名称，HiRDB のポート番号	odbc.ini ファイル※内で設定した PDHOST，PDNAMEPORT	1	−
	ユーザ環境変数 PDHOST，PDNAMEPORT	2	2
	odbc.ini ファイル※内の INIFLNAME で設定したクライアント環境定義ファイル内の PDHOST，PDNAMEPORT	3	−

接続情報の意味	設定方法	優先順位	
		A	B
	SQLDriverConnect 関数の引数で設定した接続文字列のキーワード PDHOST，PDNAMEPORT	−	1
	SQLDriverConnect 関数の引数で設定した接続文字列のキーワード INIFLNAME で設定したクライアント環境定義ファイル内の PDHOST，PDNAMEPORT	−	3
上記以外のクライアント環境定義	ユーザ環境変数	1	1
	odbc.ini ファイル※内の INIFLNAME で設定したクライアント環境定義ファイル	2	−
	SQLDriverConnect 関数の引数で設定した接続文字列のキーワード INIFLNAME で設定したクライアント環境定義ファイル	−	2

(凡例)

A：SQLConnect，SQLBrowseConnect，SQLDriverConnect（DSN 指定）で接続する場合

B：SQLDriverConnect（DRIVER 指定）で接続する場合

−：接続方法によって指定できない

注※

/usr/local/etc 下とホームディレクトリ下の odbc.ini ファイルに同一名称のデータソースを設定している場合は，ホームディレクトリ下の odbc.ini ファイルの設定が有効になります。

13.3.6 文字コード変換機能の設定

HiRDB サーバと HiRDB クライアントの文字コード種別が異なる場合，クライアント環境定義 PDCLTCNVMODE の設定が必要です。PDCLTCNVMODE の設定と使用できる文字コードの組み合わせを次の表に示します。

表 13–6 Windows 版 HiRDB ODBC3.5 ドライバを使用する場合の PDCLTCNVMODE の設定と使用できる文字コードの組み合わせ

ODBC C データ型	HiRDB サーバの文字コード種別	クライアント環境定義 PDCLTCNVMODE の指定	変換される文字コード種別
SQL_C_CHAR （文字コード種別は SJIS）	UJIS※	UJIS UJIS2	(O) − (L) SJIS←→UJIS
		UCS2_UJIS UCS2_UJIS2	(O) SJIS←→UTF-16 (L) UTF-16←→UJIS
	UTF-8※	UTF8 UTF8MS UTF8_TXT	(O) − (L) SJIS←→UTF-8

ODBC C データ型	HiRDB サーバの文字コード種別	クライアント環境定義 PDCLTCNVMODE の指定	変換される文字コード種別
		UTF8MS_TXT UTF8_EX UTF8_EX2	
		UCS2_UTF8	(O) SJIS←→UTF-16 (L) UTF-16←→UTF-8
SQL_C_WCHAR （文字コード種別は UTF-16）	SJIS	指定なし	(O) UTF-16←→SJIS (L) −
	UJIS※	UCS2_UJIS UCS2_UJIS2	(O) − (L) UTF-16←→UJIS
		UJIS UJIS2	(O) UTF-16←→SJIS (L) SJIS←→UJIS
	UTF-8※	UCS2_UTF8	(O) − (L) UTF-16←→UTF-8
		UTF8 UTF8MS UTF8_TXT UTF8MS_TXT UTF8_EX UTF8_EX2	(O) UTF-16←→SJIS (L) SJIS←→UTF-8

（凡例）

(O)：ODBC ドライバによる文字コード変換

(L)：クライアントライブラリによる文字コード変換

−：文字コード変換なし

注※

上段と下段で文字コード変換の回数が異なります。文字コード変換回数が少ない上段の指定を推奨します。

表 13–7　Linux 版 HiRDB ODBC3.5 ドライバを使用する場合の PDCLTCNVMODE の設定と使用できる文字コードの組み合わせ

ODBC C データ型	HiRDB サーバの文字コード種別	クライアント環境定義 PDCLTCNVMODE の指定	変換される文字コード種別
SQL_C_CHAR （文字コード種別は SJIS）	UJIS	UJIS UJIS2	(O) − (L) SJIS←→UJIS
	UTF-8	UTF8 UTF8MS UTF8_TXT UTF8MS_TXT UTF8_EX UTF8_EX2	(O) − (L) SJIS←→UTF-8

ODBC C データ型	HiRDB サーバの文字コード種別	クライアント環境定義 PDCLTCNVMODE の指定	変換される文字コード種別
SQL_C_WCHAR （文字コード種別は UTF-16）	表「Unicode 対応の ODBC 関数を使用する場合に使用できる文字コードの組み合わせと変換される文字コード種別」を参照してください		

（凡例）

(O)：ODBC ドライバによる文字コード変換

(L)：クライアントライブラリによる文字コード変換

−：文字コード変換なし

13.4 HiRDB が提供する ODBC 関数

HiRDB では ODBC 関数を提供していて，その ODBC 関数を利用した UAP からサーバ上の HiRDB にアクセスできます。HiRDB が提供する ODBC 関数を次の表に示します。

表 13–8 HiRDB が提供する ODBC 関数

分類	ODBC 関数	ODBC2.0 ドライバ		ODBC3.5 ドライバ	
		提供可否	拡張レベル	提供可否	拡張レベル
data source との接続	SQLAllocEnv	○	Core	−	−
	SQLAllocHandle	−	−	○	Core
	SQLAllocConnect	○	Core	−	−
	SQLConnect	○	Core	○	Core
	SQLDriverConnect	○	1	○	Core
	SQLBrowseConnect	○	2	○	1
ドライバ及び data source の情報取得	SQLDataSources	○※1	2	○※1	Core
	SQLDrivers	−	−	○※1	Core
	SQLGetInfo	○	1	○	Core
	SQLGetFunctions	−	−	○	Core
	SQLGetTypeInfo	○	1	○	Core
ドライバオプションの設定及び取得	SQLSetConnectOption	○	1	−	−
	SQLGetConnectOption	○	1	−	−
	SQLSetStmtOption	○	1	−	−
	SQLGetStmtOption	○	1	−	−
	SQLSetConnectAttr	−	−	○	Core
	SQLGetConnectAttr	−	−	○	Core
	SQLSetEnvAttr	−	−	○	Core
	SQLGetEnvAttr	−	−	○	Core
	SQLSetStmtAttr	−	1	○	Core
	SQLGetStmtAttr	−	1	○	Core
ディスクリプタ値の設定	SQLGetDescField	−	−	○	Core
	SQLGetDescRec	−	−	○	Core
	SQLSetDescField	−	−	○	Core
	SQLSetDescRec	−	−	○	Core

分類	ODBC 関数	ODBC2.0 ドライバ		ODBC3.5 ドライバ	
		提供可否	拡張レベル	提供可否	拡張レベル
	SQLCopyDesc	−	−	○	Core
SQL 要求の作成	SQLAllocStmt	○	Core	−	−
	SQLPrepare	○	Core	○	Core
	SQLBindParameter	○	1※1	○	Core
	SQLSetParam※2	○	1	−	−
	SQLGetCursorName	○	Core	○	Core
	SQLSetCursorName	○	Core	○	Core
	SQLDescribeParam	○	2	−	−
	SQLNumParam	○	2	−	−
	SQLDescribeParams	−	−	○	2
	SQLNumParams	−	−	○	Core
	SQLParamOptions	×	2	−	−
	SQLSetScrollOptions	×※3	2	×	2
SQL の実行	SQLExecute	○	Core	○	Core
	SQLExecDirect	○	Core	○	Core
	SQLNativeSql	○	2	○	Core
	SQLParamData	○	1	○	Core
	SQLPutData	○	1	○	Core
実行結果及び実行結果情報の取得	SQLRowCount	○	Core	○	Core
	SQLNumResultCols	○	Core	○	Core
	SQLDescribeCol	○	Core	○	Core
	SQLColAttributes	○	Core	−	−
	SQLColAttribute	−	−	○	Core
	SQLBindCol	○	Core	○	Core
	SQLFetch	○	Core	○	Core
	SQLFetchScroll	−	−	○※4	Core
	SQLExtendedFetch	×※3	2	○	Core
	SQLGetData	○	1	○	Core
	SQLSetPos	×※3	2	○※4	1

分類	ODBC 関数	ODBC2.0 ドライバ		ODBC3.5 ドライバ	
		提供可否	拡張レベル	提供可否	拡張レベル
	SQLBulkOperations	−	−	×	1
	SQLMoreResults※5	○	2	○	1
	SQLError	○	Core	−	−
	SQLGetDiagField	−	−	○	Core
	SQLGetDiagRec	−	−	○	Core
data source のシステム情報の取得	SQLColumnPrivileges	○	2	○	2
	SQLColumns	○	1	○	Core
	SQLForeignKeys	○※6	2	○※6	2
	SQLPrimaryKeys	○※6	2	○※6	1
	SQLProcedureColumns	○	2	○	1
	SQLProcedure	○	2	○	1
	SQLSpecialColumns	○※6	1	○※6	Core
	SQLStatistics	○	1	○	Core
	SQLTablePrivileges	○	2	○	2
	SQLTables	○	1	○	Core
SQL 実行の終了	SQLFreestmt	○	Core	○	Core
	SQLCloseCursor	−	−	○	Core
	SQLCancel	○	Core	○	Core
	SQLTransact	○	Core	○	Core
	SQLEndTran	−	−	○	Core
切り離し	SQLDisconnect	○	Core	○	Core
	SQLFreeConnect	○	Core	−	−
	SQLFreeEnv	○	Core	−	−
	SQLFreeHandle	−	−	○	Core

(凡例)

○：該当する ODBC 関数を提供しています。

×：該当する ODBC 関数を提供していません。

−：該当しません。

1：Level1 のことです。

2：Level2 のことです。

Core：コアレベルのことです。

注※1

ドライバマネージャで実装しています。

注※2

SQLSetParam の機能は，ODBC 2.0 で SQLBindParameter に含まれましたが，ODBC 2.0 に対応しないアプリケーションとの互換性を保つために提供しています。

注※3

ODBC2.0 カーソルライブラリで実装しているため，カーソルライブラリで規定された範囲の機能は使用できます。SQLExtendedFetch を使用したい場合は，カーソルライブラリの設定をしてください。カーソルライブラリの設定については，「カーソルライブラリの設定」を参照してください。

注※4

該当する ODBC 関数を使用する場合，Microsoft 提供のカーソルライブラリを使用する必要があります。

注※5

SQL ステートメントを実行したステートメントハンドルで SQLMoreResults を呼び出します。実行された SQL ステートメントが HiRDB の結果集合返却機能を使用している場合，使用できる結果セットが存在するときは SQL_SUCCESS を返却し，次の結果セットが使用可能になります。なお，次の結果セットが存在しない場合は SQL_NO_DATA を返却します。

結果集合返却機能については，マニュアル「HiRDB SQL リファレンス」を参照してください。

注※6

クライアント環境定義 PDODBSYSKEYS=YES 指定時にだけサポートします。それ以外の場合は，作成される結果セットは常に行なしとなります。

13.5　ODBC 関数のデータ型と HiRDB のデータ型との対応

13.5.1　ODBC 関数と HiRDB とのデータ型の対応

ODBC 関数とサーバ上の HiRDB とのデータ型の対応を次の表に示します。

なお，ODBC 関数のデータ型とは，ODBC 関数のアーギュメントに指定する SQL データ型のことです。

表 13–9　ODBC 関数と HiRDB とのデータ型の対応

分　類	ODBC のデータ型	HiRDB のデータ型	説　　明	可否
文字データ	SQL_CHAR	CHAR(n)	固定長文字列	○
	SQL_VARCHAR	VARCHAR(n)	可変長文字列	○
	SQL_LONGVARCHAR	VARCHAR(n)	可変長文字列	○
	SQL_CHAR	NCHAR(n)	固定長各国文字列 NATIONAL CHARACTER(n)	○
	SQL_VARCHAR	NVARCHAR(n)	可変長各国文字列	○
	SQL_CHAR	MCHAR(n)	固定長混在文字列	○
	SQL_VARCHAR	MVARCHAR(n)	可変長混在文字列	○
数データ	SQL_DECIMAL	DEC[IMAL](p，s)	固定小数点数 精度(全体のけた数)=p， 位取り(小数点以下のけた数)=s 1≦p≦15，0≦s≦p	○
	SQL_NUMERIC	－		×
	SQL_SMALLINT	SMALLINT	値の範囲が-32768～32767 の整数	○
	SQL_INTEGER	INTEGER	値の範囲が-2147483648～2147483647 の整数	○
	SQL_TINYINT	－	-256～255 の整数	×
	SQL_BIGINT	－	1 けたの符号と 19 けたの整数	×
	SQL_REAL	SMALLFLT，REAL	単精度浮動小数点数	○
	SQL_FLOAT	FLOAT， DOUBLE PRECISION	倍精度浮動小数点数	○
	SQL_DOUBLE	FLOAT， DOUBLE PRECISION	倍精度浮動小数点数	○
	SQL_BIT	－	ビット	×
	SQL_BINARY	－	固定長バイナリデータ	×

分　類	ODBC のデータ型	HiRDB のデータ型	説　　明	可否
	SQL_LONGVARBINARY	BINARY(n)	可変長バイナリデータ	○
	SQL_LONGVARBINARY	BLOB	可変長バイナリデータ	○
日付，時刻データ	SQL_TYPE_DATE	DATE	日付	○
	SQL_TYPE_TIMESTAMP	TIMESTAMP	日付/時刻	○
	SQL_TYPE_TIME	TIME	時刻	○
	–※	INTERVAL YEAR TO DAY	日間隔	×
	SQL_INTERVAL_HOUR_TO_SECOND	INTERVAL HOUR TO SECOND	時間隔	○
ユーザ定義型	–	抽象データ型	抽象データ型	×

(凡例)

–：ODBC にはないデータ型を示します。

○：使用できます。

×：使用できません。

注

データ型の最大文字列長，及び値の範囲については，マニュアル「HiRDB SQL リファレンス」を参照してください。

注※

サーバ上のデータベースのデータ型がそのまま通知されます。

(1)　ODBC 関数で使用できる機能

ODBC 関数を利用した UAP からサーバ上の HiRDB にアクセスする場合，使用できる機能が一部制限されます。使用できる機能を次の表に示します。

表 13–10　使用できる機能

機　　能	使用可否
スペシャルカラム情報の取得	–
インデクス情報の取得	○
日付，時刻データ型の使用	○※1
繰返し列の使用	×※3
配列列の使用	–
表ヘッダ，列ヘッダの取得	–
非同期処理	×

機　　能	使用可否
LIKE のエスケープ文字の使用	○
更新行数の取得	○
LOGIN タイムアウト時間設定	×
日本語データ型の使用	○※2
定義系 SQL の実行	○

（凡例）

○：使用できます。

×：使用できません。

－：DBMS に機能がありません。

注※1

INTERVAL YEAR TO DAY は使用できません。

注※2

データベースのデータ型がそのまま通知されます。

注※3

繰返し列，？パラメタが繰返し構造でない単純構造の場合，アクセスはできます。

(例)　T1 の列 C1 が繰返し列の場合

```
SELECT C1[1],C1[2] FROM T1        … ○
SELECT C1 FROM T1                 … ×
INSERT INTO T1 VALUES(ARRAY[?,?]) … ○
INSERT INTO T1 VALUES(?)          … ×

（凡例）
  ○：アクセスできます。
  ×：アクセスできません。
```

(2)　カーソルを使用した更新，又は削除する場合の設定

SQLGetCursorName は，SQLSetCursorName によってユーザが設定したカーソル名(ユーザカーソル名)を取得します。ユーザが設定しない場合に，システムが設定するカーソルは取得できません。そのため，カーソルを使った更新，又は削除は，ユーザカーソル名を設定する必要があります。

(3)　ドライバオプションの設定

SQLSetConnectOption 関数，及び SQLGetConnectOption 関数で設定する項目に制限があります。項目の設定可否を次の表に示します。

表 13–11　SQLSetConnectOption 関数，SQLGetConnectOption 関数の設定可否

fOption	設定可否
SQL_ACCESS_MODE	SQL_MODE_READ_WRITE
SQL_AUTOCOMMIT	SQL_AUTOCOMMIT_OFF 又は SQL_AUTOCOMMIT_ON
SQL_LOGIN_TIMEOUT	−
SQL_TRANSLATE_DLL	−
SQL_TRANSLATE_OPTION	−
SQL_TXN_ISOLATION	−

（凡例）−：設定できません。

13.6　ODBC 関数の各属性の指定可否

13.6.1　SQLSetConnectAttr

SQLSetConnectAttr で指定できる ODBC 接続属性を次の表に示します。

表 13–12　SQLSetConnectAttr で指定できる ODBC 接続属性

属性	指定可否	準拠レベル	備考
SQL_ATTR_ACCESS_MODE	○	Core	−
SQL_ATTR_ASYNC_ENABLE	○	Level 1	−
SQL_ATTR_AUTO_IPD	×	Level 2	−
SQL_ATTR_AUTO_COMMIT	○	Level 1	−
SQL_ATTR_CONNECTION_DEAD	×	Level 1	−
SQL_ATTR_CONNECTION_TIMEOUT	○	Level 2	値 0 だけ指定できます。それ以外はエラーとなります。
SQL_ATTR_CURRENT_CATALOG	×	Level 2	−
SQL_ATTR_LOGIN_TIMEOUT	○	Level 2	値 0 だけ指定できます。それ以外も 0 が設定されます。
SQL_ATTR_METADATA_ID	×	Core	−
SQL_ATTR_ODBC_CURSORS	○	Core	カーソルライブラリ使用時だけ指定できます
SQL_ATTR_PACKET_SIZE	×	Level 2	−
SQL_ATTR_QUIET_MODE	×	Core	−
SQL_ATTR_TRACE	○	Core	−
SQL_ATTR_TRACEFILE	○	Core	−
SQL_ATTR_TRANSLATE_LIB	×	Core	−
SQL_ATTR_TRANSLATE_OPTION	×	Core	−
SQL_ATTR_ANSI_APP	○	規定されていません。	−
SQL_ATTR_TXN_ISOLATION	○	Level 1	−
SQL_ATTR_ENLIST_IN_DTC	○	規定されていません。	−

（凡例）

○：指定できます。

×：指定できません。

－：特にありません。

13.6.2　SQLGetConnectAttr

SQLGetConnectAttr で指定できる ODBC 接続属性を次の表に示します。

表 13–13　SQLGetConnectAttr で指定できる ODBC 接続属性

属性	指定可否	準拠レベル	備考
SQL_ATTR_ACCESS_MODE	○	Core	－
SQL_ATTR_ASYNC_ENABLE	○	Level 1	－
SQL_ATTR_AUTO_IPD	○	Level 2	－
SQL_ATTR_AUTO_COMMIT	○	Level 1	－
SQL_ATTR_CONNECTION_DEAD	○	Level 1	－
SQL_ATTR_CONNECTION_TIMEOUT	×	Level 2	－
SQL_ATTR_CURRENT_CATALOG	×	Level 2	－
SQL_ATTR_LOGIN_TIMEOUT	×	Level 2	－
SQL_ATTR_METADATA_ID	○	Core	－
SQL_ATTR_ODBC_CURSORS	×	Core	－
SQL_ATTR_PACKET_SIZE	×	Level 2	－
SQL_ATTR_QUIET_MODE	×	Core	－
SQL_ATTR_TRACE	○	Core	ドライバマネージャが返却します。
SQL_ATTR_TRACEFILE	○	Core	ドライバマネージャが返却します。
SQL_ATTR_TRANSLATE_LIB	×	Core	－
SQL_ATTR_TRANSLATE_OPTION	×	Core	－
SQL_ATTR_ANSI_APP	○	規定されていません。	－
SQL_ATTR_TXN_ISOLATION	○	Level 1	－

（凡例）

○：指定できます。

×：指定できません。

－：特にありません。

13.6.3 SQLSetDescField

SQLSetDescField で指定できる ODBC ディスクリプタ属性を次の表に示します。

表 13–14 SQLSetDescField で指定できる ODBC ディスクリプタ属性

属性	指定可否	準拠レベル	備考
SQL_DESC_ALLOC_TYPE	×	Core	−
SQL_DESC_ARRAY_SIZE	○	Core	−
SQL_DESC_ARRAY_STATUS_PTR	○	Core	−
SQL_DESC_BIND_OFFSET_PTR	○	Core	−
SQL_DESC_DESC_BIND_TYPE	○	Core	−
SQL_DESC_COUNT	○	Core	−
SQL_DESC_ROWS_PROCESSED_PTR	○	Core	−
SQL_DESC_AUTO_UNIQUE_VALUE	×	Level 2	−
SQL_DESC_BASE_COLUMN_NAME	×	Core	−
SQL_DESC_BASE_TABLE_NAME	×	Level 1	−
SQL_DESC_CASE_SENSITIVE	×	Core	−
SQL_DESC_CATALOG_NAME	×	Level 2	−
SQL_DESC_CONCISE_TYPE	○	Core	−
SQL_DESC_DATA_PTR	○	Core	−
SQL_DESC_DATETIME_INTERVAL_CODE	○	Core	−
SQL_DESC_DATETIME_INTERVAL_PRECISION	○	Core	−
SQL_DESC_SQL_DESC_DISPLAY_SIZE	×	Core	−
SQL_DESC_FIXED_PREC_SCALE	×	Core	−
SQL_DESC_INDICATOR_PTR	○	Core	−
SQL_DESC_LABEL	×	Level 2	−
SQL_DESC_LENGTH	○	Core	−
SQL_DESC_LITERAL_PREFIX	×	Core	−
SQL_DESC_LITERAL_SUFFIX	×	Core	−
SQL_DESC_LOCAL_TYPE_NAME	×	Core	−
SQL_DESC_NAME	○	Core	−
SQL_DESC_NULLABLE	×	Core	−

属性	指定可否	準拠レベル	備考
SQL_DESC_NUM_PREC_RADIX	○	規定されていません。	−
SQL_DESC_OCTET_LENGTH	○	Core	−
SQL_DESC_OCTET_LENGTH_PTR	○	Core	−
SQL_DESC_PARAMETER_TYPE	○	Core	−
SQL_DESC_PRECISION	○	Core	−
SQL_DESC_ROWVER	×	Level 1	−
SQL_DESC_SCALE	○	Core	−
SQL_DESC_SCHEMA_NAME	×	Level 1	−
SQL_DESC_SEARCHABLE	×	Core	−
SQL_DESC_TABLE_NAME	×	Level 1	−
SQL_DESC_TYPE	○	Core	−
SQL_DESC_TYPE_NAME	×	Core	−
SQL_DESC_UNNAMED	○	Core	−
SQL_DESC_UNSIGNED	×	Core	−
SQL_DESC_UPDATABLE	×	Core	−

（凡例）

○：指定できます。

×：指定できません。

−：特にありません。

13.6.4 SQLGetDescField

SQLGetDescField で指定できる ODBC ディスクリプタ属性を次の表に示します。

表 13–15 SQLGetDescField で指定できる ODBC ディスクリプタ属性

属性	指定可否	準拠レベル	備考
SQL_DESC_ALLOC_TYPE	○	Core	−
SQL_DESC_ARRAY_SIZE	○	Core	−
SQL_DESC_ARRAY_STATUS_PTR	○	Core	−
SQL_DESC_BIND_OFFSET_PTR	○	Core	−
SQL_DESC_DESC_BIND_TYPE	○	Core	−
SQL_DESC_COUNT	○	Core	−

属性	指定可否	準拠レベル	備考
SQL_DESC_ROWS_PROCESSED_PTR	○	Core	−
SQL_DESC_AUTO_UNIQUE_VALUE	○	Level 2	−
SQL_DESC_BASE_COLUMN_NAME	○	Core	−
SQL_DESC_BASE_TABLE_NAME	○	Level 1	−
SQL_DESC_CASE_SENSITIVE	○	Core	−
SQL_DESC_CATALOG_NAME	○	Level 2	−
SQL_DESC_CONCISE_TYPE	○	Core	−
SQL_DESC_DATA_PTR	○	Core	−
SQL_DESC_DATETIME_INTERVAL_CODE	○	Core	−
SQL_DESC_DATETIME_INTERVAL_PRECISION	○	Core	−
SQL_DESC_SQL_DESC_DISPLAY_SIZE	○	Core	−
SQL_DESC_FIXED_PREC_SCALE	○	Core	−
SQL_DESC_INDICATOR_PTR	○	Core	−
SQL_DESC_LABEL	○	Level 2	−
SQL_DESC_LENGTH	○	Core	−
SQL_DESC_LITERAL_PREFIX	○	Core	−
SQL_DESC_LITERAL_SUFFIX	○	Core	−
SQL_DESC_LOCAL_TYPE_NAME	○	Core	−
SQL_DESC_NAME	○	Core	−
SQL_DESC_NULLABLE	○	Core	−
SQL_DESC_NUM_PREC_RADIX	○	規定されていません。	−
SQL_DESC_OCTET_LENGTH	○	Core	−
SQL_DESC_OCTET_LENGTH_PTR	○	Core	−
SQL_DESC_PARAMETER_TYPE	○	Core	−
SQL_DESC_PRECISION	○	Core	−
SQL_DESC_ROWVER	×	Level 1	−
SQL_DESC_SCALE	○	Core	−
SQL_DESC_SCHEMA_NAME	○	Level 1	−
SQL_DESC_SEARCHABLE	○	Core	−
SQL_DESC_TABLE_NAME	○	Level 1	−

属性	指定可否	準拠レベル	備考
SQL_DESC_TYPE	○	Core	−
SQL_DESC_TYPE_NAME	○	Core	−
SQL_DESC_UNNAMED	○	Core	−
SQL_DESC_UNSIGNED	○	Core	−
SQL_DESC_UPDATABLE	○	Core	−

（凡例）

○：指定できます。

×：指定できません。

−：特にありません。

13.6.5　SQLSetEnvAttr

SQLSetEnvAttr で指定できる ODBC 環境属性を次の表に示します。

表 13–16　SQLSetEnvAttr で指定できる ODBC 環境属性

属性	指定可否	準拠レベル	備考
SQL_ATTR_CONNECTION_POOLING	○	規定されていません。	Conformance Level に含まれません。
SQL_ATTR_CP_MATCH	○	規定されていません。	Conformance Level に含まれません。
SQL_ATTR_ODBC_VERSION	○	Core	−
SQL_ATTR_OUTPUT_NTS	○	規定されていません。	Conformance Level に含まれません。

（凡例）

○：指定できます。

−：特にありません。

13.6.6　SQLGetEnvAttr

SQLGetEnvAttr で指定できる ODBC 環境属性を次の表に示します。

表 13–17　SQLGetEnvAttr で指定できる ODBC 環境属性

属性	指定可否	準拠レベル	備考
SQL_ATTR_CONNECTION_POOLING	○	規定されていません。	Conformance Level に含まれません。

属性	指定可否	準拠レベル	備考
SQL_ATTR_CP_MATCH	○	規定されていません。	Conformance Level に含まれません。
SQL_ATTR_ODBC_VERSION	○	Core	−
SQL_ATTR_OUTPUT_NTS	○	規定されていません。	Conformance Level に含まれません。

（凡例）

○：指定できます。

−：特にありません。

13.6.7 SQLSetStmtAttr

SQLSetStmtAttr で指定できる ODBC ステートメント属性を次の表に示します。

表 13–18 SQLSetStmtAttr で指定できる ODBC ステートメント属性

属性	指定可否	準拠レベル	備考
SQL_ATTR_APP_PARAM_DESC	○	Core	−
SQL_ATTR_APP_ROW_DESC	○	Core	−
SQL_ATTR_ASYNC_ENABLE	○	Level 1	−
SQL_ATTR_CONCURRENCY	○	Level 2	−
SQL_ATTR_CURSOR_SCROLLABLE	○	Level 1	−
SQL_ATTR_CURSOR_SENSITIVITY	○	Level 2	−
SQL_ATTR_CURSOR_TYPE	○	Level 2	SQL_CURSOR_FORWARD_ONLY だけ指定できます。それ以外を指定しても SQL_CURSOR_FORWARD_ONLY が設定されます。
SQL_ATTR_ENABLE_AUTO_IPD	○	Level 2	−
SQL_ATTR_FETCH_BOOKMARK_PTR	×	Level 2	−
SQL_ATTR_IMP_PARAM_DESC	×	Core	−
SQL_ATTR_IMP_ROW_DESC	×	Core	−
SQL_ATTR_KEYSET_SIZE	×	Level 2	−
SQL_ATTR_MAX_LENGTH	○	Level 1	−
SQL_ATTR_MAX_ROWS	○	Level 1	−
SQL_ATTR_METADATA_ID	○	Core	−

属性	指定可否	準拠レベル	備考
SQL_ATTR_NOSCAN	○	Core	−
SQL_ATTR_PARAM_BIND_OFFSET_PTR	×	Core	−
SQL_ATTR_PARAM_BIND_TYPE	○	Core	−
SQL_ATTR_PARAM_OPERATION_PTR	×	Core	−
SQL_ATTR_PARAM_STATUS_PTR	○	Core	−
SQL_ATTR_PARAMS_PROCESSED_PTR	○	Core	−
SQL_ATTR_PARAMSET_SIZE	○	Core	−
SQL_ATTR_QUERY_TIMEOUT	×	Level 2	−
SQL_ATTR_RETRIEVE_DATA	○	Level 1	−
SQL_ATTR_ROW_ARRAY_SIZE	○	Core	−
SQL_ATTR_ROW_BIND_OFFSET_PTR	○	Core	−
SQL_ATTR_ROW_BIND_TYPE	○	Core	−
SQL_ATTR_ROW_NUMBER	×	Level 1	−
SQL_ATTR_ROW_OPERATION_PTR	○	Level 1	−
SQL_ATTR_ROW_STATUS_PTR	○	Core	−
SQL_ATTR_ROWS_FETCHED_PTR	○	Core	−
SQL_ATTR_SIMULATE_CURSOR	×	Level 2	−
SQL_ATTR_USE_BOOKMARKS	×	Level 2	−

（凡例）

○：指定できます。

×：指定できません。

−：特にありません。

13.6.8 SQLGetStmtAttr

SQLGetStmtAttr で指定できる ODBC ステートメント属性を次の表に示します。

表 13–19 SQLGetStmtAttr で指定できる ODBC ステートメント属性

属性	指定可否	準拠レベル	備考
SQL_ATTR_APP_PARAM_DESC	○	Core	−
SQL_ATTR_APP_ROW_DESC	○	Core	−
SQL_ATTR_ASYNC_ENABLE	○	Level 1	−

属性	指定可否	準拠レベル	備考
SQL_ATTR_CONCURRENCY	○	Level 2	−
SQL_ATTR_CURSOR_SCROLLABLE	○	Level 1	−
SQL_ATTR_CURSOR_SENSITIVITY	○	Level 2	−
SQL_ATTR_CURSOR_TYPE	○	Level 2	−
SQL_ATTR_ENABLE_AUTO_IPD	○	Level 2	−
SQL_ATTR_FETCH_BOOKMARK_PTR	×	Level 2	−
SQL_ATTR_IMP_PARAM_DESC	○	Core	−
SQL_ATTR_IMP_ROW_DESC	○	Core	−
SQL_ATTR_KEYSET_SIZE	×	Level 2	−
SQL_ATTR_MAX_LENGTH	○	Level 1	−
SQL_ATTR_MAX_ROWS	○	Level 1	−
SQL_ATTR_METADATA_ID	○	Core	−
SQL_ATTR_NOSCAN	○	Core	−
SQL_ATTR_PARAM_BIND_OFFSET_PTR	×	Core	−
SQL_ATTR_PARAM_BIND_TYPE	×	Core	−
SQL_ATTR_PARAM_OPERATION_PTR	○	Core	−
SQL_ATTR_PARAM_STATUS_PTR	○	Core	−
SQL_ATTR_PARAMS_PROCESSED_PTR	○	Core	−
SQL_ATTR_PARAMSET_SIZE	○	Core	−
SQL_ATTR_QUERY_TIMEOUT	×	Level 2	−
SQL_ATTR_RETRIEVE_DATA	○	Level 1	−
SQL_ATTR_ROW_ARRAY_SIZE	○	Core	−
SQL_ATTR_ROW_BIND_OFFSET_PTR	×	Core	−
SQL_ATTR_ROW_BIND_TYPE	×	Core	−
SQL_ATTR_ROW_NUMBER	×	Level 1	−
SQL_ATTR_ROW_OPERATION_PTR	○	Level 1	−
SQL_ATTR_ROW_STATUS_PTR	○	Core	−
SQL_ATTR_ROWS_FETCHED_PTR	○	Core	−
SQL_ATTR_SIMULATE_CURSOR	×	Level 2	−
SQL_ATTR_USE_BOOKMARKS	×	Level 2	−

（凡例）

○：指定できます。

×：指定できません。

−：特にありません。

13.6.9　SQLGetInfo

SQLGetInfo で指定できる情報型を次に示します。

(1)　ドライバの情報

アクティブステートメントの数，データソース名，及びインタフェース規格の合致レベルなどの ODBC ドライバについての情報を返却します。SQLGetInfo で指定できるドライバの情報を次の表に示します。

表 13–20　SQLGetInfo で指定できるドライバの情報

オプション値	指定可否	備考
SQL_ACTIVE_ENVIRONMENTS	○	−
SQL_ASYNC_MODE	○	−
SQL_BATCH_ROW_COUNT	○	−
SQL_BATCH_SUPPORT	○	−
SQL_DATA_SOURCE_NAME	○	−
SQL_DRIVER_HDBC	○	−
SQL_DRIVER_HDESC	○	ドライバマネージャが返却します。
SQL_DRIVER_HENV	○	−
SQL_DRIVER_HLIB	○	ドライバマネージャが返却します。
SQL_DRIVER_HSTMT	○	−
SQL_DRIVER_NAME	○	−
SQL_DRIVER_ODBC_VER	○	−
SQL_DRIVER_VER	○	−
SQL_DYNAMIC_CURSOR_ATTRIBUTES1	○	−
SQL_DYNAMIC_CURSOR_ATTRIBUTES2	○	−
SQL_FORWARD_ONLY_CURSOR_ATTRIBUTES1	○	−
SQL_FORWARD_ONLY_CURSOR_ATTRIBUTES2	○	−
SQL_FILE_USAGE	○	−

オプション値	指定可否	備考
SQL_GETDATA_EXTENSIONS	○	−
SQL_INFO_SCHEMA_VIEWS	○	−
SQL_KEYSET_CURSOR_ATTRIBUTES1	○	−
SQL_KEYSET_CURSOR_ATTRIBUTES2	○	−
SQL_MAX_ASYNC_CONCURRENT_STATEMENTS	○	−
SQL_MAX_CONCURRENT_ACTIVITIES	○	−
SQL_MAX_DRIVER_CONNECTIONS	○	−
SQL_ODBC_INTERFACE_CONFORMANCE	○	−
SQL_ODBC_VER	○	−
SQL_PARAM_ARRAY_ROW_COUNTS	○	−
SQL_PARAM_ARRAY_SELECTS	○	−
SQL_ROW_UPDATES	○	−
SQL_SEARCH_PATTERN_ESCAPE	○	−
SQL_SERVER_NAME	○	−
SQL_STATIC_CURSOR_ATTRIBUTES1	○	−
SQL_STATIC_CURSOR_ATTRIBUTES2	○	−

（凡例）

○：指定できます。

−：特にありません。

(2)　DBMS 製品の情報

DBMS 製品の名称やバージョンなどの情報を返却します。SQLGetInfo で指定できる DBMS 製品の情報を次の表に示します。

表 13–21　SQLGetInfo で指定できる DBMS 製品の情報

オプション値	指定可否	備考
SQL_DATABASE_NAME	○	−
SQL_DBMS_NAME	○	−
SQL_DBMS_VER	○	−

（凡例）

○：指定できます。

−：特にありません。

(3)　データソースの情報

カーソルの特性やトランザクションの機能など，データソースの情報を返却します。SQLGetInfo で指定できるデータソースの情報を次の表に示します。

表 13–22　SQLGetInfo で指定できるデータソースの情報

オプション値	指定可否	備考
SQL_ACCESSIBLE_PROCEDURES	○	–
SQL_ACCESSIBLE_TABLES	○	–
SQL_BOOKMARK_PERSISTENCE	○	–
SQL_CATALOG_TERM	○	–
SQL_COLLATION_SEQ	○	–
SQL_CONCAT_NULL_BEHAVIOR	○	–
SQL_CURSOR_COMMIT_BEHAVIOR	○	–
SQL_CURSOR_ROLLBACK_BEHAVIOR	○	–
SQL_CURSOR_SENSITIVITY	○	–
SQL_DATA_SOURCE_READ_ONLY	○	–
SQL_DEFAULT_TXN_ISOLATION	○	–
SQL_DESCRIBE_PARAMETER	○	–
SQL_MULT_RESULT_SETS	○	–
SQL_MULTIPLE_ACTIVE_TXN	○	–
SQL_NEED_LONG_DATA_LEN	○	–
SQL_NULL_COLLATION	○	–
SQL_PROCEDURE_TERM	○	–
SQL_SCHEMA_TERM	○	–
SQL_SCROLL_OPTIONS	○	–
SQL_TABLE_TERM	○	–
SQL_TXN_CAPABLE	○	–
SQL_TXN_ISOLATION_OPTION	○	–
SQL_USER_NAME	○	–

（凡例）

○：指定できます。

–：特にありません。

(4)　SQL ステートメントの情報

データソースでサポートしている SQL ステートメントについての情報を返却します。SQLGetInfo で指定できる SQL ステートメントの情報を次の表に示します。

表 13–23　SQLGetInfo で指定できる SQL ステートメントの情報

オプション値	指定可否	備考
SQL_AGGREGATE_FUNCTIONS	○	−
SQL_ALTER_DOMAIN	○	−
SQL_ALTER_SCHEMA	×	−
SQL_ALTER_TABLE	○	−
SQL_ANSI_SQL_DATETIME_LITERALS	○	−
SQL_CATALOG_LOCATION	○	−
SQL_CATALOG_NAME	○	−
SQL_CATALOG_NAME_SEPARATOR	○	−
SQL_CATALOG_USAGE	○	−
SQL_COLUMN_ALIAS	○	−
SQL_CORRELATION_NAME	○	−
SQL_CREATE_ASSERTION	○	−
SQL_CREATE_CHARACTER_SET	○	−
SQL_CREATE_COLLATION	○	−
SQL_CREATE_DOMAIN	○	−
SQL_CREATE_SCHEMA	○	−
SQL_CREATE_TABLE	○	−
SQL_CREATE_TRANSLATION	○	−
SQL_DDL_INDEX	○	−
SQL_DROP_ASSERTION	○	−
SQL_DROP_CHARACTER_SET	○	−
SQL_DROP_COLLATION	○	−
SQL_DROP_DOMAIN	○	−
SQL_DROP_SCHEMA	○	−
SQL_DROP_TABLE	○	−
SQL_DROP_TRANSLATION	○	−

オプション値	指定可否	備考
SQL_DROP_VIEW	○	−
SQL_EXPRESSIONS_IN_ORDERBY	○	−
SQL_GROUP_BY	○	−
SQL_IDENTIFIER_CASE	○	−
SQL_IDENTIFIER_QUOTE_CHAR	○	−
SQL_INDEX_KEYWORDS	○	−
SQL_INSERT_STATEMENT	○	−
SQL_INTEGRITY	○	−
SQL_KEYWORDS	○	−
SQL_LIKE_ESCAPE_CLAUSE	○	−
SQL_NON_NULLABLE_COLUMNS	○	−
SQL_SQL_CONFORMANCE	○	−
SQL_OJ_CAPABILITIES	○	−
SQL_ORDER_BY_COLUMNS_IN_SELECT	○	−
SQL_OUTER_JOINS	○	−
SQL_PROCEDURES	○	−
SQL_QUOTED_IDENTIFIER_CASE	○	−
SQL_SCHEMA_USAGE	○	−
SQL_SPECIAL_CHARACTERS	○	−
SQL_SUBQUERIES	○	−
SQL_UNION	○	−

（凡例）

○：指定できます。

×：指定できません。

−：特にありません。

(5)　SQL ステートメントの制限に関する情報

識別子の最大長や選択一覧の列の最大数など，SQL ステートメントの識別子や句適用される制限に関する情報を返却します。SQLGetInfo で指定できる SQL ステートメントの制限に関する情報を次の表に示します。

表 13–24　SQLGetInfo で指定できる SQL ステートメントの制限に関する情報

オプション値	指定可否	備考
SQL_MAX_BINARY_LITERAL_LEN	○	−

オプション値	指定可否	備考
SQL_MAX_CATALOG_NAME_LEN	○	−
SQL_MAX_CHAR_LITERAL_LEN	○	−
SQL_MAX_COLUMN_NAME_LEN	○	−
SQL_MAX_COLUMNS_IN_GROUP_BY	○	−
SQL_MAX_COLUMNS_IN_INDEX	○	−
SQL_MAX_COLUMNS_IN_ORDER_BY	○	−
SQL_MAX_COLUMNS_IN_SELECT	○	−
SQL_MAX_COLUMNS_IN_TABLE	○	−
SQL_MAX_CURSOR_NAME_LEN	○	−
SQL_MAX_IDENTIFIER_LEN	○	−
SQL_MAX_INDEX_SIZE	○	−
SQL_MAX_PROCEDURE_NAME_LEN	○	−
SQL_MAX_ROW_SIZE	○	−
SQL_MAX_ROW_SIZE_INCLUDES_LONG	○	−
SQL_MAX_SCHEMA_NAME_LEN	○	−
SQL_MAX_STATEMENT_LEN	○	−
SQL_MAX_TABLE_NAME_LEN	○	−
SQL_MAX_TABLES_IN_SELECT	○	−
SQL_MAX_USER_NAME_LEN	○	−

（凡例）

○：指定できます。

−：特にありません。

(6)　スカラ関数の情報

データソース，又はドライバがサポートしているスカラ関数の情報を返却します。SQLGetInfo で指定できるスカラ関数の情報を次の表に示します。

表 13–25　SQLGetInfo で指定できるスカラ関数の情報

オプション値	指定可否	備考
SQL_CONVERT_FUNCTIONS	○	−
SQL_NUMERIC_FUNCTIONS	○	−
SQL_STRING_FUNCTIONS	○	−

オプション値	指定可否	備考
SQL_SYSTEM_FUNCTIONS	○	−
SQL_TIMEDATE_ADD_INTERVALS	○	−
SQL_TIMEDATE_DIFF_INTERVALS	○	−
SQL_TIMEDATE_FUNCTIONS	○	−

（凡例）

○：指定できます。

−：特にありません。

(7)　変換先の SQL データ型の情報

データソースが指定された SQL データ型を CONVERT スカラ関数で変換する場合の変換先の SQL データ型を返却します。SQLGetInfo で指定できる変換先の SQL データ型の情報を次の表に示します。

表 13–26　SQLGetInfo で指定できる変換先の SQL データ型の情報

オプション値	指定可否	備考
SQL_CONVERT_BIGINT	○	−
SQL_CONVERT_BINARY	○	−
SQL_CONVERT_BIT	○	−
SQL_CONVERT_CHAR	○	−
SQL_CONVERT_DATE	○	−
SQL_CONVERT_DECIMAL	○	−
SQL_CONVERT_DOUBLE	○	−
SQL_CONVERT_FLOAT	○	−
SQL_CONVERT_INTEGER	○	−
SQL_CONVERT_INTERVAL_YEAR_MONTH	○	−
SQL_CONVERT_INTERVAL_DAY_TIME	○	−
SQL_CONVERT_LONGVARBINARY	○	−
SQL_CONVERT_LONGVARCHAR	○	−
SQL_CONVERT_NUMERIC	○	−
SQL_CONVERT_REAL	○	−
SQL_CONVERT_SMALLINT	○	−
SQL_CONVERT_TIME	○	−
SQL_CONVERT_TIMESTAMP	○	−

オプション値	指定可否	備考
SQL_CONVERT_TINYINT	○	−
SQL_CONVERT_VARBINARY	○	−
SQL_CONVERT_VARCHAR	○	−

（凡例）

○：指定できます。

−：特にありません。

(8)　ODBC3.0 以降に追加された情報型

ODBC3.0 以降に追加された情報型を次の表に示します。

表 13–27　ODBC3.0 以降に追加された SQLGetInfo で指定できる情報型

オプション値	指定可否	備考
SQL_ACTIVE_ENVIRONMENTS	○	−
SQL_AGGREGATE_FUNCTIONS	○	−
SQL_ALTER_DOMAIN	○	−
SQL_ALTER_SCHEMA	×	−
SQL_ANSI_SQL_DATETIME_LITERALS	○	−
SQL_ASYNC_MODE	○	−
SQL_BATCH_ROW_COUNT	○	−
SQL_BATCH_SUPPORT	○	−
SQL_CATALOG_NAME	○	−
SQL_COLLATION_SEQ	○	−
SQL_CONVERT_INTERVAL_YEAR_MONTH	○	−
SQL_CONVERT_INTERVAL_DAY_TIME	○	−
SQL_CREATE_ASSERTION	○	−
SQL_CREATE_CHARACTER_SET	○	−
SQL_CREATE_COLLATION	○	−
SQL_CREATE_DOMAIN	○	−
SQL_CREATE_SCHEMA	○	−
SQL_CREATE_TABLE	○	−
SQL_CREATE_TRANSLATION	○	−
SQL_CURSOR_SENSITIVITY	○	−

オプション値	指定可否	備考
SQL_DDL_INDEX	○	－
SQL_DESCRIBE_PARAMETER	○	－
SQL_DM_VER	○	ドライバマネージャが返却します。
SQL_DRIVER_HDESC	○	－
SQL_DROP_ASSERTION	○	－
SQL_DROP_CHARACTER_SET	○	－
SQL_DROP_COLLATION	○	－
SQL_DROP_DOMAIN	○	－
SQL_DROP_SCHEMA	○	－
SQL_DROP_TABLE	○	－
SQL_DROP_TRANSLATION	○	－
SQL_DROP_VIEW	○	－
SQL_DYNAMIC_CURSOR_ATTRIBUTES1	○	－
SQL_DYNAMIC_CURSOR_ATTRIBUTES2	○	－
SQL_FORWARD_ONLY_CURSOR_ATTRIBUTES1	○	－
SQL_FORWARD_ONLY_CURSOR_ATTRIBUTES2	○	－
SQL_INFO_SCHEMA_VIEWS	○	－
SQL_INSERT_STATEMENT	○	－
SQL_KEYSET_CURSOR_ATTRIBUTES1	○	－
SQL_KEYSET_CURSOR_ATTRIBUTES2	○	－
SQL_MAX_ASYNC_CONCURRENT_STATEMENTS	○	－
SQL_MAX_IDENTIFIER_LEN	○	－
SQL_PARAM_ARRAY_ROW_COUNTS	○	－
SQL_PARAM_ARRAY_SELECTS	○	－
SQL_STATIC_CURSOR_ATTRIBUTES1	○	－
SQL_STATIC_CURSOR_ATTRIBUTES2	○	－
SQL_SQL92_DATETIME_FUNCTIONS	○	－
SQL_SQL92_FOREIGN_KEY_DELETE_RULE	○	－
SQL_SQL92_FOREIGN_KEY_UPDATE_RULE	○	－
SQL_SQL92_GRANT	○	－

オプション値	指定可否	備考
SQL_SQL92_NUMERIC_VALUE_FUNCTIONS	○	−
SQL_SQL92_PREDICATES	○	−
SQL_SQL92_RELATIONAL_JOIN_OPERATORS	○	−
SQL_SQL92_REVOKE	○	−
SQL_SQL92_ROW_VALUE_CONSTRUCTOR	○	−
SQL_SQL92_STRING_FUNCTIONS	○	−
SQL_SQL92_VALUE_EXPRESSIONS	○	−
SQL_STANDARD_CLI_CONFORMANCE	○	−
SQL_XOPEN_CLI_YEAR	○	−

（凡例）

○：指定できます。

×：指定できません。

−：特にありません。

13.6.10　SQLColAttribute

SQLColAttribute で指定できる ODBC ディスクリプタ属性を次の表に示します。

表 13–28　SQLColAttribute で指定できる ODBC ディスクリプタ属

属性	指定可否	準拠レベル	備考
SQL_DESC_AUTO_UNIQUE_VALUE	○	Level 2	−
SQL_DESC_BASE_COLUMN_NAME	○	Core	ベース列名ではなく SQL の DESCRIBE 文で取得できる列名を返します。そのため，SQL_DESC_NAME と同じ列名を返します。
SQL_DESC_BASE_TABLE_NAME	○	Level 1	−
SQL_DESC_CASE_SENSITIVE	○	Core	−
SQL_DESC_CATALOG_NAME	○	Level 2	−
SQL_DESC_CONCISE_TYPE	○	Core	−
SQL_DESC_DATA_PTR	○	Core	−
SQL_DESC_DATETIME_INTERVAL_CODE	○	Core	−
SQL_DESC_DATETIME_INTERVAL_PRECISION	○	Core	−

属性	指定可否	準拠レベル	備考
SQL_DESC_SQL_DESC_DISPLAY_SIZE	○	Core	−
SQL_DESC_FIXED_PREC_SCALE	○	Core	−
SQL_DESC_INDICATOR_PTR	○	Core	−
SQL_DESC_LABEL	○	Level 2	−
SQL_DESC_LENGTH	○	Core	−
SQL_DESC_LITERAL_PREFIX	○	Core	−
SQL_DESC_LITERAL_SUFFIX	○	Core	−
SQL_DESC_LOCAL_TYPE_NAME	○	Core	−
SQL_DESC_NAME	○	Core	−
SQL_DESC_NULLABLE	○	Core	−
SQL_DESC_NUM_PREC_RADIX	○	−	−
SQL_DESC_OCTET_LENGTH	○	Core	−
SQL_DESC_OCTET_LENGTH_PTR	○	Core	−
SQL_DESC_PARAMETER_TYPE	○	Core	−
SQL_DESC_PRECISION	○	Core	−
SQL_DESC_SCALE	○	Core	−
SQL_DESC_SCHEMA_NAME	○	Level 1	−
SQL_DESC_SEARCHABLE	○	Core	−
SQL_DESC_TABLE_NAME	○	Level 1	−
SQL_DESC_TYPE	○	Core	−
SQL_DESC_TYPE_NAME	○	Core	−
SQL_DESC_UNNAMED	○	Core	−
SQL_DESC_UNSIGNED	○	Core	−
SQL_DESC_UPDATABLE	○	Core	−

（凡例）

○：指定できます。

−：特にありません。

13.7　HiRDB ODBC3.5 ドライバの接続文字列

HiRDB ODBC3.5 ドライバで，SQLBrowseConnect 及び SQLDriverConnect 関数に指定できる接続文字列のキーワードについて次の表に示します。

なお，各キーワードに対応する接続情報については，「インストール手順（Windows 版の場合）」の「データソースの設定」，又は「インストール手順（UNIX 版の場合）」の「データソースの設定」を参照してください。

表 13–29　SQLBrowseConnect 及び SQLDriverConnect 関数に指定できる接続文字列のキーワード

項番	キーワード	ODBC 関数		
		SQLBrowseConnect	SQLDriverConnect	
			DSN 指定	DRIVER 指定
1	DSN=	○	○	×
2	UID=	○	○	○
3	PWD=	○	○	○
4	DRIVER=	×	×	○
5	PDHOST=	×	×	○
6	PDNAMEPORT=	×	×	○
7	INIFLNAME=	×	×	○
8	ODBCHARSET=※	×	×	○

（凡例）

○：指定できます。

×：指定できません。

注※

Windows 版では指定できません。

13.7.1　認可識別子及びパスワードの小文字指定について

認可識別子及びパスワードを指定する場合に，大文字，小文字を区別して扱うときは，引用符（"）で囲んでください。引用符で囲まないと，小文字を指定しても大文字として扱われます。

なお，認可識別子及びパスワードを指定できるのは，次の関数です。

- SQLConnect
- SQLBroweConnect

- SQLDriverConnect

13.8　ODBC 関数の非同期実行

13.8.1　ODBC 関数の非同期実行とは

ODBC 対応アプリケーションプログラムから HiRDB をアクセスする場合，ODBC 関数を非同期に実行できます。

ODBC 関数を同期実行する場合，関数呼出しが終了するまで，ODBC ドライバはアプリケーションプログラムに制御を返せません。非同期実行の場合だと，任意にアプリケーションプログラムに制御を返せます。そのため，ODBC 関数が非同期実行されている間に，アプリケーションプログラムはほかの処理を実行できます。

非同期実行できる ODBC 関数を次に示します。

- SQLColumnPrivileges
- SQLColumns
- SQLExecute
- SQLExecDirect
- SQLParamData
- SQLProcedureColumns
- SQLFetch
- SQLStatistics
- SQLTablePrivileges
- SQLTables
- SQLProcedures

13.8.2　ODBC 関数の非同期実行の手順

ODBC 関数を非同期に実行する場合の手順を次に示します。

＜手順＞

1. 特定の hstmt（ステートメントハンドル）だけで非同期実行を有効にするオプション SQL_ASYNC_ENABLE を使用して，SQLSetStmtOption※1 を呼び出します。hdbc（接続ハンドル）について，関連するすべての hstmt で非同期実行を有効にする場合は，オプション SQL_ASYNC_ENABLE を使用して SQLSetConnectOption※2 を呼び出します。
2. 非同期実行が有効となっている hstmt を使用して非同期実行できる ODBC 関数※1 を呼び出すと，ODBC ドライバはその関数の非同期実行を開始し，SQL_STILL_EXECUTING を返します（非同期実行となら

なかった場合，又はエラーが発生した場合は，SQL_SUCCESS や SQL_ERROR などの同期実行時のコードを返します)。

3. ODBC 関数が非同期実行されている間に，アプリケーションプログラムはほかの処理を実行できます。非同期実行している hstmt と，それに関連する hdbc でアプリケーションプログラムが呼び出せる関数は，SQLAllocStmt，SQLCancel，又は SQLGetFunctions だけです。そのほかの関数（非同期実行中の関数を除く）を呼び出すと，ドライバマネージャからシーケンスエラーが返されます。
4. アプリケーションプログラムは非同期実行の ODBC 関数を呼び出して，その関数の実行が終了したかどうかを確認します。関数が引き続き実行状態の場合は SQL_STILL_EXECUTING が返され，処理が終了している場合は SQL_SUCCESS や SQL_ERROR などのリターンコードが返されます。

   確認のために関数を呼び出す場合，hstmt 以外の引数は指定しても無視されます（ただし，不当なアドレス，指定値はエラーとなる可能性があるため，有効な値でなければなりません)。例えば，INSERT 文で SQLExecDirect を非同期実行し，再度 SQLExecDirect を呼び出す場合に，UPDATE 文を指定していても戻り値は INSERT 文実行の状態が返ります。

注

特定の hstmt だけで非同期実行を無効にする場合も，オプション SQL_ASYNC_ENABLE を使用して SQLSetStmtOption を呼び出します。hdbc について，関連するすべての hstmt で非同期実行を無効にする場合は，オプション SQL_ASYNC_ENABLE を使用して SQLSetConnectOption を呼び出します。

注※1

SQLSetStmtOption での設定内容を次に示します。

fOption	設定内容
SQL_ASYNC_ENABLE	SQL_ASYNC_ENABLE_OFF，又は SQL_ASYNC_ENABLE _ON
SQL_BIND_TYPE	設定できません。
SQL_MAX_LENGTH	サーバの制限値，又はユーザの指定値
SQL_NOSCAN (Default=FALSE)	SQL_NOSCAN_OFF，又は SQL_NOSCAN_ON
SQL_QUERY_TIMEOUT	設定できません。
SQL_MAX_ROWS	サーバの制限値，又はユーザの指定値

注※2

SQLSetConnectOption での設定内容を次に示します。

fOption	設定内容
SQL_ACCESS_MODE	SQL_MODE_READ_WRITE 固定
SQL_AUTOCOMMIT	SQL_AUTOCOMMIT_OFF，又は SQL_AUTOCOMMIT_ON
SQL_LOGIN_TIMEOUT	設定できません。
SQL_OPT_TRACE	0(Off)固定。 ODBC ドライバマネージャから返されるオプションです。
SQL_OPT_TRACEFILE	NULL 固定。

fOption	設定内容
	ODBC ドライバマネージャから返されるオプションです。
SQL_TRANSLATE_DLL	設定できません。
SQL_TRANSLATE_OPTION	設定できません。
SQL_TXN_ISOLATION	SQL_TXN_READ_UNCOMMITED
SQL_ASYNC_ENABLE	SQL_ASYNC_ENABLE_OFF，又は SQL_ASYNC_ENABLE _ON

13.8.3　ODBC 関数の非同期実行のキャンセル

(1)　非同期実行をキャンセルするには

非同期実行中の ODBC 関数をキャンセルするには，SQLCancel を呼び出します。

SQLCancel は，指定された hstmt が現在非同期実行中であることを確認できた時点で，サーバに対して処理のキャンセル要求をします。

SQLCancel の戻り値では，キャンセル要求が終了したかどうかを通知し，実際に非同期実行の関数がキャンセルされたかどうかは処理中の非同期実行の関数を呼び出して，その戻り値で判断します。関数が実行中の場合は SQL_STILL_EXECUTING が返され，キャンセル処理が終了している場合は SQL_ERROR と SQLSTATE S1008（処理のキャンセル）が返されます。また，既に正常に終了していた場合や，ほかのエラーが発生した場合には，SQL_SUCCESS や SQL_ERROR などのコードが返ります。

(2)　マルチスレッドのアプリケーションプログラムでの非同期実行のキャンセル

マルチスレッドのアプリケーションプログラムでは，hstmt で同期実行している ODBC 関数をキャンセルできます。キャンセルする場合，アプリケーションプログラムはキャンセルする関数に使用されている hstmt と同じ hstmt で，異なるスレッドから SQLCancel を呼び出します。

SQLCancel の戻り値には，ドライバが要求を正しく受けたかどうかを表す値が返されます。また，元の関数の戻り値には，SQL_SUCCESS，又は SQL_ERROR と SQLSTATE S1008（処理のキャンセル）が返されます。

注意：

HiRDB のキャンセル処理は接続単位で実行され，いったんサーバとの接続が強制的に切断されます（サーバ側で KFPS00993-I：サーバプロセス終了 REQUEST=clt_attention が出力されます）。そのため，指定した hstmt に関連する hstmt の，すべてのステートメントがキャンセル（トランザクションはロールバック）されます。したがって，非同期実行中の ODBC 関数をキャンセルする場合，更新途中のデータを十分考慮する必要があります。

13.8.4　コーディング例

非同期実行のコーディング例を次に示します。

```
SQLSetStmtOption(hstmt,SQL_ASYNC_ENABLE,SQL_ASYNC_ENABLE_ON);
          :
  SQLFetchでの検索処理
rc=SQLFetch(hstmt);
while(rc==SQL_STILL_EXECUTING)
{
          :
  非同期実行中 APの処理を続行
          :
    if(処理のキャンセル要求あり)
    {
        rc=SQL_Cancel(hstmt);
        if(rc==SQL_ERROR){  キャンセル要求失敗 エラー処理へ  }
    }
   rc=SQLFetch(hstmt);
}
if(rc == SQL_ERROR){   エラー処理へ  }
  検索データ加工処理へ
       :
```

13.9　カーソルライブラリの設定

ODBC 対応の UAP で SQLExtendedFetch を使用したい場合は，カーソルライブラリの設定をする必要があります。カーソルライブラリの設定をする場合，次の二つの方法があります。

ODBC 関数の SetConnectOption を使用する場合

ODBC 関数の SetConnectOption の引数の fOption に SQL_ODBC_CURSORS，vParam に SQL_CUR_USE_ODBC を指定します。

Visual Basic の RDO を使用する場合

rdoEnvironment オブジェクトの CursorDriver プロパティに rdUseOdbc を指定します。Visual Basic の RDO を使用した場合のコーディング例を次に示します。

```
Dim mrdoEnv As rdoEnvironment
Set mrdoEnv = rdoEngine.rdoCreateEnvironment("", "", "")
    mrdoEnv.CursorDriver = rdUseOdbc
    Src = "DSN=host1;UID=USER_A;PWD=USER_A"
    Set mrdoConn = mrdoEnv.OpenConnection
                   ("", rdDriverComplete, False, Src)
        :
        :
```

ODBC ドライバのインストール用フロッピーディスク内の Sampleap ディレクトリに，簡単なサンプル UAP がありますので参照してください。

13.10　ファイル DSN について

ファイル DSN を使用すると，データソースに接続するための情報をファイルに格納するので，ODBC.INI やレジストリから情報を取得しないで接続を確立できます。

ファイルを共用することで，複数ユーザが各マシンにデータソース（従来のマシンデータソース）を登録しなくても HiRDB に接続できるようになります。なお，ファイル DSN は，ODBC コンポーネントバージョンが 3.0 以上の場合に使用できます。

ファイル DSN は，ODBC のデータソースアドミニストレータから作成できます。

ファイル DSN の作成方法

ファイル DSN を選択−追加−ドライバの選択（HiRDB 32bit Driver）−格納ファイル名指定をすることで HiRDB への接続要求がされ，SQLDriverConnect から返される完全接続文字列を基にドライバマネージャがファイルを作成します。ただし，この場合，パスワードはファイル DSN には格納されません。パスワードも共有する場合は，作成したファイル内に「PWD=パスワード」の 1 行を追加してください。

13.11　Unicode の UAP の実行

ここでは，Unicode の UAP で使用できる ODBC 関数，及び注意事項について説明します。

13.11.1　Unicode の UAP で使用できる ODBC 関数

Unicode の UAP で使用できる ODBC 関数を次の表に示します。

表 13–30　Unicode の UAP で使用できる ODBC 関数

分類	関数名	機能
データソースとの接続	SQLConnectW	データソース名，認可識別子，及びパスワードによって特定のドライバに接続します。
	SQLDriverConnectW	接続文字列によって特定のドライバと接続します。又は，ドライバマネージャとドライバに対して，ユーザ用に接続ダイアログボックスの表示を要求します。
	SQLBrowseConnectW	連続したレベルの接続属性と，有効な属性値を返します。各接続属性に値が指定されている場合はデータソースに接続します。
	SQLDriversW	インストールされているドライバと，その属性の一覧を返します。
ドライバ及びデータソースの情報取得	SQLDataSourcesW	利用できるデータソースの一覧を返します。
	SQLGetInfoW	特定のドライバとデータソースの情報を返します。
ドライバオプションの設定及び取得	SQLSetConnectAttrW	接続属性を設定します。
	SQLGetConnectAttrW	接続属性の値を返します。
	SQLSetStmtAttrW	ステートメント属性を設定します。
	SQLGetStmtAttrW	ステートメント属性の値を返します。
ディスクリプタ値の設定及び取得	SQLSetDescFieldW	一つのディスクリプタフィールドを設定します。
	SQLGetDescFieldW	一つのディスクリプタフィールドの値を返します。
	SQLSetDescRecW	複数のディスクリプタフィールドを設定します。
	SQLGetDescRecW	複数のディスクリプタフィールドの値を返します。
	SQLPrepareW	後で実行するために SQL ステートメントを準備します。
SQL 要求の作成	SQLSetCursorNameW	カーソル名を指定します。
	SQLGetCursorNameW	ステートメントハンドルに関連するカーソル名を返します。
SQL の実行	SQLExecDirectW	ステートメントを実行します。

分類	関数名	機能
	SQLNativeSqlW	ドライバが変換した SQL ステートメントのテキストを返します。
実行結果及び実行結果情報の取得	SQLDescribeColW	結果セットの列を記述します。
	SQLColAttributeW	結果セットの列の属性を記述します。
	SQLGetDiagFieldW	追加の診断情報（診断データ構造体の一つのフィールド）を返します。
	SQLGetDiagRecW	追加の診断情報（診断データ構造体の複数のフィールド）を返します。
	SQLColumnPrivilegesW	列の一覧と，一つ以上のテーブルに関連する特権を返します。
データソースのシステム情報の取得	SQLColumnsW	指定されたテーブルの列名の一覧を返します。
	SQLForeignKeysW	指定されたテーブルに外部キーがある場合，外部キーを構成する列名の一覧を返します。
	SQLPrimaryKeysW	指定されたテーブルの主キーを構成する列名の一覧を返します。
	SQLProcedureColumnsW	入力パラメタ又は出力パラメタの一覧と，指定されたプロシジャの結果セットを構成する列を返します。
	SQLProceduresW	指定されたデータソース内のプロシジャ名の一覧を返します。
	SQLSpecialColumnsW	指定されたテーブル内で行を一意に識別できる最適の列，又は行の値がトランザクションによって変更されるときに自動的に修正される列の情報を返します。
	SQLStatisticsW	単一のテーブルに関する統計情報と，テーブルに関するインデクスの一覧を返します。
	SQLTablePrivilegesW	テーブルの一覧と，各テーブルに関連する特権を返します。
	SQLTablesW	指定されたデータソース内のテーブル名の一覧を返します。

13.11.2　注意事項

クライアント環境定義の PDCLTCNVMODE に UCS2_UJIS，UCS2_UJIS2，又は UCS2_UTF8 が設定されている場合の注意事項を次に示します。

- 列属性取得時に返却される SQL データ型は次のようになります。
  HiRDB のデータ型が CHAR，MCHAR，又は NCHAR の場合：SQL_WCHAR
  HiRDB のデータ型が VARCHAR，MVARCHAR，又は NVARCHAR の場合：SQL_WVARCHAR

- 列属性取得時，HiRDB のデータ型が文字列系データ型の場合，列長は列の定義長×2 の値が設定されます。例えば，char(10)の場合は 20 が列長として返却されます。

13.12　チューニング，トラブルシュート

ここでは，ODBC 対応の UAP を使用している場合のチューニング，トラブルシュートについて説明します。

13.12.1　ODBC エラーログ

トラブルシュート情報として，ODBC エラーログを取得できます。ODBC エラーログファイルには，ODBC3.5 ドライバ内で検知した一部の障害について，障害要因が出力されます。

(1)　エラーログの対象

エラーログの対象となるエラーを次の表に示します。

SQLSTATE	エラーメッセージ	エラー内容
22003	Numeric value out of range	データ変換失敗（バッファサイズ指定誤り）
22005	Error in assignment	データ変換失敗（データ不正）
HYC00	Optional feature not implemented	HiRDB ODBC では未サポートのオプション要求

(2)　ODBC エラーログの取得方法

ODBC エラーログの取得方法及び仕様について次に示します。

取得方法

クライアント環境定義 PDODBERRTRC で指定します。

- YES：ODBC エラーログを出力します。
- NO：ODBC エラーログを出力しません（従来の動作）。

出力先ファイル名

下記 2 つのファイルを作成し，ラップアラウンドして出力します。

- pdodberr1.trc
- pdodberr2.trc

出力先ディレクトリ

クライアント環境定義 PDCLTPATH で指定します。省略時は UAP 実行ディレクトリに出力します。

出力先ディレクトリが存在する又は書き込み権限がある場合に出力します。

出力先ファイルサイズ

クライアント環境定義 PDODBERRTRCSIZE で指定します。

エラーログ出力時，出力先ファイルサイズが PDODBERRTRCSIZE を超えていた場合，出力先が切り替わります。

(3) ODBC エラーログの見方

ODBC エラーログの出力例とその説明を次に示します。

[出力例]

```
                [1]          [2]                          [3]     [4]
  ** HiRDB ODBC DRIVER ERROR CLT(ODBC):10-09     SVR:10-09   US UTF8 **
  CONNECTION STATUS :[5]
    CURHOST(XXX.XXX.XXX.XXX)
    CURPORT(22201) SVRNAME(sds01)
    CNCTNO(242) SVRPID(29580) CLTPID(31508) CLTTID(33476) CLTCNCTHDL(0xf84a06e0)
  ODBC ENVIRONMENT :[6]
    PDCLTCNVMODE(NOUSE);PDJETCOMPATIBLE(YES);PDODBSTATCACHE(0);PDODBESCAPE(0);PDGDATAOPT(NO)
;PDODBLOCATOR(YES);PDODBSPLITSIZE(102400);PDODBCWRNSKIP(YES);PDODBGINFOSUPPRESS(YES);PDODBST
ANDARDARGSIZE(YES);PDODBSTANDARDSQLSTATE(YES);PDODBSTANDARDDESCCOL(YES);PDODBSTANDARDGTYPEIN
FO(YES);PDODBCOLSIZEUNIT(CHAR);PDODBSETATEXECLEN(YES);PDODBSYSUNSIGNED(YES);PDODBSYSKEYS(YES
)
  HIRDBINIPATH : C:¥windows¥HiRDB.ini[7]
  2024/10/02 18:32:16[8]
  od3dcnvl.c(1769)[9]
  SECNO(1)[10]
  *SQL* CALL P1(?)[11]

  SQLExecDirect([12]
    StatementHandle        0x000002114D9348E0
    StatementText          0x000002114D92AE60
    TextLength                             10
  )
  DIAG [22005] [HITACHI][HiRDB]Error in assignment.[13]

  SQLBindParameter([14]
      StatementHandle      0x000002114D9348E0
      ParameterNumber                       1
      InputOutputType                       2
      ValueType                            -8
      ParameterType                        91
      ColumnSize                           10
      DecimalDigits                         0
      ParameterValuePtr    0x00007FF6C80AD248
      BufferLength                         10
      StrLen_or_IndPtr     0x00007FF6C80AD750
  )
DATA=32 00 30 00 32 00 34 00 2d 00                    *2.0.2.4.-.      *[15]
```

[説明]

1. タイトル

   ODBC エラーログのタイトルを表示します。

2. ODBC 作成バージョン

   ODBC ドライバの作成バージョンを表示します。

3. サーババージョン

接続先 HiRDB のバージョンを表示します。

4. サーバの文字コード
   接続先 HiRDB サーバの文字コードを表示します。

5. UAP 実行ステータス
   ODBC ドライバを使用したアプリケーションを実行したときのサーバとの接続状態を表示します。
   - CURHOST：接続先ホスト名
   - CURPORT：接続ポート番号
   - SRVNAME：フロントエンドサーバ名，又はシングルサーバ名
   - CNCTNO：コネクト通番
   - SVRPID：接続サーバのプロセス番号
   - CLTPID：UAP のプロセス番号
   - CLTTID：UAP のスレッド番号
   - CLTCNCTHDL：コネクションハンドル

   なお，取得できる情報の値だけを表示します。

6. UAP 実行環境
   ODBC ドライバを使用したアプリケーションを実行したときの ODBC 関連のクライアント環境変数の値を表示します。

7. HiRDB.ini のパス
   有効となっている HiRDB.ini のパスを表示します。

   Windows 版：

   データソースで設定した HiRDB クライアント環境変数ファイル名，又は Windows ディレクトリ下の HiRDB.ini

   UNIX 版：

   データソースで設定した HiRDB クライアント環境変数ファイル名

8. エラー検知日時
   エラーを検知した日時を表示します。

9. エラー発生個所
   ODBC ドライバ内でエラーを検知したファイル名及び行番号を表示します。

10. セクション番号
    SQL の対応を格納するための番号を表示します。番号は ODBC ドライバが自動で付与します。

11. SQL 文

エラー発生時に SQL 文を表示します。エラー検知箇所によっては SQL 文が表示されないこともあります。

12. エラーを検知した ODBC 関数

    エラーを検知した ODBC 関数について，引数情報を表示します。なお引数の値については，ODBC 仕様を参照してください。

13. エラーメッセージ

    エラーメッセージを表示します。

14. パラメタ又は結果セットのバインド情報

    SQLSTATE が 22003 又は 22005 エラーでは，エラーを検知したパラメタや結果セットに対するバインド情報（SQLBindParameter 及び SQLBindCol の引数）を表示します。なお引数の値については，ODBC 仕様を参照してください。

15. パラメタ又は結果セット

    SQLSTATE が 22003 又は 22005 エラーでは，エラーとなったパラメタや結果セットのデータを表示します。

13.12.2　複数行検索をする UAP で性能が出ない場合

ブロック転送機能を使用してください。ブロック転送機能を使用する場合は，クライアント環境定義の PDBLKF オペランドを指定します。指定値には，40〜50 を指定することをお勧めします。これより大きな値を指定しても，通信回数を削減する効果はあまりなく，逆に処理上のオーバヘッドによって遅くなる可能性があります。ブロック転送機能については，「ブロック転送機能」を参照してください。

13.12.3　頻繁に CONNECT，DISCONNECT をする UAP の場合

高速接続機能を使用してください。高速接続機能を使用する場合は，クライアント環境定義の PDFESHOST，PDSERVICEPORT，及び PDSERVICEGRP オペランドを指定します。高速接続機能は，HiRDB への接続時間を短縮する機能です。クライアント環境定義の PDFESHOST，PDSERVICEPORT，及び PDSERVICEGRP オペランドについては，「クライアント環境定義の設定内容」を参照してください。

13.12.4　HiRDB に要求される SQL 文を確認したい場合

ODBC 経由で HiRDB をアクセスする場合，UAP を作成する環境によって，UAP 上で記述した SQL 文と HiRDB に要求される SQL 文とが異なることがあります。どのような SQL 文が HiRDB に対して発行されているのか確認したい場合は，SQL トレース機能を使用します。SQL トレース機能を使用する場合は，クライアント環境定義の PDSQLTRACE オペランドを指定します。また，このとき同時に PDCLTPATH オペラン

ドにトレース出力先ディレクトリを指定することをお勧めします。SQL トレース機能については，「SQL トレース機能」を参照してください。

13.12.5　その他

- Microsoft Access などのアプリケーションプログラムで，検索の SQL 文で排他オプションを指定すると，Microsoft Access 側で構文エラーとなることがあります。このような場合，システム定義の pd_isolation_level オペランド又はクライアント環境定義の PDISLLVL オペランドを指定することで対処できないかどうか，検討することをお勧めします。
- Microsoft Access を使用して HiRDB をアクセスする場合，UAP の作成方法によっては，更新するときに HiRDB で排他エラーとなることがあります。これは，Microsoft Access が HiRDB に対して複数接続をして，別々の接続から参照及び更新を，同一行に対して実行するために発生する現象です。これを回避するためには，システム定義の pd_isolation_level オペランド又はクライアント環境定義の PDISLLVL オペランドに，0 又は 1 を指定して対処してください。ODBC ドライバのインストール用フロッピーディスク内の Sampleap ディレクトリに，Visual Basic の DAO（Data Access Object）を使用した，アクセス時に排他エラーが発生しない簡単なサンプル UAP があるので参照してください。

13.13　ODBC 経由で HiRDB をアクセスする場合に使用できない機能について

ODBC 経由で HiRDB をアクセスする場合，使用できない機能が幾つかあります。

- 行インタフェースを使用したアクセス
  ROW 指定の問合せ，UPDATE 文，及び INSERT 文は実行できません。
- カーソルを使用した更新，削除
  CURRENT OF カーソル名を使用した更新，削除はできません。ただし，カーソルライブラリの機能を使用した場合は，カーソルライブラリが CURRENT OF カーソル名を WHERE 条件に変更するため，実行できる場合もあります。
- ホールダブルカーソル
  ホールダブルカーソル（WITH HOLD 指定のカーソル，又は UNTIL DISCONNECT 指定の問合せによって定義するカーソル）は使用できません。
- ?パラメタに NULL 述語を指定
  ?パラメタに NULL 述語を指定した SQL（? IS〔NOT〕 NULL）は使用できません。
- HiRDB のコマンド，又はユティリティの実行
  CALL COMMAND 文は実行できません。

13.14　UNIX 版 HiRDB ODBC3.5 ドライバを使用する場合の留意事項

- Linux 版 HiRDB ODBC3.5 ドライバは，SQL_C_WCHAR 型を扱う場合，1 文字を 2 バイトの UTF-16LE 形式で処理します。そのため，Linux 版 HiRDB ODBC3.5 ドライバへ SQL_C_WCHAR 型のデータを渡す場合には，データは UTF-16LE 形式にしてください。また，以降の UTF-16 は UTF-16LE と読み替えてください。
- Unicode 対応の ODBC 関数を使用した場合，UNIX 版 HiRDB ODBC3.5 ドライバは接続先の HiRDB サーバの文字コードを判定し，ドライバマネージャから受け取った文字列を UTF-16 形式から HiRDB サーバの文字コードに変換します。また，HiRDB サーバから受け取ったデータは HiRDB サーバの文字コードから UTF-16 形式に変換し，ドライバマネージャに返却します。ただし，クライアント環境定義 PDCLTCNVMODE を指定した場合は，接続先の HiRDB サーバの文字コードに関係なく PDCLTCNVMODE の指定に従って文字コード変換します。Unicode 対応の ODBC 関数を使用する場合に使用できる文字コードの組み合わせと変換される文字コード種別を次の表に示します。

表 13–31　Unicode 対応の ODBC 関数を使用する場合に使用できる文字コードの組み合わせと変換される文字コード種別

項番	クライアント環境定義 PDCLTCNVMODE の指定	HiRDB サーバの文字コード種別	変換される文字コード種別	
			ODBC 環境定義 ODBCHARSET を指定しない場合	ODBC 環境定義 ODBCHARSET を指定した場合
1	UJIS UJIS2	UJIS	UTF-16←→SJIS （クライアントライブラリで SJIS←→UJIS 変換）	
2	TF8 UTF8MS UTF8_TXT UTF8MS_TXT UTF8_EX UTF8_EX2	UTF-8	UTF-16←→SJIS （クライアントライブラリで SJIS←→UTF-8 変換）	
3	UCS2_UJIS UCS2_UJIS2	UJIS	ODBC ドライバ内は Unicode で扱う （クライアントライブラリで UTF-16←→UJIS 変換）	
4	UCS2_UTF8	UTF-8	ODBC ドライバ内は Unicode で扱う （クライアントライブラリで UTF-16←→UTF-8 変換）	
5	指定なし	SJIS	UTF-16←→SJIS	UTF-16←→ODBCHARSET に指定した文字セット
6		UJIS	UTF-16←→EUC-JISX0213 （EUC-JISX0213 を使用できない環境の場合は EUC-JP）	UTF-16←→ODBCHARSET に指定した文字セット
7		UTF-8	UTF-16←→UTF-8	UTF-16←→ODBCHARSET に指定した文字セット
8		上記以外	未サポート	

- Unicode に対応していない ODBC 関数を使用する場合，UNIX 版 HiRDB ODBC3.5 ドライバは，引数として渡された文字列に対して文字コード変換をしません。この場合は，UAP の実行環境と接続先の HiRDB サーバの文字コードを合わせる必要があります。ただし，クライアント環境定義 PDCLTCNVMODE を指定した場合は，PDCLTCNVMODE の指定に従って文字コード変換します。
- UNIX 版 HiRDB ODBC3.5 ドライバでは，Unicode 対応の ODBC 関数の接続関数を使用した場合，データソース名は実行環境の LANG 環境変数の文字コードに変換します。したがって，実行環境の LANG 環境変数と登録されているデータソース名の文字コードが異なる場合は，ASCII 文字以外のデータソース名を読み取れないことがあります。
- UNIX 版 HiRDB ODBC3.5 ドライバでは，ダイアログボックスの表示をサポートしていません。そのため，HiRDB サーバと接続する場合に，SQL_DRIVER_NOPROMPT 以外を指定して SQLDriverConnect を呼び出した場合，SQL_ERROR を返します。
- UNIX 版 HiRDB ODBC3.5 ドライバは，unixODBC が提供する GUI ツール ODBC Config を使用したデータソースの登録をサポートしていません。データソースを登録する際は，「インストール手順（UNIX 版の場合）」に従い，登録する必要があります。

13.15 .NET Framework Data Provider for ODBC による SQL 文の自動生成

.NET Framework Data Provider for ODBC は，Microsoft が提供する.NET Framework クラスライブラリの一つです。.NET Framework Data Provider for ODBC では，OdbcCommandBuilder クラスを使用することで，データソースに対する更新用の SQL 文を自動で生成することができます。

HiRDB ODBC3.5 ドライバを使用した場合の，.NET Framework Data Provider for ODBC の SQL 文の自動生成を実行するときの制限事項を次に示します。

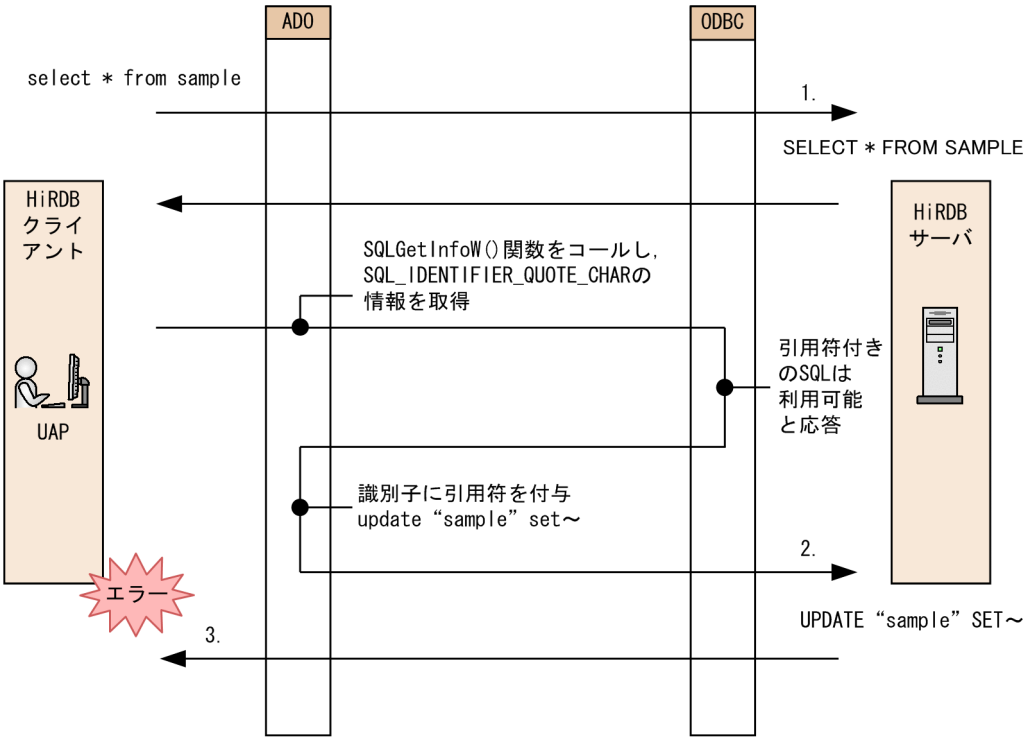
- .NET Framework Data Provider for ODBC を介した SQL 文の自動生成は，HiRDB のバージョン 08-04 以降で動作します。08-04 より前のバージョンでの動作は保証しません。
- SQL 文の自動生成の基となる OdbcDataAdapter クラスの SelectCommand プロパティに指定する SELECT 文で，表名に相関名を指定した場合，HiRDB サーバは HiRDB ODBC3.5 ドライバに実表名を返却します。そのため，HiRDB サーバに対して有効な SQL 文が自動生成されます。しかし，表名に別名を指定した場合，HiRDB サーバは HiRDB ODBC3.5 ドライバに表名を返却しません。そのため，HiRDB ODBC3.5 ドライバの SQLColAttribute 関数の SQL_DESC_BASE_TABLE_NAME の情報を必要とする.NET Framework Data Provider for ODBC へ表名が返却できません。この場合は SQL 文の自動生成は保証されません。
- SQL 文の自動生成の基となる OdbcDataAdapter クラスの SelectCommand プロパティに指定する SELECT 文で，列名に別名を指定した場合，HiRDB サーバは HiRDB ODBC3.5 ドライバに列の別名を返却します。そのため，自動生成される SQL 文は，列の別名が使用された状態となります。この列の別名が使用された SQL 文を実行した場合，その列は実際には存在しないため，HiRDB サーバ側ではエラーを返却します。

13.16 ADO 及び ADO.NET から ODBC を経由して HiRDB サーバにアクセスする場合の留意事項

ADO 及び ADO.NET のインタフェースを利用した UAP が，HiRDB ODBC3.5 ドライバ経由で HiRDB サーバに接続するときの留意事項について説明します。なお，HiRDB ODBC3.5 ドライバ，及び HiRDB サーバのバージョンが共に 08-04 以降の場合は，この現象は発生しないため対策する必要はありません。

UAP から ADO 及び ADO.NET のインタフェースを利用した場合，ユーザが SQL 文に指定した認可識別子，及び表識別子を，ADO が自動的に引用符で囲むことがあります。HiRDB サーバでは引用符が付与されている識別子を半角英大文字と半角英小文字で区別するため，ADO が自動的に識別子を引用符で囲んだ場合，SQL 文がエラーとなることがあります。

ADO が自動的に引用符を付与した SQL 文がエラーになる例を次の図に示します。この例では，表識別子が SAMPLE で定義されているとします。



[説明]

1. UAP から，select * from sample を実行します。HiRDB サーバは，SQL を SELECT * FROM SAMPLE と受け取り，識別子を「SAMPLE」と認識します。
2. ADO は，識別子を引用符で囲めるかどうかを確認するために HiRDB ODBC3.5 ドライバに問い合わせます。

   ADO が内部的に SQLGetInfoW()関数をコールして，SQL_IDENTIFIER_QUOTE_CHAR の情報を取得します。HiRDB サーバは，引用符付きの識別子を使用できるため，HiRDB ODBC3.5 ドライバは，ADO からの要求に対して引用符付きの識別子が利用できると応答します。この応答を受けて，ADO

は SQL 文を組立てる際に，自動的に識別子に引用符を付与します。HiRDB サーバは，識別子を「"sample"」と受け取ります。

3. HiRDB サーバは引用符付きの識別子は半角英大文字と半角英小文字を区別します。HiRDB サーバで定義されている識別子は「SAMPLE」であるため，2.で受け取った識別子「sample」を不一致と認識し，HiRDB サーバは UAP にエラーを返します。

ADO が自動的に引用符を付与した SQL 文がエラーになる条件を次に示します。

表 13–32　SQL 文がエラーになる条件の組み合わせ

SQL 文に指定した識別子	HiRDB サーバに格納された識別子	
	すべて半角英大文字	半角英小文字を含む
すべて半角英大文字	○	×
半角英小文字を含む	×	○

（凡例）

○：エラーは発生しません。

×：エラーが発生します。

この場合，クライアント環境定義 PDODBGINFOSUPPRESS に YES を指定することで，ADO が自動的に識別子に引用符を付与することを抑止できます。クライアント環境定義 PDODBGINFOSUPPRESS については，「PDODBGINFOSUPPRESS」を参照してください。

13.17　ODBC3.5 ドライバが返却する SQLSTATE

ODBC3.5 ドライバが返却する SQLSTATE について次の表に示します。表中で参照しているメッセージについては，マニュアル「HiRDB メッセージ」を参照してください。

表 13–33　ODBC3.5 ドライバが返却する SQLSTATE

項番	ODBC 規格の SQLSTATE	意味	返却する SQLSTATE	
			HiRDB 独自※1	ODBC 準拠※2
1	01000	General Warning 一般警告	HY000	01000
2	01001	Cursor operation conflict カーソル操作の競合	−	−
3	01002	Disconnect error 接続解除エラー	−	−
4	01003	NULL value eliminated in set function 集合関数で NULL 値が削除された	−	−
5	01004	String data, right-truncated 文字列データの右側が切り捨てられた	01004	01004
6	01006	Privilege not revoked 特権が破棄されない	−	−
7	01007	Privilege not granted 特権が与えられない	−	−
8	01S00	Invalid connection string attribute 無効な接続文字列属性	01S00	01S00
9	01S01	Error in row 行のエラー	−	−
10	01S02	Option value changed オプション値の変更	01S02	01S02
11	01S06	Attempt to fetch before the result set returned the first rowset 結果セットが最初の行セットを返す前にフェッチを試みた	−	−
12	01S07	Fractional truncation 小数点以下切り捨て	−	−
13	01S08	Error saving File DSN ファイル DSN の保存エラー	−	−
14	01S09	Invalid keyword (DM) 無効なキーワード	−	−

項番	ODBC 規格の SQLSTATE	意味	返却する SQLSTATE	
			HiRDB 独自※1	ODBC 準拠※2
15	07001	Wrong number of parameters パラメタ数不正	07001 （ODBC ドライバでエラー検知した場合） HY000 （HiRDB サーバでエラー検知した場合）	07001
16	07002	COUNT field incorrect COUNT フィールドが不正	HY000	07002
17	07005	Prepared statement not a cursor-specification 準備されたステートメントが cursor-specification でない	07005 （ODBC ドライバでエラー検知した場合） HY000 （HiRDB サーバでエラー検知した場合）	07005
18	07006	Restricted data type attribute violation データ型属性の制限違反	07006 （ODBC ドライバでエラー検知した場合） HY000 （HiRDB サーバでエラー検知した場合）	07006
19	07009	Invalid descriptor index 無効なディスクリプタインデクス	07009	07009
20	07S01	Invalid use of default parameter デフォルトパラメタの不正使用	－	－
21	08001	Client unable to establish connection クライアントが接続を確立できない	HY000	08001
22	08002	Connection name in use(DM) 接続名が使用中である	08002 （ODBC ドライバでエラー検知した場合） HY000 （HiRDB サーバでエラー検知した場合）	08002

項番	ODBC 規格の SQLSTATE	意味	返却する SQLSTATE	
			HiRDB 独自※1	ODBC 準拠※2
23	08003	Connection not open(DM) 接続が存在しない	08003 （ODBC ドライバでエラー検知した場合） HY000 （HiRDB サーバでエラー検知した場合）	08003
24	08004	Server rejected the connection サーバが接続を拒否した	HY000	08004
25	08007	Connection failure during transaction トランザクション時の接続失敗	HY000	08007
26	08S01	Communication link failure 通信リンク失敗	HY000	08S01
27	21S01	Insert value list does not match column list 挿入する値の一覧と列の一覧の不一致	HY000	21S01
28	21S02	Degree of derived table does not match column list 導出したテーブルの次数が列の一覧と不一致	HY000	21S02
29	22001	String data, right-truncated 文字列データの右側が切り捨てられた	22001 （ODBC ドライバでエラー検知した場合） HY000 （HiRDB サーバでエラー検知した場合）	22001
30	22002	Indicator variable required but not supplied 必要な標識変数が提供されない	HY000	22002
31	22003	Numeric value out of range 数値が範囲外である	22003 （ODBC ドライバでエラー検知した場合） HY000 （HiRDB サーバでエラー検知した場合）	22003
32	22007	Invalid datetime format 無効な日付時刻形式	22007 （ODBC ドライバでエラー検知した場合）	22007

項番	ODBC 規格の SQLSTATE	意味	返却する SQLSTATE	
			HiRDB 独自※1	ODBC 準拠※2
			HY000 （HiRDB サーバでエラー検知した場合）	
33	22008	Datetime field overflow 日付時刻フィールドのオーバーフロー	22008 （ODBC ドライバでエラー検知した場合） HY000 （HiRDB サーバでエラー検知した場合）	22008
34	22012	Division by zero ゼロ除算	HY000	22012
35	22015	Interval field overflow 間隔フィールドのオーバーフロー	HY000	22015
36	22018	Invalid character value for cast specification キャスト指定値に対する無効な文字値	HY000	22018
37	22019	Invalid escape character 無効なエスケープ文字	HY000	22019
38	22025	Invalid escape sequence 無効なエスケープシーケンス	HY000	22025
39	22026	String data, length mismatch 文字列データの長さが一致しない	－	－
40	23000	Integrity constraint violation 整合性の制約違反	HY000	23000
41	24000	Invalid cursor state 無効なカーソル状態	24000 （ODBC ドライバでエラー検知した場合） HY000 （HiRDB サーバでエラー検知した場合）	24000
42	25000	Invalid transaction state 無効なトランザクション状態	25000 （ODBC ドライバでエラー検知した場合） HY000	25000

項番	ODBC 規格の SQLSTATE	意味	返却する SQLSTATE	
			HiRDB 独自※1	ODBC 準拠※2
			（HiRDB サーバでエラー検知した場合）	
43	25S01	Transaction state トランザクション状態が不明	－	－
44	25S02	Transaction is still active トランザクションがアクティブである	－	－
45	25S03	Transaction is rolled back トランザクションがロールバックされた	－	－
46	28000	Invalid authorization specification 無効な認証指定	28000 （ODBC ドライバでエラー検知した場合） HY000 （HiRDB サーバでエラー検知した場合）	28000
47	34000	Invalid cursor name 無効なカーソル名	34000 （ODBC ドライバでエラー検知した場合） HY000 （HiRDB サーバでエラー検知した場合）	34000
48	3C000	Duplicate cursor name カーソル名の重複	3C000 （ODBC ドライバでエラー検知した場合） HY000 （HiRDB サーバでエラー検知した場合）	3C000
49	3D000	Invalid catalog name 無効なカタログ名	－	－
50	3F000	Invalid schema name 無効なスキーマ名	HY000	3F000
51	40001	Serialization failure 直列化の失敗	HY000	40001

項番	ODBC 規格の SQLSTATE	意味	返却する SQLSTATE	
			HiRDB 独自※1	ODBC 準拠※2
52	40002	Integrity constraint violation 一貫性制約に違反した	−	−
53	40003	Statement completion unknown ステートメントの完了が不明	−	−
54	42000	Syntax error or access violation 構文エラー又はアクセス違反	HY000	42000
55	42S01	Base table or view already exists ベーステーブル又はビューが既に存在する	HY000	42S01
56	42S02	Base table or view not found ベーステーブル又はビューが見つからない	HY000	42S02
57	42S11	Index already exists インデクスが既に存在する	HY000	42S11
58	42S12	Index not exists インデクスが見つからない	HY000	42S12
59	42S21	Column already exists 列が既に存在する	HY000	42S21
60	42S22	Column not foun 列が見つからない	HY000	42S22
61	44000	WITH CHECK OPTION violation WITH CHECK OPTION 違反	−	−
62	HY000	General error 一般エラー	HY000	HY000
63	HY001	Memory allocation error メモリ割り当てエラー	HY001	HY001
64	HY003	Invalid application buffer type 無効なアプリケーションバッファのデータ型	HY003	HY003
65	HY004	Invalid SQL data type 無効な SQL データ型	HY004	HY004
66	HY007	Associated statement is not prepared 関連されたステートメントが準備されていない	HY007	HY007
67	HY008	Operation canceled 動作がキャンセルされた	−	−
68	HY009	Invalid use of null pointer NULL ポインタ不正使用	HY009	HY009

項番	ODBC 規格の SQLSTATE	意味	返却する SQLSTATE	
			HiRDB 独自※1	ODBC 準拠※2
69	HY010	Function sequence error 関数シーケンスエラー	HY010	HY010
70	HY011	Attribute cannot be set now ここでは属性を設定できない	HY011	HY011
71	HY012	Invalid transaction operation code 無効なトランザクション処理コード	HY012	HY012
72	HY013	Memory management error メモリ管理エラー	HY013	HY013
73	HY014	Limit on the number of handles exceeded ハンドル数の上限を超過	−	−
74	HY015	No cursor name available 利用できるカーソル名がない	−	−
75	HY016	Cannot modify an implementation row descriptor インプリメンテーション行ディスクリプタを変更できない	HY016	HY016
76	HY017	Invalid use of an automatically allocated descriptor handle 自動的に割り当てられるディスクリプタハンドルの不正使用	HY017	HY017
77	HY018	Server declined cancel request サーバがキャンセル要求を拒否した	−	−
78	HY019	Non-character and non-binary data sent in pieces 文字データとバイナリデータ以外のデータが分割送信された	−	−
79	HY020	Attempt to concatenate a null value NULL 値の連結を試みた	−	−
80	HY021	Inconsistent descriptor information ディスクリプタ情報の不一致	−	−
81	HY024	Invalid attribute value 無効な属性値	HY024	HY024
82	HY090	Invalid string or buffer length 無効な文字列長又は無効なバッファ長	HY090	HY090
83	HY091	Invalid descriptor field identifier 無効なディスクリプタフィールド識別子	HY091	HY091
84	HY092	Invalid attribute/option identifier 無効な属性識別子又は無効なオプション識別子	HY092	HY092

項番	ODBC 規格の SQLSTATE	意味	返却する SQLSTATE	
			HiRDB 独自※1	ODBC 準拠※2
85	HY095	Function type out of range 関数型が範囲外である	HY095	HY095
86	HY096	Invalid information type 無効な情報型	−	−
87	HY097	Column type out of range 列の種類が範囲外である	HY097	HY097
88	HY098	Scope type out of range 引数 scope の値が範囲外である	−	−
89	HY099	Nullable type out of range 引数 Nullable の値が範囲外である	HY099	HY099
90	HY100	Uniqueness option type out of range 一意性のオプションが範囲外である	HY100	HY100
91	HY101	Accuracy option type out of range 精度のオプションが範囲外である	HY101	HY101
92	HY103	Invalid retrieval code 無効な取得コード	−	−
93	HY104	Invalid precision or scale value 無効な精度又は無効なスケール値	−	−
94	HY105	Invalid parameter type 無効なパラメタの種類	HY105	HY105
95	HY106	Fetch type out of range フェッチの種類が範囲外である	−	−
96	HY107	Row value out of range 行の値が範囲外である	−	−
97	HY109	Invalid cursor position 無効なカーソル位置	−	−
98	HY110	Invalid driver completion 引数 DriverCompletion が無効である	−	−
99	HY111	Invalid bookmark value 無効なブックマーク値	−	−
100	HYC00	Optional feature not implemented オプション機能は実装されていない	HYC00	HYC00
101	HYT00	Timeout expired タイムアウト終了	−	−

項番	ODBC 規格の SQLSTATE	意味	返却する SQLSTATE	
			HiRDB 独自※1	ODBC 準拠※2
102	HYT01	Connection timeout expired 接続タイムアウト終了	−	−
103	IM001	Driver does not support this function ドライバはこの関数をサポートしていない	−	−
104	IM002	Data source name not found and no default driver specified データソース名が見つからずデフォルトのドライバが指定されていない	−	−
105	IM003	Specified driver could not be loaded 指定のドライバがロードできない	−	−
106	IM004	Driver's SQLAllocHandle on SQL_HANDLE_ENV failed ドライバが SQL_HANDLE_ENV に対する SQLAllocHandle に失敗した	−	−
107	IM005	Driver's SQLAllocHandle on SQL_HANDLE_DBC failed ドライバが SQL_HANDLE_DBC に対する SQLAllocHandle に失敗した	−	−
108	IM006	Driver's SQLSetConnectAttr failed ドライバが SQLSetConnectAttr に失敗した	−	−
109	IM007	No data source or driver specified; dialog prohibited データソース又はドライバが指定されず，ダイアログが禁止された	−	−
110	IM008	Dialog failed ダイアログの失敗	−	−
111	IM009	Unable to load translation DLL トランスレータ DLL をロードできない	−	−
112	IM010	Data source name too long データソース名が長過ぎる	−	−
113	IM011	Driver name too long ドライバ名が長過ぎる	−	−
114	IM012	DRIVER keyword syntax error DRIVER キーワードの構文エラー	−	−
115	IM013	Trace file error トレースファイルのエラー	−	−
116	IM014	Invalid name of File DSN ファイル DSN の名前が無効である	−	−

項番	ODBC 規格の SQLSTATE	意味	返却する SQLSTATE	
			HiRDB 独自※1	ODBC 準拠※2
117	IM015	Corrupt file data source ファイルデータソースが壊れている	−	−

（凡例）

−：該当なし。

注※1

クライアント環境定義 PDODBSTANDARDSQLSTATE に NO を指定した場合，又は指定を省略した場合

注※2

クライアント環境定義 PDODBSTANDARDSQLSTATE に YES を指定した場合

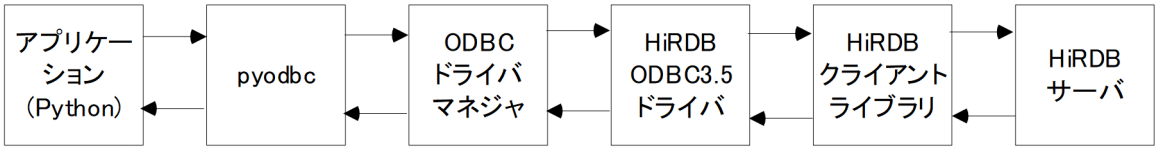
なお，HiRDB サーバから ODBC 規格にない SQLSTATE が返却された場合，HY000 にマッピングされます。

13.18　Python アプリケーションからの HiRDB アクセス

Python アプリケーションから HiRDB サーバに接続するには，pyodbc を使用して HiRDB ODBC3.5 ドライバを経由します。

pyodbc とは，Python からデータベースに接続するための規格，Python Database API 2.0 の実装の一つです。ODBC ドライバを使用してデータベースにアクセスできます。

図 13–1　Python アプリケーションからの HiRDB アクセス



13.18.1　サポート対象

サポート対象を次の表に示します。

表 13–34　サポート対象

項目	サポート対象
プラットフォーム	Windows,Linux
アドレッシングモード	64 ビット

13.18.2　データ型

(1)　検索データ取得時のマッピング

HiRDB から検索データを取得した場合の，HiRDB の SQL データ型と，取得される Python オブジェクトのクラスのマッピングを次の表に示します。

表 13–35　HiRDB の SQL データ型と，取得される Python オブジェクトのクラスのマッピング

HiRDB の SQL データ型	Python オブジェクトのクラス
INT	int
SMALLINT	int
DEC	decimal.Decimal
FLOAT	float
REAL	float

HiRDB の SQL データ型	Python オブジェクトのクラス
CHAR	str
VARCHAR	str
NCHAR	str
NVARCHAR	str
MCHAR	str
MVARCHAR	str
BINARY	bytes
BLOB	bytes
DATE	datetime.date
TIME	datetime.time
TIMESTAMP	datetime.datetime
INTERVAL HOUR TO SECOND	−
INTERVAL YEAR TO DAY	−

（凡例）

−：未サポート

(2)　？パラメタ設定時のマッピング

SQL の？パラメタとして設定する Python オブジェクトのクラスと，HiRDB の SQL データ型のマッピングを次の表に示します。

表 13–36　パラメタに設定する Python オブジェクトのクラスと，HiRDB の SQL データ型のマッピング（1/2）

Python オブジェクトのクラス	INT SMALLINT	DEC	FLOAT REAL	CHAR VARCHAR MCHAR MVARCHAR
int	◎	○	○	○
decimal.Decimal	○	◎	○	○
float	○	○	◎	○
str	○	○	○	◎
bytes	×	×	×	×
datetime.date	×	×	×	×

Python オブジェクトのクラス	INT SMALLINT	DEC	FLOAT REAL	CHAR VARCHAR MCHAR MVARCHAR
datetime.time	×	×	×	×
datetime.datetime	×	×	×	×

表 13–37　パラメタに設定する Python オブジェクトのクラスと，HiRDB の SQL データ型のマッピング（2/2）

Python オブジェクトのクラス	NCHAR NVARCHAR	BINARY BLOB	DATE	TIME	TIMESTAMP
int	×	×	×	×	×
decimal.Decimal	×	×	×	×	×
float	×	×	×	×	×
str	◎	×	×	×	○※2
bytes	×	◎※1	×	×	×
datetime.date	×	×	◎	×	×
datetime.time	×	×	×	◎	×
datetime.datetime	×	×	×	×	◎※2

（凡例）

◎：マッピングすることを推奨します。

○：マッピングできます。ただし，変換元データの形式によっては，データの欠落や変換エラーとなることがあるため，注意してください。

×：マッピングできません。

注※1

データ型として SQL_LONGVARBINARY を指定してください。

指定方法は，Cursor クラスの setinputsizes メソッドの引数の，パラメタ番号に対応する位置に(pyodbc.SQL_LONGVARBINARY,定義長)を指定してください。

BINARY(100000)の列に第 2 パラメタで INSERT する場合の例を示します。

```
cur.setinputsizes([None,(pyodbc.SQL_LONGVARBINARY,100000)])
cur.execute("INSERT INTO TABLE1 VALUES (?,?,?)",123,b"ABC",456)
```

注※2

datetime.datetime をパラメタ値として指定すると，小数秒が設定されている場合に小数秒が切り捨てられて設定されます。小数秒まで含む値を設定したい場合は，str をパラメタ値として使用し，データ型として SQL_CHAR（長さ 26）を指定してください。

指定方法は，Cursor クラスの setinputsizes メソッドの引数の，パラメタ番号に対応する位置に（pyodbc.SQL_CHAR,26）を指定してください。

TIMESTAMP(6)の列に第 2 パラメタで INSERT する場合の例を示します。

```
cur.setinputsizes([None,(pyodbc.SQL_CHAR,26)])
cur.execute("INSERT INTO TABLE2 VALUES (?,?,?)",123,"2000-12-31 12:34:56.123456",456)
```

(3) HiRDB サーバの文字コードとエンコーディング

HiRDB サーバの文字コードと，それに対応した UAP の設定について次の表に示します。

表 13–38 HiRDB サーバの文字コードに対応した UAP の設定

プラットフォーム	サーバの文字コード		
	EUC	UTF-8	SJIS
Windows	クライアント環境定義 PDCLTCNVMODE に UCS2_UJIS を設定してください。	クライアント環境定義 PDCLTCNVMODE に UCS2_UTF8 を設定してください。	Connection クラスの setencoding メソッドに"shift-jis"を指定してください。
Linux			Connection クラスの setencoding メソッドに"shift-jis"を指定してください※。

注※

Connection クラスの setencoding メソッドに"shift-jis"を指定した場合，SQL 文に全角文字が含まれるとエラーとなります。このため，SQL 文に全角文字が含まれる場合，setencoding メソッドは指定しないで，パラメタとして文字列を指定する個所で，データ型として SQL_VARCHAR を指定してください。

指定方法は，Cursor クラスの setinputsizes メソッドの引数の，パラメタ番号に対応する位置に（pyodbc.SQL_VARCHAR,列のバイト長）を指定してください。

CHAR(10)の列に第 2 パラメタで INSERT する場合の例を示します。

```
cur.setinputsizes([None,(pyodbc.SQL_VARCHAR,10)])
cur.execute("INSERT INTO TABLE3 VALUES (?,?,?)",123,"あいう",456)
```

13.18.3 使用できない機能

(1) 使用できないメソッド

Cursor クラスの次のメソッドは使用できません。

- skip
- rowIdColumns

- rowVerColumns
- primaryKeys
- foreignKeys

(2) タイムアウトの設定

pyodbc.connect 関数の timeout 引数及び Connection クラスの timeout 属性は，指定しても有効になりません。

接続時のタイムアウト及び SQL 実行時のタイムアウトは，クライアント環境定義で指定してください。

接続時のタイムアウト：

PDNBLOCKWAITTIME 及び PDCONNECTWAITTIME

SQL 実行時のタイムアウト：

PDCWAITTIME

(3) unixODBC のコネクションプーリング

unixODBC 使用時は，コネクションプーリングを使用しないでください。

odbcinst.ini でコネクションプーリングを有効（Pooling = Yes）にしている場合は，pyodbc.pooling に False を設定してください。

(4) pyodbc.connect 関数の ansi 引数

pyodbc.connect 関数の ansi 引数は指定しないでください。

14 OLE DB 対応アプリケーションプログラムからの HiRDB アクセス

この章では，OLE DB の概要，接続インタフェース，スキーマ情報，及び障害対策について説明します。

14.1 概要

14.1.1 OLE DB の概要

(1) OLE DB とは

OLE DB とは，ODBC と同様に，広範囲なデータソースにアクセスするための API です。また，ODBC とは異なり，SQL データ以外のデータアクセスに適したインタフェースも定義されています。

(2) HiRDB OLE DB プロバイダ

OLE DB 対応アプリケーションプログラムから HiRDB へアクセスする場合，HiRDB OLE DB プロバイダを経由してアクセスします。HiRDB OLE DB プロバイダは，HiRDB/Run Time，及び HiRDB/Developer's Kit に含まれています。

(3) HiRDB OLE DB プロバイダのインストール

HiRDB/Run Time，又は HiRDB/Developer's Kit のインストール時に，「セットアップ タイプ」画面で「標準」を選択するか，又は「カスタム」を選択し，「機能の選択」画面で「HiRDB OLE DB プロバイダ」をチェックすると，HiRDB OLE DB プロバイダがインストールされます。

HiRDB OLE DB プロバイダをインストールすると，次のファイルが作成されます。

- PDOLEDB.DLL
- PDCLTM32.DLL

HiRDB OLE DB プロバイダは，32 ビット版だけサポートしています。

(4) HiRDB OLE DB プロバイダ名

HiRDB OLE DB プロバイダのプロバイダ名（プロバイダプログラム ID）は，「HiRDBProvider」です。

プロバイダ名を指定するインタフェース（例えば，ADO（ActiveX Data Object））を使用する場合，connection オブジェクトの Provider プロパティにこのプロバイダ名を設定することで，HiRDB の OLE DB プロバイダを使用できます。

14.2　接続インタフェース

ここでは，レジストリ情報と接続プロパティについて説明します。

14.2.1　レジストリ情報

(1)　HKEY_CLASSES_ROOT キーに追加

(a)　プロバイダプログラム ID ＝プロバイダ名称

```
"HiRDBProvider"="Hitachi HiRDB OLE DB Provider"
```

(b)　プロバイダクラス ID

```
"HiRDBProvider¥¥CLSID"="{6A708561-748A-11d3-B810-0000E2212E58}"
```

(2)　HKEY_CLASSES_ROOT¥CLSID サブキーに追加

(a)　プロバイダプログラム ID

```
{"CLSID¥¥{6A708561-748A-11d3-B810-0000E2212E58}"="HiRDBProvider"
```

(b)　プロバイダ名称

```
"CLSID¥¥{6A708561-748A-11d3-B810-0000E2212E58}¥¥ProgID"
  ="HiRDBProvider"
```

(c)　バージョン別プログラム ID

```
"CLSID¥¥{6A708561-748A-11d3-B810-0000E2212E58}
  ¥¥VersionIndependentProgID"="HiRDBProvider"
```

(d)　プロバイダ DLL 名称

```
"CLSID¥¥{6A708561-748A-11d3-B810-0000E2212E58}¥¥InprocServer32"
  ="pdoledb.dll"
"CLSID¥¥{6A708561-748A-11d3-B810-0000E2212E58}¥¥InprocServer32
  ¥¥ThreadingModel"="Both"
```

(e)　コメント

```
"CLSID¥¥{6A708561-748A-11d3-B810-0000E2212E58}¥¥OLE DB Provider"
  ="Hitachi HiRDB OLE DB Provider"
```

(f)　拡張エラー名称

```
"CLSID¥¥{6A708561-748A-11d3-B810-0000E2212E58}¥¥ExtendedErrors"
  ="Hitachi HiRDB OLE DB Provider"
```

(g)　拡張エラーコメント

```
"CLSID¥¥{6A708561-748A-11d3-B810-0000E2212E58}
  ¥¥ExtendedErrors¥¥{5F6D492E-40BA-11D3-BD66-0000E21F878E}"
  = "Hitachi HiRDB OLE DB Provider"
```

(3)　HKEY_CLASSES_ROOT キーに追加

(a)　プロバイダエラープログラム ID

```
" HiRDBProviderErrors"="Hitachi HiRDB OLE DB Provider"
```

(b)　プロバイダエラークラス ID

```
"HiRDBProviderErrors¥¥ClSID"="{5F6D492E-40BA-11D3-BD66-0000E21F878E}"
```

(4)　HKEY_CLASSES_ROOT¥CLSID サブキーに追加

(a)　プロバイダエラープログラム ID

```
"CLSID¥¥{5F6D492E-40BA-11D3-BD66-0000E21F878E}"
  ="HiRDBProvider Error Lookup"
```

(b)　プロバイダエラールックアップ名称

```
"CLSID¥¥{5F6D492E-40BA-11D3-BD66-0000E21F878E}¥¥ProgID"
  ="HiRDBProvider Error Lookup"
```

(c)　バージョン別エラールックアッププログラム ID

```
"CLSID¥¥{5F6D492E-40BA-11D3-BD66-0000E21F878E}
  ¥¥VersionIndependentProgID"="HiRDBProvider Error Lookup"
```

(d)　プロバイダエラールックアップ DLL 名称

```
"CLSID¥¥{5F6D492E-40BA-11D3-BD66-0000E21F878E}¥¥InprocServer32"
  ="pdoledb.dll"
"CLSID¥¥{5F6D492E-40BA-11D3-BD66-0000E21F878E}¥¥InprocServer32
  ¥¥ThreadingModel"="Both"
```

14.2.2　接続プロパティ

接続に使用するプロパティは，Initialization プロパティの次の三つです。ただし，この三つは省略できます。

(1)　DBPROP_INIT_DATASOURCE

クライアントの環境変数グループ名です。

省略した場合は HiRDB.INI を使用します。クライアントの環境変数グループについては，「環境変数のグループ登録」を参照してください。

(2)　DBPROP_AUTH_USERID

接続に使用する認可識別子です。

省略した場合，DBPROP_INIT_DATASOURCE の指定があれば，該当するクライアントの環境変数グループの PDUSER から認可識別子を取得します。DBPROP_INIT_DATASOURCE の指定がない場合は HiRDB.INI から認可識別子を取得します。

(3)　DBPROP_AUTH_PASSWORD

接続に使用するパスワードです。

省略した場合，DBPROP_INIT_DATASOURCE の指定があれば，該当するクライアントの環境変数グループの PDUSER からパスワードを取得します。DBPROP_INIT_DATASOURCE の指定がない場合は HiRDB.INI からパスワードを取得します。

14.3　スキーマ情報

HiRDB OLE DB プロバイダが提供するスキーマ情報を次の表に示します。

表 14–1　HiRDB OLE DB プロバイダが提供するスキーマ情報

OLE DB スキーマ情報種別	説明	提供
ASSERTIONS	アサーション情報	×
CATALOGS	カタログ情報	×
CHARACTER_SETS	文字セットの識別	×
CHECK_CONSTRAINTS	CHECK 制約の識別	×
COLLATIONS	文字照合の識別	×
COLUMN_DOMAIN_USAGE	ドメインに依存する列情報	×
COLUMN_PRIVILEGES	列の特権情報	×
COLUMNS	列情報	○(必要)
CONSTRAINT_COLUMN_USAGE	各制約(参照，UNIQUE，CHECK)列情報	×
CONSTRAINT_TABLE_USAGE	各制約(参照，UNIQUE，CHECK)表情報	×
FOREIGN_KEYS	外部キー情報	×
INDEXES	インデクス情報	○
KEY_COLUMN_USAGE	キー列情報	×
PRIMARY_KEYS	主キー情報	×
PROCEDURE_COLUMNS	プロシジャが返す行セットの列情報	×
PROCEDURE_PARAMETERS	プロシジャのパラメタ情報	○
PROCEDURES	プロシジャ情報	○
PROVIDER_TYPES	プロバイダデータ型識別	○(必要)
REFERENTIAL_CONSTRAINTS	参照制約情報	×
SCHEMATA	スキーマ情報	○
SQL_LANGUAGES	SQL の実装を処理する合致レベル，及び言語の識別	×
STATISTICS	統計情報	×
TABLE_CONSTRAINTS	テーブル制約情報	×
TABLE_PRIVILEGES	テーブル特権情報	○
TABLES	テーブル情報	○(必要)
TRANSLATIONS	文字変換識別	×

OLE DB スキーマ情報種別	説明	提供
USAGE_PRIVILEGES	ユーザ権限情報	×
VIEW_COLUMN_USAGE	ビュー列情報	×
VIEWS	ビュー情報	×

（凡例）

○：提供しています。

×：提供していません。

14.4　データ型の対応

HiRDB のデータ型と OLE DB 型標識の対応を次の表に示します。

表 14–2　HiRDB のデータ型と OLE DB 型標識の対応

HiRDB のデータ型	OLE DB 型標識
CHAR，MCHAR，及び NCHAR	DBTYPE_STR
VARCHAR，MVARCHAR，及び NVARCHAR	
DECIMAL(p,s)	DBTYPE_NUMERIC
SMALLINT (signed)	DBTYPE_I2
INTEGER (signed)	DBTYPE_I4
REAL	DBTYPE_R4
SMALLFLT	
FLOAT	DBTYPE_R8
DOUBLE PRECISION	
BLOB	DBTYPE_BYTES
BINARY	
DATE	DBTYPE_DBDATE
TIME	DBTYPE_DBTIME
TIMESTAMP	DBTYPE_DBTIMESTAMP
INTERVAL YEAR TO DAY	DBTYPE_DECIMAL
INTERVAL HOUR TO SECOND	DBTYPE_DECIMAL

14.5　障害対策

14.5.1　トラブルシュート機能

コンシューマから発行される OLE DB インタフェース（メソッド単位）のトレースを取得します。

(1)　取得方法

次のレジストリキーに値を設定します。

HKEY_LOCAL_MACHINE

Software¥HITACHI¥HiRDB¥oleprovtrc の値が 1 の場合にだけトレースを取得します。

Software¥HITACHI¥HiRDB¥oletrcfile に出力ファイル名を絶対パスで指定します（oletrcfile 省略時は"c:¥temp¥pdoletrc.txt"に出力します）。

Software¥HITACHI¥HiRDB¥oletrcdumpsize に GetData()で出力，Execute()で入力されます。void*型データのダンプ出力サイズをバイト単位で指定します（oletrcdumpsize 省略時は 256 が仮定されます）。

14.6　留意事項

14.6.1　ADO でのカーソルについて

HiRDB では，ADO でのカーソルの後方スクロールやカーソルを使用した更新を行いたい場合，クライアントカーソル（RecordSet オブジェクトの CursorLocation プロパティに adUseClient を設定）を使用します。

その他の場合，サーバカーソル（RecordSet オブジェクトの CursorLocation プロパティに adUseServer を設定）を使用してください。

なお，プロシジャからの結果セットはサポートしていないため，CALL 文実行時は必ずサーバカーソルを使用してください。

15 ADO.NET 対応アプリケーションプログラムからの HiRDB アクセス

この章では，ADO.NET 対応アプリケーションプログラムから HiRDB をアクセスする場合に必要な，HiRDB データプロバイダ for .NET Framework のインストール，機能，UAP 例などについて説明します。

なお，.NET Framework 1.1 に対応した HiRDB データプロバイダ for .NET Framework は廃止になりました。現在はアプリケーションの互換性を保つために使用できますが，将来は削除されます。代わりに.NET Framework 2.0 以降に対応した HiRDB データプロバイダ for .NET Framework を使用してください。

15.1　概要

15.1.1　HiRDB データプロバイダ for .NET Framework

.NET Framework では，プラットフォームや開発言語に依存しない共通言語ランタイム，及び.NET Framework クラスライブラリを提供します。ADO.NET とは，データベースにアクセスする.NET Framework アプリケーション作成時に利用できるライブラリのことをいいます。

HiRDB では，ADO.NET を使用して HiRDB をアクセスするために必要な **HiRDB データプロバイダ for .NET Framework** を提供しています。HiRDB データプロバイダ for .NET Framework は，ADO.NET 仕様に準拠したデータプロバイダです。

HiRDB データプロバイダ for .NET Framework は，.NET Framework の System.Data 空間で提供されている共通基本インタフェース群を実装しています。また，HiRDB データプロバイダ for .NET Framework 独自の拡張機能として，配列を使用した INSERT 機能，及び繰返し列へのアクセスを実装しています。

15.1.2　HiRDB データプロバイダ for .NET Framework の前提プログラム

(1)　稼働プラットフォーム

.NET Framework 1.1 対応（32 ビットモード）：

- Windows 10

.NET Framework 2.0 対応（32 ビットモード）：

- Windows 10

.NET Framework 2.0 対応（64 ビットモード）：

- Windows 10

.NET Framework 4 対応（32 ビットモード）：

- Windows Server 2019
- Windows Server 2022
- Windows Server 2025
- Windows 10
- Windows 11

.NET Framework 4 対応（64 ビットモード）：

- Windows Server 2019
- Windows Server 2022
- Windows Server 2025
- Windows 10
- Windows 11

(2) 必要なプログラム

アプリケーションプログラムの開発及び実行には，次の環境が必要です。

●.NET Framework 1.1 対応の UAP を開発する場合：

開発環境

次のどちらかを選択してください。

- Microsoft Visual Studio .NET 2003
- Microsoft Visual Studio 2005

実行環境

次のどれかを選択してください。

- .NET Framework Version 1.1 再頒布可能パッケージ
- .NET Framework Version 2.0 再頒布可能パッケージ
- .NET Framework Version 3.0 再頒布可能パッケージ

●.NET Framework 2.0 対応の UAP を開発する場合：

開発環境

次のどれかを選択してください。

- Microsoft Visual Studio 2005
- Microsoft Visual Studio 2008
- Microsoft Visual Studio 2010
- Microsoft Visual Studio 2012
- Microsoft Visual Studio 2013

実行環境

次のどれかを選択してください。

- .NET Framework Version 2.0 再頒布可能パッケージ
- .NET Framework Version 3.0 再頒布可能パッケージ
- .NET Framework 3.5 再頒布可能パッケージ
- .NET Framework 4 Client Profile 再頒布可能パッケージ

- .NET Framework 4 再頒布可能パッケージ
- .NET Framework 4.5 再頒布可能パッケージ
- .NET Framework 4.5.1 再頒布可能パッケージ
- .NET Framework 4.5.2 再頒布可能パッケージ

.NET Framework 4～4.5.2 だけがインストールされた環境で.NET Framework 2.0 対応の HiRDB データプロバイダ for .NET Framework を使用する場合，アプリケーション構成ファイルに次の設定をしてください。

- <startup>要素の useLegacyV2RuntimeActivationPolicy 属性を true にする
- <supportedRuntime>要素の version 属性を v4.0 にする

●.NET Framework 2.0（64 ビットモード）対応の UAP を開発する場合：

開発環境

次のどれかを選択してください。

- Microsoft Visual Studio 2005
- Microsoft Visual Studio 2008
- Microsoft Visual Studio 2010
- Microsoft Visual Studio 2012
- Microsoft Visual Studio 2013

実行環境

次のどれかを選択してください。

- .NET Framework Version 2.0 再頒布可能パッケージ
- .NET Framework Version 3.0 再頒布可能パッケージ
- .NET Framework 3.5 再頒布可能パッケージ
- .NET Framework 4 Client Profile 再頒布可能パッケージ
- .NET Framework 4 再頒布可能パッケージ
- .NET Framework 4.5 再頒布可能パッケージ
- .NET Framework 4.5.1 再頒布可能パッケージ
- .NET Framework 4.5.2 再頒布可能パッケージ

.NET Framework 4～4.5.2 だけがインストールされた環境で.NET Framework 2.0 対応の HiRDB データプロバイダ for .NET Framework を使用する場合，アプリケーション構成ファイルに次の設定をしてください。

- <startup>要素の useLegacyV2RuntimeActivationPolicy 属性を true にする
- <supportedRuntime>要素の version 属性を v4.0 にする

●.NET Framework 4 対応の UAP を開発する場合：

開発環境

次のどれかを選択してください。

- Microsoft Visual Studio 2010
- Microsoft Visual Studio 2012
- Microsoft Visual Studio 2013
- Microsoft Visual Studio 2015
- Microsoft Visual Studio 2017
- Microsoft Visual Studio 2019
- Microsoft Visual Studio 2022

実行環境

次のどれかを選択してください。

- .NET Framework Version 4 Client Profile 再頒布可能パッケージ
- .NET Framework Version 4 再頒布可能パッケージ
- .NET Framework Version 4.5 再頒布可能パッケージ
- .NET Framework Version 4.5.1 再頒布可能パッケージ
- .NET Framework Version 4.5.2 再頒布可能パッケージ
- .NET Framework Version 4.6 再頒布可能パッケージ
- .NET Framework Version 4.6.1 再頒布可能パッケージ
- .NET Framework Version 4.6.2 再頒布可能パッケージ
- .NET Framework Version 4.7 再頒布可能パッケージ
- .NET Framework Version 4.7.1 再頒布可能パッケージ
- .NET Framework Version 4.7.2 再頒布可能パッケージ
- .NET Framework Version 4.8 再頒布可能パッケージ
- .NET Framework Version 4.8.1 再頒布可能パッケージ
- .NET 6
- .NET 8

●.NET Framework 4（64 ビットモード）対応の UAP を開発する場合：

開発環境

次のどれかを選択してください。

- Microsoft Visual Studio 2010
- Microsoft Visual Studio 2012

- Microsoft Visual Studio 2013
- Microsoft Visual Studio 2015
- Microsoft Visual Studio 2017
- Microsoft Visual Studio 2019
- Microsoft Visual Studio 2022

実行環境

次のどれかを選択してください。

- .NET Framework Version 4 Client Profile 再頒布可能パッケージ
- .NET Framework Version 4 再頒布可能パッケージ
- .NET Framework Version 4.5 再頒布可能パッケージ
- .NET Framework Version 4.5.1 再頒布可能パッケージ
- .NET Framework Version 4.5.2 再頒布可能パッケージ
- .NET Framework Version 4.6 再頒布可能パッケージ
- .NET Framework Version 4.6.1 再頒布可能パッケージ
- .NET Framework Version 4.6.2 再頒布可能パッケージ
- .NET Framework Version 4.7 再頒布可能パッケージ
- .NET Framework Version 4.7.1 再頒布可能パッケージ
- .NET Framework Version 4.7.2 再頒布可能パッケージ
- .NET Framework Version 4.8 再頒布可能パッケージ
- .NET Framework Version 4.8.1 再頒布可能パッケージ
- .NET 6
- .NET 8

.NET Framework の再頒布可能パッケージは Windows Update で導入できます。なお，Microsoft Visual Studio .NET 2002 ＋ .NET Framework SDK Version 1.1 での動作は保証しません。

15.1.3 HiRDB データプロバイダ for .NET Framework と対応する ADO.NET のバージョン

HiRDB データプロバイダ for .NET Framework は，次の表で示すバージョンの ADO.NET に対応しています。

表 15–1 HiRDB データプロバイダ for .NET Framework と対応する ADO.NET のバージョン

HiRDB データプロバイダ for .NET Framework	対応する ADO.NET のバージョン
.NET Framework 1.1 対応	ADO.NET 1.1

HiRDB データプロバイダ for .NET Framework	対応する ADO.NET のバージョン
.NET Framework 2.0 対応	ADO.NET 2.0
.NET Framework 4 対応	

15.2　HiRDB データプロバイダ for .NET Framework のインストール

15.2.1　インストール手順

HiRDB/Run Time，又は HiRDB/Developer's Kit のインストール時に，「セットアップタイプ」画面で「すべて」を選択するか，又は「カスタム」を選択して「機能の選択」画面で「HiRDB.NET データプロバイダ」をチェックすると，HiRDB データプロバイダ for .NET Framework がインストールされます。

このとき，HiRDB データプロバイダ for .NET Framework の DLL 群と発行者ポリシーがグローバルアセンブリキャッシュへ配置されます。また，発行者ポリシーによってインストールしている最新の HiRDB データプロバイダ for .NET Framework が使用されるため，HiRDB データプロバイダ for .NET Framework をバージョンアップした場合でも，UAP を再ビルドする必要はありません。

HiRDB データプロバイダ for .NET Framework のグローバルアセンブリキャッシュへの配置については，「グローバルアセンブリキャッシュへの配置」を参照してください。

15.2.2　インストールされるファイル

HiRDB データプロバイダ for .NET Framework をインストールすると，次のファイルが作成されます。

HiRDB データプロバイダ for .NET Framework（.NET Framework 1.1 対応）の場合：

- pddndp.dll
- pddndpcore.dll

HiRDB データプロバイダ for .NET Framework（.NET Framework 2.0 対応）の場合：

32 ビットのとき

- pddndp20.dll
- pddndpcore20.dll

64 ビットのとき

- pddndp20x.dll
- pddndpcore20x.dll

HiRDB データプロバイダ for .NET Framework（.NET Framework 4 対応）の場合：

32 ビットのとき

- pddndp40.dll
- pddndpcore40.dll

64 ビットのとき

- pddndp40x.dll

- pddndpcore40x.dll

HiRDB データプロバイダ for .NET Framework を使用する場合は，上記のファイルを参照設定に追加してください。

15.2.3　バージョン情報の確認

HiRDB データプロバイダ for .NET Framework のバージョン情報は，HiRDB データプロバイダ for .NET Framework の提供 DLL のプロパティで確認できます。

15.3　HiRDB データプロバイダ for .NET Framework のクラス一覧

HiRDB データプロバイダ for .NET Framework は，ADO.NET の仕様に準拠しています。

HiRDB データプロバイダ for .NET Framework のクラス一覧を次の表に示します。

クラスは名前空間「Hitachi.HiRDB」に属しています。

表 15–2　HiRDB データプロバイダ for .NET Framework クラス一覧

クラス	機能	対応する ADO.NET のバージョン
HiRDBCommand	データベースに対して実行する SQL ステートメント又はストアドプロシジャを表します。	1.1 以降
HiRDBCommandBuilder	データベースに関連づけられた DataSet への変更を調整するための単一テーブルコマンドを自動的に生成します。	1.1 以降
HiRDBConnection	データベースへの開いた接続を表します。	1.1 以降
HiRDBDataAdapter	DataSet へのデータの格納及びデータベースの更新に使用される，一連のデータコマンド及びデータベース接続を表します。	1.1 以降
HiRDBDataReader	データベースからデータ行の前方向ストリームを読み取る方法を提供します。	1.1 以降
HiRDBException	HiRDB データプロバイダ for .NET Framework から警告又はエラーが返された場合に生成される例外です。	1.1 以降
HiRDBParameter	HiRDBCommand のパラメタと，オプションとして DataColumn に対するマップを表します。	1.1 以降
HiRDBParameterCollection	HiRDBCommand に関連するパラメタコレクション，及び DataSet 列に対する各パラメタのマップを表します。	1.1 以降
HiRDBProviderFactory	HiRDB データプロバイダ for .NET Framework が提供する各クラスのインスタンスを生成します。	2.0 以降
HiRDBRowUpdatedEventArgs	RowUpdated イベントのデータを提供します。	1.1 以降
HiRDBRowUpdatingEventArgs	RowUpdating イベントのデータを提供します。	1.1 以降
HiRDBTransaction	データベースで実行するトランザクションを表します。	1.1 以降

15.4　HiRDB データプロバイダ for .NET Framework のメンバー覧

HiRDB データプロバイダ for .NET Framework の提供するインタフェースのメンバー覧を示します。

15.4.1　HiRDBCommand のメンバー覧

(1)　コンストラクタ

HiRDBCommand

(2)　継承クラス

対応する ADO.NET のバージョンが 1.1 の場合：

Component，IDbCommand，ICloneable

対応する ADO.NET のバージョンが 2.0 の場合：

DbCommand

(3)　プロパティ

メンバ	機能	対応する ADO.NET のバージョン
CommandText	データベースに対して実行するテキストコマンドを取得又は設定します。	1.1 以降
CommandTimeout	SQL 実行のタイムアウト時間（秒数）を取得又は設定します。	1.1 以降
CommandType	CommandText プロパティの解釈方法を示す値を取得又は設定します。	1.1 以降
Connection	この HiRDBCommand で使用される HiRDBConnection を取得又は設定します。	1.1 以降
DesignTimeVisible	インタフェースコントロールに HiRDBCommand オブジェクトを連携した場合に，HiRDBCommand オブジェクトをコントロール上に表示するかどうかを示す値を取得又は設定します。	2.0 以降
Parameters	HiRDBParameterCollection を取得します。	1.1 以降
Transaction	この HiRDBCommand が実行される HiRDBTransaction を取得又は設定します。	1.1 以降
UpdatedRowSource	HiRDBDataAdapter の Update メソッドがコマンド結果を使用するときに，コマンド結果を DataRow に適用する方法を取得又は設定します。	1.1 以降

(4)　メソッド

メンバ	機能	対応する ADO.NET のバージョン
Cancel ()	HiRDBCommand の実行中の SQL をキャンセルします。	1.1 以降
Clone ()	現在のインスタンスのコピーである新しいオブジェクトを作成します。	1.1 以降
CreateParameter ()	HiRDBParameter オブジェクトの新しいインスタンスを作成します。	1.1 以降
Dispose()	HiRDBCommand によって使用されているすべてのリソースを解放します。	1.1 以降
ExecuteNonQuery ()	HiRDBConnection オブジェクトに対して SQL ステートメントを実行し，影響を受けた行の数を返します。	1.1 以降
ExecuteNonQuery (int)		1.1 以降
ExecuteReader ()	HiRDBConnection に対して CommandText を実行し，HiRDBDataReader を構築します。	1.1 以降
ExecuteReader (CommandBehavior)		1.1 以降
ExecuteScalar ()	クエリを実行し，そのクエリが返す結果セットの最初の行にある最初の列を返します。余分な列又は行は無視されます。	1.1 以降
Prepare ()	準備されたバージョンのコマンド（コンパイル済み）をデータベースに作成します。	1.1 以降

15.4.2　HiRDBCommandBuilder のメンバー一覧

(1)　コンストラクタ

HiRDBCommandBuilder

(2)　継承クラス

対応する ADO.NET のバージョンが 1.1 の場合：

　Component

対応する ADO.NET のバージョンが 2.0 の場合：

　DbCommandBuilder

(3)　プロパティ

メンバ	機能	対応する ADO.NET のバージョン
CatalogLocation	サーバ名，カタログ名，スキーマ名，テーブル名で修飾テーブル名を示すときのカタログ名の位置を示します。	2.0 以降

メンバ	機能	対応する ADO.NET のバージョン
CatalogSeparator	HiRDBCommandBuilder オブジェクトのカタログ区切り記号として使用する文字列を設定又は取得します。	2.0 以降
ConflictOption	該当するオブジェクトが生成する UpdateCommand と DeleteCommand の WHERE 句に追加する列の組み合わせ（同時実行チェックの種類）を設定又は取得します。	2.0 以降
DataAdapter	SQL ステートメントの自動生成の対象となる HiRDBDataAdapter オブジェクトを取得又は設定します。	1.1 以降
QuotePrefix	列やテーブルの識別子を指定するための開始文字を取得又は設定します。	2.0 以降
QuoteSuffix	列やテーブルの識別子を指定するための終了文字を取得又は設定します。	2.0 以降
SchemaSeparator	認可識別子とそのほかの識別子との間の区切り記号に使用する文字を取得又は設定します。	2.0 以降
SetAllValues	UPDATE 文で値を更新する列がすべての列かどうかを取得又は設定します。	2.0 以降

(4) メソッド

メンバ	機能	対応する ADO.NET のバージョン
Dispose()	HiRDBCommandBuilder によって使用されているすべてのリソースを解放します。	1.1 以降
GetDeleteCommand ()	データベースで削除処理を実行するための，自動生成された HiRDBCommand オブジェクトを取得します。	2.0 以降
GetDeleteCommand (bool)		2.0 以降
GetDeleteCommand (string)		1.1 以降
GetInsertCommand ()	データベースで挿入処理を実行するための，自動生成された HiRDBCommand オブジェクトを取得します。	2.0 以降
GetInsertCommand (bool)		2.0 以降
GetInsertCommand (string)		1.1 以降
GetUpdateCommand ()	データベースで更新処理を実行するための，自動生成された HiRDBCommand オブジェクトを取得します。	2.0 以降
GetUpdateCommand (bool)		2.0 以降
GetUpdateCommand (string)		1.1 以降
QuoteIdentifier (string)	指定された文字列を HiRDBCommandBuilder#QuotePrefix プロパティの値と HiRDBCommandBuilder#QuoteSuffix プロパティの値で囲って返します。	2.0 以降
RefreshSchema ()	INSERT，UPDATE，又は DELETE ステートメントを生成するための，データベーススキーマ情報を更新します。	2.0 以降
RefreshSchema (string)		1.1 以降

メンバ	機能	対応する ADO.NET のバージョン
UnquoteIdentifier (string)	指定された文字列から先頭の HiRDBCommandBuilder#QuotePrefix プロパティの値と終端の HiRDBCommandBuilder#QuoteSuffix プロパティの値を除いたものを返します。	2.0 以降

15.4.3 HiRDBConnection のメンバー一覧

(1) コンストラクタ

HiRDBConnection

(2) 継承クラス

対応する ADO.NET のバージョンが 1.1 の場合：

Component，IDbConnection，ICloneable

対応する ADO.NET のバージョンが 2.0 の場合：

DbConnection

(3) プロパティ

メンバ	機能	対応する ADO.NET のバージョン
ConnectionString	データベースを開くために使用する文字列を取得又は設定します。	1.1 以降
ConnectionTimeout	試行を中断して，エラーを生成する前に接続の確立時に待機する時間を取得します。	1.1 以降
Database	現在のデータベース，又は接続が開いてから使用するデータベースの名前を取得します。	1.1 以降
DataSource	接続しているデータベースサーバの名前を取得します。	2.0 以降
LifeTime	実際に切断するまでの時間を取得又は設定します。	1.1 以降
Pooling	プーリングを行うかどうかを取得又は設定します。	1.1 以降
ServerVersion	接続中のサーバのバージョンを取得します。	2.0 以降
State	現在の接続状態を取得します。	1.1 以降
ConnectionRecover	接続プーリング機能使用時に，接続回復機能を使用するかどうかを取得又は設定します。	2.0 以降

(4) メソッド

メンバ	機能	対応する ADO.NET のバージョン
BeginTransaction ()	データベーストランザクションを開始します。	1.1 以降
BeginTransaction (IsolationLevel)		1.1 以降
ChangeDatabase (string)	開いている HiRDBConnection オブジェクトの現在のデータベースを変更します。	1.1 以降
Clone ()	現在のインスタンスのコピーである新しいオブジェクトを作成します。	1.1 以降
Close ()	データベースへの接続を閉じます。	1.1 以降
CreateCommand ()	接続に関連づけられた HiRDBCommand オブジェクトを作成し，返します。	1.1 以降
Dispose()	HiRDBConnection によって使用されているすべてのリソースを解放します。	1.1 以降
EnlistTransaction (Transaction)	指定されたトランザクションに登録します。	2.0 以降
GetSchema ()	スキーマ情報を返します。	2.0 以降
GetSchema (string)		2.0 以降
GetSchema (string, string[])		2.0 以降
Open ()	HiRDBConnection オブジェクトの ConnectionString プロパティで指定されている設定で，データベース接続を開きます。	1.1 以降

15.4.4 HiRDBDataAdapter のメンバー一覧

(1) コンストラクタ

HiRDBDataAdapter

(2) 継承クラス

対応する ADO.NET のバージョンが 1.1 の場合：

DbDataAdapter，IDbDataAdapter

対応する ADO.NET のバージョンが 2.0 の場合：

DbDataAdapter

(3) プロパティ

メンバ	機能	対応する ADO.NET のバージョン
DeleteCommand	データセットからレコードを削除する SQL ステートメントを取得又は設定します。	1.1 以降
InsertCommand	データベースに新しいレコードを挿入する SQL ステートメントを取得又は設定します。	1.1 以降
SelectCommand	データベース内のレコードを選択する SQL ステートメントを取得又は設定します。	1.1 以降
UpdateCommand	データベース内のレコードを更新する SQL ステートメントを取得又は設定します。	1.1 以降

(4) イベント

メンバ	機能	対応する ADO.NET のバージョン
RowUpdated	更新処理時に，データソースに対してコマンドを実行した後に発生します。	1.1 以降
RowUpdating	更新処理時に，データソースに対してコマンドを実行する前に発生します。	1.1 以降

15.4.5 HiRDBDataReader のメンバー一覧

(1) コンストラクタ

HiRDBDataReader

(2) 継承クラス

対応する ADO.NET のバージョンが 1.1 の場合：

MarshalByRefObject，IEnumerable，IDataReader，IDisposable，IDataRecord

対応する ADO.NET のバージョンが 2.0 の場合：

DbDataReader

(3) プロパティ

メンバ	機能	対応する ADO.NET のバージョン
Depth	現在の行の入れ子の深さを示す値を取得します。	1.1 以降
FieldCount	現在の行の列数を取得します。	1.1 以降

メンバ	機能	対応する ADO.NET のバージョン
HasRows	該当する HiRDBDataReader に 1 行以上の行が格納されているかどうかを示す値を取得します。	2.0 以降
IsClosed	データリーダが閉じているかどうかを示す値を取得します。	1.1 以降
Item[int]	HiRDBDataReader オブジェクトを配列のように扱い，データを取得します。	1.1 以降
Item[int,int]		1.1 以降
Item[string]		1.1 以降
RecordsAffected	SQL ステートメントの実行によって，変更，挿入，又は削除された行の数を取得します。	1.1 以降
VisibleFieldCount	HiRDBDataReader の非表示ではない列数を取得します。	2.0 以降

(4) メソッド

メンバ	機能	対応する ADO.NET のバージョン
Close ()	HiRDBDataReader オブジェクトを閉じます。	1.1 以降
Dispose()	HiRDBDataReader によって使用されているすべてのリソースを解放します。	1.1 以降
GetBoolean (int)	指定した列の値をブール値として取得します。	1.1 以降
GetByte (int)	指定した列の 8 ビット符号なし整数値を取得します。	1.1 以降
GetBytes (int, long, byte[], int, int)	指定したバッファオフセットを開始位置として，指定した列オフセットからバッファに，バイトのストリームを配列として読み込みます。	1.1 以降
GetChar (int)	指定した列の文字値を取得します。	1.1 以降
GetChars (int, long, byte[], int, int)	指定したバッファオフセットを開始位置として，指定した列オフセットからバッファに，文字のストリームを配列として読み込みます。	1.1 以降
GetData (int)	このメンバは，.NET Framework インフラストラクチャのサポートを目的としています。独自に作成したコード内で直接使用できません。	1.1 以降
GetDataTypeName (int)	指定したフィールドのデータ型情報を取得します。	1.1 以降
GetDateTime (int)	指定したフィールドの日時のデータ値を取得又は設定します。	1.1 以降
GetDecimal (int)	指定したフィールドの固定位置数値を取得します。	1.1 以降
GetDouble (int)	指定したフィールドの倍精度浮動小数点数を取得します。	1.1 以降
GetEnumerator ()	コレクションを反復処理できる列挙子を返します。	1.1 以降
GetFieldArrayCount (int)	フィールドの配列の大きさを取得します。	1.1 以降

メンバ	機能	対応する ADO.NET のバージョン
GetFieldType (int)	GetValue から返される Object の型に対応する Type 情報を取得します。	1.1 以降
GetFloat (int)	指定したフィールドの単精度浮動小数点数を取得します。	1.1 以降
GetGuid (int)	指定したフィールドの GUID 値を返します。	1.1 以降
GetInt16 (int)	指定したフィールドの 16 ビット符号付き整数値を取得します。	1.1 以降
GetInt32 (int)	指定したフィールドの 32 ビット符号付き整数値を取得します。	1.1 以降
GetInt64 (int)	指定したフィールドの 64 ビット符号付き整数値を取得します。	1.1 以降
GetName (int)	検索するフィールドの名前を取得します。	1.1 以降
GetOrdinal (string)	指定したフィールドのインデクスを返します。	1.1 以降
GetProviderSpecificFieldType (int)	指定した列のデータ型を取得します。	2.0 以降
GetProviderSpecificValue (int)	指定した列の値を Object のインスタンスとして取得します。	2.0 以降
GetProviderSpecificValues (Object[])	現在の行のコレクション内にあるすべての属性列を取得します。	2.0 以降
GetSchemaTable ()	HiRDBDataReader の列メタデータを説明する DataTable を返します。	1.1 以降
GetString (int)	指定したフィールドの文字列値を取得します。	1.1 以降
GetValue (int)	指定した列の値を Object のインスタンスとして取得します。	1.1 以降
GetValue (int, int)	指定した列の指定した要素の値を Object 型オブジェクトとして取得します。	1.1 以降
GetValues (object[])	現在のレコードコレクション内のすべての属性フィールドを取得します。	1.1 以降
IsDBNull (int)	指定したフィールドが null に設定されているかどうかを示す値を返します。	1.1 以降
NextResult ()	バッチ SQL ステートメントの結果を読み込むときに，データリーダを次の結果に進めます。	1.1 以降
Read ()	HiRDBDataReader を次のレコードに進めます。	1.1 以降

15.4.6 HiRDBException のメンバー覧

(1) コンストラクタ

HiRDBException

(2)　継承クラス

対応する ADO.NET のバージョンが 1.1 の場合：

Exception

対応する ADO.NET のバージョンが 2.0 の場合：

DbException

(3)　プロパティ

メンバ	機能	対応する ADO.NET のバージョン
ErrorCode	エラーのコード部分を int として取得します。	1.1 以降
Message	エラーの完全なテキストを取得します。	1.1 以降

15.4.7　HiRDBParameter のメンバー一覧

(1)　コンストラクタ

HiRDBParameter

(2)　継承クラス

対応する ADO.NET のバージョンが 1.1 の場合：

MarshalByRefObject，IDbDataParameter，IDataParameter，ICloneable

対応する ADO.NET のバージョンが 2.0 の場合：

DbParameter，IDbDataParameter

(3)　プロパティ

メンバ	機能	対応する ADO.NET のバージョン
DbType	パラメタの DbType を取得又は設定します。設定時には，表「DbType プロパティ設定時に自動設定される HiRDBType プロパティの値」に従って，HiRDBType プロパティに該当するデータタイプを設定します。	1.1 以降
Direction	パラメタが入力専用，出力専用，双方向，又はストアドプロシジャの戻り値パラメタかどうかを示す値を取得又は設定します。	1.1 以降

メンバ	機能	対応する ADO.NET のバージョン
HiRDBType	HiRDB でのデータタイプを示す列挙体を取得，又は設定します。設定時には，表「HiRDBType プロパティ設定時に自動設定される DbType プロパティの値」に従って，該当するデータタイプを DbType プロパティに設定します。 ［HiRDBType 列挙体］ Integer，SmallInt，Decimal，Float，SmallFlt，Char，VarChar，NChar，NVarChar， MChar，MVarChar，Date，Time，TimeStamp，IntervalYearToDay， IntervalHourToSecond，Blob，Binary	1.1 以降
IsNullable	パラメタが null 値を受け入れるかどうかを示す値を取得します。	1.1 以降
ParameterName	HiRDBParameter の名前を取得又は設定します。	1.1 以降
Precision	DECIMAL 型パラメタの定義長の有効けた数（小数部けた数を含む）を取得又は設定します。HiRDBType プロパティの値が HiRDBType.Decimal の場合，必ず設定してください。	1.1 以降
Repetition	HiRDB での配列構造を取得又は設定します。	1.1 以降
Scale	DECIMAL 型パラメタの定義長の小数部けた数を取得又は設定します。HiRDBType プロパティの値が HiRDBType.Decimal の場合，必ず設定してください。	1.1 以降
Size	パラメタの定義長を取得又は設定します。固定長（数値型や日時型など）の場合は 0 を，可変長（文字列型など）の場合はテーブルに格納するバイト数又は列の最大長を設定してください。TIMESTAMP（DateTime）の場合は，小数部のけた数となります。 なお，Size プロパティの設定よりも長い文字列を入力した場合は，Size プロパティに設定した長さまでの文字列が格納されます。例外は発生しません。	1.1 以降
SourceColumn	DataSet に割り当てられていて，Value を読み込むとき，又は戻すときに使用されるソース列の名前を取得又は設定します。	1.1 以降
SourceColumnNullMapping	パラメタに対応する DataTable オブジェクトの列が NULL 値を許すかどうかを示す値を取得又は設定します。	1.1 以降
SourceVersion	Value の読み込みに使用する DataRowVersion を取得又は設定します。	1.1 以降
Value	パラメタの値を取得又は設定します。	1.1 以降

(4) メソッド

メンバ	機能	対応する ADO.NET のバージョン
Clone ()	現在のインスタンスのコピーである新しいオブジェクトを作成します。	1.1 以降

メンバ	機能	対応する ADO.NET のバージョン
ResetDbType ()	DbType プロパティの値を初期値に戻します。	2.0 以降

15.4.8 HiRDBParameterCollection のメンバー一覧

(1) コンストラクタ

HiRDBParameterCollection

(2) 継承クラス

対応する ADO.NET のバージョンが 1.1 の場合：

MarshalByRefObject，IDataParameterCollection，IList，ICollection，IEnumerable

対応する ADO.NET のバージョンが 2.0 の場合：

DbParameterCollection

(3) プロパティ

メンバ	機能	対応する ADO.NET のバージョン
Count	HiRDBParameterCollection に格納されている HiRDBParameter オブジェクトの数を取得します。	1.1 以降
IsFixedSize	HiRDBParameterCollection が固定サイズかどうかを示す値を取得します。	1.1 以降
IsReadOnly	HiRDBParameterCollection が読み取り専用かどうかを示す値を取得します。	1.1 以降
IsSynchronized	HiRDBParameterCollection へのアクセスが同期されている（スレッドセーフである）かどうかを示す値を取得します。	1.1 以降
Item[int]	指定したインデクスの HiRDBParameter オブジェクトを取得，又は指定したインデクスに HiRDBParameter オブジェクトを設定します。	1.1 以降
Item[string]		1.1 以降
SyncRoot	HiRDBParameterCollection へのアクセスを同期するために使用できるオブジェクトを取得します。	1.1 以降

(4) メソッド

メンバ	機能	対応する ADO.NET のバージョン
Add (object)	HiRDBParameterCollection に項目を追加します。	1.1 以降
Add (HiRDBParameter)		1.1 以降
Add (string, object)		1.1 以降
Add (string, HiRDBType)		1.1 以降
Add (string, HiRDBType, int)		1.1 以降
Add (string, HiRDBType, int, string)		1.1 以降
Add2 (HiRDBParameter)		2.0 以降
Add2(string, object)		2.0 以降
Add2(string, HiRDBType)		2.0 以降
Add2(string, HiRDBType, int)		2.0 以降
Add2(string, HiRDBType, int, string)		2.0 以降
AddRange (Array)	HiRDBParameter オブジェクトの配列を HiRDBParameterCollection に追加します。	1.1 以降
AddRange (HiRDBParameter[])		1.1 以降
Clear ()	HiRDBParameterCollection からすべての項目を削除します。	1.1 以降
Contains (object)	HiRDBParameter がコレクション内にあるかどうかを示す値を取得します。	1.1 以降
Contains (HiRDBParameter)		1.1 以降
Contains (string)		1.1 以降
CopyTo (Array, int)	Array の特定のインデクスを開始位置として，Array に HiRDBParameterCollection の要素をコピーします。	1.1 以降
GetEnumerator ()	コレクションを反復処理できる列挙子を返します。	1.1 以降
IndexOf (object)	コレクション内の HiRDBParameter の位置を取得します。	1.1 以降
IndexOf (string)		1.1 以降
Insert (int, object)	HiRDBParameterCollection 内の指定した位置に項目を挿入します。	1.1 以降
Insert (int, HiRDBParameter)		1.1 以降
Remove	HiRDBParameterCollection 内にある特定のオブジェクトのうち，最初に出現するオブジェクトを削除します。	1.1 以降
RemoveAt (int)	HiRDBParameter をコレクションから削除します。	1.1 以降
RemoveAt (string)		1.1 以降

15.4.9　HiRDBProviderFactory のメンバ一覧

(1)　コンストラクタ

HiRDBProviderFactory

(2)　継承クラス

DbProviderFactory

(3)　フィールド

Instance

(4)　プロパティ

メンバ	機能	対応する ADO.NET のバージョン
CanCreateDataSourceEnumerator	DbDataSourceEnumerator クラスから派生したクラスをサポートするかどうかを示します。	2.0 以降

(5)　メソッド

メンバ	機能	対応する ADO.NET のバージョン
CreateCommand ()	HiRDBCommand オブジェクトを作成して返します。	2.0 以降
CreateCommandBuilder ()	HiRDBCommandBuilder オブジェクトを作成して返します。	2.0 以降
CreateConnection ()	HiRDBConnection オブジェクトを作成して返します。	2.0 以降
CreateConnectionStringBuilder ()	DbConnectionStringBuilder オブジェクトを作成して返します。	2.0 以降
CreateDataAdapter ()	HiRDBDataAdapter オブジェクトを作成して返します。	2.0 以降
CreateDataSourceEnumerator ()	System.NotSupportedException を返します。	2.0 以降
CreateParameter ()	HiRDBParameter オブジェクトを作成して返します。	2.0 以降

15.4.10　HiRDBRowUpdatedEventArgs のメンバ一覧

(1)　コンストラクタ

HiRDBRowUpdatedEventArgs

(2) 継承クラス

RowUpdatedEventArgs

(3) プロパティ

メンバ	機能	対応する ADO.NET のバージョン
Command	Update の呼び出し時に実行される HiRDBCommand を取得します。	1.1 以降

15.4.11 HiRDBRowUpdatingEventArgs のメンバー一覧

(1) コンストラクタ

HiRDBRowUpdatingEventArgs

(2) 継承クラス

RowUpdatingEventArgss

(3) プロパティ

メンバ	機能	対応する ADO.NET のバージョン
Command	Update 処理中に実行する HiRDBCommand を取得又は設定します。	1.1 以降

15.4.12 HiRDBTransaction メンバー一覧

(1) コンストラクタ

HiRDBTransaction

(2) 継承クラス

対応する ADO.NET のバージョンが 1.1 の場合：

MarshalByRefObject，IDbTransaction，IDisposable

対応する ADO.NET のバージョンが 2.0 の場合：

DbTransaction

(3) プロパティ

メンバ	機能	対応する ADO.NET のバージョン
Connection	トランザクションを関連づける HiRDBConnection オブジェクトを取得します。	1.1 以降
IsCompleted	トランザクションが完了しているかどうかを取得します。	1.1 以降
IsolationLevel	このトランザクションの IsolationLevel を指定します。	1.1 以降

(4) メソッド

メンバ	機能	対応する ADO.NET のバージョン
Commit ()	データベーストランザクションをコミットします。	1.1 以降
Dispose()	HiRDBTransaction によって使用されているすべてのリソースを解放します	1.1 以降
Rollback ()	保留中の状態からデータベーストランザクションをロールバックします。	1.1 以降

15.5　HiRDB データプロバイダ for .NET Framework のインタフェース

15.5.1　HiRDBCommand

(1)　コンストラクタ

(a)　HiRDBCommand

void HiRDBCommand ()

説明：HiRDBCommand の新しいインスタンスを初期化します。

void HiRDBCommand (string)

引数

string cmdText：SQL テキスト(CommandText プロパティ)

説明：SQL テキストを指定して，HiRDBCommand クラスの新しいインスタンスを初期化します。

void HiRDBCommand (string, Hitachi.HiRDB.HiRDBConnection)

引数

string cmdText：SQL テキスト(CommandText プロパティ)

HiRDBConnection rConnection：データベースへの接続を表す HiRDBConnection オブジェクト(Connection プロパティ)

説明：SQL テキストと HiRDBConnection オブジェクトを使用して，HiRDBCommand クラスの新しいインスタンスを初期化します。

void HiRDBCommand (string, Hitachi.HiRDB.HiRDBConnection, Hitachi.HiRDB.HiRDBTransaction)

引数

string cmdText：SQL テキスト（CommandText プロパティ）

HiRDBConnection rConnection：データベースへの接続を表す HiRDBConnection オブジェクト(CommandText プロパティ)

HiRDBTransaction rTransaction：HiRDBCommand を実行する HiRDBTransaction オブジェクト（Transaction プロパティ）

説明：SQL テキスト，HiRDBConnection オブジェクト，及び HiRDBTransaction オブジェクトを使用して，HiRDBCommand クラスの新しいインスタンスを初期化します。

(2)　プロパティ

(a)　CommandText

型：string

既定値：""

説明：データベースに対して実行するテキストコマンドを取得又は設定します。

(b) CommandTimeout

型：int

既定値：30

説明：SQL 実行のタイムアウト時間（秒数）を取得又は設定します。

例外：HiRDBException

(c) CommandType

型：System.Data.CommandType

既定値：CommandType.Text

説明：CommandText プロパティの解釈方法を取得又は設定します。

(d) Connection

型：HiRDBConnection

既定値：null

説明：この HiRDBCommand で使用される HiRDBConnection を取得又は設定します。

例外：HiRDBException

(e) DesignTimeVisible

型：bool

既定値：true

説明：インタフェースコントロールに HiRDBCommand オブジェクトを連携した場合に，HiRDBCommand オブジェクトをコントロール上に表示するかどうかを示す値を取得又は設定します。

(f) Parameters

型：HiRDBParameterCollection

説明：HiRDBParameterCollection を取得します（読み取り専用）。

(g) Transaction

型：HiRDBTransaction

既定値：null

説明：この HiRDBCommand が実行される HiRDBTransaction を取得又は設定します。

(h)　UpdatedRowSource

型：System.Data.UpdateRowSource

既定値：UpdatedRowSource.None

説明：HiRDBDataAdapter の Update メソッドがコマンド結果を使用するときに，コマンド結果を DataRow に適用する方法を取得又は設定します。

例外：HiRDBException

(3)　メソッド

(a)　Cancel

void Cancel ()

Return：void

説明：HiRDBCommand の実行中の SQL をキャンセルします。

(b)　Clone

object Clone ()

Return

object：このインスタンスのコピーである新しいオブジェクト

説明：現在のインスタンスのコピーである新しいオブジェクトを作成します。

(c)　CreateParameter

Hitachi.HiRDB.HiRDBParameter CreateParameter ()

Return

HiRDBParameter：HiRDBParameter オブジェクト

説明：HiRDBParameter オブジェクトの新しいインスタンスを作成します。

(d)　Dispose

void Dispose()

Return：void

説明：HiRDBCommand が使用しているすべてのリソースを解放します。

(e) ExecuteNonQuery

int ExecuteNonQuery ()

Return

int：影響を受けた行の数

説明：HiRDBConnection オブジェクトに対して SQL ステートメントを実行し，影響を受けた行の数を返します。

例外：HiRDBException

int ExecuteNonQuery (int)

引数

int nArraySize：配列要素数

Return

int：影響を受けた行の数

説明：配列を使用した INSERT 機能を使用し，HiRDBConnection オブジェクトに対して SQL ステートメントを実行し，影響を受けた行の数を返します。

例外：HiRDBException

(f) ExecuteReader

Hitachi.HiRDB.HiRDBDataReader ExecuteReader ()

Return

HiRDBDataReader：HiRDBDataReader オブジェクト

説明：HiRDBConnection に対して CommandText を実行し，HiRDBDataReader を構築します。

例外：HiRDBException

ExecuteReader (System.Data.CommandBehavior)

引数

System.DataCommandBehavior behavior：CommandBehavior 値の一つ

Return

HiRDBDataReader：HiRDBDataReader オブジェクト

説明：HiRDBConnection に対して CommandText を実行し，CommandBehavior 値の一つを使用して HiRDBDataReader を構築します。

例外：HiRDBException

(g) ExecuteScalar

object ExecuteScalar ()

Return

object：結果セットの最初の行にある最初の列

説明：クエリを実行し，そのクエリが .NET Framework のデータ型で返す結果セットの最初の行の最初の列を返します。残りの列又は行は無視されます。

例外：HiRDBException

(h) Prepare

void Prepare ()

Return：void

説明：準備されたバージョンのコマンド（コンパイル済み）をデータベースに作成します。

例外：HiRDBException

15.5.2 HiRDBCommandBuilder

(1) コンストラクタ

(a) HiRDBCommandBuilder

void HiRDBCommandBuilder ()

説明：HiRDBCommandBuilder の新しいインスタンスを初期化します。

void HiRDBCommandBuilder (HiRDBDataAdapter adapter)

引数

HiRDBDataAdapter adapter：HiRDBDataAdapter オブジェクト（DataAdapter プロパティ）

説明：HiRDBDataAdapter オブジェクトを指定して，HiRDBCommandBuilder の新しいインスタンスを初期化します。

(2) プロパティ

(a) CatalogLocation

型：CatalogLocation

既定値：CatalogLocation.Start

説明：サーバ名，カタログ名，スキーマ名，テーブル名で修飾テーブル名を示すときのカタログ名の位置を示します。

例外：HiRDBException

(b) CatalogSeparator

型：string

既定値：""

説明：HiRDBCommandBuilder オブジェクトのカタログ区切り記号として使用する文字列を設定又は取得します。

例外：HiRDBException

(c) ConflictOption

型：ConflictOption

既定値：ConflictOption.CompareAllSearchableValues

説明：該当するオブジェクトが生成する UpdateCommand と DeleteCommand の WHERE 句に追加する列の組み合わせ（同時実行チェックの種類）を設定又は取得します。

例外：HiRDBException

(d) DataAdapter

型：HiRDBDataAdapter

既定値：null

説明：SQL ステートメントを自動生成する対象の HiRDBDataAdapter オブジェクトを取得又は設定します。

(e) QuotePrefix

型：string

既定値：”（引用符）

説明：列やテーブルの識別子を指定するための開始文字を取得又は設定します。

例外：HiRDBException

(f) QuoteSuffix

型：string

既定値：”（引用符）

説明：列やテーブルの識別子を指定するための終了文字を取得又は設定します。

例外：HiRDBException

(g) SchemaSeparator

型：HiRDBDataAdapter

既定値：.（ピリオド）

説明：認可識別子とそのほかの識別子との間の区切り記号に使用する文字を取得又は設定します。

例外：HiRDBException

(h) SetAllValues

型：bool

既定値：true

説明：UPDATE 文で値を更新する列がすべての列かどうかを示す値を，取得又は設定します。

(3) メソッド

(a) Dispose

void Dispose()

Return：void

説明：HiRDBCommandBuilder が使用しているすべてのリソースを解放します。

(b) GetDeleteCommand

HiRDBCommand GetDeleteCommand()

Return

HiRDBCommand：削除を実行するための，自動生成された HiRDBCommand オブジェクト

説明：データベースで削除処理を実行するための，自動生成された HiRDBCommand オブジェクトを取得します。DataAdapter プロパティの SelectCommand に指定する検索 SQL 文が次に示す場合は，削除 SQL 文を作成できません。

- 単一表に対する検索ではない
- 表に別名を指定している

例外：HiRDBException

HiRDBCommand GetDeleteCommand(bool)

引数

useColumnsForParameterNames

true：列名に基づいたパラメタ名（@ID など）

false：@pX 形式のパラメタ名（X：1 からの序数）

Return

HiRDBCommand：削除を実行するための，自動生成された HiRDBCommand オブジェクト

説明：データベースで削除処理を実行するための，自動生成された HiRDBCommand オブジェクトを取得します。DataAdapter プロパティの SelectCommand に指定する検索 SQL 文が次に示す場合は，削除 SQL 文を作成できません。

- 単一表に対する検索ではない
- 表に別名を指定している

例外：HiRDBException

HiRDBCommand GetDeleteCommand (string)

引数

string sTableName：テーブル名

Return

HiRDBCommand：削除を実行するための，自動生成された HiRDBCommand オブジェクト

説明：データベースで削除処理を実行するための，自動生成された HiRDBCommand オブジェクトを取得します。

例外：HiRDBException

(c) GetInsertCommand

HiRDBCommand GetInsertCommand ()

Return

HiRDBCommand：挿入を実行するための自動生成された HiRDBCommand オブジェクト

説明：データベースで挿入処理を実行するための，自動生成された HiRDBCommand オブジェクトを取得します。DataAdapter プロパティの SelectCommand に指定する検索 SQL 文が次に示す場合，挿入 SQL 文を作成できません。

- 単一表に対する検索ではない。
- 表に別名を指定している。

例外：HiRDBException

HiRDBCommand GetInsertCommand (bool)

引数

useColumnsForParameterNames

true：列名に基づいたパラメタ名（@ID など）

false：@pX 形式のパラメタ名（X：1 からの序数）

Return

HiRDBCommand：挿入を実行するための，自動生成された HiRDBCommand オブジェクト

説明：データベースで挿入処理を実行するための，自動生成された HiRDBCommand オブジェクトを取得します。DataAdapter プロパティの SelectCommand に指定する検索 SQL 文が次に示す場合，挿入 SQL 文を作成できません。

- 単一表に対する検索ではない
- 表に別名を指定している

例外：HiRDBException

HiRDBCommand GetInsertCommand (string)

引数

string sTableName：テーブル名

Return

HiRDBCommand：挿入を実行するための，自動生成された HiRDBCommand オブジェクト

説明：データベースで挿入処理を実行するための，自動生成された HiRDBCommand オブジェクトを取得します。

例外：HiRDBException

(d) GetUpdateCommand

HiRDBCommand GetUpdateCommand ()

Return

HiRDBCommand：更新を実行するための，自動生成された HiRDBCommand オブジェクト

説明：データベースで更新処理を実行するための，自動生成された HiRDBCommand オブジェクトを取得します。DataAdapter プロパティの SelectCommand に指定する検索 SQL 文が次に示す場合，更新 SQL 文を作成できません。

- 単一表に対する検索ではない。
- 表に別名を指定している。

例外：HiRDBException

HiRDBCommand GetUpdateCommand (bool)

引数

useColumnsForParameterNames：

true：列名に基づいたパラメタ名（@ID など）

false：@pX 形式のパラメタ名（X：1 からの序数）

Return

HiRDBCommand：更新を実行するための自動生成された HiRDBCommand オブジェクト

説明：データベースで更新処理を実行するための，自動生成された HiRDBCommand オブジェクトを取得します。DataAdapter プロパティの SelectCommand に指定する検索 SQL 文が次に示す場合，更新 SQL 文を作成できません。

- 単一表に対する検索ではない
- 表に別名を指定している

例外：HiRDBException

HiRDBCommand GetUpdateCommand (string)

引数

string sTableName：テーブル名

Return

HiRDBCommand：更新を実行するための自動生成された HiRDBCommand オブジェクト

説明：データベースで更新処理を実行するための，自動生成された HiRDBCommand オブジェクトを取得します。

例外：HiRDBException

(e) QuoteIdentifier

string QuoteIdentifier(string)

引数

string unquotedIdentifier：HiRDBCommandBuilder#QuotePrefix プロパティの値と HiRDBCommandBuilder#QuoteSuffix プロパティの値で囲む文字列

Return

string：HiRDBCommandBuilder#QuotePrefix プロパティの値と HiRDBCommandBuilder#QuoteSuffix プロパティの値で囲んだ文字列

説明：指定された文字列を HiRDBCommandBuilder#QuotePrefix プロパティの値と HiRDBCommandBuilder#QuoteSuffix プロパティの値で囲って返します。

例外：HiRDBException

(f) RefreshSchema

void RefreshSchema ()

Return：void

説明：INSERT，UPDATE，又は DELETE ステートメントを生成するための，データベースのスキーマ情報を更新します。

void RefreshSchema (string)

引数

string sTableName：テーブル名

Return：void

説明：INSERT，UPDATE，又は DELETE ステートメントを生成するための，データベースのスキーマ情報を更新します。

例外：HiRDBException

(g) UnquoteIdentifier

string UnquoteIdentifier (string)

引数

string quotedIdentifier：HiRDBCommandBuilder#QuotePrefix プロパティの値と HiRDBCommandBuilder#QuoteSuffix プロパティの値で囲んだ文字列

Return

string：先頭の HiRDBCommandBuilder#QuotePrefix プロパティの値と終端の HiRDBCommandBuilder#QuoteSuffix プロパティの値を取った文字列

説明：指定された文字列から先頭の HiRDBCommandBuilder#QuotePrefix プロパティの値と終端の HiRDBCommandBuilder#QuoteSuffix プロパティの値を除いたものを返します。

例外：HiRDBException

15.5.3 HiRDBConnection

(1) コンストラクタ

(a) HiRDBConnection

void HiRDBConnection ()

説明：HiRDBConnection の新しいインスタンスを初期化します。

void HiRDBConnection (string)

引数

string ConnectionString：接続設定を格納している文字列（ConnectionString プロパティ）

説明：接続文字列を指定して，HiRDBConnection クラスの新しいインスタンスを初期化します。

(2) プロパティ

(a) ConnectionString

型：string

既定値：""

説明：データベースを開くために使用する文字列を取得又は設定します。

例外：HiRDBException

このプロパティには一つの string 型引数を指定する必要があります。指定する文字列は接続文字列と呼ばれるもので，これは ADO や ADO.NET の Connection で使用する接続文字列と同種のものです。指定できる文字列を次に示します。

文字列	内容
• datasource • dsn • env	使用するレジストリの設定。HiRDB クライアント環境変数登録ツールで作成した環境変数グループ名，又は環境変数グループファイル名（絶対パス）を指定します。
• uid • userid	DB 接続で使用する認可識別子。
• password • Pwd	DB 接続で使用するパスワード。
• PD*	クライアント環境定義の設定。

指定の規則を次に示します。このプロパティ以外にも指定方法があります。指定値の優先順位に関する詳細は，「接続情報の優先順位」を参照してください。

- 各指定値は，セミコロンで区切ります。
- 大文字小文字の区別はありません。区別させたい場合は，その部分を引用符で囲んでください。
- 空白やタブは無視されます（引用符で囲まれた部分は除きます）。
- 接続文字列と無関係な文字列を指定すると，例外が発生します。ただし，「Provider」については，例外は発生しないで無視されます。これは DataProvider 層で OleDb Data Provider との互換性を保つためです。

(b) ConnectionTimeout

型：int

既定値：

対応する ADO.NET のバージョンが 1.1 の場合は 15

対応する ADO.NET のバージョンが 2.0 の場合は 0

説明：試行を中断してエラーを生成する前に，接続の確立時に待機する時間を取得します（読み取り専用）。

(c) Database

型：string

既定値：""

説明：現在のデータベース，又は接続が開いてから使用するデータベースの名前を取得します（読み取り専用）。

(d) DataSource

型：string

既定値：""

説明：接続しているデータベースサーバの名前を取得します（読み取り専用）。

(e) ServerVersion

型：string

既定値：""

説明：接続中のサーバのバージョンを取得します（読み取り専用）。

String.Compare()を使用して比較できる正規化された形式で返します。バージョンの形式を次に示します。

XX.YY.ZZZZ

XX：メジャーバージョン

YY：マイナーバージョン

ZZZZ："0000"固定

(f) LifeTime

型：int

既定値：60

説明：実際に切断するまでの時間を取得又は設定します。指定値の単位は秒で，0～2147483 の間で指定してください。0 を指定した場合，又は 2147483 を超える値を指定した場合は即座に切断します。負の値を指定した場合は指定値を無視します。

例外：HiRDBException

(g) Pooling

型：bool

既定値：true

説明：プーリングを行うかどうかを取得又は設定します。

プーリングを行う場合は true，それ以外の場合は false となります。

例外：HiRDBException

(h)　State

型：System.Data.ConnectionState

既定値：ConnectionState.Closed

説明：現在の接続状態を取得します（読み取り専用）。

(i)　ConnectionRecover

型：bool

既定値：true

説明：接続プーリング機能使用時に，接続回復機能を使用するかどうかを取得又は設定します。

接続回復機能を使用する場合は true，使用しない場合は false となります。接続回復機能については，「HiRDB データプロバイダ for .NET Framework の接続プーリングにおける接続回復機能」を参照してください。

例外：HiRDBException

(3)　メソッド

(a)　BeginTransaction

BeginTransaction ()

Return

HiRDBTransaction：新しいトランザクションを表すオブジェクト

説明：データベースでトランザクションを開始します。

例外：HiRDBException

BeginTransaction (System.Data.IsolationLevel)

引数

System.Data.IsolationLevel：IsolationLevel 値の一つ

Return

HiRDBTransaction：新しいトランザクションを表すオブジェクト

説明：指定した IsolationLevel 値を使用して，データベースでトランザクションを開始します。

例外：HiRDBException

(b)　ChangeDatabase

void ChangeDatabase (string)

引数

string databaseName：変更するデータベースの名前

Return：void

説明：開いている HiRDBConnection オブジェクトの現在のデータベースを変更します。

例外：HiRDBException

(c) Clone

HiRDBConnection Clone()

Return：無条件に null を返します。

説明：無条件に null を返します。

(d) Close

void Close ()

Return：void

説明：データベースへの接続を閉じます。

(e) CreateCommand

Hitachi.HiRDB.HiRDBCommand CreateCommand ()

Return

HiRDBCommand：HiRDBCommand オブジェクト

説明：接続に関連づけられた HiRDBCommand オブジェクトを作成し，返します。

(f) Dispose

void Dispose()

Return：void

説明：HiRDBConnection が使用しているすべてのリソースを解放します。

(g) EnlistTransaction

void EnlistTransaction (Transaction)

引数

transaction：登録先となる既存の Transaction オブジェクト

Return：void

説明：指定されたトランザクションに登録します。

例外：HiRDBException

(h) GetSchema

DataTable GetSchema ()

Return：DataTable オブジェクト

説明：スキーマ情報を返します。

DataTable GetSchema (string)

引数

collectionName：返すスキーマの名前

Return：DataTable オブジェクト

説明：スキーマ情報を返します。

DataTable GetSchema (string, string[])

引数

collectionName：返すスキーマの名前

restrictionValues：要求したスキーマの制限値

Return：DataTable オブジェクト

説明：スキーマ情報を返します。

(i) Open

void Open ()

Return：void

説明：HiRDBConnection オブジェクトの ConnectionString プロパティで指定されている設定で，データベース接続を開きます。

例外：HiRDBException

15.5.4　HiRDBDataAdapter

(1)　コンストラクタ

(a)　HiRDBDataAdapter

void HiRDBDataAdapter ()

説明：HiRDBDataAdapter クラスの新しいインスタンスを初期化します。

void HiRDBDataAdapter (Hitachi.HiRDB.HiRDBCommand)

引数

HiRDBCommand selectCommand：SQL SELECT ステートメントを表す HiRDBCommand オブジェクト（SelectCommand プロパティ）

説明：指定した HiRDBCommand を使用して，HiRDBDataAdapter クラスの新しいインスタンスを初期化します。

void HiRDBDataAdapter (string, Hitachi.HiRDB.HiRDBConnection)

引数

string selectCommandText：SQL SELECT ステートメント

HiRDBConnection selectConnection：接続を表す HiRDBConnection オブジェクト

説明：SQL SELECT ステートメントを指定した HiRDBConnection を使用して，HiRDBCommand を作成します（SelectCommand プロパティ）。HiRDBDataAdapter クラスの新しいインスタンスを初期化します。

void HiRDBDataAdapter (string, string)

引数

string selectCommandText：SQL SELECT ステートメント

string selectConnectionString：接続文字列

説明：接続文字列を使用して，HiRDBConnection を作成します。SQL SELECT ステートメントと，作成した HiRDBConnection を使用して，HiRDBCommand を作成します（SelectCommand プロパティ）。HiRDBDataAdapter クラスの新しいインスタンスを初期化します。

(2)　プロパティ

(a)　DeleteCommand

型：HiRDBCommand

既定値：null

説明：データセットからレコードを削除する SQL ステートメントを取得又は設定します。

(b) InsertCommand

型：HiRDBCommand

既定値：null

説明：データベースに新しいレコードを挿入する SQL ステートメントを取得又は設定します。

(c) SelectCommand

型：HiRDBCommand

既定値：null

説明：データベース内のレコードを選択する SQL ステートメントを取得又は設定します。

(d) UpDateCommand

型：HiRDBCommand

既定値：null

説明：データベース内のレコードを更新する SQL ステートメントを取得又は設定します。

(3) イベント

(a) RowUpdated

型：HiRDBRowUpdatedEventHandler

説明：更新処理時に，データソースに対してコマンドを実行した後に発生するイベントです。

(b) RowUpdating

型：HiRDBRowUpdatingEventHandler

説明：更新処理時に，データソースに対してコマンドを実行する前に発生するイベントです。

15.5.5 HiRDBDataReader

(1) コンストラクタ

HiRDBDataReader

説明：HiRDBDataReader を作成するには，コンストラクタを直接使用しないで，HiRDBCommand オブジェクトの ExecuteReader メソッドを呼び出す必要があります。

(2) プロパティ

(a) Depth

型：int

既定値：0

説明：現在の行の入れ子の深さを示す値を取得します。

(b) FieldCount

型：int

説明：現在の行の列数を取得します。

(c) HasRows

型：bool

既定値：false

説明：該当する HiRDBDataReader に 1 行以上の行が格納されているかどうかを示す値を取得します。1 行以上の行が HiRDBDataReader に含まれている場合は true，それ以外の場合は false となります。

(d) IsClosed

型：bool

既定値：false

説明：データリーダが閉じているかどうかを示す値を取得します。閉じている場合は true，それ以外の場合は false となります。

(e) Item

Item[string]

型：Object this[string name]

説明：HiRDBDataReader オブジェクトを配列のように扱い，データを取得します。

Item[int]

型：Object this[int ordinal]

説明：HiRDBDataReader オブジェクトを配列のように扱い，データを取得します。

Item[int, int]

型：Object this[int colIdx, int elmIdx]

説明：HiRDBDataReader オブジェクトを配列のように扱い，データを取得します。

(f)　RecordsAffected

型：int

既定値：0

説明：SQL ステートメントの実行によって変更，挿入，又は削除された行の数を取得します。

(g)　VisibleFieldCount

型：int

説明：HiRDBDataReader の非表示ではない列数を取得します。

(3)　メソッド

(a)　Close

void Close ()

Return：void

説明：HiRDBDataReader オブジェクトを閉じます。

(b)　Dispose

void Dispose()

Return：void

説明：HiRDBDataReader が使用しているすべてのリソースを解放します。

(c)　GetBoolean

bool GetBoolean (int)

引数

　int i：列の 0 から始まる序数

Return

　bool：列の値

説明：指定した列の値をブール値として取得します。

例外：HiRDBException

(d)　GetByte

byte GetByte (int)

引数

int i：列の 0 から始まる序数

Return

byte：指定した列の 8 ビット符号なし整数値

説明：指定した列の 8 ビット符号なし整数値を取得します。

例外：HiRDBException

(e) GetBytes

long GetBytes (int, long, byte[ ], int,int)

引数

int i：列の 0 から始まる序数

long fieldOffset：読み取り操作を開始する行内のインデクス

byte[] buffer：バイトのストリームを読み込むバッファ

int bufferoffset：読み取り操作を開始する buffer のインデクス

int length：読み込むバイト数

Return

long：実際に読み込んだバイトの数

説明：指定したバッファオフセットを開始位置として，指定した列オフセットからバッファに，バイトのストリームを配列として読み込みます。

例外：HiRDBException

(f) GetChar

char GetChar (int)

引数

int i：列の 0 から始まる序数

Return

char：指定した列の文字値

説明：指定した列の文字値を取得します。

例外：HiRDBException

(g) GetChars

long GetChars (int, long,char[ ], int, int)

引数

int i：列の 0 から始まる序数

long fieldOffset：読み取り操作を開始する行内のインデクス

char[] buffer：バイトのストリームを読み込むバッファ

int bufferoffset：読み取り操作を開始する buffer のインデクス

int length：読み込むバイト数

Return

long：実際に読み込んだ文字数

説明：指定したバッファオフセットを開始位置として，指定した列オフセットからバッファに，文字のストリームを配列として読み込みます。

例外：HiRDBException

(h) GetData

GetData (int)

引数

int i：列の 0 から始まる序数

Return：現在サポートされていません。

説明：このメンバは，.NET Framework インフラストラクチャのサポートを目的としています。独自に作成したコード内で直接使用できません。

(i) GetDataTypeName

string GetDataTypeName (int)

引数

int i：検索するフィールドのインデクス

Return

string：指定したフィールドのデータ型情報

説明：指定したフィールドのデータ型情報を取得します。

例外：HiRDBException

(j) GetDateTime

System.DateTime GetDateTime (int)

引数

int i：検索するフィールドのインデクス

Return

System.DateTime：指定したフィールドの日時のデータ値

説明：指定したフィールドの日時のデータ値を取得します。

例外：HiRDBException

(k) GetDecimal

decimal GetDecimal (int)

引数

int i：検索するフィールドのインデクス

Return

decimal：指定したフィールドの固定位置数値

説明：指定したフィールドの固定位置数値を取得します。

例外：HiRDBException

(l) GetDouble

double GetDouble (int)

引数

int i：検索するフィールドのインデクス

Return

double：指定したフィールドの倍精度浮動小数点数

説明：指定したフィールドの倍精度浮動小数点数を取得します。

例外：HiRDBException

(m) GetEnumerator

System.Collections.IEnumerator GetEnumerator ()

Return

System.Collections.IEnumerator：コレクションを反復処理するために使用できる IEnumerator

説明：コレクションを反復処理できる列挙子を返します。

(n) GetFieldArrayCount

int GetFieldArrayCount (int)

引数

int i：検索するフィールドのインデクス

Return

int：フィールドの配列の大きさ

説明：フィールドの配列の大きさを取得します。

例外：HiRDBException

(o) GetFieldType

System.Type GetFieldType (int)

引数

int i：検索するフィールドのインデクス

Return

System.Type：GetValue から返される object の型に対応する Type 情報

説明：GetValue から返される，Object の型に対応する Type 情報を取得します。

例外：HiRDBException

(p) GetFloat

float GetFloat (int)

引数

int i：検索するフィールドのインデクス

Return

float：指定したフィールドの単精度浮動小数点数

説明：指定したフィールドの単精度浮動小数点数を取得します。

例外：HiRDBException

(q) GetGuid

System.Guid GetGuid (int)

引数

int i：検索するフィールドのインデクス

Return

System.Guid：指定したフィールドの GUID 値

説明：指定したフィールドの GUID 値を返します。

(r)　GetInt16

short GetInt16 (int)

引数

　int i：検索するフィールドのインデクス

Return

　short：指定したフィールドの 16 ビット符号付き整数値

説明：指定したフィールドの 16 ビット符号付き整数値を取得します。

例外：HiRDBException

(s)　GetInt32

int GetInt32 (int)

引数

　int i：検索するフィールドのインデクス

Return

　int：指定したフィールドの 32 ビット符号付き整数値

説明：指定したフィールドの 32 ビット符号付き整数値を取得します。

例外：HiRDBException

(t)　GetInt64

long GetInt64 (int)

引数

　int i：検索するフィールドのインデクス

Return

　long：指定したフィールドの 64 ビット符号付き整数値

説明：指定したフィールドの 64 ビット符号付き整数値を取得します。

例外：HiRDBException

(u)　GetName

string GetName (int)

引数

　int i：検索するフィールドのインデクス

Return

string：フィールドの名前（返される値がない場合は空の文字列（""））

説明：検索するフィールドの名前を取得します。

例外：HiRDBException

(v) GetOrdinal

int GetOrdinal (string)

引数

string name：検索するフィールドの名前

Return

int：指定したフィールドのインデクス

説明：指定したフィールドのインデクスを返します。

例外：HiRDBException

(w) GetProviderSpecificFieldType

Object GetProviderSpecificFieldType (int)

引数

int ordinal：列の 0 から始まる序数

Return

Object：指定した列のデータ型

説明：指定した列のデータ型を取得します。HiRDB データプロバイダ for .NET Framework は独自のデータ型をサポートしていないため，.NET Framework の共通言語ランタイムに用意されているデータ型の Type オブジェクトを返します。HiRDBDataReader#GetFieldType メソッドと同じ動作をします。

例外：HiRDBException

(x) GetProviderSpecificValue

Object GetProviderSpecificValue (int)

引数

int ordinal：列の 0 から始まる序数

Return

Object：指定された列の値を持つオブジェクト

説明：指定した列の値を Object のインスタンスとして取得します。

例外：HiRDBException

(y) GetProviderSpecificValues

int GetProviderSpecificValues (Object[])

引数

values：属性列のコピー先の Object 配列（現在行の各列データを格納する Object 型の配列）

Return

int：配列の Object インスタンス数

説明：現在の行のコレクション内にあるすべての属性列を取得します。

例外：HiRDBException

(z) GetSchemaTable

System.Data.DataTable GetSchemaTable ()

Return

System.Data.DataTable：列メタデータを説明する DataTable

説明：HiRDBDataReader の列メタデータを説明する DataTable を返します。

例外：HiRDBException

(aa) GetString

string GetString (int)

引数

int i：検索するフィールドのインデクス

Return

string：指定したフィールドの文字列値

説明：指定したフィールドの文字列値を取得します。

例外：HiRDBException

(ab) GetValue

object GetValue (int)

引数

int i：検索するフィールドのインデクス

Return

object：フィールドの値が返されたときにその値を格納する Object

説明：指定したフィールドの値を返します。

例外：HiRDBException

object GetValue (int, int)

引数

int i：検索するフィールドのインデクス

int j：検索するフィールドのインデクス

Return

object：フィールドの値が返されたときにその値を格納する Object

説明：指定したフィールドの値を返します（配列用）。

例外：HiRDBException

(ac) GetValues

int GetValues (object[ ])

引数

object values：属性フィールドのコピー先である Object の配列

Return

int：配列の Object のインスタンス数

説明：現在のレコードコレクション内のすべての属性フィールドを取得します。

(ad) IsDBNull

bool IsDBNull (int)

引数

int i：検索するフィールドのインデクス

Return

bool：指定したフィールドが NULL 値の場合は true，それ以外の場合は false となります。繰返し列の場合は，列全体が NULL 値の場合は true，それ以外の場合は false となります。

説明：指定したフィールドが NULL 値かどうかを示す値を返します。繰返し列の場合は，列全体が NULL 値かどうかを示す値を返します。

繰返し列の NULL 値判定

繰返し列の NULL 値判定は，次の手順で行います。

1. HiRDBDataReader.IsDBNull(int)メソッドで，繰返し列全体が NULL 値かどうかを判定します。

2. 1.の戻り値が true の場合は，列全体が NULL 値であるため，各要素の値はありません。false の場合は，HiRDBDataReader.GetFieldArrayCount()メソッドで現在要素数を取得します。
3. 2.で取得した要素数分 HiRDBDataReader.IsDBNull(int,int)メソッドを実行して，各要素が NULL 値かどうかを判定します。そこで true を返却された要素が NULL 値となります。

例外：HiRDBException

(ae)　NextResult

bool NextResult ()

Return

bool：もっと多くの行がある場合は true，それ以外の場合は false となります。

説明：バッチ SQL ステートメントの結果を読み込むときに，データリーダを次の結果に進めます。

例外：HiRDBException

(af)　Read

bool Read ()

Return

bool：もっと多くの行がある場合は true，それ以外の場合は false となります。

説明：HiRDBDataReader を次のレコードに進めます。

例外：HiRDBException

(ag)　IsDBNull

bool IsDBNull(int ,int)

引数

int colIdx：列の 0 から始まる序数

int elmIdx：繰返し列の要素番号（0 から始まる序数）

Return

bool：指定した繰返し列の，指定した要素が NULL 値の場合 true，それ以外の場合は false となります。

説明：指定した繰返し列の，指定した要素が NULL 値かどうかを示す値を返します。

例外：HiRDBException

15.5.6　HiRDBException

(1)　プロパティ

(a)　ErrorCode

型：int

既定値：0

説明：エラーのコード部分を int として取得します。

(b)　Message

型：String

既定値：""

説明：エラーの完全なテキストを取得します。

15.5.7　HiRDBParameter

(1)　コンストラクタ

(a)　HiRDBParameter

void HiRDBParameter ()

説明：HiRDBParameter クラスの新しいインスタンスを初期化します。

void HiRDBParameter (string, object)

引数

string name：割り当てるパラメタの名前（ParameterName プロパティ）

object value：新しい HiRDBParameter オブジェクトの値（Value プロパティ）

説明：パラメタ名と HiRDBParameter オブジェクトの値を指定して，HiRDBParameter クラスの新しいインスタンスを初期化します。

void HiRDBParameter (string, Hitachi.HiRDB.HiRDBType)

引数

string name：割り当てるパラメタの名前（ParameterName プロパティ）

Hitachi.HiRDB.HiRDBType dataType：HiRDBType 値の一つ（HiRDBType プロパティ）

説明：パラメタ名とデータ型を指定して，HiRDBParameter クラスの新しいインスタンスを初期化します。

void HiRDBParameter (string, Hitachi.HiRDB.HiRDBType, int)

引数

string name：割り当てるパラメタの名前（ParameterName プロパティ）

Hitachi.HiRDB.HiRDBType dataType：HiRDBType 値の一つ（HiRDBType プロパティ）

int size：パラメタの定義長（Size プロパティ）

説明：パラメタ名，データ型，及び長さを使用して，HiRDBParameter クラスの新しいインスタンスを初期化します。

void HiRDBParameter (string, Hitachi.HiRDB.HiRDBType, int, string)

引数

string name：割り当てるパラメタの名前（ParameterName プロパティ）

Hitachi.HiRDB.HiRDBType dataType：HiRDBType 値の一つ（HiRDBType プロパティ）

int size：パラメタの定義長（Size プロパティ）

string srcColumn：ソース列の名前（SourceColumn プロパティ）

説明：パラメタ名，データ型，長さ，及びソース列名を指定して，HiRDBParameter クラスの新しいインスタンスを初期化します。

void HiRDBParameter (string, Hitachi.HiRDB.HiRDBType, int, System.Data.ParameterDirection, byte, byte, string, System.Data.DataRowVersion, object)

引数

string parameterName：パラメタの名前（ParameterName プロパティ）

Hitachi.HiRDB.HiRDBType dataType：HiRDBType 値の一つ（HiRDBType プロパティ）

int size：パラメタの定義長（Size プロパティ）

System.Data.ParameterDirection direction：ParameterDirection 値の一つ（Direction プロパティ）

byte precision：Value を解決するための，小数点の左右の合計けた数（Precision プロパティ）

byte scale：Value を解決するための，小数部のけた数（Scale プロパティ）

string srcColumn：ソース列の名前（SourceColumn プロパティ）

System.Data.DataRowVersion srcVersion：DataRowVersion 値の一つ（SourceVersion プロパティ）

object value：HiRDBParameter の値である Object（Value プロパティ）

説明：パラメタ名，データ型，長さ，ソース列名，パラメタの方向，数値の精度，及びそのほかのプロパティを指定して，HiRDBParameter クラスの新しいインスタンスを初期化します。

(2) プロパティ

(a) DbType

型：System.Data.DbType

既定値：DbType.String

説明：パラメタの DbType を取得又は設定します。設定時には，表「DbType プロパティ設定時に自動設定される HiRDBType プロパティの値」に従って，該当するデータタイプを HiRDBType プロパティに設定します。Value プロパティに値を指定した場合，このプロパティの指定を省略できます。このプロパティの指定を省略した場合，HiRDB データプロバイダ for .NET Framework が DbType プロパティの値を仮定します。DbType プロパティの仮定値の規則については，「パラメタのデータ長及びデータタイプの仮定」を参照してください。

(b) Direction

型：System.Data.ParameterDirection

既定値：ParameterDirection.Input

説明：パラメタが入力専用，出力専用，双方向，又はストアドプロシジャの戻り値パラメタのどれであるかを示す値を取得又は設定します。

(c) HiRDBType

型：Hitachi.HiRDB.HiRDBType

既定値：HiRDBType.MVarChar

説明：HiRDB でのデータタイプを示す列挙体を取得，又は設定します。設定時には，表「HiRDBType プロパティ設定時に自動設定される DbType プロパティの値」に従って，該当するデータタイプを DbType プロパティに設定します。Value プロパティに値を指定した場合，このプロパティの指定を省略できます。このプロパティの指定を省略した場合，HiRDB データプロバイダ for .NET Framework が HiRDBType プロパティの値を仮定します。HiRDBType プロパティの仮定値の規則については，「パラメタのデータ長及びデータタイプの仮定」を参照してください。

［HiRDBType 列挙体］

Integer，SmallInt，Decimal，Float，SmallFlt，Char，VarChar，NChar，NVarChar， MChar，MVarChar，Date，Time，TimeStamp，IntervalYearToDay，IntervalHourToSecond，Blob，Binary

(d) IsNullable

型：bool

既定値：true（固定）

説明：パラメタが null 値を受け付けるかどうかを示す値を取得します（読み取り専用）。null 値を受け付ける場合は true，それ以外の場合は false となります。

(e) ParameterName

型：string

既定値：""

説明：HiRDBParameter の名前を取得又は設定します。

(f)　Precision

型：byte

既定値：0

説明：DECIMAL 型パラメタの定義長の有効けた数（小数部けた数を含む）を取得又は設定します。

HiRDBType プロパティの値が HiRDBType.Decimal の場合：

パラメタに対応する HiRDB サーバでの DECIMAL 型列の精度を設定してください。

HiRDBType プロパティの値が HiRDBType.Decimal 以外の場合：

設定値は無視されます。

Value プロパティに値を指定した場合，このプロパティの指定を省略できます。このプロパティの指定を省略した場合，HiRDB データプロバイダ for .NET Framework が Precision プロパティの値を仮定します。Precision プロパティの仮定値の規則については，「パラメタのデータ長及びデータタイプの仮定」を参照してください。

(g)　Repetition

型：short

既定値：1

説明：HiRDB での配列構造を取得又は設定します。1 の場合は対象列を非繰返し列，2 以上の場合は繰返し列の最大要素数として扱います。

(h)　Scale

型：byte

既定値：0

説明：DECIMAL 型パラメタの定義長の小数部けた数を取得又は設定します。

HiRDBType プロパティの値が HiRDBType.Decimal の場合：

パラメタに対応する HiRDB サーバでの DECIMAL 型列の位どりを設定してください。

HiRDBType プロパティの値が HiRDBType.Decimal 以外の場合：

設定値は無視されます。

Value プロパティに値を指定した場合，このプロパティの指定を省略できます。このプロパティの指定を省略した場合，HiRDB データプロバイダ for .NET Framework が Scale プロパティの値を仮定します。

Scale プロパティの仮定値の規則については，「パラメタのデータ長及びデータタイプの仮定」を参照してください。

(i) Size

型：int

既定値：0

説明：パラメタの定義長を取得又は設定します。固定長（数値型や日時型など）の場合は 0 を，可変長（文字列型など）の場合はテーブルに格納するバイト数又は列の最大長を設定してください。

TIMESTAMP（DateTime）の場合は，小数部のけた数となります。

注

Size プロパティの設定よりも長い文字列を入力した場合は，Size プロパティに設定した長さまでの文字列が格納されます。例外は発生しません。

Value プロパティに値を指定した場合，このプロパティの指定を省略できます。このプロパティの指定を省略した場合，HiRDB データプロバイダ for .NET Framework が Size プロパティの値を仮定します。Size プロパティの仮定値の規則については，「パラメタのデータ長及びデータタイプの仮定」を参照してください。

(j) SourceColumn

型：string

既定値：""

説明：DataSet に割り当てられ，Value を読み込むとき，又は戻すときに使用されるソース列の名前を取得又は設定します。

(k) SourceColumnNullMapping

型：bool

既定値：true

説明：パラメタに対応する DataTable オブジェクトの列が NULL 値を許すかどうかを示す値を取得又は設定します。

(l) SourceVersion

型：System.Data.DataRowVersion

既定値：DataRowVersion.Default

説明：Value の読み込みに使用する DataRowVersion を取得又は設定します。

(m)　Value

型：object

既定値：null

説明：パラメタの値を取得又は設定します。入力，又は入出力用の HiRDBParameter オブジェクトに Value プロパティを指定した場合，次のプロパティの指定を省略できます。

- DbType プロパティ
- HiRDBType プロパティ
- Size プロパティ
- Precision プロパティ
- Scale プロパティ

これらのプロパティの指定を省略した場合，各プロパティの値は HiRDB データプロバイダ for .NET Framework が仮定します。ただし，これらのプロパティを明示的に指定した場合は，HiRDB データプロバイダ for .NET Framework は値を仮定しません。そのため，Value プロパティに指定する値に合わせて各プロパティを設定する必要があります。詳細は，「パラメタのデータ長及びデータタイプの仮定」を参照してください。

(3)　メソッド

(a)　Clone

object Clone ()

Return

object：このインスタンスのコピーである新しいオブジェクト

説明：現在のインスタンスのコピーである新しいオブジェクトを作成します。

(b)　ResetDbType

void ResetDbType ()

Return：void

説明：Value プロパティが指定されていない場合，DbType プロパティの値を初期値に戻します。Value プロパティに値が指定されている場合，HiRDB データプロバイダ for .NET Framework が，Value プロパティの値から仮定する DbType の値を設定します。データタイプの仮定値の規則については，「パラメタのデータ長及びデータタイプの仮定」を参照してください。DbType プロパティの値と HiRDBType プロパティの値は 1 対 1 で対応しているため，このメソッドを実行すると HiRDB データプロバイダ for .NET Framework が DbType プロパティの値に対応する値を HiRDBType に設定します。

15.5.8　HiRDBParameterCollection

(1)　コンストラクタ

(a)　HiRDBParameterCollection

void HiRDBParameterCollection ()

説明：HiRDBParameterCollection クラスの新規インスタンスを初期化します。

(2)　プロパティ

(a)　Count

型：int

既定値：0

説明：HiRDBParameterCollection に格納されている HiRDBParameter オブジェクトの数を取得します（読み取り専用）。

(b)　IsFixedSize

型：bool

既定値：false

説明：HiRDBParameterCollection が固定サイズかどうかを示す値を取得します（読み取り専用）。常に false となります。

(c)　IsReadOnly

型：bool

既定値：false

説明：HiRDBParameterCollection が読み取り専用かどうかを示す値を取得します（読み取り専用）。常に false となります。

(d)　IsSynchronized

型：bool

既定値：false

説明：HiRDBParameterCollection へのアクセスが同期されている（スレッドセーフである）かどうかを示す値を取得します（読み取り専用）。常に false となります。

(e) Item

Item[int]

型：HiRDBParameter this[int index]

説明：指定したインデクスの HiRDBParameter オブジェクトを取得します。又は，指定したインデクスに HiRDBParameter オブジェクトを設定します。

Item[string]

型：HiRDBParameter this[string parameterName]

説明：引数で指定したパラメタ名を持つ HiRDBParameter オブジェクトを取得します。又は，引数で指定したパラメタ名を持つ HiRDBParameter オブジェクトのインデクスに新しい HiRDBParameter オブジェクトを設定します。

(f) SyncRoot

型：object

既定値：null

説明：HiRDBParameterCollection へのアクセスを同期するために使用できるオブジェクトを取得します（読み取り専用）。

(3) メソッド

(a) Add

int Add (object)

引数

object value：HiRDBParameterCollection に追加する HiRDBParameter オブジェクト

Return

int：追加した HiRDBParameter オブジェクトのコレクション内での 0 から始まるインデクス

説明：HiRDBParameterCollection に項目を追加します。

int Add (Hitachi.HiRDB.HiRDBParameter)

引数

HiRDBParameter value：HiRDBParameterCollection に追加する HiRDBParameter

Return

int：追加した HiRDBParameter の 0 から始まるインデクス

説明：HiRDBParameterCollection に項目を追加します。

int Add (string, object)

引数

string parameterName：パラメタの名前

object parameterValue：パラメタの値

Return

int：追加した HiRDBParameter の 0 から始まるインデクス

説明：パラメタの名前と値を指定して，HiRDBParameterCollection に項目を追加します。

int Add (string, HiRDBType)

引数

string parameterName：パラメタの名前

HiRDBType dataType：HiRDBType 値の一つ

Return

int：追加した HiRDBParameter の 0 から始まるインデクス

説明：パラメタの名前とデータ型を指定して，HiRDBParameterCollection に項目を追加します。

int Add (string, HiRDBType, int)

引数

string parameterName：パラメタの名前

HiRDBType dataType：HiRDBType 値の一つ

int size：パラメタのサイズ

Return

int：追加した HiRDBParameter の 0 から始まるインデクス

説明：パラメタの名前，データ型，及びサイズを指定して，HiRDBParameterCollection に項目を追加します。

int Add (string, HiRDBType, int, string)

引数

string parameterName：パラメタの名前

HiRDBType dataType：HiRDBType 値の一つ

int size：パラメタのサイズ

string srcColumn：ソース列の名前

Return

int：追加した HiRDBParameter の 0 から始まるインデクス

説明：パラメタの名前，データ型，サイズ，及びソース列の名前を指定して，HiRDBParameterCollection に項目を追加します。

(b) Add2

Hitachi.HiRDB.HiRDBParameter Add2 (Hitachi.HiRDB.HiRDBParameter)

引数

HiRDBParameter value：HiRDBParameterCollection に追加する HiRDBParameter

Return

HiRDBParameter：追加した HiRDBParameter オブジェクト

説明：HiRDBParameterCollection に項目を追加します。

Hitachi.HiRDB.HiRDBParameter Add 2(string, object)

引数

string parameterName：パラメタの名前

object parameterValue：パラメタの値

Return

HiRDBParameter：追加した HiRDBParameter オブジェクト

説明：パラメタの名前と値を指定して，HiRDBParameterCollection に項目を追加します。

Hitachi.HiRDB.HiRDBParameter Add2 (string, HiRDBType)

引数

string parameterName：パラメタの名前

HiRDBType dataType：HiRDBType 値の一つ

Return

HiRDBParameter：追加した HiRDBParameter オブジェクト

説明：パラメタの名前とデータ型を指定して，HiRDBParameterCollection に項目を追加します。

Hitachi.HiRDB.HiRDBParameter Add2 (string, HiRDBType, int)

引数

string parameterName：パラメタの名前

HiRDBType dataType：HiRDBType 値の一つ

int size：パラメタのサイズ

Return

HiRDBParameter：追加した HiRDBParameter オブジェクト

説明：パラメタの名前，データ型，及びサイズを指定して，HiRDBParameterCollection に項目を追加します。

Hitachi.HiRDB.HiRDBParameter Add 2(string, HiRDBType, int, string)

引数

string parameterName：パラメタの名前

HiRDBType dataType：HiRDBType 値の一つ

int size：パラメタのサイズ

string srcColumn：ソース列の名前

Return

HiRDBParameter：追加した HiRDBParameter オブジェクト

説明：パラメタの名前，データ型，サイズ，及びソース列の名前を指定して，HiRDBParameterCollection に項目を追加します。

(c) AddRange

void AddRange(Array)

引数

values：HiRDBParameterCollection に追加する HiRDBParameter オブジェクトの配列

Return：void

説明：指定された HiRDBParameter オブジェクトの配列を HiRDBParameterCollection に追加します。

例外：HiRDBException

void AddRange(HiRDBParameter[])

引数

value：HiRDBParameterCollection に追加する HiRDBParameter オブジェクトの配列

Return：void

説明：指定された HiRDBParameter オブジェクトの配列を HiRDBParameterCollection に追加します。

例外：HiRDBException

(d) Clear

void Clear ()

Return：void

説明：HiRDBParameterCollection からすべての項目を削除します。

(e) Contains

bool Contains (object)

引数

object value：HiRDBParameterCollection 内で検索される Object

Return

bool：Object が HiRDBParameterCollection にある場合は true，それ以外の場合は false となります。

説明：HiRDBParameter がコレクション内にあるかどうかを示す値を取得します。

例外：HiRDBException

bool Contains (HiRDBParameter)

引数

HiRDBParameter value：HiRDBParameterCollection 内で検索する HiRDBParameter オブジェクト

Return

bool：Object が HiRDBParameterCollection にある場合は true，それ以外の場合は false となります。

説明：HiRDBParameter がコレクション内にあるかどうかを示す値を取得します。

例外：HiRDBException

bool Contains (string)

引数

string parameterName：パラメタの名前

Return

bool：コレクションにパラメタが格納されている場合は true，それ以外の場合は false となります。

説明：HiRDBParameter がコレクション内にあるかどうかを示す値を取得します。

例外：HiRDBException

(f) CopyTo

void CopyTo (System.Array, int)

引数

System.Array array：HiRDBParameterCollection から要素がコピーされる 1 次元の Array

int index：value を挿入する位置の，0 から始まるインデクス番号

Return：void

説明：Array の特定のインデクスを開始位置として，HiRDBParameterCollection の要素を Array にコピーします。

(g) GetEnumerator

System.Collections.IEnumerator GetEnumerator ()

Return

System.Collections.Ienumerator：コレクションを反復処理するために使用できる IEnumerator

説明：コレクションを反復処理できる列挙子を返します。

(h) IndexOf：overload

int IndexOf (string)

引数

string parameterName：パラメタの名前

Return

int：コレクション内の HiRDBParameterCollection の 0 から始まる位置

説明：コレクション内の HiRDBParameter の位置を取得します。

例外：HiRDBException

int IndexOf (object)

引数

object value：HiRDBParameterCollection 内で検索される Object

Return

int：リストにある場合は value のインデクス，それ以外の場合は-1 となります。

説明：コレクション内の HiRDBParameter の位置を取得します。

(i) Insert

void Insert(int, object)

引数

int index：value を挿入する位置の，0 から始まるインデクス番号

object value：HiRDBParameterCollection に追加する HiRDBParameter

Return：void

説明：HiRDBParameterCollection 内の指定した位置に項目を挿入します。

例外：HiRDBException

void Insert (int, Hitachi.HiRDB.HiRDBParameter)

引数

int index：value を挿入する位置の，0 から始まるインデクス番号

HiRDBParameter value：HiRDBParameterCollection に追加する HiRDBParameter

Return：void

説明：HiRDBParameterCollection 内の指定した位置に項目を挿入します。

例外：HiRDBException

(j) Remove

void Remove (object)

引数

object value：HiRDBParameterCollection から削除する HiRDBParameter

Return：void

説明：HiRDBParameterCollection 内にある特定のオブジェクトのうち，最初に出現するオブジェクトを削除します。

(k) RemoveAt

void RemoveAt (string)

引数

string parameterName：パラメタの名前

Return：void

説明：HiRDBParameter をコレクションから削除します。

例外：HiRDBException

void RemoveAt (int)

引数

int index：削除する項目の 0 から始まるインデクス

Return：void

説明：HiRDBParameter をコレクションから削除します。

15.5.9 HiRDBProviderFactory

(1) コンストラクタ

(a) HiRDBProviderFactory

HiRDBProviderFactory()

説明：HiRDBProviderFactory クラスの新しいインスタンスを初期化します。

(2) フィールド

(a) Instance

型：HiRDBProviderFactory

説明：HiRDBProviderFactory のインスタンスを保持します（読み取り専用）。

(3) プロパティ

(a) CanCreateDataSourceEnumerator

型：bool

説明：DbDataSourceEnumerator クラスから派生したクラスをサポートするかどうかを示します（読み取り専用）。サポートする場合は true，それ以外は false となります。

(4) メソッド

(a) CreateCommand

DbCommand CreateCommand ()

Return：HiRDBCommand オブジェクト

説明：HiRDBCommand オブジェクトを作成して返します。

(b) CreateCommandBuilder

DbCommandBuilder CreateCommandBuilder ()

Return：HiRDBCommandBuilder オブジェクト

説明：HiRDBCommandBuilder オブジェクトを作成して返します。

(c) CreateConnection

DbConnection CreateConnection ()

Return：HiRDBConnection オブジェクト

説明：HiRDBConnection オブジェクトを作成して返します。

(d) CreateConnectionStringBuilder

DbConnectionStringBuilder CreateConnectionStringBuilder ()

Return：DbConnectionStringBuilder オブジェクト

説明：DbConnectionStringBuilder オブジェクトを作成して返します。

(e) CreateDataAdapter

DbDataAdapter CreateDataAdapter ()

Return：HiRDBDataAdapter オブジェクト

説明：HiRDBDataAdapter オブジェクトを作成して返します。

(f)　CreateDataSourceEnumerator

DbDataSourceEnumerator CreateDataSourceEnumerator ()

Return：必ず例外となるため，戻り値はありません。

説明：HiRDB の列挙体を提供しないため未サポートです。無条件で System.NotSupportedException を返します。

例外：System.NotSupportedException

(g)　CreateParameter

DbParameter CreateParameter ()

Return：HiRDBParameter オブジェクト

説明：HiRDBParameter オブジェクトを作成して返します。

15.5.10　HiRDBRowUpdatedEventArgs

(1)　コンストラクタ

(a)　HiRDBRowUpdatedEventArgs

void HiRDBRowUpdatedEventArgs (System.Data.DataRow, System.Data.IdbCommand, System.Data.StatementType, System.Data.Common.DataTableMapping)

引数

System.Data.DataRow dataRow：Update を通じて送信された DataRow

System.Data.IDbCommand command：Update の呼び出し時に実行された IDbCommand

System.Data.StatementType statementType：実行された SQL ステートメントの種類

System.Data.Common.DataTableMapping tableMapping：Update を通じて送信された DataTableMapping

説明：HiRDBRowUpdatedEventArgs クラスの新しいインスタンスを初期化します。

(2)　プロパティ

(a)　Command

型：HiRDBCommand

既定値：null

説明：Update の呼び出し時に実行される HiRDBCommand を取得します（読み取り専用）。

15.5.11　HiRDBRowUpdatingEventArgs

(1)　コンストラクタ

(a)　HiRDBRowUpdatingEventArgs

void HiRDBRowUpdatingEventArgs (System.Data.DataRow, System.Data.IDbCommand, System.Data.StatementType, System.Data.Common.DataTableMapping)

引数

System.Data.DataRow dataRow：Update を実行する DataRow

System.Data.IDbCommand command：Update の呼び出し時に実行する IDbCommand

System.Data.StatementType statementType：実行する SQL ステートメントの種類

System.Data.Common.DataTableMapping tableMapping：Update を通じて送信する DataTableMapping

説明：HiRDBRowUpdatingEventArgs クラスの新しいインスタンスを初期化します。

(2)　プロパティ

(a)　Command

型：HiRDBCommand

既定値：null

説明：Update 処理中に実行する HiRDBCommand を取得又は設定します。

15.5.12　HiRDBTransaction

(1)　プロパティ

(a)　Connection

型：HiRDBConnection

既定値：null

説明：トランザクションを関連づける HiRDBConnection オブジェクトを指定します（読み取り専用）。

(b) IsCompleted

型：bool

既定値：false

説明：トランザクションが完了しているかを取得します（読み取り専用）。完了している場合は true，それ以外の場合は false となります。

(c) IsolationLevel

型：System.Data.IsolationLevel

既定値：

対応する ADO.NET のバージョンが 1.1 の場合は IsolationLevel.ReadCommitted

対応する ADO.NET のバージョンが 2.0 の場合は IsolationLevel.RepeatableRead

説明：このトランザクションの IsolationLevel を指定します（読み取り専用）。

(2) メソッド

(a) Commit

void Commit ()

Return：void

説明：データベーストランザクションをコミットします。

例外：HiRDBException

(b) Dispose

void Dispose()

Return：void

説明：HiRDBTransaction が使用しているすべてのリソースを解放します。

(c) Rollback

void Rollback ()

Return：void

説明：保留中の状態からデータベーストランザクションをロールバックします。

例外：HiRDBException

15.6　HiRDB データプロバイダ for .NET Framework の留意事項

15.6.1　グローバルアセンブリキャッシュへの配置

(1)　UAP のビルド，実行での留意事項

UAP ビルド時には，実行ファイル格納ディレクトリへの HiRDB データプロバイダ for .NET Framework の DLL 群のコピーは行われません。また，UAP 実行時には，グローバルアセンブリキャッシュに配置されている DLL 群を参照します。そのため，UAP を実行する場合には，HiRDB データプロバイダ for .NET Framework の DLL 群を UAP と同じディレクトリに配置する必要はありません。

グローバルアセンブリキャッシュには，HiRDB データプロバイダ for .NET Framework の DLL 群に関する発行者ポリシーも配置されます。発行者ポリシーには，UAP 実行時に，インストールされているバージョンの HiRDB クライアントの HiRDB データプロバイダ for .NET Framework の DLL 群に対して，バージョンのリダイレクトを行うための規則が書かれています。このため，作成済みの UAP を再ビルドしなくても，インストールした新しいバージョンの HiRDB データプロバイダ for .NET Framework を参照して，UAP を実行できるようになります。ただし，.NET 5 以降（HiRDB データプロバイダ for .NET Framework でサポートしているバージョンは.NET 6 以降）では，グローバルアセンブリキャッシュの仕様は廃止になりました。

UAP 実行時に参照する HiRDB データプロバイダ for .NET Framework のバージョンを次に示します。

UAP ビルド時に使用した HiRDB データプロバイダ for .NET Framework のバージョン	UAP ビルド及び実行時に使用した.NET Framework 又は.NET のバージョン	UAP 実行時に参照する HiRDB データプロバイダ for .NET Framework のバージョン
09-00 以前	.NET Framework（バージョン 5 より前）	ビルドしたバージョンの HiRDB データプロバイダ for .NET Framework を参照します。新しいバージョンの HiRDB データプロバイダ for .NET Framework を使用するためには，新しいバージョンの HiRDB クライアントのインストール後に，再ビルドする必要があります。
09-01 以降	.NET Framework（バージョン 5 より前）	インストールしている HiRDB クライアントの，HiRDB データプロバイダ for .NET Framework を常に参照します。
	.NET（バージョン 5 以降）	ビルドしたバージョンの HiRDB データプロバイダ for .NET Framework を参照します。新しいバージョンの HiRDB データプロバイダ for .NET Framework を使用するためには，新しいバージョンの HiRDB クライアントのインストール後に，再ビルドする必要があります。

(2)　発行者ポリシーの適用回避の方法

発行者ポリシーは，Windows のデフォルト設定では有効となるように設定されています。しかし，ユーザ側で独自の設定を追加することで，HiRDB クライアントで配置した発行者ポリシーでのバージョンのリダイレクトを，意図的に回避することもできます。

適用回避の方法を次に示します。なお，詳細は Microsoft 提供のドキュメントを参照してください。

(a)　アプリケーション構成ファイルでの設定

UAP 開発時に，アプリケーション構成ファイル（{実行ファイル名}.config）を作成し，アプリケーションとともに配置することができます。アプリケーション構成ファイルに，発行者ポリシーを無効化する設定（<publisherPolicy apply=no/>要素）を追加することで，アプリケーション単位で発行者ポリシーの適用を回避できます。

(b)　コントロールパネルでの設定

[コントロール パネル]–[管理ツール]から，[Microsoft .NET Framework 1.1 Configuration]又は[Microsoft .NET Framework 2.0 Configuration]を選択し，アプリケーションのプロパティを変更することで，アプリケーション単位で発行者ポリシーの適用を回避できます。

(3)　グローバルアセンブリキャッシュへの対応状況

HiRDB データプロバイダ for .NET Framework と発行者ポリシーのグローバルアセンブリキャッシュへの対応状況を次の表に示します。

表 15–3　グローバルアセンブリキャッシュへの対応状況

機能名称		ファイル名	HiRDB クライアントの稼働プラットフォーム	
			Windows (x86)	Windows (x64)
.NET Framework 1.1 対応（32 ビットモード）	データプロバイダ本体	pddndp.dll	○	○
		pddndpcore.dll		
	発行者ポリシー	policy.vv.rr.pddndp.dll※	○	○
		policy.vv.rr.pddndpcore.dll※		
.NET Framework 2.0 対応（32 ビットモード）	データプロバイダ本体	pddndp20.dll	○	○
		pddndpcore20.dll		
	発行者ポリシー	policy.vv.rr.pddndp20.dll※	○	○
		policy.vv.rr.pddndpcore20.dll※		
.NET Framework 2.0 対応（64 ビットモード）	データプロバイダ本体	pddndp20x.dll	–	○
		pddndpcore20x.dll		

機能名称		ファイル名	HiRDB クライアントの稼働プラットフォーム	
			Windows (x86)	Windows (x64)
	発行者ポリシー	policy.vv.rr.pddndp20x.dll※	−	○
		policy.vv.rr.pddndpcore20x.dll※		
.NET Framework 4 対応（32 ビットモード）	データプロバイダ本体	pddndp40.dll	○	○
		pddndpcore40.dll		
	発行者ポリシー	policy.vv.rr.pddndp40.dll※	○	○
		policy.vv.rr.pddndpcore40.dll※		
.NET Framework 4 対応（64 ビットモード）	データプロバイダ本体	pddndp40x.dll	−	○
		pddndpcore40x.dll		
	発行者ポリシー	policy.vv.rr.pddndp40x.dll※	−	○
		policy.vv.rr.pddndpcore40x.dll※		

（凡例）

○：サポートしています。

×：サポートしていません。

−：該当しません。

注※

HiRDB データプロバイダ for .NET Framework 用発行者ポリシーは，バージョン 09-01 以降のリビジョンごとのファイルを累積します。

vv はバージョン番号，rr はリビジョン番号を示します。vv，rr ともにゼロサプレスします。

15.6.2　各メソッド，プロパティについての留意事項

HiRDB データプロバイダ for .NET Framework の各メソッド，プロパティについての留意事項を次の表に示します。

表 15–4　各メソッド，プロパティについての留意事項

オブジェクト	メソッド又はプロパティ	詳細項目
HiRDBCommand	CommandTimeout プロパティ	実行時のタイムアウトは，クライアント環境定義（PDCWAITTIME，PDSWAITTIME，PDSWATCHTIME）の設定に依存するため，設定値は無効になります。
	Cancel メソッド	キャンセルする機能がないため，System.NotSupportedException が返ります。

オブジェクト	メソッド又はプロパティ	詳細項目
	ExecuteReader メソッド	列情報だけの取得や，主キー情報だけの取得をする機能がないため，CommandBehavior.KeyInfo，CommandBehavior. SchemaOnly，CommandBehavior. SequentialAccess を引数指定した場合，CommandBehavior.Default として処理します。
	UpdatedRowSource プロパティ	行を返すバッチクエリ機能がないため，UpdatedRowSource.Both，UpdatedRowSource.FirstReturnedRecord を指定した場合，HiRDBException が返ります。
HiRDBCommandBuilder	CatalogLocation プロパティ	カタログ機能がないため，このプロパティの値が影響する機能はありません。
	CatalogSeparator プロパティ	カタログ機能がないため，このプロパティの値が影響する機能はありません。
	SetAllValues プロパティ	このプロパティの値は，常に true になり，UPDATE 文はすべての列情報を含みます。
HiRDBConnection	ConnectionTimeout プロパティ	常に次の値が返ります。 • 対応する ADO.NET のバージョンが 1.1 の場合は 15 • 対応する ADO.NET のバージョンが 2.0 の場合は 0
	Database プロパティ	DB 名を取得する機能がないため，常に空文字となります。
	State プロパティ	製品の将来のバージョンで使用するための予約値のため，ConnectionState.Connecting，ConnectionState.Executing，ConnectionState.Fetching，ConnectionState.Broken となることはありません。
	BeginTransaction メソッド	SQL ごとに設定，又は HiRDB 環境変数から取得するため，IsolationLevel の指定は無効です。 また，一つのコネクションで複数のトランザクションを実行できません。
	ChangeDatabase メソッド	接続 DB 変更機能がないため，System.NotSupportedException が返ります。
	EnlistTransaction メソッド	分散トランザクション機能がないため，System.NotSupportedException が返ります。
	GetSchema メソッド	スキーマ情報を返しません。常に空の DataTable オブジェクトを返します。
HiRDBDataReader	Depth プロパティ	階層の概念がないため，常に 0 になります。
	VisibleFieldCount プロパティ	検索系 SQL で指定された列以外の列を取得することがないため，FieldCount と同じ値になります。
	GetBoolean メソッド	対応する型がないため，NotSupportedException が返ります。
	GetByte メソッド	対応する型がないため，NotSupportedException が返ります。
	GetChar メソッド	対応する型がないため，NotSupportedException が返ります。
	GetData メソッド	対応する型がないため，NotSupportedException が返ります。

オブジェクト	メソッド又はプロパティ	詳細項目
	GetGuid メソッド	対応する型がないため，NotSupportedException が返ります。
	GetProviderSpecificFieldType	独自のデータ型をサポートしていないため，.NET Framework の共通言語ランタイムに用意されているデータ型のオブジェクトを返します。
	GetProviderSpecificValue	独自のデータ型をサポートしていないため，.NET Framework の共通言語ランタイムに用意されているデータ型のオブジェクトを返します。
	GetProviderSpecificValues	独自のデータ型をサポートしていないため，.NET Framework の共通言語ランタイムに用意されているデータ型のオブジェクトを返します。
	NextResult メソッド	複数レコードセット機能がないため，false が返ります。
HiRDBParameter	DbType プロパティ	対応する型がないため，DbType.Boolean，DbType.Currency，DbType.Guid，DbType.VarNumeric を指定した場合，HiRDBException が返ります。
	Direction プロパティ	ストアドプロシジャの戻り値取得機能がないため，Direction.ReturnValue が指定された状態で，HiRDBCommand クラスの ExecuteNonQuery メソッド，ExecuteReader メソッド，ExecuteScalar メソッド，又は Prepare メソッドのどれかを実行した場合，HiRDBException が返ります。
	IsNullable プロパティ	取得だけで設定はできません（常に null 値の指定はできます）。
HiRDBProviderFactory	CreateDataSourceEnumerator メソッド	HiRDB の列挙体を提供しないため，NotSupportedException が返ります。
HiRDBTransaction	IsolationLevel プロパティ	常に次の値が返ります。 • 対応する ADO.NET のバージョンが 1.1 の場合は IsolationLevel.ReadCommitted • 対応する ADO.NET のバージョンが 2.0 の場合は IsolationLevel.RepeatableRead

15.6.3　UAP 開発時のプラットフォームターゲットの設定

Microsoft Visual Studio 2005 以降の開発環境を使用する場合，アプリケーションのコンパイル時にプラットフォームターゲットを指定します。この指定値によって，UAP 実行時のプロセスのアドレッシングモードが変わります。プラットフォームターゲットの指定値と UAP 実行時のプロセスのアドレッシングモードを次の表に示します。

表 15–5　プラットフォームターゲットの指定値と UAP 実行時のアドレッシングモード

コンパイル時に指定したプラットフォームターゲット	UAP 実行時のプラットフォーム	
	Windows（x86）	Windows（x64）
Any CPU	32 ビットモード	64 ビットモード
x86	32 ビットモード	
x64	64 ビットモード	

Any CPU を指定した場合，プロセスのアドレッシングモードは，UAP 実行時のプラットフォームに依存します。

HiRDB データプロバイダ for .NET Framework でサポートするプラットフォームターゲットの値を次の表に示します。

表 15–6　HiRDB データプロバイダ for .NET Framework でサポートするプラットフォームターゲットの値

HiRDB データプロバイダ for .NET Framework	UAP 実行時のプラットフォーム	
	Windows（x86）	Windows（x64）
.NET Framework 1.1 対応（32 ビットモード）	Any CPU，x86	x86
.NET Framework 2.0 対応（32 ビットモード）	Any CPU，x86	x86
.NET Framework 2.0 対応（64 ビットモード）	該当しません。	Any CPU，x64
.NET Framework 4 対応（32 ビットモード）	Any CPU，x86	x86
.NET Framework 4 対応（64 ビットモード）	該当しません。	Any CPU，x64

15.7　HiRDB データプロバイダ for .NET Framework のデータ型

15.7.1　DbType プロパティと HiRDBType プロパティ

HiRDBParameter クラスの DbType プロパティを設定すると，自動的に同クラスの HiRDBType プロパティも設定されます。また，HiRDBType プロパティを設定すると，自動的に DbType プロパティも設定されます。DbType プロパティの設定時に自動設定される HiRDBType プロパティの値を次の表に示します。

表 15–7　DbType プロパティ設定時に自動設定される HiRDBType プロパティの値

DbType プロパティ	HiRDBType プロパティ
AnsiString	VarChar
AnsiStringFixedLength	Char
Binary	Binary
Boolean	［NotSupportedException 例外］
Byte	SmallInt
Currency	［NotSupportedException 例外］
Date	Date
DateTime	TimeStamp
Decimal	Decimal
Double	Float
Guid	［NotSupportedException 例外］
Int16	SmallInt
Int32	Integer
Int64	Decimal
Object	Binary
SByte	SmallInt
Single	SmallFlt
String	MvarChar
StringFixedLength	Mchar
Time	Time
UInt16	Integer
UInt32	Decimal
UInt64	Decimal

DbType プロパティ	HiRDBType プロパティ
VarNumeric	［NotSupportedException 例外］

HiRDBType プロパティの設定時に自動設定される DbType プロパティの値を次の表に示します。

表 15–8　HiRDBType プロパティ設定時に自動設定される DbType プロパティの値

HiRDBType プロパティ	DbType プロパティ
Binary	Object
Blob	Object
Char	AnsiStringFixedLength
Date	Date
Decimal	Decimal
Float	Double
Integer	Int32
IntervalYearToDay	String
IntervalHourToSecond	String
MChar	StringFixedLength
MVarChar	String
NChar	StringFixedLength
NVarChar	String
SmallFlt	Single
SmallInt	Int16
Time	Time
TimeStamp	DateTime
VarChar	AnsiString

15.7.2　UAP で使用するデータ型とアクセサ

INSERT 実行時などで HiRDBParameter クラスの Value プロパティに設定するデータの型，及び SELECT 実行時などで使用する HiRDBDataReader クラスの GetXXXX メソッドを次の表に示します。なお，HiRDB の NULL は，.NET Framework 型の DBNull.Value で表現されます。

表 15–9　HiRDB 型に対する UAP の使用型及びアクセサ

分類	HiRDB のデータ型	INSERT などで UAP が使用する.NET Framework 型	SELECT などで UAP が使用するアクセサ
文字	CHAR[ACTER]	String	GetString()
	VARCHAR/CHAR[ACTER]VARYING	String	GetString()
	NCHAR/NATIONAL CHAR[ACTER]	String	GetString()
	NVARCHAR/NCHAR VARYING	String	GetString()
	MCHAR	String	GetString()
	MVARCHAR	String	GetString()
数値	[LARGE]DEC[IMAL]/NUMERIC	Decimal	GetDecimal()
	SMALLINT	Int16	GetInt16()
	INT[EGER]	Int32	GetInt32()
	SMALLFLT/REAL	Single	GetFloat()
	FLOAT/DOUBLE PRECISION	Double	GetDouble()
日時	DATE	DateTime	GetDateTime()
	TIME	DateTime	GetDateTime()
	TIMESTAMP	DateTime	GetDateTime()
その他	BINARY	Byte[]	GetBytes()
	BLOB	Byte[]	GetBytes()
	INTERVAL YEAR TO DAY	String	GetString()
	INTERVAL HOUR TO SECOND	TimeSpan	GetString()

15.7.3　HiRDB データプロバイダ for .NET Framework の型変換

表「HiRDB 型に対する UAP の使用型及びアクセサ」の.NET Framework 型及びアクセサを使用しない場合，HiRDB データプロバイダ内部で自動的に型変換されます。NET Framework 型及びアクセサを使用しない場合とは，CHAR 属性の項目を持つ表に Int32 型のデータを INSERT したり，GetInt32 メソッドで取得したりすることをいいます。

INSERT 時の型変換一覧を次の表に示します。

- 表「INSERT 時の型変換一覧（1/2）」
- 表「INSERT 時の型変換一覧（2/2）」

SELECT 時の型変換一覧を次の表に示します。

- 表「SELECT 時の型変換一覧（1/2）」
- 表「SELECT 時の型変換一覧（2/2）」

なお，これらの表の記号の意味については，「記号の意味」を参照してください。

表 15–10　INSERT 時の型変換一覧（1/2）

.NET Framework 型	HiRDB のデータ型								
	I	SI	DE	F	SF	C	VC	NC	NVC
Boolean	×1	×1	×1	×1	×1	×1	×1	×1	×1
Int16	○	○	○	○	○	○	○	×1	×1
Int32	○	△1	○	○	○	○	○	×1	×1
Int64	△2	△1	○	○	○	○	○	×1	×1
UInt16	○	△1	○	○	○	○	○	×1	×1
UInt32	△2	△1	○	○	○	○	○	×1	×1
UInt64	△2	△1	○	○	○	○	○	×1	×1
Single 小数部ありデータ	△4	△3	○	○	○	○	○	×1	×1
Single 小数部なしデータ	△2	△1	○	○	○	○	○	×1	×1
Double 小数部ありデータ	△4	△3	○	○	○	○	○	×1	×1
Double 小数部なしデータ	△2	△1	○	○	○	○	○	×1	×1
Decimal 小数部ありデータ	△4	△3	○	○	○	○	○	×1	×1
Decimal 小数部なしデータ	△2	△1	○	○	○	○	○	×1	×1
Char	○1	○1	×1	×1	×1	○	○	○	○
Char[]	×1	×1	×1	×1	×1	×1	×1	×1	×1
String	△2	△1	○	○	○	○	○	○	○
DateTime	×1	×1	×1	×1	×1	○	○	×1	×1
TimeSpan	×1	×1	×1	×1	×1	○	○	×1	×1
Guid	×1	×1	×1	×1	×1	○	○	×1	×1
Byte	○	○	○	○	○	○	○	×1	×1
Byte[]	×1	×1	×1	×1	×1	×1	×1	×1	×1
Sbyte	○	○	○	○	○	○	○	×1	×1
SByte[]	×1	×1	×1	×1	×1	×1	×1	×1	×1

表 15–11　INSERT 時の型変換一覧（2/2）

.NET Framework 型	HiRDB のデータ型								
	MC	MVC	DA	T	TS	IY	IHS	BI	BL
Boolean	×1	×1	×1	×1	×1	×1	×1	×1	×1
Int16	○	○	×1	×1	×1	×2	×2	×1	×1
Int32	○	○	×1	×1	×1	×2	×2	×1	×1
Int64	○	○	×1	×1	×1	×2	×2	×1	×1
UInt16	○	○	×1	×1	×1	×2	×2	×1	×1
UInt32	○	○	×1	×1	×1	×2	×2	×1	×1
UInt64	○	○	×1	×1	×1	×2	×2	×1	×1
Single 小数部ありデータ	○	○	×1	×1	×1	×2	×2	×1	×1
Single 小数部なしデータ	○	○	×1	×1	×1	×2	×2	×1	×1
Double 小数部ありデータ	○	○	×1	×1	×1	×2	×2	×1	×1
Double 小数部なしデータ	○	○	×1	×1	×1	×2	×2	×1	×1
Decimal 小数部ありデータ	○	○	×1	×1	×1	×2	×2	×1	×1
Decimal 小数部なしデータ	○	○	×1	×1	×1	×2	×2	×1	×1
Char	○	○	×1	×1	×1	×2	×2	×1	×1
Char[]	×1	×1	×1	×1	×1	×2	×2	×1	×1
String	○	○	○	○	○	○	○	×1	×1
DateTime	○	○	○	○	○	×2	×2	×1	×1
TimeSpan	○	○	×1	×1	×1	×2	○	×1	×1
Guid	○	○	×1	×1	×1	×2	×2	×1	×1
Byte	○	○	×1	×1	×1	×2	×2	○	○
Byte[]	×1	×1	×1	×1	×1	×2	×2	○	○
Sbyte	○	○	×1	×1	×1	×2	×2	○	○
SByte[]	×1	×1	×1	×1	×1	×2	×2	○	○

注 1　NCHAR/NVARCHAR への INSERT 時の注意事項

S-JIS 変換後のサイズが奇数バイトのデータは，［Hitachi.HiRDB.HiRDBException］KFPZ24026-E 形式変換エラーになります。

注 2　配列 INSERT 時の注意事項

Object 配列型以外は［Hitachi.HiRDB.HiRDBException］KFPZ24026-E 形式変換エラーになります。また，BLOB への配列インサートはできないため，同じエラーになります。

表 15–12　SELECT 時の型変換一覧（1/2）

アクセサ	HiRDB のデータ型								
	I	SI	DE	F	SF	C	VC	NC	NVC
GetBoolean	×3	×3	×3	×3	×3	×3	×3	×3	×3
GetByte	×3	×3	×3	×3	×3	×3	×3	×3	×3
GetBytes	○	○	×1	○	○	○	○	○	○
GetChar	×3	×3	×3	×3	×3	×3	×3	×3	×3
GetChars	×1	×1	×1	×1	×1	○	○	○	○
GetData	×3	×3	×3	×3	×3	×3	×3	×3	×3
GetDateTime	×1	×1	×1	×1	×1	△6	△6	△6	△6
GetDecimal	○	○	○	○	○	△7	△7	△7	△7
GetDouble	○	○	○	○	○	△8	△8	△8	△8
GetFloat	○	○	○	○	○	△9	△9	△9	△9
GetGuid	×3	×3	×3	×3	×3	×3	×3	×3	×3
GetInt16	△1	○	△1	△1	△1	△1	△1	△1	△1
GetInt32	○	○	△2	△2	△2	△2	△2	△2	△2
GetInt64	○	○	△10	△10	△10	△10	△10	△10	△10
GetString	○	○	○	○	○	○	○	○	○
GetValue	○	○	○	○	○	○	○	○	○
GetValues	○	○	○	○	○	○	○	○	○

表 15–13　SELECT 時の型変換一覧（2/2）

アクセサ	HiRDB のデータ型								
	MC	MVC	DA	T	TS	IY	IHS	BI	BL
GetBoolean	×3	×3	×3	×3	×3	×3	×3	×3	×3
GetByte	×3	×3	×3	×3	×3	×3	×3	×3	×3
GetBytes	○	○	×1	×1	×1	×1	×1	○	○
GetChar	×3	×3	×3	×3	×3	×3	×3	×3	×3
GetChars	○	○	×1	×1	×1	×1	×1	×1	×1
GetData	×3	×3	×3	×3	×3	×3	×3	×3	×3
GetDateTime	△6	△6	○	○	○	×1	×1	×1	×1
GetDecimal	△7	△7	×1	×1	×1	×1	×1	×1	×1

アクセサ	HiRDB のデータ型								
	MC	MVC	DA	T	TS	IY	IHS	BI	BL
GetDouble	△8	△8	×1	×1	×1	×1	×1	×1	×1
GetFloat	△9	△9	×1	×1	×1	×1	×1	×1	×1
GetGuid	×3	×3	×3	×3	×3	×3	×3	×3	×3
GetInt16	△1	△1	×1	×1	×1	×1	×1	×1	×1
GetInt32	△2	△2	×1	×1	×1	×1	×1	×1	×1
GetInt64	△10	△10	×1	×1	×1	×1	×1	×1	×1
GetString	○	○	○	○	○	○	○	○	○
GetValue	○	○	○	○	○	○	○	○	○
GetValues	○	○	○	○	○	○	○	○	○

注 1　DATE 取得時の注意事項

GetDateTime メソッドで取得した場合，時刻領域は 0 時 0 分 0 秒が入ります。GetString メソッドで取得した場合，YYYY/MM/DD のフォーマットで入ります。

注 2　TIME/TIMESTAMP 取得時の注意事項

GetDateTime メソッドで取得した場合，日付領域は現在年月日が入ります。GetString メソッドで取得した場合，次のフォーマットで入ります。

TIME：hh:mm:ss

TIMESTAMP(0)：YYYY/MM/DD hh:mm:ss

TIMESTAMP(2)：YYYY/MM/DD hh:mm:ss.nn

TIMESTAMP(4)：YYYY/MM/DD hh:mm:ss.nnnn

TIMESTAMP(6)：YYYY/MM/DD hh:mm:ss.nnnnnn

注 3　INTERVALYEARTODAY 取得時の注意事項

GetString メソッドで取得した場合，±YYYY/MM/DD のフォーマットで入ります。

注 4　INTERVALHOURTOSECOND 取得時の注意事項

GetString メソッドで取得した場合，±hh:mm:ss のフォーマットで入ります。

(1)　記号の意味

(a)　HiRDB のデータ型

HiRDB のデータ型の，記号の意味を次に示します。

記号	意味
I	INTEGER
SI	SMALLINT

記号	意味
DE	DECIMAL，及び LARGE DECIMAL
F	FLOAT/DOUBLE PRECISION
SF	SMALLFLT，及び REAL
C	CHARACTER
VC	VARCHAR
NC	NCHAR，及び NATIONAL CHARACTER
NVC	NVARCHAR
MC	MCHAR
MVC	MVARCHAR
DA	DATE
T	TIME
TS	TIMESTAMP
IY	INTERVAL YEAR TO DAY
IHS	INTERVAL HOUR TO SECOND
BI	BINARY
BL	BLOB

(b)　型変換可否

○は正常，△は条件付きで動作，×はエラーとなります。さらに，これらに番号が付いている場合があります。○，△，及び×に番号が付いている場合の記号の意味を次に示します。

記号	意味
○	数字の文字コードが入ります。
△1	Int32 型，Int64 型，Single 小数部なしデータ型，Double 小数部なしデータ型，Decimal 小数部なしデータ型，String 小数部なしデータ型の場合 -32768～32767：正常 UInt16 型，UInt32 型，UInt64 型の場合 0～32767：正常 範囲外：［Hitachi.HiRDB.HiRDBException］KFPZ24026-E 形式変換エラー
△2	Int64 型，Single 小数部なしデータ型，Double 小数部なしデータ型，Decimal 小数部なしデータ型，String 小数部なしデータ型の場合 -2147483648～2147483647：正常 UInt32 型，UInt64 型の場合 0～2147483647：正常 範囲外：［Hitachi.HiRDB.HiRDBException］KFPZ24026-E 形式変換エラー

記号	意味
△3	-32768～32767：正常（小数点第一位四捨五入） 範囲外：［Hitachi.HiRDB.HiRDBException］KFPZ24026-E 形式変換エラー
△4	-2147483648～2147483647：正常（小数点第一位四捨五入） 範囲外：［Hitachi.HiRDB.HiRDBException］KFPZ24026-E 形式変換エラー
△5	0～255：正常 範囲外：［Hitachi.HiRDB.HiRDBException］KFPZ24026-E 形式変換エラー
△6	DateTime 形式データの場合：正常 DateTime 形式データ以外：［Hitachi.HiRDB.HiRDBException］KFPZ24026-E 形式変換エラー
△7	Decimal 形式データの場合：正常 Decimal 形式データ以外：［Hitachi.HiRDB.HiRDBException］KFPZ24026-E 形式変換エラー
△8	Double 形式データの場合：正常 Double 形式データ以外：［Hitachi.HiRDB.HiRDBException］KFPZ24026-E 形式変換エラー
△9	Float 形式データの場合：正常 Float 形式データ以外：［Hitachi.HiRDB.HiRDBException］KFPZ24026-E 形式変換エラー
△10	-9223372036854775808～9223372036854775807：正常 範囲外：［Hitachi.HiRDB.HiRDBException］KFPZ24026-E 形式変換エラー
×1	［Hitachi.HiRDB.HiRDBException］KFPZ24026-E 形式変換エラー
×2	［Hitachi.HiRDB.HiRDBException］KFPZ24107-E Decimal，日時，時間間隔型オーバフロー ［Hitachi.HiRDB.HiRDBException］KFPZ24106-E 日時，時間間隔型フォーマットエラー
×3	［System.NotSupportedException］ 未サポートエラー

15.8　接続プーリング機能

15.8.1　接続プーリング機能とは

接続プーリング機能とは，再利用できる接続を保持しておくことによって，新しく開く接続の数を削減する機能のことです。

UAP が Connection オブジェクトの Open メソッドを呼び出すと，HiRDB データプロバイダ for .NET Framework は接続プールに利用可能な接続があるかどうかを確認します。プールされた接続が利用できる場合は，新しい接続を開かないで，プールされた接続を呼び出し元に返します。プールされた接続が利用できない場合は，新しい接続を開きます。また，UAP が Connection オブジェクトの Close メソッドを呼び出しても，HiRDB データプロバイダ for .NET Framework は実際には接続を閉じません。そして，次の Open メソッド呼び出し時には新しい接続を開かないで，接続プールに保持された接続を再利用します。

なお，接続プールでは接続を一定時間保持しますが，その間に再利用されない場合は，接続を破棄します。

ポイント

Connection オブジェクトの Open メソッドと Close メソッドを頻繁に使用する場合は，接続をプールすると UAP のパフォーマンスとスケーラビリティを大幅に改善できる可能性があります。

15.8.2　使用方法

接続プーリング機能を使用する場合は，次の設定をしてください。

- HiRDBConnection.Pooling プロパティに true（デフォルト値）を設定します。
- HiRDBConnection.LifeTime プロパティに，接続プールで接続を保持する時間を設定します。

上記の設定をすると，プールに存在する接続が次の条件をすべて満たす場合，その接続は再利用されます。

- 現在使用されていない接続である
- 接続文字列が完全に一致する

15.8.3　HiRDB データプロバイダ for .NET Framework の接続プーリングにおける接続回復機能

HiRDB データプロバイダ for .NET Framework の接続プーリング機能では，プール中に HiRDB サーバとの接続が切断され，未接続状態となった HiRDBConnection オブジェクトに対して SQL 実行要求を行うと，接続障害が発生します。HiRDB データプロバイダ for .NET Framework の接続プーリングでの接続回復機

能は，HiRDBConnection オブジェクトの Open メソッド呼び出し後，初回の SQL（以降，ファースト SQL と呼ぶ）の実行時に接続障害を検知した場合，HiRDB データプロバイダ for .NET Framework で一度だけ再接続を行い，プール中の接続を回復する機能です。

(1) 使用方法

この機能は，.NET Framework2.0 以降に対応した HiRDB データプロバイダ for .NET Framework でだけ有効になります。

この機能を使用する場合は，次の設定をしてください。

- HiRDBConnection.ConnectionRecover プロパティに true（デフォルト値）を設定する

ただし，クライアントライブラリが提供する自動再接続機能を使用している場合は，自動再接続機能による再接続を優先するため，この機能は無効になります。この機能が有効になる条件を次の表に示します。

表 15–14　接続回復機能が有効になる条件

自動再接続機能の使用有無	接続プーリング機能の使用有無	SQL 種別	SQL 実行結果	ConnectionRecover の値	
				true	false
有	–	–	–	×	×
無	有	ファースト SQL	接続障害※	○	×
			上記以外	×	×
		上記以外	–	×	×
	無	–	–	×	×

（凡例）

○：接続回復機能が有効になります。

×：接続回復機能が無効になります。

–：該当しません。

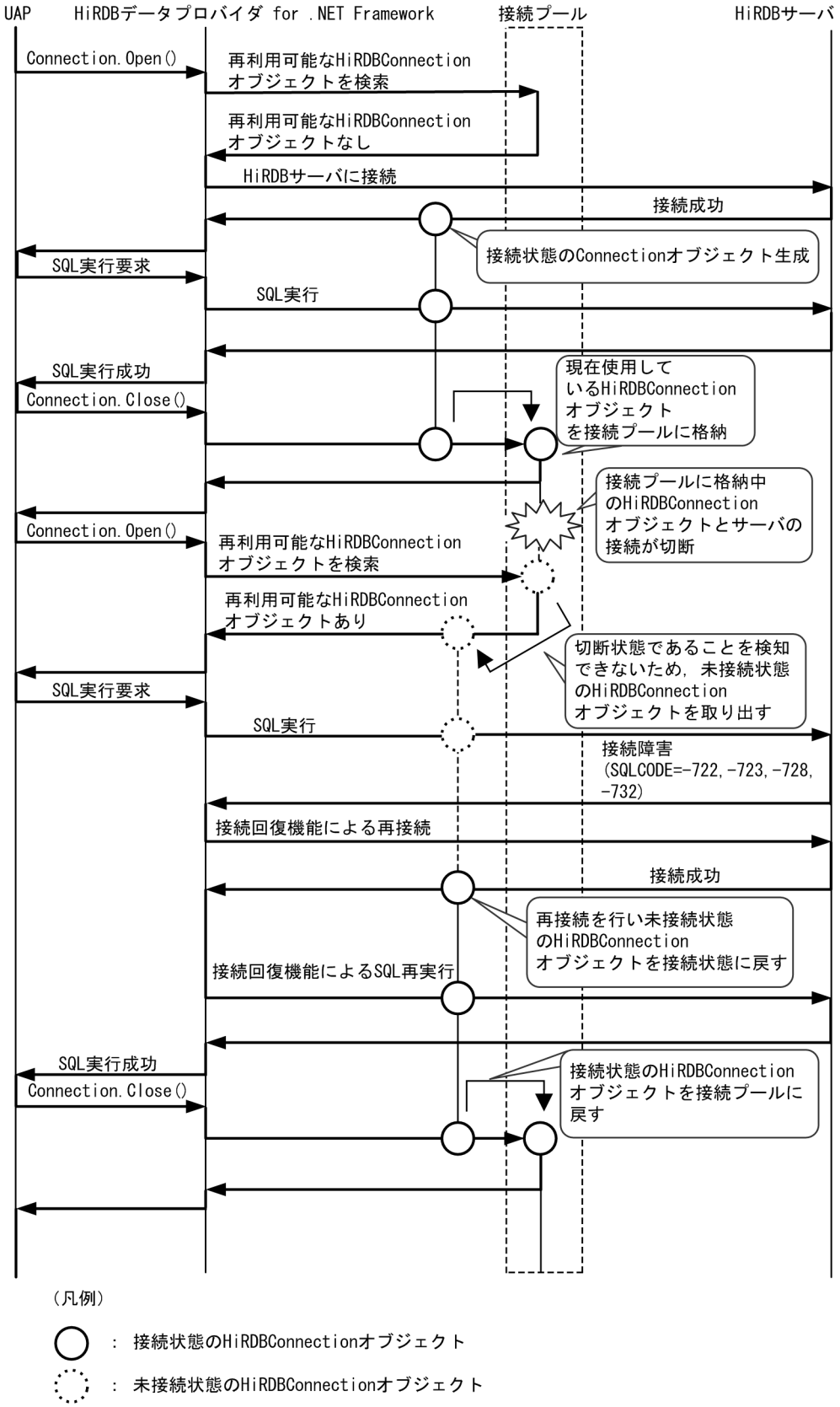
注※

SQLCODE が-722，-723，-728，-732 のどれかの場合を示します。

(2) 接続回復契機

この機能の接続回復契機を次の図に示します。この機能による接続処理で障害が発生した場合は，UAP へエラーを返却します。

図 15–1　接続回復機能の接続回復契機



（凡例）

○ ： 接続状態のHiRDBConnectionオブジェクト

◌ ： 未接続状態のHiRDBConnectionオブジェクト

15.9 DbProviderFactory を使用したプロバイダに依存しないコード

ADO.NET2.0 では，DbProviderFactory インスタンスによって，Command や Parameter などのほかのクラスのインスタンスを生成できるようになりました。プロバイダの名前空間を示す文字列（以降，プロバイダ名と呼びます）を指定して DbProviderFactory インスタンスを生成すると，あらかじめ提供されている情報に基づいて各プロバイダに特化したインスタンスを生成できます。なお，指定するプロバイダ名は構成ファイルに登録し，取得できます。これによって，プロバイダに依存しないコードを作成し，実行時にプロバイダを選択できます。

15.9.1 DbProviderFactory を使用したプロバイダに依存しないコードの作成方法

(1) プロバイダ情報の追加

DbProviderFactory インスタンスを DbProviderFactories クラスの GetFactory メソッドを呼び出して生成します。.NET Framework Version2.0/3.0 のインストール時に，このメソッドに指定できるプロバイダ名は次の 4 種類です。

- System.Data.Odbc
- System.Data.OleDb
- System.Data.OracleClient
- System.Data.SqlClient

.NET Framework Version2.0/3.0 のインストール時に組み込まれる machine.config ファイルについて，system.data セクションの DbProviderFactories 要素にプロバイダ情報を追加すると GetFactory メソッドに指定できるようになります。

また，HiRDB データプロバイダ for .NET Framework の場合，指定する invariant 値は"Hitachi.HiRDB"です。

HiRDB データプロバイダ for .NET Framework の情報を追加する場合の例を次に示します。

- プロバイダ情報の追加

```
<system.data>
  <DbProviderFactories>
           .
           .
           .
    <add name="HiRDB Data Provider" invariant="Hitachi.HiRDB"
      description=".NET Framework Data Provider for HiRDB"
      type="Hitachi.HiRDB.HiRDBProviderFactory, pddndp20,
      Version=X.X.X.X, Culture=neutral,
      PublicKeyToken=YYYYYYYYYYYYYYYY" />
```

```
  </DbProviderFactories>
</system.data>
```

（凡例）

X.X.X.X：アセンブリのバージョン。アセンブリのバージョンは，pddndp20.dll のプロパティで確認できます。

YYYYYYYYYYYYYYYY：アセンブリ公開キートークン。アセンブリ公開キートークンは，コマンドプロンプト又は MS-DOS プロンプトから次のコマンドを入力することで確認できます。

```
sn -T pddndp20.dll
```

(2) 構成ファイルを使用したプロバイダ名の指定

プロバイダ名を構成ファイルに登録し，DbProviderFactory インスタンス生成時に取得します。構成ファイルについては，.NET Framework のマニュアルを参照してください。

HiRDB データプロバイダ for .NET Framework の場合，指定するプロバイダ名（value）は"Hitachi.HiRDB"です。なお，プロバイダ名を DbProviderFactories クラスの GetFactory メソッドの引数に直接指定する場合，構成ファイルは不要です。

構成ファイルの例を次に示します。なお，キー名（provider）は任意の文字列です。

- 構成ファイルの例

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <appSettings>
    <add key="provider" value="Hitachi.HiRDB" />  ...HiRDBデータプロバイダ for .NET Frame
work
  </appSettings>
</configuration>
```

(3) DbProviderFactory インスタンスの生成

(a) 構成ファイルを使用しない場合

プロバイダ名（"Hitachi.HiRDB"）を DbProviderFactories クラスの GetFactory メソッドの引数に直接指定して，DbProviderFactory インスタンスを生成します。コーディング例を次に示します。

```
DbProviderFactory dataFactory =
          DbProviderFactories.GetFactory( “Hitachi.HiRDB” );
```

(b) 構成ファイルを使用する場合

構成ファイルからキー名（provider）の値を取得し，その値を DbProviderFactories クラスの GetFactory メソッドの引数に指定して，DbProviderFactory インスタンスを生成します。コーディング例を次に示します。

```
DbProviderFactory dataFactory =
          DbProviderFactories.GetFactory(ConfigurationManager.AppSettings["provider"]);
```

(4)　コーディング例

プロバイダに依存するプロバイダ名や接続文字列を構成ファイルから取得することによって，プログラムを変更することなく異なるプロバイダを使用できます。

構成ファイルを使用する場合のコーディング例を次に示します。

```
try{
    // 指定したプロバイダ名のDbProviderFactoryの
    // オブジェクトを作成
    DbProviderFactory dataFactory =
      DbProviderFactories.GetFactory(
        ConfigurationManager.AppSettings["provider"]);
    // 接続オブジェクトを作成する。
    DbConnection cn = dataFactory.CreateConnection();
    // 接続文字列を設定
    cn.ConnectionString =
      ConfigurationManager.AppSettings["connect"];
    // DBに接続する
    cn.Open();
    // SQLを実行するコマンドオブジェクトを作成
    DbCommand cm = dataFactory.CreateCommand();
    cm.Connection = cn;
    cm.CommandText = "select C1,C2 from ex where C1 = ?";
    // パラメタオブジェクトを作成する
    DbParameter pr = dataFactory.CreateParameter();
    // パラメタの設定をする
    pr.ParameterName = "@p1";
    pr.DbType = DbType.Int32;
    pr.Value = 1;
    cm.Parameters.Add(pr);
    // DataReaderオブジェクトを作成する
    DbDataReader rd = cm.ExecuteReader();
    while (rd.Read())
    {
      Console.WriteLine("C1: " + rd.GetValue(0));
      Console.WriteLine("C2: " + rd.GetValue(1));
    }
    // DBから切断する
    cn.Close();
 }
 catch (System.Exception ex)
 {
    Console.WriteLine(ex);
    throw ex;
 }
```

1.
2.
3.
4.

```
構成ファイル

<?xml version="1.0"
 encoding="utf-8" ?>
<configuration>
 <appSettings>
  <add key="provider"
   value="Hitachi.HiRDB" />
  <add key="connect"
   value="DSN=pc" />
 </appSettings>
</configuration>
```

一致

```
machine.configファイル

<system.data>
 <DbProviderFactories>
 --------省略---------
  <add name=
   "HiRDB Data Provider"
   invariant=
   "Hitachi.HiRDB"
 --------省略---------
  />
 </DbProviderFactories>
</system.data>
```

[説明]
1.キー名（provider）で構成ファイルの検索
2.値（Hitachi.HiRDB）の返却
3.キー名（connect）で構成ファイルの検索
4.値（DSN=pc）の返却

15.10　HiRDB データプロバイダ for .NET Framework のトラブルシュート機能

.NET Framework 2.0 以降に対応した HiRDB データプロバイダ for .NET Framework では，トラブルシュート情報としてメソッドトレースを取得できます。

15.10.1　メソッドトレースの取得方法

メソッドトレース情報は，クライアント環境定義の PDCLTPATH 及び PDDNDPTRACE に値を設定することで取得できます。各クライアント環境定義については，「クライアント環境定義（環境変数の設定）」を参照してください。

15.10.2　メソッドトレースの出力規則

メソッドトレースの出力規則を次に示します。

- 情報を取得するメソッドトレースファイルは，指定したディレクトリに二つ作成されます。
- 次の時に出力されます。
  - メソッドの呼び出し時
  - メソッドの戻り時
  - プロパティの設定時
  - プロパティの取得時
- 文字コードは UTF-8 です。
- 作成されるファイル名称は，pddndpxxxxx_yyyy_1.trc，及び pddndpxxxxx_yyyy_2.trc です。xxxxx にはプロセス ID，yyyy にはコネクト通番が入ります。

15.10.3　メソッドトレース情報の見方

出力されるメソッドトレースの例とその説明を次に示します。

ヘッダ

```
[1][1742][sds01][12345678][HiRDB_Data_Provider20][08.04.0.0]
 1  2     3      4         5                      6
```

[説明]

ヘッダはファイルの先頭に出力されます。

1. コネクト通番です。

2. 接続先サーバのプロセス ID です。

3. シングルサーバ名，又はフロントエンドサーバ名です。

4. UAP のプロセス ID です。

5. トレース識別情報です。使用する HiRDB データプロバイダ for .NET Framework によって，出力内容が異なります。出力内容の違いを次に示します。

HiRDB データプロバイダ for .NET Framework 種別	アドレッシングモード	トレースの識別情報
HiRDB データプロバイダ for .NET Framework (.NET Framework 2.0)	32	HiRDB_Data_Provider20
	64	HiRDB_Data_Provider20x
HiRDB データプロバイダ for .NET Framework (.NET Framework 4)	32	HiRDB_Data_Provider40
	64	HiRDB_Data_Provider40x

6. HiRDB データプロバイダ for .NET Framework のアセンブリのバージョンです。

メソッドトレース情報

```
[0000000001][E][HiRDBCommand@12345678 ExecuteNonQuery][SID(2)][2008/08/27 1:29:10.123]
 1           2  3            4        5                6       7

  [Return=0]
   8
  [nArraySize=10]                                  ┐
                                                   │
        :                                          ├9
                                                   │
  [CommandText=INSERT INTO T1 VALUES(100)]         ┘

  [MessageText=KFPA11117-E Number of insert values not equal to number of insert columns]
 10
  [SQLCODE=-117] 11
  [SQLWARN=0000] 12
  場所 Hitachi.HiRDB.native.HiRDBcore.ClearSectionItems()        ┐
  場所 Hitachi.HiRDB.HiRDBConnection.Close()                     │
  場所 Hitachi.HiRDB.HiRDBConnection.Dispose(Boolean disposing)  ├13
  場所 Hitachi.HiRDB.HiRDBConnection.Finalize()                  ┘
```

[説明]

メソッドトレース情報は，メソッドの呼び出し時及び戻り時，プロパティの設定時及び取得時ごとに出力されます。

1. スレッド ID です。

2. アクセス種別です。

   E：メソッドの呼び出し

   R：メソッドからの戻り

   S：プロパティへの値設定

G：プロパティの値取得

アクセス種別によって，出力内容が異なります。アクセス種別による出力内容の違いを次に示します。

アクセス種別		呼び出し種別	引数値又は プロパティ値	戻り値	エラー情報
メソッド	E	呼び出し	○	×	×
	R	戻り（正常時）	×	○	×
		戻り（エラー時）	×	×	○
プロパティ	S	設定（正常時）	○	×	×
		設定（エラー時）	○	×	○
	G	取得（正常時）	×	○	×
		取得（エラー時）	×	×	○

（凡例）

○：出力されます。

×：出力されません。

3. クラス名です。プロバイダ名を省略して出力します。
4. ハッシュコードです。クラス名と@で結合して出力します。
5. メソッド名，又はプロパティ名です。
6. セクション番号です。SQL の実行に関係しないメソッド，プロパティの場合は特定できないため，*を出力します。
7. トレースの取得日時です。
8. 戻り値です。例外発生時は Exception クラス名を出力します。
9. 引数名と引数値，又はプロパティ名とプロパティ値です。＝で連結して出力します。
10. エラーメッセージです。
11. SQLCODE です。SQL 文を実行した結果，発生した SQLCODE を表示します。
12. SQLWARN です。警告情報を 16 進数表記で表示します。詳細については，「SQL トレース情報の見方」を参照してください。
13. スタックトレースです。

一部のメソッドではプロパティ情報が出力されます。アクセス種別ごとの出力形式を次に示します。

アクセス種別		呼び出し元	形式	備考
メソッド	E	HiRDBCommand.Execute HiRDBCommand.ExecuteDbDataReader HiRDBCommand.ExecuteNonQuery HiRDBCommand.ExecuteReader HiRDBCommand.ExecuteScalar	CommandText=VALUE※1 Parameters.Count=VALUE PARAMETER_VALUE※2	引数が存在する場合は，引数情報を出力します。

アクセス種別		呼び出し元	形式	備考
		その他	ARGUMENT※3=VALUE	−
	R	HiRDBConnection.Open	ConnectionString=VALUE ServerVersion=VALUE	−
		その他	Return=VALUE	−
プロパティ	S	−	PROPERTY※4=VALUE	−
	G	−	Return=VALUE	−

（凡例）

−：該当しません。

注※1

VALUE はプロパティ値，引数値，戻り値として設定，又は取得した値です。

注※2

PARAMETER_VALUE は HiRDBParameterCollection に登録されている各パラメタの情報です。情報の内容を次に示します。

ParameterName，HiRDBType，Value，Precision，Scale，Repetition

注※3

ARGUMENT は引数名です。

注※4

PROPERTY はプロパティ名です。

15.10.4　メソッドトレースファイルのバックアップの取得

メソッドトレース情報を出力してメソッドトレースファイルの容量が指定したサイズを超えると，次のエントリからはもう一方のメソッドトレースファイルに出力されます。このとき，切り替え先のメソッドトレースファイルに格納されている古いメソッドトレース情報から順に消去され，新しいメソッドトレース情報に書き換えられます。このため，必要な情報は UAP 終了時にメソッドトレースファイルの内容をコピーしてバックアップを取得しておいてください。

なお，現在使用しているメソッドトレースファイルを知りたい場合は，ファイルの最終更新日時を調べてください。最終更新日時の新しい方が現在使用しているメソッドトレースファイルになります。dir コマンド又はエクスプローラで，ファイルの最終更新日時を調べてください。

15.11　HiRDB データプロバイダ for .NET Framework を使用した UAP 例

HiRDB データプロバイダ for .NET Framework を使用した UAP 例について説明します。

15.11.1　データベースへの接続

HiRDB に接続し，そのまま切断する例を次に示します。

- Visual C# .NET で記述した例

```
using System;
using Hitachi.HiRDB;

namespace test_C
{
    class Sample
    {
        [STAThread]
        static void Main(string[] args)
        {
            try
            {
                // Connectionオブジェクトを作成する
                HiRDBConnection cn = new HiRDBConnection(”dsn=pc;”);  ...1

                // DBに接続する
                cn.Open();  .....................................2

                // DBから切断する
                cn.Close();  .....................................3
                cn.Dispose();
            }
            catch ( HiRDBException ex)
            {
                Console.WriteLine(ex);
            }
            catch ( System.Exception ex)
            {
                Console.WriteLine(ex);
            }  .....................................4
        }
    }
}
```

- Visual Basic.NET で記述した例

```
Imports System
Imports System.Data
Imports Hitachi.HiRDB

Module Module1

    Sub Main()
```

```
        Dim cn As HiRDBConnection
        Dim cm As HiRDBCommand
        Try
            ’ Connectionオブジェクトを作成する
            cn = New HiRDBConnection(”dsn=pc;”)  .....................1

            ’ DBに接続する
            cn.Open()  ...............................................2

            ’ DBから切断する
            cn.Close()  ..............................................3
            cn.Dispoxe()
        Catch ex As HiRDBException
            Console.WriteLine(ex)

        Catch ex As System.Exception

            Console.WriteLine(ex)

        End Try  .....................................................4

    End Sub

End Module
```

[説明]

1. 最初に HiRDBConnection のオブジェクトを作成します。このオブジェクトが HiRDB との通信をすべて管理することになります。

   このメソッドには一つの string 型引数を指定する必要があります。指定する文字列は接続文字列と呼ばれるもので，これは ADO や ADO.NET の Connection で使用する接続文字列と同種のものです。指定できる文字列については，「ConnectionString」を参照してください。

2. DB へ接続するには Open メソッドを使用します。

3. 切断する場合には Close メソッドを使用します。接続していない状態で Close メソッドを使用しても例外は発生しません。

4. サーバが起動していなかったり，通信ができなかったり，SQL 文が不正だったりした場合など，例外が発生します。基本的には，HiRDB データプロバイダ for .NET Framework を使用するブロックは，try～catch で例外を検出して例外メッセージを表示させるようにします。

   HiRDB 全般のエラーでは System.Exception が，HiRDB データプロバイダ for .NET Framework 固有のエラーでは HiRDBException が発生します。なお，System.Exception は Exception と省略しないでください。

   HiRDB データプロバイダ for .NET Framework が生成する例外オブジェクトのプロパティ「ErrorCode」には，HiRDB Client Library 又は HiRDB データプロバイダ for .NET Framework 固有のエラーコードが格納されます。

   エラーコードが 3 けた(-XXX)又は 4 けた(-XXXX)の場合は KFPA1XXXX を示し，5 けた(-24XXX)の場合は KFPZ24XXX を示します。

15.11.2　SQL 文の実行

表 ex を作成する例を次に示します。

- Visual C# .NET で記述した例

```
using System;
using Hitachi.HiRDB;

namespace test_C
{
    class Sample
    {
        [STAThread]
        static void Main(string[] args)
        {
            try
            {
                // Connectionオブジェクトを作成する
                HiRDBConnection cn = new HiRDBConnection("dsn=pc;");

                // DBに接続する
                cn.Open();

                // Commandオブジェクトを作成する
                HiRDBCommand cm = new HiRDBCommand();
                // 表を作成する
                cm.Connection = cn;
                cm.CommandText = "create table ex (a int)";
                cm.ExecuteNonQuery();  .................................1

                // DBから切断する
                cn.Close();
                cn.Dispose();
            }
            catch ( HiRDBException ex)
            {
                Console.WriteLine(ex);
            }
            catch ( System.Exception ex)
            {
                Console.WriteLine(ex);
            }
        }
    }
}
```

- Visual Basic.NET で記述した例

```
Imports System
Imports System.Data
Imports Hitachi.HiRDB

Module Module1

    Sub Main()
```

```
        Dim cn As HiRDBConnection
        Dim cm As HiRDBCommand
        Try
            ’ Connectionオブジェクトを作成する
            cn = New HiRDBConnection(”dsn=pc;”)

            ’ DBに接続する
            cn.Open()

            ’ Commandオブジェクトに作成する
            cm = New HiRDBCommand()
            ’ 表を作成する
            cm.Connection = cn
            cm.CommandText = ”create table ex (a int)”
            cm.ExecuteNonQuery()  .....................................1

            ’ DBから切断する
            cn.Close()
            cn.Dispose()
        Catch ex As HiRDBException
            Console.WriteLine(ex)

        Catch ex As System.Exception

            Console.WriteLine(ex)

        End Try

    End Sub

End Module
```

[説明]

1. SQL 文を実行する場合，Execute メソッドを使用します。HiRDBCommand の CommandText プロパティに，string 型の SQL 文をそのまま記述します。このメソッドでほとんどの SQL 文が実行できます。ただし，「commit」などの特殊な SQL 文はこのメソッドでは実行できません。また，結果セットを受け取る必要のある「select」も実行できません。これらの SQL 文を実行する場合は，専用のメソッドを使用します。

15.11.3　トランザクションの実行

表 ex にデータ「1」を挿入する例を次に示します。

- Visual C# .NET で記述した例

```
using System;
using System.Data;
using Hitachi.HiRDB;

namespace test_C
{
    class Sample
```

```
    {
        [STAThread]
        static void Main(string[] args)
        {
            // Connectionオブジェクトを作成する
            HiRDBConnection cn = new HiRDBConnection("dsn=pc;");

            // DBに接続する
            cn.Open();

            // Transactionオブジェクトを作成する
            HiRDBTransaction tran;
            // トランザクション開始
            tran = cn.BeginTransaction(IsolationLevel.ReadCommitted);  ..1
            // Commandオブジェクトを作成する
            HiRDBCommand cm = new HiRDBCommand();
            cm.Connection = cn;
            cm.Transaction = tran;
            try
            {
                // 表にデータを挿入する
                cm.CommandText = "insert into ex values (1)";
                cm.ExecuteNonQuery();

                // トランザクション成功
                tran.Commit();  .....................................2

                // DBから切断する
                cn.Close();
                cn.Dispose();
            }
            catch ( HiRDBException ex)
            {
                // トランザクション失敗
                if(tran.IsCompleted != true){
                    tran.Rollback();  ..............................3

                Console.WriteLine(ex);
            }
            catch ( System.Exception ex)
            {
                // トランザクション失敗
                if(tran.IsCompleted != true){
                    tran.Rollback();  ..............................3

                Console.WriteLine(ex);
            }
        }
    }
}
```

- Visual Basic.NET で記述した例

```
Imports System
Imports System.Data
Imports Hitachi.HiRDB

Module Module1
```

```
    Sub Main()

        Dim cn As HiRDBConnection
        Dim tran As HiRDBTransaction
        Dim cm As HiRDBCommand
        ’ Connectionオブジェクトを作成する
        cn = New HiRDBConnection(”dsn=pc;”)

        ’ DBに接続する
        cn.Open()

        ’ トランザクション開始
        tran = cn.BeginTransaction(IsolationLevel.ReadCommitted)  .......1
        ’ Commandオブジェクトを作成する
        cm = New HiRDBCommand()
        cm.Connection = cn
        cm.Transaction = tran

        Try
            ’ 表にデータを挿入する
            cm.CommandText = ”insert into ex values (1)”
            cm.ExecuteNonQuery()

            ’ トランザクション成功
            tran.Commit()  ..............................................2

            ’ DBから切断する
            cn.Close()
            cn.Dispose()

        Catch ex As HiRDBException
            ’ トランザクション失敗
            If tran.IsCompleted <> True Then
                tran.Rollback()  ........................................3
            End If
            Console.WriteLine(ex)

        Catch ex As System.Exception

            ’ トランザクション失敗
            If tran.IsCompleted <> True Then
                tran.Rollback()  ........................................3
            End If
            Console.WriteLine(ex)

        End Try

    End Sub

End Module
```

[説明]

1. トランザクションを開始する場合，BeginTransaction メソッドを使用します。
2. トランザクションを完了する場合，Commit メソッドを呼びます。

3. 元に戻す場合は，Rollback メソッドを呼びます。

15.11.4 検索文の実行

表のデータをすべて表示する例を次に示します。

プログラム例は Visual C# .NET で記述していますが，Visual Basic.NET でもほぼ同じ内容です。必要に応じて，読み替えてください。

```
using System;
using System.Data;
using Hitachi.HiRDB;

namespace test_C
{
    class Sample
    {
        [STAThread]
        static void Main(string[] args)
        {
            try
            {
                // Connectionオブジェクトを作成する
                HiRDBConnection cn = new HiRDBConnection("dsn=pc;");

                // DBに接続する
                cn.Open();

                // Commandオブジェクトを作成する
                HiRDBCommand cm = new HiRDBCommand();
                cm.Connection = cn;
                cm.CommandText = "select a from ex";
                // DataReaderオブジェクトを作成する
                HiRDBDataReader rd = cm.ExecuteReader();  ..............1
                int i;
                while(rd.Read())
                {
                    for (i = 0 ; i < rd.FieldCount ; i++)
                    {
                     Console.WriteLine(rd.GetName(i) + " - " +rd.GetValue(i));
                    }
                }  ..................................................2
                // DBから切断する
                cn.Close();
                cn.Dispose();
            }
            catch ( HiRDBException ex)
            {
                Console.WriteLine(ex);
            }
            catch ( System.Exception ex)
            {
                Console.WriteLine(ex);
            }
```

```
        }
    }
}
```

[説明]

1. 検索を実行する場合，ExecuteReader メソッドを使用し，HiRDBDataReader を構築します。
2. Read メソッドを使用し，次の行に進めます。列の名前を取得する場合は GetName メソッドを使用し，列の値を取得する場合は GetValue メソッドを使用します。

15.11.5　配列を使用した INSERT 機能の実行

表 ex に「123」，「200」，及び「null」を挿入する例を次に示します。

プログラム例は Visual C# .NET で記述していますが，Visual Basic.NET でもほぼ同じ内容です。必要に応じて，読み替えてください。

```
    // 接続オブジェクトなどを作成する
    HiRDBConnection    pConn = new HiRDBConnection(”接続文字列”);
    HiRDBCommand    pCom = pConn.CreateCommand();

    // DBに接続する
    pConn.Open();

    // パラメタオブジェクトを作成する
    HiRDBParameter    pPar = pCom.CreateParameter();

    // パラメタの設定をする
    pPar.Direction = ParameterDirection.Input;
    pPar.HiRDBType = HiRDBType.Integer;
    object [] aValue = new object[3];
    aValue[0] = 123;
    aValue[1] = 200;
    aValue[2] = null;
    pPar.Value = aValue;
    pCom.Parameters.Add(pPar);  .....................................1
    // パラメタを使用してSQL文を実行する
    pCom.CommandText = ”insert into ex values(?)”;
    pCom.ExecuteNonQuery(aValue.Length);  ...........................2

    // DBから切断する
    pConn.Close();
    pConn.Dispose();
```

[説明]

1. パラメタの value には，パラメタの値を設定します。value は object 型のため，すべての型を参照できます。通常の INSERT 文では Int32 型を指定しますが，配列を使用した INSERT 文の場合は object の配列を value に設定します。そして，object 配列の各要素が Int32 型を指すように設定します。ほかの型を使う場合も同様で，value には必ず object の配列を設定します。

2. SQL の実行には ExecuteNonQuery の：overload を使用します。通常の ExecuteNonQuery には引数はありませんが，配列を使用した INSERT 文を使用する場合は，配列の大きさを指定します。

注

配列を使用した場合と使用しない場合とでは，パラメタの value を設定する部分と，SQL を実行する部分が異なります。

15.11.6　繰返し列の実行

表 ex の 2 列目に「123」,「456」，及び「NULL」を挿入する例を次に示します。

プログラム例は Visual C# .NET で記述していますが，Visual Basic.NET でもほぼ同じ内容です。必要に応じて，読み替えてください。

```
    // 接続オブジェクトなどを作成する
    HiRDBConnection    pConn = new HiRDBConnection(”接続文字列”);
    HiRDBCommand    pCom = pConn.CreateCommand();

    // DBに接続する
    pConn.Open();

    // 表を作成する
    pCom.CommandText = ”create table ex(a int,b char(10) array[3])”;
    pCom.ExecuteNonQuery();

    // パラメタオブジェクトを作成する
    HiRDBParameter    pPar1 = pCom.CreateParameter();
    HiRDBParameter    pPar2 = pCom.CreateParameter();

    // パラメタの設定をする
    pPar1.Direction = ParameterDirection.Input;
    pPar1.HiRDBType = HiRDBType.Integer;
    pPar1.Value = 1;

    pPar2.Direction = ParameterDirection.Input;
    pPar2.HiRDBType = HiRDBType.Char;
    pPar2.Size = 10;
    object [] aValue = new object[3];
    aValue[0] = ”abc”;
    aValue[1] = ”def”;
    aValue[2] = null;
    pPar2.Value = aValue;

    // 表exの列aの最大要素数を設定する
    pPar2.Repetition =3;

    pCom.Parameters.Add(pPar1);
    pCom.Parameters.Add(pPar2);

    // パラメタを使用してSQL文を実行する
    pCom.CommandText = ”insert into ex values(?,?)”;
    pCom.ExecuteNonQuery();
```

```
    // select文を実行する
    pCom.CommandText = ”select * from ex”;
    HiRDBDataReader pReader = pCom.ExecuteReader();

    // フェッチする
    pReader.Read();

    // 1列目のデータを取得する
    if(!pReader.IsDBNull(0))
    {
        Console.WriteLine(”1列目の値は” + pReader.GetInt32(0));
    }
    else
    {
        Console.WriteLine(”1列目の値はNULL”);
    }

    // 2列目（繰返し列）のデータを取得する
    if(!pReader.IsDBNull(1))
    {
        for (int i = 0; i < pReader.GetFieldArrayCount(1); i++)
        {
            if (!pReader.IsDBNull(1, i))
            {
                Console.WriteLine(”2列目の” + (i+1) + ”番目の要素は”+ pReader.GetValue(1, i)
);
            }
            else
            {
                Console.WriteLine(”2列目の” + (i+1) + ”番目の要素は NULL”);
            }
        }
    }
    else
    {
        Console.WriteLine(”2列目の値はNULL”);
    }

    // カーソルクローズ
    pReader.Close();
    // DBから切断する
    pConn.Close();
    pConn.Dispose();
```

[説明]

1. value に object 配列を設定するのは，配列を使用した INSERT 文と同じです。繰返し列の場合，更に拡張プロパティ Repetition を設定します。これには繰返し列の数を指定します。そのため，SQL 実行時の引数は必要ありません。
2. FETCH の場合，DataReader にも繰返し列のための拡張メソッドが用意されています。まず，FETCH したデータの繰返し列の個数を GetFieldArrayCount で取得します。さらに，FETCH したデータの値を GetValue の：overload で取得します。第 2 引数には繰返し列の列番号を指定します。また，このメソッドと等価なインデクサ[int,int]も用意しています。

注

繰返し列の使用方法は，配列を使用した INSERT 機能に似ています。異なるのは，パラメタに繰り返し回数を指定する部分と，SQL 文を実行する部分です。

15.11.7　SQL 文のエラー判定とエラー情報の取得

SQL 文のエラー判定をして，エラーがある場合はエラー情報を取得する例を次に示します。

プログラム例は Visual C# .NET で記述していますが，Visual Basic.NET でもほぼ同じ内容です。必要に応じて，読み替えてください。

```
using System;
using System.IO;
using System.Data;
using System.Windows.Forms;
using Hitachi.HiRDB;

namespace SAMPLE
{
    public class SAMPLE
    {
        static void Main()
        {

            HiRDBConnection connection1 = new HiRDBConnection
            (”datasource=C:¥¥Windows¥¥HiRDB.ini;UID=USER1;PWD=USER1;”);
            HiRDBCommand cm = new HiRDBCommand();

                try
                {
                    // 接続実行  .....................................1
                    connection1.Open();
                    cm.Connection = connection1;

                    // ******************************************
                    // SAMPLE1(C1 INT,C2 INT,C3 VARCHAR(30))の検索例  ...2
                    // ******************************************
                    // パラメタオブジェクトを作成する
                    HiRDBParameter    par = cm.CreateParameter();
                    // パラメタ属性設定
                    // 入力パラメタ
                    par.Direction = ParameterDirection.Input;
                    // INTEGER型
                    par.HiRDBType = HiRDBType.Integer;
                    int aValue;
                    aValue = 200;
                    // パラメタ値設定
                    par.Value = aValue;

                    // SQL設定
                    cm.CommandText = ”SELECT C2,C3 FROM SAMPLE1 WHERE C1=?
                                      WITH EXCLUSIVE LOCK NO WAIT”;
                    // パラメタオブジェクトの割り当て
```

```
                    cm.Parameters.Add(par);
                    // DataReaderオブジェクト取得
                    HiRDBDataReader dr = cm.ExecuteReader();
                    int cnt=1;
                    Console.WriteLine("**** 検索実行 ****");
                    while(dr.Read())
                    {
                        Console.WriteLine("**** "+cnt+"行目検索 ***");
                        // C2のデータ表示
                        Console.WriteLine("C2="+dr.GetInt32(0));
                        // C3のデータ表示
                        Console.WriteLine("C3="+dr.GetString(1));
                        cnt ++;
                    }
                    // DataReaderの解放
                    dr.Close();
                    // 切断  .....................................3
                    connection1.Close();
                    connection1.Dispose();

                }
                catch (HiRDBException ex)  ..........................4
                {
                    // エラー情報出力
                    Console.WriteLine(ex);
                    // 個別情報は以下の処理で取得
                    // SQLCODEの出力
                    Console.WriteLine("SQLCODE="+ex.ErrorCode);
                    // SQLERRM(SQLメッセージ)の出力
                    Console.WriteLine("SQLERRM=" + ex.Message);
                }
        }
    }
}
```

[説明]

1. Open メソッドを使用して HiRDB へ接続します。
2. 指定した条件と一致する行を表示させる SQL 文を実行します。
3. Close メソッドを使用して HiRDB から切断します。
4. エラーになった場合は SQLException を返し，エラー情報を出力します。

15.12　パラメタのデータ長及びデータタイプの仮定

.NET Framework 2.0 以降に対応した HiRDB データプロバイダ for .NET Framework では，HiRDBParameter オブジェクトの Value プロパティを指定した場合，指定値からデータ長及びデータタイプを仮定するため，データ長及びデータタイプのプロパティの設定を省略できます。

15.12.1　省略できるプロパティ

HiRDBParameter オブジェクトの Value プロパティ指定時に省略できる HiRDBParameter クラスのプロパティは，パラメタ種別によって異なります。Value プロパティ指定時に省略できるプロパティ及びパラメタ種別の関係を次に示します。

表 15–15　Value プロパティ指定時に省略できるプロパティ及びパラメタ種別

プロパティ名	説明	パラメタ種別		
		入力パラメタ	出力パラメタ	入出力パラメタ
DbType	.NET Framework でのデータタイプ	●	×	●※2
Direction	パラメタの方向種別	○	×	×
HiRDBType	HiRDB でのデータタイプ	●	×	●※2
IsNullable	null 値の受け入れ可否	○	○	○
ParameterName	HiRDBParameter の名前	○	○	○
Precision	DECIMAL 型の有効けた数	●	×	●※2
Repetition	HiRDB での配列構造数	○	○	○
Scale	DECIMAL 型の少数部けた数	●	×	●※2
Size	パラメタの定義長	●	×※1	●※2
SourceColumn	DataSet 連携時に使用されるソース列の名前	○	○	○
SourceColumnNullMapping	パラメタに対応する DataTable の列の null 値の受け入れ可否	○	○	○
SourceVersion	Value プロパティ値の読み込みに使用する DataRowVersion	○	○	○

（凡例）

○：省略できます。指定を省略した場合，既定値で動作します。既定値については，「HiRDBParameter」 を参照してください。

●：省略できます。指定を省略した場合，HiRDB データプロバイダ for .NET Framework が Value プロパティの指定値から値を仮定します。ただし，明示的に値を指定した場合は，HiRDB データプロバイダ for .NET Framework は以降の処理でプロパティの値を仮定しません。

×：省略できません。

注※1

HiRDB でのデータタイプが次の場合は，指定する必要はありません。

- DECIMAL
- SMALLINT
- INTEGER
- FLOAT
- DATE
- TIME
- INTERVAL YEAR TO DAY
- INTERVAL HOUR TO SECOND

注※2

入力パラメタとして扱う場合に仮定した値を，出力パラメタとして扱う場合にも適用します。

15.12.2 HiRDB データプロバイダ for .NET Framework が仮定するデータタイプ

HiRDBParameter オブジェクトの Value プロパティの指定値から，HiRDB データプロバイダ for .NET Framework が仮定する HiRDBType，及び HiRDBType に対応する DbType を次に示します。

表 15–16　Value プロパティ指定値から仮定する HiRDBType 及び DbType

Value プロパティ指定値のデータ型	HiRDB データプロバイダ for .NET Framework が仮定する HiRDBType	HiRDBType に対応する DbType
Boolean	–	–
Int16	SmallInt	Int16
Int32	Integer	Int32
Int64	Decimal	Int64
UInt16	–	–
UInt32	–	–
UInt64	–	–
Single	SmallFlt	Single
Double	Float	Double
Decimal	Decimal	Decimal
Char	MChar	StringFiexedLength

Value プロパティ指定値のデータ型	HiRDB データプロバイダ for .NET Framework が仮定する HiRDBType	HiRDBType に対応する DbType
Char[]	−	−
String	MVarChar	String
DateTime	TimeStamp	DateTime
TimeSpan	IntervalHourToSecond	String
Guid	−	−
Byte	SmallInt	Byte
Byte[]	Binary	Object
Sbyte	SmallInt	SByte
SByte[]	Binary	Object

（凡例）

−：該当するデータ型からの仮定はサポートしていません。

「−」に該当するデータ型が Value プロパティに指定された場合，HiRDBType プロパティは既定値である MVarChar を適用します。

15.12.3　HiRDB データプロバイダ for .NET Framework が仮定するデータ長

HiRDBParameter オブジェクトの Value プロパティの指定値から，HiRDB データプロバイダ for .NET Framework が仮定するデータ長を次に示します。

Value プロパティ指定値のデータ型	HiRDB データプロバイダ for .NET Framework が仮定するデータ長		
	Size プロパティ	Presition プロパティ	Scale プロパティ
Boolean	×	×	×
Int16	2	−	−
Int32	4	−	−
Int64	−	○	○
UInt16	×	×	×
UInt32	×	×	×
UInt64	×	×	×
Single	4	−	−
Double	8	−	−
Decimal	−	○	○
Char	1	−	−

Value プロパティ指定値のデータ型	HiRDB データプロバイダ for .NET Framework が仮定するデータ長		
	Size プロパティ	Presition プロパティ	Scale プロパティ
Char[]	×	×	×
String	○	−	−
DateTime	○	−	−
TimeSpan	4	−	−
Guid	×	×	×
Byte	1	−	−
Byte[]	○	−	−
Sbyte	1	−	−
SByte[]	○	−	−

（凡例）

○：HiRDB データプロバイダ for .NET Framework が値を仮定します。

×：該当するデータ型からの仮定はサポートしていません。

数値：データタイプから無条件で求めるデータ長になります。

−：HiRDB データプロバイダ for .NET Framework が値を仮定しません。

HiRDB データプロバイダ for .NET Framework が実行時に仮定した値は，Size，Precision 及び，Scale プロパティには設定しません。

15.12.4 プロパティの明示的な指定の有無と HiRDB データプロバイダ for .NET Framework による仮定可否

HiRDB データプロバイダ for .NET Framework が仮定を行うプロパティに対して，明示的に値を指定した場合，HiRDB データプロバイダ for .NET Framework は以降の処理でプロパティの値を仮定しません。各プロパティへの明示的な指定の有無に対する HiRDB データプロバイダ for .NET Framework の仮定可否を次に示します。

表 15–17　プロパティの明示的な指定の有無に対するプロパティの仮定可否

パラメタ種別	各プロパティの明示的な指定の有無					プロパティの仮定可否				
Direction	HiRDBType	DbType	Size	Precision	Scale	HiRDBType	DbType	Size	Precision	Scale
Input InputOutput	有	有	有	有	有	×	×	×	×	×
					無	×	×	×	×	○
				無	有	×	×	×	○	×
					無	×	×	×	○	○

パラメタ種別	各プロパティの明示的な指定の有無					プロパティの仮定可否				
Direction	HiRDBType	DbType	Size	Precision	Scale	HiRDBType	DbType	Size	Precision	Scale
			無	有	有	×	×	○	×	×
					無	×	×	○	×	○
				無	有	×	×	○	○	×
					無	×	×	○	○	○
		無	有	有	有	×	×	×	×	×
					無	×	×	×	×	○
				無	有	×	×	×	○	×
					無	×	×	×	○	○
			無	有	有	×	×	○	×	×
					無	×	×	○	×	○
				無	有	×	×	○	○	×
					無	×	×	○	○	○
	無	有	有	有	有	×	×	×	×	×
					無	×	×	×	×	○
				無	有	×	×	×	○	×
					無	×	×	×	○	○
			無	有	有	×	×	○	×	×
					無	×	×	○	×	○
				無	有	×	×	○	○	×
					無	×	×	○	○	○
		無	有	有	有	○	○	×	×	×
					無	○	○	×	×	○
				無	有	○	○	×	○	×
					無	○	○	×	○	○
			無	有	有	○	○	○	×	×
					無	○	○	○	×	○
				無	有	○	○	○	○	×
					無	○	○	○	○	○
Output	−	−	−	−	−	×	×	×	×	×

（凡例）

○：HiRDB データプロバイダ for .NET Framework が値を仮定します。

×：HiRDB データプロバイダ for .NET Framework が値を仮定しません。

−：プロパティの指定値に依存しません。

15.12.5　仮定を適用するプロパティの初期化

HiRDB データプロバイダ for .NET Framework は，プロパティに対して明示的な値が指定された場合，以降の処理でプロパティの値を仮定しません。しかし，HiRDBParameter クラスの ResetDbType メソッドを実行した場合は，HiRDBParameter オブジェクトの DbType プロパティを初期値に戻すため，再び HiRDBParameter クラスの DbType プロパティ及び HiRDBType プロパティについて仮定ができます。データ長については初期化しないため，明示的に指定した場合，HiRDB データプロバイダ for .NET Framework が再びデータ長を仮定することはありません。HiRDB データプロバイダ for .NET Framework による仮定を適用するプロパティの初期化可否を次に示します。

表 15–18　仮定を適用するプロパティの初期化可否

仮定するプロパティ	初期化可否
HiRDBType	○
DbType	○
Size	×
Precision	×
Scale	×

（凡例）

○：初期化できます。

×：初期化できません。

15.12.6　仮定値の取得

HiRDBParameter オブジェクトの Value プロパティに値を指定し，HiRDB データプロバイダ for .NET Framework がデータタイプ及びデータ長を仮定して SQL を実行した場合，SQL 実行時に HiRDB データプロバイダ for .NET Framework が仮定したデータ長は Size，Precision，Scale の各プロパティにアクセスしても取得できません。HiRDB データプロバイダ for .NET Framework が仮定したプロパティ値の取得可否を次に示します。

表 15–19　プロパティ仮定値の取得可否及び取得値

仮定するプロパティ	仮定値の取得可否	実際に取得できる値
HiRDBType	○	仮定値

仮定するプロパティ	仮定値の取得可否	実際に取得できる値
DbType	○	仮定値
Size	×	0
Precision	×	0
Scale	×	0

（凡例）

○：取得できます。

×：取得できません。

15.12.7　注意事項

SQL 実行時に HiRDB データプロバイダ for .NET Framework が仮定するデータタイプが対象表の列定義と異なる場合，マニュアル「HiRDB SQL リファレンス」の「変換（代入，比較）できるデータ型」に示す規則に従い，HiRDB サーバが変換します。この変換によるオーバヘッドを削減する場合は，対象表の列定義に合わせてデータ長及びデータタイプを明示的に指定してください。

15.13　構成ファイル

.NET Framework 2.0 以降に対応した HiRDB データプロバイダ for .NET Framework では，.NET Framework の構成ファイルに，HiRDBConnection クラスのプロパティ指定と同様の指定ができます。構成ファイルを使用することで，アプリケーションを変更することなく，各機能の適用有無を変更できるようになります。

15.13.1　適用契機

構成ファイルに指定した HiRDB データプロバイダ for .NET Framework の設定値は，HiRDBConnection クラスの Open メソッド実行時に適用します。

15.13.2　サポートする構成ファイル

HiRDB データプロバイダ for .NET Framework がサポートする.NET Framework の構成ファイルを次の表に示します。

表 15–20　HiRDB データプロバイダ for .NET Framework がサポートする構成ファイル

構成ファイル種別	説明
アプリケーション構成ファイル	アプリケーション単位に適用する設定を記述するファイル
マシン構成ファイル	コンピュータ全体に適用する設定を記述するファイル

15.13.3　HiRDB データプロバイダ for .NET Framework がサポートする指定可能なキー

構成ファイルに指定可能なキー，及びキーに対応する HiRDBConnection クラスのプロパティを次の表に示します。各キーの詳細は，「HiRDBConnection」の「プロパティ」を参照してください。

表 15–21　HiRDB データプロバイダ for .NET Framework がサポートするキー

キー	プロパティ	説明
HiRDB_ConnectionString	ConnectionString	接続文字列を指定します。
HiRDB_Pooling	Pooling	接続プーリング機能の適用有無を指定します。
HiRDB_LifeTime	LifeTime	接続プーリング機能使用時の接続待機時間を指定します。
HiRDB_ConnectionRecover	ConnectionRecover	接続プーリング機能使用時の自動再接続機能の適用有無を指定します。

15.13.4　指定方法

HiRDB データプロバイダ for .NET Framework での構成ファイルの指定は，構成ファイルの<appSettings>要素に指定します。指定には<appSettings>要素の子要素<add>を使用し，<add>要素には「key」と「value」の二つの属性を指定する必要があります。「key」属性には表「HiRDB データプロバイダ for .NET Framework がサポートするキー」に示すキーを，「value」属性にはキーに指定する値を指定してください。「key」及び「value」属性に対する指定の規則を次に示します。

(1)　key に対する指定の規則

- 英大文字と英小文字を区別しません。
- 空白文字など，指定値はすべて有効な文字として扱います。
- 表「HiRDB データプロバイダ for .NET Framework がサポートするキー」に示すキー以外を指定した場合，HiRDB データプロバイダ for .NET Framework はその指定を無視します。

(2)　value に対する指定の規則

- HiRDB_ConnectionString 以外の指定値は，英大文字と英小文字を区別しません。
- 空白文字など，指定値はすべて有効な文字として扱います。
- HiRDB_ConnectionString の指定値は HiRDBConnection クラスの ConnectionString プロパティの指定規則に準じます。詳細は，「HiRDBConnection」の「プロパティ」の「ConnectionString」を参照してください。
- 空文字("")を指定した場合，HiRDB データプロバイダ for .NET Framework は対象の key について未指定として扱います。

15.13.5　指定値の優先順位

HiRDB データプロバイダ for .NET Framework での HiRDBConnection クラスのプロパティと各構成ファイルの優先順位を次の表に示します。

表 15–22　HiRDB データプロバイダ for .NET Framework における指定値の優先順位

優先順位	指定値の指定場所
1	アプリケーション構成ファイル
2	マシン構成ファイル
3	HiRDBConnection クラスのプロパティ

ただし，アプリケーション構成ファイルの任意の key に対応する value に空文字("")を指定した場合，マシン構成ファイルのアプリケーション構成ファイルと同一の key に対する指定は.NET Framework によって無視されます。この場合，HiRDB データプロバイダ for .NET Framework は前述のとおり対象の key を

未指定として扱うため，マシン構成ファイルに有効な値を指定しているときでも，HiRDBConnection クラスのプロパティへの指定が有効となることに注意してください。他の設定方法も含めた指定値の優先順位に関する詳細は，「接続情報の優先順位」を参照してください。

15.13.6　指定例

HiRDB データプロバイダ for .NET Framework での構成ファイルの指定例を次に示します。

HiRDB データプロバイダ for .NET Framework における構成ファイルの指定例

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
    <appSettings>
        <add key="HiRDB_Pooling"            value="true" />
        <add key="HiRDB_LifeTime"           value="3600" />
        <add key="HiRDB_ConnectionRecover"  value="true" />
    </appSettings>
</configuration>
```

15.13.7　例外

構成ファイル指定時に発生する可能性のある例外 HiRDBException について，発生条件を次の表に示します。

表 15–23　構成ファイル指定時に発生する可能性のある例外の発生条件

発生条件	出力される例外メッセージ
HiRDBConnection クラスの Open メソッド実行時に，マシン構成ファイル又はアプリケーション構成ファイルの読み込みに失敗しました。	KFPZ24037-E
HiRDB データプロバイダ for .NET Framework がサポートする構成ファイルに指定可能なキーに誤った値を指定しました。	KFPZ24038-E

15.13.8　.NET Framework の仕様に関連する注意事項

- マシン構成ファイルは，.NET Framework のバージョンごとに存在します。マシン構成ファイルを使用する場合は，アプリケーションを実行する.NET Framework のバージョンに対応したマシン構成ファイルを編集してください。
- アプリケーションの実行中に構成ファイルの指定を変更した場合，変更した値はアプリケーションの再起動まで有効になりません。構成ファイルの指定を変更する場合，次の手順で行ってください。
  1. アプリケーションが実行中の場合は終了する。
  2. 構成ファイルの指定を変更する。

3. アプリケーションを起動する。

- 構成ファイルに重複したキーを指定した場合，最後に指定したキーに対応する値が有効となります。

15.14　接続情報の優先順位

HiRDB データプロバイダ for .NET Framework では，意味が同じ接続情報を複数の設定方法で指定できます。同時に複数の設定方法で設定された場合の接続情報の優先順位を次の表に示します。

表 15–24　接続情報の優先順位

接続情報の意味	設定方法		優先順位
接続時の認可識別子，パスワード	アプリケーション構成ファイルのキー HiRDB_ConnectionString の指定	「uid 又は userid」及び「password 又は Pwd」	1
		「PDUSER」	2
		「datasource」,「dsn」又は「env」で設定した，環境変数グループ又は環境変数グループファイル内の PDUSER	3
	マシン構成ファイルのキー HiRDB_ConnectionString の指定	「uid 又は userid」及び「password 又は Pwd」	4
		「PDUSER」	5
		「datasource」,「dsn」又は「env」で設定した，環境変数グループ又は環境変数グループファイル内の PDUSER	6
	HiRDBConnection クラスのプロパティ ConnectionString の指定	「uid 又は userid」及び「password 又は Pwd」	7
		「PDUSER」	8
		「datasource」,「dsn」又は「env」で設定した，環境変数グループ又は環境変数グループファイル内の PDUSER	9
	ユーザ環境変数 PDUSER		10
	システム環境変数 PDUSER		11
	Windows ディレクトリ下の HiRDB.INI ファイル内の PDUSER		12
接続プーリング機能を使用するかどうか	アプリケーション構成ファイルのキー HiRDB_Pooling		1
	マシン構成ファイルのキー HiRDB_Pooling		2
	HiRDBConnection クラスのプロパティ Pooling		3
接続プーリング機能でのプーリング時間	アプリケーション構成ファイルのキー HiRDB_LifeTime		1
	マシン構成ファイルのキー HiRDB_LifeTime		2
	HiRDBConnection クラスのプロパティ LifeTime		3
接続プーリングの接続回復機能を使用するかどうか	アプリケーション構成ファイルのキー HiRDB_ConnectionRecover		1
	マシン構成ファイルのキー HiRDB_ConnectionRecover		2
	HiRDBConnection クラスのプロパティ ConnectionRecover		3
PDUSER 以外のクライアント環境定義	アプリケーション構成ファイルのキー	「PD で始まるクライアント環境定義」	1

接続情報の意味	設定方法		優先順位
	HiRDB_ConnectionString の指定	「datasource」,「dsn」又は「env」で設定した，環境変数グループ又は環境変数グループファイル	2
	マシン構成ファイルのキー HiRDB_ConnectionString の指定	「PD で始まるクライアント環境定義」	3
		「datasource」,「dsn」又は「env」で設定した，環境変数グループ又は環境変数グループファイル	4
	HiRDBConnection クラスのプロパティ ConnectionString の指定	「PD で始まるクライアント環境定義」	5
		「datasource」,「dsn」又は「env」で設定した，環境変数グループ又は環境変数グループファイル	6
	ユーザ環境変数		7
	システム環境変数		8
	Windows ディレクトリ下の HiRDB.INI ファイル		9

16 Type2 JDBC ドライバ

この章では，Type2 JDBC ドライバのインストール，環境設定，JDBC の機能などについて説明します。

以降，この章では，Type2 JDBC ドライバを単に「JDBC ドライバ」と表記します。

なお，Type2 JDBC ドライバは廃止になりました。現在はアプリケーションの互換性を保つために使用できますが，将来は削除されます。代わりに Type4 JDBC ドライバを使用してください。

Type2 JDBC ドライバを使用する場合は，HiRDB サーバとの接続時にパスワードを指定するユーザインタフェースで，29 バイト以上のパスワードを引用符で囲んだ指定はできません。

16.1　インストールと環境設定

16.1.1　インストール

JDBC ドライバのインストールは，HiRDB インストール時に選択します。

JDBC ドライバのインストールディレクトリとファイルを次の表に示します。

表 16–1　JDBC ドライバのインストールディレクトリとファイル

プラットフォーム	種別	インストールディレクトリ	ファイル
UNIX	HiRDB サーバ	$PDDIR/client/lib/	pdjdbc.jar※ libjjdbc.sl(libjjdbc.so)※
	HiRDB クライアント	/HiRDB/client/lib/	pdjdbc.jar※ libjjdbc.sl(libjjdbc.so)※
Windows	HiRDB サーバ	%PDDIR%¥CLIENT¥UTL¥	pdjdbc.jar※ jjdbc.dll※
	HiRDB クライアント	¥HiRDB¥CLIENT¥UTL¥	pdjdbc.jar※ jjdbc.dll※

注

　下線で示す部分は，HiRDB クライアントのインストールディレクトリとなります。

注※

　Windows (x64)版，Linux(EM64T)版の場合は，32Bit モードで動作させてください。

16.1.2　環境設定

JDBC ドライバが動作するときに必要となる，環境変数の設定を次に示します。

(1)　UNIX 環境の場合

実行環境の環境変数に，次の内容を設定してください。

```
CLASSPATH=$CLASSPATH:[インストールディレクトリ]/pdjdbc.jar
```

(2) Windows 環境の場合

［コントロールパネル］－［システム］－［システムのプロパティ］の「環境」に，次の内容を設定してください。

```
CLASSPATH=%CLASSPATH%;[インストールディレクトリ]¥pdjdbc.jar
```

16.1.3 メソッドの略記について

- 先頭に「get」が付くメソッドをまとめて表す場合，getXXX メソッドと表記します。
- 先頭に「set」が付くメソッドをまとめて表す場合，setXXX メソッドと表記します。

16.2　JDBC1.0 機能

16.2.1　Driver クラス

(1)　概要

Driver クラスでは，次の機能が提供されます。

- DB 接続
- 指定した URL の妥当性チェック
- DriverManager.getConnection メソッドで指定する接続プロパティの情報取得
- ドライババージョンの返却

Driver クラスで提供される各メソッドの詳細，使用方法については，JDBC の関連ドキュメントを参照してください。ここでは，DB 接続をするときの手順，及びこの JDBC ドライバ独自の URL の構文について説明します。

(2)　DriverManager を使用した DB 接続

DB 接続は，Java 実行環境が提供する DriverManager クラスを使用して，次の手順で実行します。

1. Driver クラスを Java 仮想マシンに登録します。
2. 接続情報を引数にして，DriverManager.getConnection メソッドを呼び出します。

(a)　Driver クラスの Java 仮想マシンへの登録

Class.forName メソッドの使用，又はシステムプロパティへの登録で，Driver クラスを Java 仮想マシンに登録します。登録するときに指定する，JDBC ドライバのパッケージ名称と Driver クラス名称を次に示します。

パッケージ名称：JP.co.Hitachi.soft.HiRDB.JDBC

Driver クラス名称：PrdbDriver

- **Class.forName メソッドの使用**

アプリケーション内で，次のように Class.forName メソッドを呼び出します。

```
Class.forName("JP.co.Hitachi.soft.HiRDB.JDBC.PrdbDriver");
```

- **システムプロパティへの登録**

アプリケーション内で，次のように System.setProperty メソッドを呼び出します。

```
System.setProperty("jdbc.drivers","JP.co.Hitachi.soft.HiRDB.JDBC.PrdbDriver");
```

(b)　接続情報の設定と DB 接続

DB と接続する場合，次のどれかの方法で実行してください。

- **DriverManager.getConnection メソッドの使用**

```
  Connection con = DriverManager.getConnection(String url, String user, String password)
;
  又は
  Connection con = DriverManager.getConnection(String url, Properties info) ;
```

- **内部ドライバでの記述**

内部ドライバで記述する場合，認可識別子などの接続情報は，HiRDB 側でルーチンを呼び出した外部ドライバの情報が仮定されます。ただし，JDBC ドライバ内部でトレースなどを取得する場合は，認可識別子として"INNER"が仮定されます。

```
＜内部ドライバ限定の記述＞
  Connection con = DriverManager.getConnection(String url) ;
```

- **Driver クラスの connect メソッドの直接呼び出し**

```
  Driver drv = new JP.co.Hitachi.soft.HiRDB.JDBC.PrdbDriver();
  Connection con = drv.connect(String url, Properties info) ;
```

上記の各メソッドの引数（url，user，password，及び info）には，DB 接続で必要な接続情報を設定します。

JDBC ドライバは，正常に DB 接続がされた場合，上記の各メソッドの呼び出しの結果として，Connection オブジェクトを返却します。必要な接続情報を各引数に設定していない場合，又は接続情報の内容が不正な場合は，上記の各メソッドの呼び出しの結果として，SQLException を投入します。

getConnection メソッドの引数の内容を次の表に示します。

表 16–2　getConnection メソッドの引数の内容

引数	内容	指定可否
String url	URL。URL については，「URL の構文」を参照してください。	○
String user	認可識別子※1	○※2
String password	パスワード	△
Properties info	表「Properties info の設定内容」を参照してください。	－

（凡例）

○：必ず指定してください。

△：指定は任意です。

－：該当しません。

注※1

認可識別子に null 又は空文字を指定した場合，SQLException を投入します。また，ドライバが文字コードを変換した結果，認可識別子に指定した文字列のサイズが 31 バイト以上となった場合，SQLException を投入します。文字コードの変換については，「文字コード変換機能」を参照してください。

注※2

内部ドライバで記述する場合は省略できます。

Properties info の設定内容を次の表に示します。

表 16–3　Properties info の設定内容

キー	内容	指定可否
user	認可識別子※1 です。	○※2
password	パスワードです。	△
ENCODELANG	Java プログラム内では，文字コードは Unicode で扱うため，HiRDB との文字データ処理時に，JDBC ドライバが HiRDB の文字データと Unicode との相互文字コード変換をします。この文字コード変換処理で，JDBC ドライバは Java 仮想マシンが提供するエンコーダ及びデコーダを利用します。この Java 仮想マシンが提供するエンコーダ及びデコーダに対して，JDBC ドライバが指定する文字セット名称を指定します。指定値は Java がサポートしている文字セット（MS932 など）です。 この Properties info で"OFF"を指定した場合，又は指定をしなかった場合（DataSource.setEncodeLang メソッドでの設定，及び URL の ENCODELANG での設定も含む）の動作については，「setEncodeLang」を参照してください。	△
COMMIT_BEHAVIOR	HiRDB がコミットをした場合に，次に示すクラスをコミット実行後も有効とするかどうかを設定します。 • ResultSet クラス • Statement クラス，PreparedStatement クラス，及び CallableStatement クラス 指定値については，「setCommit_Behavior」を参照してください。 注意事項： 「COMMIT_BEHAVIOR についての注意事項」を参照してください。	△
BLOCK_UPDATE	?パラメタを使用したデータベースの更新で，複数のパラメタセットを一度に処理するかどうかを設定します。省略した場合，"FALSE"が仮定されます。 "TRUE"： 一度に処理します。 "FALSE"： パラメタセットを一つずつ分割して処理します。	△

キー	内容	指定可否
	上記以外： "FALSE"を指定したものとみなします。 注意事項： • "TRUE"を設定した場合，バッチ更新機能で，HiRDB の配列を使用した機能を使用できます。 • 配列を使用した機能を利用できる SQL は，INSERT 文，UPDATE 文，及び DELETE 文だけです。それ以外の SQL の場合，一括実行されないで逐次実行で処理されます。 • 配列を使用した機能を利用できる SQL でも，配列を使用した機能の使用条件を満たさない場合は一括実行されないで，逐次実行で処理されます。 • 配列を使用した機能を利用する場合，「バッチ更新」を参照してください。 • 配列を使用した機能については，「配列を使用した機能」を参照してください。 • この機能は，システムプロパティの HiRDB_for_Java_BLOCK_UPDATE でも指定できます。ただし，BLOCK_UPDATE を設定した場合は，システムプロパティの HiRDB_for_Java_BLOCK_UPDATE の設定は無効となります。	
LONGVARBINARY_ACCESS	LONGVARBINARY（列属性が BLOB 又は BINARY）のデータベースのアクセス方法を指定します。省略した場合，"REAL"が仮定されます。 "REAL"： HiRDB から実データでアクセスします。 "LOCATOR"： HiRDB の位置付け子を使用してアクセスします。 上記以外： "REAL"を指定したものとみなします。	△
HiRDB_for_Java_SQL_IN_NUM	実行する SQL の入力，又は入出力?パラメタの最大数を指定します。この指定は，SQL の前処理時に取得する，入力，又は入出力?パラメタ情報の数となります。 実際の入力，又は入出力?パラメタの数が，このプロパティの指定値よりも多い場合，SQL の前処理の後に入力，又は入出力?パラメタ情報を取得します。 指定値は，1～30,000 です（デフォルトは 64）。これ以外の値，又は数字以外を指定した場合はエラーとなります。 注意事項： • この項目は，システムプロパティの HiRDB_for_Java_SQL_IN_NUM でも指定できます。ただし，Properties info に HiRDB_for_Java_SQL_IN_NUM を設定した場合は，システムプロパティの設定が無効となります。 • 入力，又は入出力?パラメタのある SQL 文を実行しない場合は，1 を指定することをお勧めします。 • このプロパティの指定値は，バージョン 07-02 以降の HiRDB サーバと接続している場合に有効です。	△
HiRDB_for_Java_SQL_OUT_NUM	実行する SQL の出力項目数の最大数を指定します。この指定は，SQL の前処理時に取得する出力項目情報の数となります。 実際の出力項目情報の数が，このプロパティの指定値よりも多い場合，SQL の前処理の後に出力項目情報を取得します。 指定値は，1～30,000 です（デフォルトは 64）。これ以外の値，又は数字以外を指定した場合はエラーとなります。	△

キー	内容	指定可否
	注意事項： • この機能は，システムプロパティの HiRDB_for_Java_SQL_OUT_NUM でも指定できます。ただし，Properties info に HiRDB_for_Java_SQL_OUT_NUM を設定した場合は，システムプロパティの設定が無効となります。 • 検索項目，又は出力若しくは入出力?パラメタのある SQL 文を実行しない場合は，1 を指定することをお勧めします。 • このプロパティの指定値は，バージョン 07-02 以降の HiRDB サーバと接続している場合に有効です。	
HiRDB_for_Java_SQLWARNING_LEVEL	SQL 実行時に発生した警告情報の保持レベルを指定します。次の値を警告保持レベルとして指定できます。 • IGNORE • SQLWARN（デフォルト） • ALLWARN なお，このメソッドでは，引数に指定した内容の大文字，小文字を区別しません。 上記の値については，「SQLWarning クラス」を参照してください。	△
HiRDB_for_Java_CLEAR_ENV	DB 接続時に，OS の環境変数として設定した HiRDB のクライアント環境定義を無効にするかどうかを指定します。 TRUE： OS の環境変数として登録した HiRDB のクライアント環境定義を，プロセス起動後の最初の DB 接続時に無効にします。TRUE を指定することで，OS の環境変数以外の方法（環境変数グループなど）で設定した HiRDB のクライアント環境定義の値を適用できます。 FALSE（デフォルト）： OS の環境変数として登録した HiRDB のクライアント環境定義を無効にしません。 注意事項： • このメソッドでは，引数に指定した内容の大文字，小文字を区別しません。 • 最初の DB 接続の後，C 言語などで実装したネイティブメソッド内で，OS の環境変数として設定した HiRDB のクライアント環境定義は，指定値を TRUE にしても無効にはなりません。 • 複数回 DB に接続する場合，最初に TRUE を指定して DB 接続をした後は，次の接続で FALSE を指定しても，クライアント環境定義の値は無効になったまま元には戻りません。	△

（凡例）

○：必ず指定してください。

△：指定は任意です。

注※1

認可識別子に null 又は空文字を指定した場合，SQLException を投入します。また，ドライバが文字コードを変換した結果，認可識別子に指定した文字列のサイズが 31 バイト以上となった場合，SQLException を投入します。文字コードの変換については，「文字コード変換機能」を参照してください。

注※2

内部ドライバで記述する場合は省略できます。

COMMIT_BEHAVIOR についての注意事項

注意事項を次に示します。

- CLOSE 又は PRESERVE を指定して，SELECT，INSERT，DELETE，UPDATE，PURGE TABLE，又は CALL でアクセスする資源（表やインデクスなど）に対して，ほかのユーザが定義系 SQL を実行した場合，クライアント環境定義 PDDDLDEAPRPEXE が NO，かつ PDDDLDEAPRP が NO のとき，資源にアクセスしていたコネクションを DISCONNECT するまでの間，その定義系 SQL は排他待ちの状態になります。

  クライアント環境定義 PDDDLDEAPRPEXE が YES，又は PDDDLDEAPRP が YES のときは，前処理結果が無効となります。前処理結果が無効となった SQL を実行すると，SQLException 例外（getErrorCode メソッドで取得できる値は-1542）が発生します。

- PRESERVE を指定した場合，JDBC ドライバは HiRDB のホールダブルカーソルを使用します。

- CLOSE 又は PRESERVE を指定※1 することで，コミット後※2 もプリコンパイルした SQL 文が有効になる SQL 文は，SELECT，INSERT，DELETE，UPDATE，PURGE TABLE，及び CALL だけです（Connection.prepareStatement メソッドの実行，及び Connection.prepareCall メソッドの実行によって，SQL 文をプリコンパイルできます）。

  上記以外の SQL 文では，COMMIT_BEHAVIOR に CLOSE 又は PRESERVE を指定しても，コミット時にプリコンパイルした SQL 文は無効になります。

  それらの無効になった SQL 文を格納した PreparedStatement クラスのオブジェクト，及び CallableStatement クラスのオブジェクトで SQL 文を実行するとエラーになります。エラーになる例を次に示します。

  [実行例]

```
PreparedStatement pstmt1 = con.prepareStatement("lock table tb1");
PreparedStatement pstmt2 = con.prepareStatement("lock table tb2");
pstmt1.execute();        //エラーにならない。
con.commit();
pstmt2.execute();        //エラーになる。
pstmt1.close();
pstmt2.close();
```

  [説明]

  実行する SQL 文が LOCK 文であるため，COMMIT_BEHAVIOR が CLOSE を指定していても，コミット後は PreparedStatement が無効となり，エラーが発生します。

  注※1

  次のどれかの指定の場合が該当します。

  ・getConnection メソッドで指定する URL に COMMIT_BEHAVIOR=CLOSE を指定する。

  ・getConnection メソッドで指定する URL に COMMIT_BEHAVIOR=PRESERVE を指定する。

  ・JdbhDataSource，JdbhConnectionPoolDataSource，及び JdbhXADataSource クラスの setCommit_Behavior メソッドで CLOSE を指定する。

・JdbhDataSource，JdbhConnectionPoolDataSource，及び JdbhXADataSource クラスの setCommit_Behavior メソッドで PRESERVE を指定する。

注※2

次の場合が該当します。

・commit メソッドによる明示的なコミット

・自動コミットによる暗黙的なコミット

・定義系 SQL 文の実行

・PURGE TABLE 文の実行

・rollback メソッドによる明示的なロールバック

・SQL 実行エラーによる暗黙的なロールバック

(3) URL の構文

JDBC ドライバで指定できる URL の構文について説明します。なお，URL 内の各項目及び項目間には空白を入れないでください。接続付加情報項目と DB ホスト名称項目の両方を指定する場合は，接続付加情報項目と DB ホスト名称項目との間にコンマを指定してください。

(a) URL の構文

```
jdbc:hitachi:PrdbDrive[://[DBID=接続付加情報]
                       [[{://|,}]DBHOST=DBホスト名称]
                       [[{://|,}]ENCODELANG=変換文字セット]
                       [[{://|,}]COMMIT_BEHAVIOR=カーソル動作モード]
                       [[{://|,}]CLEAR_ENV=環境変数無効化指定]]
```

(b) URL の各項目の説明

jdbc:hitachi:PrdbDrive：

プロトコル名称及びサブプロトコル名称です。必ず指定してください。

接続付加情報：

HiRDB のポート番号を指定します（クライアント環境定義の PDNAMEPORT に相当します）。又は，HiRDB の環境変数グループを指定します。

省略した場合，PDNAMEPORT のデフォルト値となります。

＜接続付加情報に HiRDB の環境変数グループを指定する場合の注意事項＞

- HiRDB の環境変数グループ名を指定する場合は，グループ名の先頭に@を付けます。
- 環境変数グループ名に半角空白文字，及び半角@文字を含む場合，半角引用符（"）で囲んでください。環境変数グループ名を半角引用符で囲んだ場合，最後の半角引用符から次の設定項目，又は文字終端までの文字は無視されます。また，半角引用符，及び半角コンマを含む環境変数グループ名は指定できません。
- 環境変数グループに登録された環境変数は，ユーザ環境変数や HiRDB.INI で登録した環境変数よりも優先されます。

- 接続付加情報，及び DB ホスト名称の指定と，接続先の優先順位を次に示します。

  1.接続付加情報に指定した HiRDB の環境変数グループ

  2.DB ホスト名称，又は接続付加情報に指定したポート番号

  例えば，URL 中の DBHOST に DB ホスト名称を指定しても，DBID に HiRDB の環境変数グループ名を指定している場合は，HiRDB の環境変数グループの内容が優先されます。この場合，HiRDB の環境変数グループ内に PDHOST の指定がないときは接続エラーとなります。

＜接続付加情報に HiRDB のポート番号を指定する場合の注意事項＞

接続先のポート番号が 65535 である場合は，クライアント環境定義の PDNAMEPORT 又は環境変数グループを使用して接続先を指定してください。

DB ホスト名称：

HiRDB のホスト名を指定します。クライアント環境定義の PDHOST に相当します。

省略した場合，PDHOST のデフォルト値となります。

変換文字セット：

文字型変換で使用する変換文字セットを指定します。

カーソル動作モード：

カーソルが COMMIT をわたって有効かどうかを指定します。

環境変数無効化指定：

DB 接続時に，OS の環境変数として設定した HiRDB のクライアント環境定義を無効にするかどうかを指定します。指定値，及び注意事項については，表「Properties info の設定内容」の HiRDB_for_Java_CLEAR_ENV を参照してください。

(c)　接続付加情報に HiRDB の環境変数グループ名を指定する場合の例

- **UNIX 版の場合**

HiRDB の環境変数グループ名のパスが「/HiRDB_P/Client/HiRDB.ini」の指定例を次に示します。

```
String url = ”jdbc:hitachi:PrdbDrive://DBID=@HIRDBENVGRP=/HiRDB_P/Client/HiRDB.ini”;
```

- **Windows 版の場合**

1. HiRDB クライアント環境変数登録ツールで登録した環境変数グループ名が HiRDB_ENV_GROUP の場合の指定例を次に示します。

   ```
   String url = ”jdbc:hitachi:PrdbDrive://DBID=@HIRDBENVGRP=HiRDB_ENV_GROUP”;
   ```

2. HiRDB の環境変数グループ名のパスが「C:¥HiRDB_P¥Client¥HiRDB.ini」の場合の指定例を次に示します。

   ```
   String url = ”jdbc:hitachi:PrdbDrive://DBID=@HIRDBENVGRP=C:¥¥HiRDB_P¥¥Client¥¥HiRDB.ini”;
   ```

3. HiRDB の環境変数グループ名のパスが「C:¥Program△Files¥HITACHI¥HiRDB¥HiRDB.ini」の場合の指定例を次に示します（△は半角空白文字）。

```
String url = ”jdbc:hitachi:PrdbDrive://DBID=@HIRDBENVGRP=” +
             ”¥”C:¥Program△Files¥¥HITACHI¥¥HiRDB¥¥HiRDB.ini¥””;
```

16.2.2 Connection クラス

(1) 概要

Connection クラスでは，次の機能が提供されます。

- Statement クラス，PreparedStatement クラス，及び CallableStatement クラスのオブジェクト生成
- トランザクションの決着（COMMIT 又は ROLLBACK）
- AUTO コミットモードの設定

Connection クラスで提供される各メソッドの詳細，使用方法については，JDBC の関連ドキュメントを参照してください。

(2) 注意事項

(a) カタログ

JDBC ドライバでは，接続 DB 種別に関係なく，カタログは使用できません。そのため，getCatalog メソッドは無条件に null を返却し，setCatalog メソッドは何も処理をしません。

(b) アクセスモード

JDBC ドライバでは，アクセスモードの変更はできません。そのため，isReadOnly メソッドは無条件に false を返却し，setReadOnly メソッドは何も処理をしません。

(c) トランザクション分離モード

JDBC ドライバでは，トランザクション分離モードの変更はできません。そのため，getTransactionIsolation メソッドは無条件に TRANSACTION_READ_COMMITTED を返却し，setTransactionIsolation メソッドは何も処理をしません。

16.2.3 Statement クラス

(1) 概要

Statement クラスでは，次の機能が提供されます。

- SQL の実行

- 検索結果としての結果セット（ResultSet オブジェクト）の生成
- 更新結果としての更新行数の返却

Statement クラスで提供される各メソッドの詳細，使用方法については，JDBC の関連ドキュメントを参照してください。

(2) 注意事項

(a) マルチスレッド

一つの Statement オブジェクトを複数のスレッドで使用する場合，「SQL の実行～結果セットの取得～結果セットのクローズ」の処理を，スレッドごとにシリアライズする必要があります。シリアライズしないで並行して処理した場合，動作は保証されません。したがって，スレッドごとに別々の Statement オブジェクトを割り当てることをお勧めします。

(b) カーソル名称

JDBC ドライバでは，位置決めされた更新及び削除は使用できません。そのため，setCursorName メソッドは何も処理をしません。

(c) 検索制限時間

JDBC ドライバでは，検索の時間監視はできません。そのため，setQueryTimeout メソッドで指定した値は無効となります。

(d) 最大検索行数の設定

JDBC ドライバでは，最大検索行数の設定はできません。

16.2.4 PreparedStatement クラス

(1) 概要

PreparedStatement クラスでは，次の機能が提供されます。

- ？パラメタ指定の SQL の実行
- ？パラメタの設定
- 検索結果としての ResultSet オブジェクトの生成，返却
- 更新結果としての更新行数の返却

また，PreparedStatement クラスは Statement クラスのサブクラスであるため，Statement クラスの機能をすべて継承します。

PreparedStatement クラスで提供される各メソッドの詳細，使用方法については，JDBC の関連ドキュメントを参照してください。

(2)　注意事項

PreparedStatement クラスは Statement クラスのサブクラスであるため，Statement クラスの注意事項はすべて該当します。それ以外の PreparedStatement クラスの注意事項を次に示します。

(a)　？パラメタの設定

？パラメタの設定に使用する setXXX メソッドについては，「？パラメタ設定時のデータマッピング」を参照してください。接続 DB がどの JDBC SQL タイプを使用できるかについては，「データ型，文字コード」を参照してください。

(b)　複数の結果セット

複数の結果セットを返却する機能は使用できません。そのため，getMoreResults メソッドは無条件に false を返却し，現在オープンしている結果セットがあるときはその結果セットをクローズします。

16.2.5　CallableStatement クラス

(1)　概要

CallableStatement クラスでは，次の機能が提供されます。

- Java ストアドルーチンの実行
- IN パラメタ及び INOUT パラメタの設定（PreparedStatement クラスの setXXX メソッドを使用）
- OUT パラメタ及び INOUT パラメタの登録
- OUT パラメタ及び INOUT パラメタ値の取得
- 結果セットの取得

また，CallableStatement クラスは PreparedStatement クラスのサブクラスであるため，PreparedStatement クラス及び Statement クラスの機能をすべて継承します。ただし，Java ストアドルーチン内の DatabaseMetaData クラスで取得した結果セットは，Java ストアドルーチン内でだけ使用できます。CallableStatement クラスの getResultSet では，動的結果セットとして取得できません。

CallableStatement クラスで提供される各メソッドの詳細，使用方法については，JDBC の関連ドキュメントを参照してください。

(2)　注意事項

1. CallableStatement クラスは PreparedStatement クラスのサブクラスであるため，PreparedStatement クラス及び Statement クラスの注意事項はすべて該当します。

2. clearParameters メソッドを実行した場合，clearParameters メソッドを実行した時点でパラメタの情報を消去します。execute メソッド実行後，getXXX メソッドを実行するまでの間に clearParameters メソッドを実行した場合，getXXX メソッドの実行で KFPJ20506-E メッセージを出力します。
3. Java ストアドルーチンの INOUT パラメタを使用する場合，registerOutParameter メソッドで指定する java.sql.Types クラスの型と setXXX メソッドで設定するデータの型は同一にしてください。

16.2.6　ResultSet クラス

(1)　概要

ResultSet クラスでは，次の機能が提供されます。

- 行単位の結果セット内の移動
- 結果データの返却
- 検索結果データが NULL 値かどうかの通知

ResultSet クラスで提供される各メソッドの詳細，使用方法については，JDBC の関連ドキュメントを参照してください。

(2)　注意事項

(a)　マルチスレッド

一つの ResultSet オブジェクトを複数のスレッドで並行して使用する場合，動作は保証されません。したがって，一つの ResultSet オブジェクトは一つのスレッドで処理することをお勧めします。

(b)　データマッピング（変換）

結果取得時の getXXX メソッドについては，「検索データ取得時のデータマッピング」を参照してください。接続 DB がどの JDBC SQL タイプを使用できるかについては，「データ型，文字コード」を参照してください。

16.2.7　ResultSetMetaData クラス

(1)　概要

ResultSetMetaData クラスでは，次の機能が提供されます。

- ResultSet（結果セット）の各列に対するデータ型及びデータ長などのメタ情報の返却

(2)　メソッドの詳細

(a)　isSearchable (int column)メソッド

パラメタ column で指定された列を WHERE 句で使用できる場合は true，できない場合は false を戻り値とします。すべてのデータ型の列を WHERE 句で使用できるため，常に true を返します。ただし，Array.getResultSet メソッドの戻り値である ResultSet の 1 番目の列の場合は，false を返します。getResultSet については，「Array クラス」を参照してください。

（例）

表 T1 に列 C1 があります。C1 がどのデータ型の場合でも，次に示すように WHERE 句で使用できます。

```
SELECT * FROM T1 WHERE LENGTH (C1) > 5
```

(b)　getColumnDisplaySize (int column)メソッド

パラメタ column で指定された列を文字列表現した場合の，最大文字数を戻り値とします。ただし，Array.getResultSet メソッドの戻り値である ResultSet の 1 番目の列の場合は，10 を返します。HiRDB の各 SQL データ型に対するこのメソッドの戻り値を次の表に示します。

表 16–4　HiRDB の各 SQL データ型に対する getColumnDisplaySize メソッドの戻り値

HiRDB の SQL データ型	戻り値（int）	戻り値の計算式
INTEGER	11	符号 1 文字＋最大けた数 10 けた
SMALLINT	6	符号 1 文字＋最大けた数 5 けた
DECIMAL（m，n） NUMERIC（m，n） • m：精度（全体のけた数） • n：位取り（小数点以下のけた数）	m ＋ 2	符号 1 文字＋精度 m ＋小数点 1 文字
FLOAT DOUBLE PRECISION	23	符号 1 文字＋最大有効けた数 17 けた＋小数点 1 文字＋指数部最大文字数 4
SMALLFLT REAL	13	符号 1 文字＋最大有効けた数 8 けた＋小数点 1 文字＋指数部最大文字数 3
CHAR（n） • n：定義長のバイト数	n	－
VARCHAR（n） CHAR VARYING（n） • n：最大長のバイト数	n	－
NCHAR（n） NATIONAL CHAR（n） • n：定義長の文字数	n	－

HiRDB の SQL データ型	戻り値（int）	戻り値の計算式
NVARCHAR（n） NATIONAL CHAR VARYING（n） NCHAR VARYING（n） • n：最大長の文字数	n	−
MCHAR（n） • n：定義長のバイト数	n	−
MVARCHAR（n） • n：最大長のバイト数	n	−
DATE	10	"yyyy-mm-dd"の 10 文字
TIME	8	"hh:mm:ss"の 8 文字
TIMESTAMP（p） • p：小数秒のけた数	(1) p が 0 の場合： 19 (2) p が 2，4，又は 6 の場合： 20 ＋ p	(1) "yyyy-mm-dd hh:mm:ss"の 19 文字 (2) (1)の 19 文字＋小数点 1 文字＋小数部けた数 p
BLOB（n［K ｜ M ｜ G］） • n：最大長 • K：キロバイト単位 • M：メガバイト単位 • G：ギガバイト単位 単位を省略した場合はバイト単位	単位の指定を省略した場合： n 単位に K を指定した場合： n×1024※ 単位に M を指定した場合： n×1024×1024※ 単位に G を指定した場合： n×1024×1024×1024※	−
BINARY（n） • n：最大長のバイト数	n	−

（凡例）

−：該当しません。

注※

計算結果が 2147483648 の場合は，2147483648 以上になります。

16.2.8 DatabaseMetaData クラス

DatabaseMetaData クラスでは，次の機能が提供されます。

- 接続 DB に関する各種情報の返却
- 一覧系情報（表一覧，列一覧など）の ResultSet（結果セット）への格納，返却

ただし，Java ストアドルーチン内の DatabaseMetaData クラスで取得した結果セットは，Java ストアドルーチン内でだけ使用できます。

DatabaseMetaData クラスで提供される各メソッドの詳細，使用方法については，JDBC の関連ドキュメントを参照してください。実際に返却する値については，「制限事項があるクラスとメソッド」を参照してください。なお，各メソッドが返却する値は，常に JDBC ドライバのバージョンと同一のバージョンの HiRDB サーバに関する情報です。

16.2.9　SQLWarning クラス

(1)　概要

SQLWarning クラスでは，次の機能が提供されます。

- データベースアクセスの警告に関する情報の提供

SQLWarning オブジェクトは，警告が報告される原因となったメソッドのオブジェクトに，例外での通知なしで蓄積されます。

(2)　注意事項

(a)　蓄積された SQLWarning オブジェクトの解放

SQLWarning オブジェクトは，警告が報告される原因となったメソッドのオブジェクト（Connection，Statement，PreparedStatement，CallableStatement，又は ResultSet）から，チェーンによって蓄積されます。

蓄積された SQLWarning オブジェクトを明示的に解放するには，チェーンをつないでいるオブジェクトから，clearWarnings を実行する必要があります。

(b)　SQLWarning オブジェクトの生成条件

SQL の実行で発生した警告が，JDBC ドライバ内で保持することを，警告保持レベルで指定している場合，SQLWarning オブジェクトを生成して警告情報を保持します。SQLWarning の生成条件を次に示します。

SQL の実行結果	警告保持レベル		
	IGNORE	SQLWARN	ALLWARN
SQLCODE>0，かつ SQLCODE が 100，110，及び 120 以外	×	×	○
SQL 連絡領域の SQLWARN0 が W（SQLWARN6 が W である場合を除く）	×	○	○
JDBC ドライバ内での警告発生	×	○	○

（凡例）

○：生成します。

×：生成しません。

注

警告保持レベルは，プロパティの HiRDB_for_Java_SQLWARNING_LEVEL，又はメソッドの setSQLWarningLevel で指定できます。デフォルトは SQLWARN です。

(c)　警告メッセージ

SQLWarning から取得できるメッセージを次に示します。

条件	getMessage で取得するメッセージ
SQLWARN0 が W	KFPJ01074-W
SQLWARN0 が'△'，かつ SQLCODE>0（SQLCODE=100，110，及び 120 を除く）	KFPAXXXXX-X
JDBC ドライバ内で警告が発生	KFPJXXXXX-W

(d)　バッチ更新

バッチ更新実行中に複数行の更新で警告が発生しても，SQLWarning は 1 個しか生成しません。

16.3　JDBC2.0 基本機能

16.3.1　結果セットの拡張

JDBC2.0 基本規格では，結果セット（ResultSet クラス）の拡張機能として「スクロール」と「並行処理」が追加されました。

(1)　スクロールタイプ

結果セットのスクロールタイプには，次の 3 種類があります。

(a)　順方向専用型

JDBC1.0 からの標準のスクロールタイプです。結果セット内を順方向（上から下）にだけスクロールできます。

(b)　スクロール非反映型

JDBC2.0 で追加されたスクロールタイプです。結果セット内を順方向又は逆方向にスクロールできます。また，現在の位置からの相対位置指定の移動，又は絶対位置への移動もできます。

「非反映型」とは，結果セットが開かれている間に加えられた変更が，その結果セットに反映されないということを意味します。つまり，基盤となるデータの静的なビューを提供するだけで，結果セットに含まれる行，その順序，及び列の値は，結果セットの作成時に固定されます。

(c)　スクロール反映型

JDBC2.0 で追加されたスクロールタイプです。結果セットが開かれている間に加えられた変更が，その結果セットに反映されます。

「変更の反映」という点では，「その結果セット自身による変更の反映」，「同一トランザクション内でのほかの結果セットによる変更の反映」，「他トランザクションによる変更の反映」などがあります。どこまで保証されるかは，ドライバの実装レベル，及び DBMS のトランザクション遮断レベルに依存します。

(2)　並行処理タイプ

結果セットの並行処理タイプには，次の 2 種類があります。

(a)　読み取り専用型

JDBC1.0 からの標準の並行処理タイプです。結果セットからの更新はできません。

(b)　更新可能型

JDBC2.0 で追加された並行処理タイプです。結果セットからの更新（UPDATE，INSERT，及び DELETE）ができます。

(3)　結果セットタイプ

スクロールタイプと並行処理タイプを組み合わせると，結果セットタイプは 6 種類になります。結果セットタイプは，Connection クラスの createStatement メソッド，prepareStatement メソッド，又は prepareCall メソッドで Statement クラス（又はそのサブクラス）のインスタンスを取得する場合に指定します。

結果セットタイプと JDBC ドライバでの提供可否を次の表に示します。

表 16–5　結果セットタイプと JDBC ドライバでの提供可否

結果セットタイプ		JDBC ドライバでの提供可否
スクロールタイプ	並行処理タイプ	
順方向専用型	読み取り専用型	○
	更新可能型	×
スクロール非反映型	読み取り専用型	○
	更新可能型	×
スクロール反映型	読み取り専用型	×
	更新可能型	×

（凡例）

○：提供されます。

×：提供されません。

注 1

提供されていない結果セットタイプを指定した場合，エラーにはなりません。この場合，指定した結果セットタイプに一番近い結果セットタイプを使用して Statement クラス（又はそのサブクラス）のインスタンスを生成し，警告メッセージを Connection クラスの SQLWarning に格納します。

注 2

更新可能型の並行処理タイプは JDBC ドライバでは提供されていないため，ResultSet クラスには使用できないメソッドがあります。これらの使用できないメソッドが呼び出された場合，無条件に SQLException を投入します。使用できないメソッドについては，「制限事項があるクラスとメソッド」を参照してください。

(4)　スクロール型結果セット使用時の注意点

スクロール型結果セットでは，すべての検索データを JDBC ドライバ内でキャッシングします。そのため，データ量が多い場合は，メモリ不足や性能劣化となる可能性が高くなります。したがって，スクロール型結果セットを使用する場合は，「SQL に条件を付加する」など，検索データ量をあらかじめ抑制しておく必要があります。

16.3.2　バッチ更新

JDBC2.0 基本規格では，Statement クラス，PreparedStatement クラス，及び CallableStatement クラスにバッチ更新機能が追加されました。バッチ更新機能によって，複数の SQL，又は複数のパラメタ値を登録し，一括して実行できるようになります。

バッチ更新機能を使用する場合，Connection クラスの AUTO コミットモードを OFF にする必要があります。これは，バッチ更新の途中でエラーが発生した場合に，そのトランザクションの有効・無効をアプリケーション側で制御する必要があるためです。AUTO コミットモードが ON（初期状態）の場合，バッチ更新の途中でエラーが発生しても，エラーが発生する一つ前までの SQL 実行は有効となります。

バッチ更新を実行する場合，HiRDB の配列を使用した機能が使用できます。

配列を使用した機能は，HiRDB に対して大量のデータを高速に更新したい場合に有効です。なお，配列を使用した機能については，「配列を使用した機能」を参照してください。

配列を使用した機能を利用する場合の注意事項：

1. 配列を使用した機能は，バージョン 07-01 以降の HiRDB の場合に使用できます。
2. Connect 時に，プロパティとして BLOCK_UPDATE=TRUE を指定（DataSource を使用する場合は setBlockUpdate(true)），又は JdbcDbpsvPreparedStatement の setBlockUpdate(true)を指定する必要があります。
3. システムプロパティの HiRDB_for_Java_BLOCK_UPDATE=TRUE を指定した場合，配列機能を有効にできます。HiRDB_for_Java_BLOCK_UPDATE については，表「Properties info の設定内容」の BLOCK_UPDATE を参照してください。
4. 実行する SQL には，?パラメタが一つ以上なければなりません（ストアドプロシジャでは使用できません）。また，PreparedStatement クラス，又は CallableStatement クラスの addBatch()メソッドを使用する必要があります（Statement クラスの addBatch(String sql)メソッドを使用すると，HiRDB でエラーとなります）。

   実行できる SQL 文は，INSERT 文，UPDATE 文，又は DELETE 文です。それ以外の SQL 文の場合，一括実行は行われないで逐次実行されます。
5. addBatch()メソッドで登録したパラメタセットが 2 件以上なければなりません。1 件の場合は，一括処理は行われないで通常の処理となります。また，パラメタセットが 30,000 件を超える場合，30,000 件ごとに分割して実行されます。

6. ?パラメタにデータ長が 32,001 バイト以上の BINARY データを指定した場合，配列を使用した機能は適用されないため，逐次実行されます。
7. HiRDB の BLOB 型の列に対して 32,001 バイト以上のデータを指定した場合，配列を使用した機能は適用されないため，逐次実行されます。※2
8. 各列に指定するデータ型はすべて同じにしてください。※1
9. DECIMAL 型データを挿入する場合，配列に指定する DECIMAL 型データの精度及び位取りは，HiRDB の表定義の属性に置き換えられます。配列に指定する DECIMAL 型データの整数部けた数が，HiRDB の表定義属性の整数部けた数より大きい場合，オーバフローが発生してエラーとなります。
10. ?パラメタに HiRDB の繰返し列を指定している場合，配列を使用した機能は利用できません。
11. 配列を使用した機能を利用してバッチ更新の途中でエラーが発生した場合，エラーが発生する直前までの実行結果はすべて無効となります。
12. Cosminexus J2EE サーバモードのベーシックモードからは，配列を使用した機能は利用できません。
13. Cosminexus から使用する場合，PreparedStatement の setBlockUpdate メソッドは使用できません。
14. addBatch 機能を使用して大量のデータを一度に更新すると，大量の Java メモリを使用します。そのため，Java メモリの性能によってはバッチ更新の効果が得られない場合があるので注意が必要です。また，大量のデータを使用する場合，Java 起動時のヒープサイズを指定してください（java -Xms32m JavaUP：Java 起動時の Java ヒープを 32 メガバイトと設定）。

注※1

例えば，列 1 のデータの 1 件目の addBatch を setInt()で指定した場合，2 件目以降の addBatch も setInt()を使用する必要があります。

注※2

配列を使用した機能を利用する場合，HiRDB の BLOB 型列に対して?パラメタを指定するときは，次のことに注意してください。

- ?パラメタに指定するデータ長が 32,001 バイト未満の場合は，JDBC ドライバ内部で BINARY 型データとして扱うため，配列を使用した機能が実行されます。32,001 バイト以上の場合は，配列を使用した機能は実行されません。

(1) Statement クラスでのバッチ更新

Statement クラスでのバッチ更新の留意点を次に示します。

- 複数の更新系 SQL を，addBatch メソッドで登録します。
- 登録した更新系 SQL を，executeBatch メソッドで一括して実行します。
- 一括実行の結果として，それぞれの更新系 SQL で更新された行数の配列を返却します。
- 一括実行の途中でエラーが発生した場合，BatchUpdateException を投入します。

- 登録した SQL 中に検索系 SQL がある場合，executeBatch メソッド呼び出し時に BatchUpdateException を投入します。

なお，JDBC ドライバでは，複数の SQL を一括実行できないため，登録された SQL を逐次実行することになります。

(2) PreparedStatement クラスでのバッチ更新

PreparedStatement クラスでのバッチ更新の留意点を次に示します。

- PreparedStatement インスタンス生成時に指定した更新系 SQL に対する？パラメタを，通常の手順（setXXX メソッド）で設定します。
- addBatch メソッドで？パラメタのセットを登録します。
- 登録した複数セットの？パラメタを，executeBatch メソッドで一括して実行します。
- 一括実行の結果として，それぞれの？パラメタのセットで更新した行数の配列を返却します。
- 一括実行の途中でエラーが発生した場合，BatchUpdateException を投入します。
- PreparedStatement インスタンス生成時に指定した SQL が検索系 SQL の場合，executeBatch メソッド呼び出し時に BatchUpdateException を投入します。

JDBC ドライバでは，配列を使用した機能を利用した場合，複数行の？パラメタの一括実行ができます。配列を使用した機能を利用しない場合は，複数行の？パラメタを逐次実行することになります。

注意事項：

- HiRDB の配列を使用した機能を利用する場合，「バッチ更新」の注意事項を参照してください。
- 2 件目以降の addBatch で，setXXX メソッドで指定するパラメタ数が不足していた場合，前回セットした値が引き継がれるため注意が必要です。例を次に示します。

例：INTEGER 型列が 2 列(列 1，列 2)ある場合

```
prepstmt.setInt(1,100);
prepstmt.setInt(2,100);
prepstmt.addBatch();
prepstmt.setInt(1,200);
prepstmt.addBatch();
prepstmt.executeBatch();
```

[説明]

- 1 件目の addBatch で設定される値は，列 1 ＝ 100，列 2 ＝ 100 となります。
  1 件目の addBatch でパラメタ数が不足している場合は，エラーが発生します。
- 2 件目の addBatch で設定される値は，列 1 ＝ 200，列 2 ＝ 100 となります。
  2 件目の addBatch で，列 2 の情報が更新されていないため，1 件目の addBatch の情報が引き継がれます。

(3) CallableStatement クラスでのバッチ更新

CallableStatement クラスでのバッチ更新の留意点を次に示します。

- CallableStatement インスタンス生成時に指定した Java ストアドルーチンに対する入力パラメタを，通常の手順（setXXX メソッド）で設定します。
- addBatch メソッドで入力パラメタのセットを登録します。
- 登録した複数セットの入力パラメタを，executeBatch メソッドで一括して実行します。
- 一括実行の結果として，それぞれの入力パラメタのセットで実行した Java ストアドルーチンの返却値（更新行数）の配列を返却します。
- 一括実行の途中でエラーが発生した場合，BatchUpdateException を投入します。
- CallableStatement インスタンス生成時に指定した Java ストアドルーチンが，更新行数を返却するルーチンでない場合，executeBatch メソッド呼び出し時に BatchUpdateException を投入します。
- CallableStatement インスタンス生成時に指定した Java ストアドルーチンが，出力パラメタ及び入出力パラメタを持つ場合，addBatch メソッド呼び出し時に BatchUpdateException を投入します。

なお，JDBC ドライバでは，複数行のストアドプロシジャの？パラメタを一括実行できないため，複数行のストアドプロシジャの？パラメタを逐次実行することになります。

注意事項：

- ストアドプロシジャのバッチ更新は，IN パラメタでだけ使用できます。OUT パラメタ，INOUT パラメタ，及び結果セット(ResultSet)を持つ場合はエラーとなります。
- 結果セット(ResultSet)を返すストアドプロシジャは，バッチ更新ではストアドプロシジャを実行するまで結果セットを返すかどうか分からないため，ストアドプロシジャ内で更新をしている場合，更新が反映されることがあるので注意してください。※1
- 配列を使用した機能は，ストアドプロシジャでは利用できません。?パラメタを持つ SQL でだけ使用できます。
- 2 件目以降の addBatch で，setXXX メソッドで指定するパラメタ数が不足している場合，前回セットした値が引き継がれるため注意が必要です。※2
- 配列を使用した機能を利用する場合，「バッチ更新」の配列を使用した機能を利用する場合の注意事項を参照してください。

注※1

例えば，更新後にその結果を検索し取得するストアドプロシジャを，バッチ更新で実行すると，BatchUpdateException が発生するが，更新は反映されてしまうことがあります。

注※2

INTEGER 型列が 2 列(列 1，列 2)ある場合の例を次に示します。

```
callstmt.setInt(1,100);
callstmt.setInt(2,100);
callstmt.addBatch();
```

```
callstmt.setInt(1,200);
callstmt.addBatch();
callstmt.executeBatch();
```

［説明］

- 1 件目の addBatch で設定される値は，列 1 ＝ 100，列 2 ＝ 100 となります。
  1 件目の addBatch でパラメタ数が不足している場合は，エラーが発生します。
- 2 件目の addBatch で設定される値は，列 1 ＝ 200，列 2 ＝ 100 となります。
  2 件目の addBatch で列 2 の情報が更新されていないため，1 件目の addBatch の情報が引き継がれます。

16.3.3　追加されたデータ型

JDBC2.0 基本規格では，幾つかの新たな JDBC SQL タイプが追加されました。追加された JDBC SQL タイプを次に示します。

- BLOB
- CLOB
- ARRAY
- REF
- DISTINCT
- STRUCT
- JAVA OBJECT

ただし，JDBC ドライバでは ARRAY 以外の JDBC SQL タイプは使用できません。

(1)　検索データ取得時のデータマッピング

ResultSet クラス及び CallableStatement クラスの getXXX メソッドと各 JDBC SQL タイプとのマッピングを次の表に示します。

マッピングできない JDBC SQL タイプに対して getXXX メソッドが呼び出された場合，SQLException を投入します。接続 DB がどの JDBC SQL タイプを使用できるかについては「データ型，文字コード」を参照してください。

なお，getUnicodeStream メソッドが JDBC2.0 基本規格で推奨されないメソッドとなったため，代わりに getCharacterStream が追加されました。

表 16–6　ResultSet クラス及び CallableStatement クラスの getXXX メソッドと JDBC SQL タイプとのマッピング（1/2）

getXXX メソッド	JDBC SQL タイプ					
	SMALLINT	INTEGER	FLOAT	REAL	DECIMAL	CHAR
getByte	○	○	○	○	○	○※2
getShort	◎	○	○	○	○	○※2
getInt	○	◎	○	○	○	○※2
getLong	○	○	○	○	○	○※2
getFloat	○	○	○	◎	○	○※2
getDouble	○	○	◎	○	○	○※2
getBigDecimal	○	○	○	○	◎	○※2
getBoolean	○	○	○	○	○	○
getString	○	○	○	○	○	◎
getBytes	×	×	×	×	×	×
getDate	×	×	×	×	×	○※2
getTime	×	×	×	×	×	○※2
getTimestamp	×	×	×	×	×	○※2
getAsciiStream	×	×	×	×	×	○
getUnicodeStream	×	×	×	×	×	○
getBinaryStream	×	×	×	×	×	×
getObject	○	○	○	○	○	○
getCharacterStream	×	×	×	×	×	○
getArray	×	×	×	×	×	×
getBlob	×	×	×	×	×	×
getClob※1	×	×	×	×	×	×
getRef※1	×	×	×	×	×	×

（凡例）

◎：マッピングすることを推奨します。

○：マッピングできます。

×：マッピングできません。

注※1

JDBC ドライバでは使用できません。

注※2

文字列データからのデータ変換で，データベースから取得した文字列データの前後に半角スペースが存在する場合は，半角スペースを取り除いた後，getXXX メソッドが返却する Java のデータ型に変換します。

Java のデータ型に変換する場合の注意事項を次に示します。

- 文字列データに小数点以下の表現がある場合，getByte，getInt，getShort，又は getLong のどれかのメソッドを実行すると，小数点以下を切り捨てて整数だけを変換し，返却します。
- 文字列データに全角文字が含まれている場合は，SQLException を投入します。全角文字には，NCHAR 型の列に列の定義長よりも短い文字列を格納している場合に補完する全角スペースも含みます。
- 文字列データを Java のデータ型に変換した結果，オーバフローが発生する場合は，SQLException を投入します。
- UAP の実行環境が JDK 又は JRE 1.2 で，文字列データが指数の表記法（1.23E-23 など）の場合，getLong メソッド，又は getBigDecimal メソッドのどちらかを実行すると，SQLException を投入します。

表 16–7　ResultSet クラス及び CallableStatement クラスの getXXX メソッドと JDBC SQL タイプとのマッピング（2/2）

getXXX メソッド	JDBC SQL タイプ					
	VARCHAR	DATE	TIME	TIMESTAMP	LONGVARBINARY	ARRAY
getByte	○※2	×	×	×	×	×
getShort	○※2	×	×	×	×	×
getInt	○※2	×	×	×	×	×
getLong	○※2	×	×	×	×	×
getFloat	○※2	×	×	×	×	×
getDouble	○※2	×	×	×	×	×
getBigDecimal	○※2	×	×	×	×	×
getBoolean	○	×	×	×	×	×
getString	◎	○	○	○	○	×
getBytes	×	×	×	×	○	×
getDate	○※2	◎※3	×	○	×	×
getTime	○※2	×	◎	○	×	×

getXXX メソッド	JDBC SQL タイプ					
	VARCHAR	DATE	TIME	TIMESTAMP	LONGVARBINARY	ARRAY
getTimestamp	○※2	○	×	◎	×	×
getAsciiStream	○	×	×	×	○	×
getUnicodeStream	○	×	×	×	○	×
getBinaryStream	×	×	×	×	◎	×
getObject	○	○	○	○	○	○
getCharacterStream	○	×	×	×	○	×
getArray	×	×	×	×	×	◎
getBlob	×	×	×	×	○	×
getClob※1	×	×	×	×	×	×
getRef※1	×	×	×	×	×	×

（凡例）

◎：マッピングすることを推奨します。

○：マッピングできます。

×：マッピングできません。

注※1

JDBC ドライバでは使用できません。

注※2

文字列データからのデータ変換で，データベースから取得した文字列データの前後に半角スペースがある場合は，半角スペースを取り除いた後，getXXX メソッドが返却する Java のデータ型に変換します。

Java のデータ型に変換する場合の注意事項を次に示します。

- 文字列データに小数点以下の表現がある場合，getByte，getInt，getShort，又は getLong のどれかのメソッドを実行すると，小数点以下を切り捨てて整数だけを変換し，返却します。
- 文字列データに全角文字が含まれている場合は，SQLException を投入します。全角文字には，NCHAR 型の列に列の定義長よりも短い文字列を格納している場合に補完する全角スペースも含みます。
- 文字列データを Java のデータ型に変換した結果，オーバフローが発生する場合は，SQLException を投入します。
- UAP の実行環境が JDK 又は JRE 1.2 で，文字列データが指数の表記法（1.23E-23 など）の場合，getLong メソッド，又は getBigDecimal メソッドのどちらかを実行すると，SQLException を投入します。

注※3

JDBC SQL タイプが DATE 型の場合，setDate メソッドに java.util.Calendar オブジェクトを指定して実行すると，指定した java.util.Calendar オブジェクトを使用してデータを変換し，時刻データを切り捨てて日付データだけをデータベースに格納します。このとき，時刻データを切り捨てるため，getDate メソッドに java.util.Calendar オブジェクトを指定して，setDate メソッドで格納したデータを取得しても，setDate メソッドに指定した日付と異なる日付を取得する場合があります。

（例）

日本標準時をデフォルトのタイムゾーンとする UAP で，setDate メソッド，及び getDate メソッドに世界標準時のタイムゾーンを持つ java.util.Calendar オブジェクトを指定した場合の例を次に示します。

setDate メソッドに「2005-10-03」を表す java.sql.Date オブジェクトを指定して実行した場合，JDBC ドライバは時刻部分に「00:00:00」を補完した後，タイムゾーンの違いによって 9 時間遅らせて「2005-10-02 15:00:00」とし，日付部分「2005-10-02」をデータベースに格納します。このデータを getDate メソッドで取得した場合，データベースから日付部分「2005-10-02」を取得し，時刻部分に「00:00:00」を補完した後，タイムゾーンの違いによって 9 時間進めて「2005-10-02 09:00:00」とします。これによって，getDate メソッドの戻り値の java.sql.Date オブジェクトには，「2005-10-02」が設定されるため，setDate メソッドに指定した「2005-10-03」とは異なります。

(2)　？パラメタ設定時のデータマッピング

PreparedStatement クラス及び CallableStatement クラスの setXXX メソッドと，マッピングされる JDBC SQL タイプを次の表に示します。使用できない JDBC SQL タイプの場合，setXXX メソッドは SQLException を投入します。接続 DB がどの JDBC SQL タイプを使用できるかについては「データ型，文字コード」を参照してください。

なお，setUnicodeStream メソッドが JDBC2.0 基本規格で推奨されないメソッドとなったため，代わりに setCharacterStream が追加されました。

表 16–8　PreparedStatement クラスの setXXX メソッドと，マッピングされる JDBC SQL タイプ

PreparedStatement クラスの setXXX メソッド	マッピングされる JDBC SQL タイプ
setCharacterStream	CHAR，VARCHAR，又は LONGVARCHAR
setRef※	REF
setBlob	LONGVARBINARY
setClob※	CLOB
setArray	ARRAY

注※

JDBC ドライバでは使用できません。

PreparedStatement クラス及び CallableStatement クラスの setXXX メソッドと各 JDBC SQL タイプとのマッピングを次の表に示します。

表 16–9　PreparedStatement クラス及び CallableStatement クラスの setXXX メソッドと各 JDBC SQL タイプとのマッピング（1/2）

setXXX メソッド	JDBC SQL タイプ					
	SMALLINT	INTEGER	FLOAT	REAL	DECIMAL	CHAR
setByte	○	○	○	○	○	○
setShort	◎	○	○	○	○	○
setInt	○	◎	○	○	○	○
setLong	○	○	○	○	○	○
setFloat	○	○	○	◎	○	○
setDouble	○	○	◎	○	○	○
setBigDecimal	○	○	○	○	◎	○
setBoolean	○	○	○	○	○	○
setString	○	○	○	○	○	◎
setBytes	×	×	×	×	×	×
setDate	×	×	×	×	×	○
setTime	×	×	×	×	×	○
setTimestamp	×	×	×	×	×	○
setAsciiStream	×	×	×	×	×	○
setUnicodeStream	×	×	×	×	×	○
setBinaryStream	×	×	×	×	×	×
setObject	○	○	○	○	○	○
setCharacterStream	×	×	×	×	×	○
setArray	×	×	×	×	×	×
setBlob	×	×	×	×	×	×
setClob※	×	×	×	×	×	×
setRef※	×	×	×	×	×	×

（凡例）

◎：マッピングすることを推奨します。

○：マッピングできます。なお，変換元データの形式によっては，データの欠落や変換エラーとなることがあるため，注意してください。

×：マッピングできません。

注※

JDBC ドライバでは使用できません。

表 16–10　PreparedStatement クラス及び CallableStatement の setXXX メソッドと各 JDBC SQL タイプとのマッピング（2/2）

setXXX メソッド	JDBC SQL タイプ					
	VARCHAR	DATE	TIME	TIMESTAMP	LONGVARBINARY	ARRAY
setByte	○	×	×	×	×	×
setShort	○	×	×	×	×	×
setInt	○	×	×	×	×	×
setLong	○	×	×	×	×	×
setFloat	○	×	×	×	×	×
setDouble	○	×	×	×	×	×
setBigDecimal	○	×	×	×	×	×
setBoolean	○	×	×	×	×	×
setString	◎	○	○	○	○	×
setBytes	×	×	×	×	○	×
setDate	○	◎※2	×	○	×	×
setTime	○	×	◎	○	×	×
setTimestamp	○	○	×	◎	×	×
setAsciiStream	○	×	×	×	○	×
setUnicodeStream	○	×	×	×	○	×
setBinaryStream	×	×	×	×	◎	×
setObject	○	○	○	○	○	○
setCharacterStream	○	×	×	×	○	×
setArray	×	×	×	×	×	◎
setBlob	×	×	×	×	○	×
setClob※1	×	×	×	×	×	×
setRef※1	×	×	×	×	×	×

（凡例）

◎：マッピングすることを推奨します。

○：マッピングできます。なお，変換元データの形式によっては，データの欠落や変換エラーとなることがあるため，注意してください。

×：マッピングできません。

注※1

JDBC ドライバでは使用できません。

注※2

JDBC SQL タイプが DATE 型の場合，setDate メソッドに java.util.Calendar オブジェクトを指定して実行すると，指定した java.util.Calendar オブジェクトを使用してデータを変換し，時刻データを切り捨てて日付データだけをデータベースに格納します。このとき，時刻データを切り捨てるため，getDate メソッドに java.util.Calendar オブジェクトを指定して，setDate メソッドで格納したデータを取得しても，setDate メソッドに指定した日付と異なる日付を取得する場合があります。

（例）

日本標準時をデフォルトのタイムゾーンとする UAP で，setDate メソッド，及び getDate メソッドに世界標準時のタイムゾーンを持つ java.util.Calendar オブジェクトを指定した場合の例を次に示します。

setDate メソッドに「2005-10-03」を表す java.sql.Date オブジェクトを指定して実行した場合，JDBC ドライバは時刻部分に「00:00:00」を補完した後，タイムゾーンの違いによって 9 時間遅らせて「2005-10-02 15:00:00」とし，日付部分「2005-10-02」をデータベースに格納します。このデータを getDate メソッドで取得した場合，データベースから日付部分「2005-10-02」を取得し，時刻部分に「00:00:00」を補完した後，タイムゾーンの違いによって 9 時間進めて「2005-10-02 09:00:00」とします。これによって，getDate メソッドの戻り値の java.sql.Date オブジェクトには，「2005-10-02」が設定されるため，setDate メソッドに指定した「2005-10-03」とは異なります。

16.4　JDBC2.0 Optional Package

16.4.1　DataSource と JNDI を使用した DB 接続

DataSource と JNDI を使用した DB 接続は，JDBC2.0 Optional Package で使用できるようになりました。

必ずしも JNDI を使用する必要はありませんが，JNDI を使用することで接続情報の設定が 1 回で済むというメリットがあります。DataSource クラスのインタフェース定義，及び JNDI は，JDK に標準で含まれていないため，AP 開発をする場合には，JavaSoft の Web サイトから入手する必要があります。

DataSource と JNDI を使用した DB 接続の手順を次に示します。

1. DataSource オブジェクトの生成
2. 接続情報の設定
3. JNDI への DataSource の登録
4. JNDI からの DataSource の取得
5. DB 接続

JNDI を使用しない場合は，3 及び 4 の操作をする必要はありません。

JNDI を使用する場合，1～3 の操作は 1 回だけ実行します。その後，4 及び 5 の操作をするだけで，DB 接続ができます。また，4 の操作の後，必要に応じて接続情報を変更できます。

(1)　DataSource オブジェクトの生成

JDBC ドライバが提供する，DataSource クラスのオブジェクトを生成します。

DataSource クラスのオブジェクト生成で必要となる，JDBC ドライバの DataSource クラス名は **JdbhDataSource** となります。

DataSource クラスのオブジェクトの生成例を次に示します。

```
JP.co.Hitachi.soft.HiRDB.JDBC.JdbhDataSource  ds = null ;
ds = new JP.co.Hitachi.soft.HiRDB.JDBC.JdbhDataSource() ;
```

(2)　接続情報の設定

DataSource オブジェクトに対して，接続情報設定用メソッドを呼び出し，接続情報の設定をします。接続情報取得用のメソッドも使用できるため，現在の接続情報の確認もできます。接続情報設定／取得メソッドについては，「接続情報設定／取得インタフェース」を参照してください。

(3) JNDI への DataSource の登録

DataSource オブジェクトを JNDI に登録します。

JNDI は，その実行環境によって幾つかのサービスプロバイダを選択できます。

DataSource オブジェクトの JNDI への登録例を次に示します（Windows の場合の例です）。なお，登録例では，サービスプロバイダの一つである，File System サービスプロバイダを使用しています。ほかのサービスプロバイダについては，JNDI のドキュメントを参照してください。

```
// JDBCドライバが提供するDataSourceクラスのオブジェクトを生成する
JP.co.Hitachi.soft.HiRDB.JDBC.JdbhDataSource  ds;
ds = new JP.co.Hitachi.soft.HiRDB.JDBC.JdbhDataSource();

// 接続情報を設定する
        :
// システムプロパティを取得する
Properties sys_prop = System.getProperties() ;

// File Systemサービスプロバイダのプロパティを設定する
sys_prop.put(Context.INITIAL_CONTEXT_FACTORY,
             ”com.sun.jndi.fscontext.RefFSContextFactory”);
// File Systemサービスプロバイダで使用するディレクトリを設定する
//（この場合，c:¥JNDI_DIRの下に登録される）
sys_prop.put(Context.PROVIDER_URL, ”file:c:¥¥” + ”JNDI_DIR”);

// システムプロパティを更新する
System.setProperties(sys_prop) ;
// JNDIを初期化する
Context ctx = new InitialContext();

// HiRDBドライバが提供するDataSourceクラスのオブジェクトを，
// jdbc/TestDataSourceという論理名称でJNDIに登録する
ctx.bind(”jdbc” + ”¥¥” + ”TestDataSource”, ds);
                      :
```

なお，JDBC2.0 規格では，JNDI に登録する論理名称は，"jdbc"というサブコンテキスト下（登録例では jdbc/TestDataSource）に登録するように推奨されています。

(4) JNDI からの DataSource の取得

JNDI から DataSource オブジェクトを取得します。

JNDI からの DataSource オブジェクトの登録例を次に示します（Windows の場合の例です）。なお，登録例では，サービスプロバイダの一つである，File System サービスプロバイダを使用しています。ほかのサービスプロバイダについては，JNDI のドキュメントを参照してください。

```
// システムプロパティを取得する
Properties sys_prop = System.getProperties() ;

// File Systemサービスプロバイダのプロパティを設定する
sys_prop.put(Context.INITIAL_CONTEXT_FACTORY,
```

```
            "com.sun.jndi.fscontext.RefFSContextFactory");
// File Systemサービスプロバイダで使用するディレクトリを設定する
//（この場合，c:¥JNDI_DIRの下に登録されている）
sys_prop.put(Context.PROVIDER_URL, "file:c:¥¥" + "JNDI_DIR");
// システムプロパティを更新する
System.setProperties(sys_prop) ;

// JNDIを初期化する
Context ctx = new InitialContext();
// jdbc/TestDataSourceという論理名称のオブジェクトをJNDIから取得する
Object obj = ctx.lookup("jdbc" + "¥¥" + "TestDataSource") ;

// 取り出したオブジェクトを，DataSourceクラスの型にキャストする
DataSource ds = (DataSource)obj;
              :
```

(5)　DB 接続

DataSource オブジェクトに対して，getConnection メソッドを呼び出します。

getConnection メソッドの呼び出し例を次に示します。

```
DataSource ds

// JNDIからDataSourceオブジェクトを取得する
                    :
// getConnectionメソッドを発行する
Connection con = ds.getConnection();
  又は
Connection con = ds.getConnection("USERID", "PASSWORD");※
```

注※

メソッドの引数（認可識別子，パスワード）は，DataSource オブジェクトに設定した接続情報よりも優先されます。必要な接続情報が DataSource オブジェクトに設定されていない場合，接続情報の内容が不正な場合，又は HiRDB サーバとの接続に失敗した場合，getConnection メソッドは SQLException を投入します。

JNDI から DataSource オブジェクトを取得後，必要に応じて接続情報を再度設定できます。この場合，DataSource オブジェクトを，JDBC ドライバが提供する DataSource クラスの型にキャストしてから設定する必要があります。例を次に示します。

```
DataSource ds
JP.co.Hitachi.soft.HiRDB.JDBC.JdbhDataSource  hirdb_ds;

// JNDIからDataSourceオブジェクトを取得する
                    :
// DataSourceオブジェクトを，JDBCドライバが提供する
// DataSourceクラスの型にキャストする
dbp_ds = (JP.co.Hitachi.soft.HiRDB.JDBC.JdbhDataSource)ds;
```

```
// 接続情報を再設定する
        :
```

16.4.2　接続プール

JDBC2.0 Optional Package では，DB との接続をプールする機能が規定されています。接続プールの概要を次に示します。

- 既存のアプリケーションに影響を与えません。つまり，アプリケーションは接続プールの有無を意識する必要がありません。ただし，DB 接続が DriverManager を使用する方法ではなく，JDBC2.0 Optional Package で導入された DataSource と JNDI を使用する方法で行っていることが前提となります。
- 接続プールを行う機能自体は，JDBC 規格の機能範囲外です。これは，システム構築時に，接続プール機能をユーザが任意に選択できるようにするためです（ユーザが自分で作成するか，APServer ベンダ提供のものを使用するか，又は JDBC ベンダが提供するものを使用します）。
- 接続プール機能では，アプリケーションとのインタフェースとして DataSource クラスを使用できます。この DataSource クラスは，JDBC ドライバが提供する DataSource クラスとは別のものです。
- JDBC ドライバでは，接続プール機能とのインタフェースとして，ConnectionPoolDataSource クラスと PooledConnection クラスを使用できます。
- JDBC ドライバが提供する ConnectionPoolDataSource クラスは，JDBC ドライバが提供する DataSource クラスと同様に，接続情報の設定／取得用のメソッドを使用できます。

接続プールに関連するクラスを次の表に示します。

表 16–11　接続プールに関連するクラス

クラス	概　要
javax.sql.DataSource	• 接続プール機能が提供します。 • DB 接続のとき，アプリケーションとのインタフェースとして使用します。 • 通常，接続プールの制御はこのクラスで行います。 • 通常，JNDI に登録して使用します。 • JDBC ドライバが提供する DataSource クラスとは別のものです。
javax.sql.ConnectionPoolDataSource	• JDBC ドライバが提供します。 • DB 接続に必要な，接続情報設定／取得用のメソッドを使用できます。 • 通常，アプリケーションから直接使用することはなく，接続プール機能で使用されます。 • 通常，JNDI に登録して使用します。 • 接続プール機能は，このクラスのオブジェクトから，PooledConnection オブジェクトを取得します。
javax.sql.PooledConnection	• JDBC ドライバが提供します。 • 通常，アプリケーションから直接使用することはなく，接続プール機能で使用されます。

クラス	概　要
	• 接続プール機能は，このクラスのオブジェクトをプールの対象とします。 • 接続プール機能は，このクラスのオブジェクトから，アプリケーションが使用する Connection オブジェクトを取得します。
javax.sql.ConnectionEventListener	• 接続プール機能が提供します。 • 接続プール機能は，接続の切断／SQL エラーなどを，このクラスのオブジェクトを通して検知することで，接続プールの契機とします。

表「接続プールに関連するクラス」のクラスのインタフェース定義は，JDK のバージョンによっては JDK に標準で含まれていないため，接続プール機能を使用する場合は JavaSoft の Web サイトで確認する必要があります。

表「接続プールに関連するクラス」の，JDBC ドライバが提供するクラスのパッケージ名称とクラス名称を次に示します。

パッケージ名称：**JP.co.Hitachi.soft.HiRDB.JDBC**

ConnectionPoolDataSource クラス名称：**JdbhConnectionPoolDataSource**

PooledConnection クラス名称：**JdbhPooledConnection**

なお，JDBC ドライバが提供する ConnectionPoolDataSource クラスの接続情報の設定は，JDBC ドライバが提供する DataSource クラスの接続情報の設定と同じです。

16.4.3　分散トランザクション

JDBC2.0 Optional Package では，接続プール機能の拡張として，X/Open の XA 規格を基にしたトランザクションマネジャ（TM）との連携による，分散トランザクションが規定されています。分散トランザクションの概要を次に示します。

- 既存のアプリケーションには，ほとんど影響を与えません。ただし，「直接 commit してはいけません」などの制限があります。また，接続プールの場合と同様に，DB 接続は DriverManager を使用する方法ではなく，JDBC2.0 Optional Package で導入された DataSource と JNDI を使用する方法で行っていることが前提となります。
- TM との連携をするトランザクション連携機能は，接続プールの場合と同様に，JDBC 規格の機能範囲外です。
- 通常，トランザクション連携機能は，接続プール機能の拡張として実装され，TM とのインタフェースとして，TM が提供する JTA や JTS を使用します。なお，JTA 規格 1.0 に準拠した動作は保証しません。
- トランザクション連携機能では，接続プールの場合と同様に，アプリケーションとのインタフェースとして DataSource クラスを使用できます。この DataSource クラスは，JDBC ドライバが提供する DataSource クラスとは別のものです。

- JDBC ドライバでは，トランザクション連携機能とのインタフェースとして，XADataSource クラスと XAConnection クラスを使用できます。また，TM とのインタフェースとして XAResource クラスを使用できます。
- JDBC ドライバが提供する XADataSource クラスは，JDBC ドライバが提供する DataSource クラスと同様に，接続情報の設定／取得用のメソッドを使用できます。

接続プールの場合と同様に，アプリケーションで使用する Connection オブジェクトは XAConnection クラスが生成します。ただし，PooledConnection クラスや JDBC ドライバが提供する，DataSource クラスが生成する Connection オブジェクトと比べて，次の点が異なります。

- Connection クラスの commit メソッド及び rollback メソッドの呼び出しは，SQLException となります。つまり，アプリケーションから直接トランザクションの決着はできません。
- AutoCommit のデフォルトモードは OFF です。
- AutoCommit のモードを ON にする Connection クラスの setAutoCommit（true）メソッドの発行は，SQLException となります。

分散トランザクションに関連するクラスを次の表に示します。

表 16–12　分散トランザクションに関連するクラス

クラス	概　要
javax.sql.DataSource	• トランザクション連携機能が提供します。 • DB 接続のとき，アプリケーションとのインタフェースとして使用します。 • 通常，TM との連携と，接続プールの制御は，このクラスで行われます。 • 通常，JNDI に登録して使用します。 • JDBC ドライバが提供する DataSource クラスとは別のものです。
javax.sql.XADataSource	• JDBC ドライバが提供します。 • DB 接続に必要な，接続情報設定／取得用のメソッドを使用できます。 • 通常，アプリケーションから直接使用されることはなく，トランザクション連携機能によって使用されます。 • 通常，JNDI に登録して使用します。 • トランザクション連携機能は，このクラスのオブジェクトから XAConnection オブジェクトを取得します。
javax.sql.XAConnection	• JDBC ドライバが提供します。 • PooledConnection クラスのサブクラスです。つまり，接続プールに関連するメソッドをすべて引き継ぎます。 • 通常，アプリケーションから直接使用されることはなく，トランザクション連携機能によって使用されます。 • トランザクション連携機能は，このクラスのオブジェクトをプールの対象とします。 • トランザクション連携機能は，このクラスのオブジェクトからアプリケーションが使用する，Connection オブジェクトを取得します。
javax.sql.ConnectionEventListener	• トランザクション連携機能が提供します。

クラス	概　要
	• トランザクション連携機能は，接続の切断／SQL エラーなどをこのクラスのオブジェクトを通して検知することで，接続プールの契機とします。
javax.transaction.xa.XAResource	• JDBC ドライバが提供します。 • TM で使用される XA 関連のメソッドを使用できます。
javax.transaction.xa.Xid	• JDBC ドライバ及び TM が提供します。 • XAResource クラスのメソッドの引数／戻り値として使用します。

表「分散トランザクションに関連するクラス」のクラスのインタフェース定義は JDK に標準で含まれていないため，トランザクション連携機能の開発の際には，JavaSoft の Web サイトから入手する必要があります。

表「分散トランザクションに関連するクラス」の JDBC ドライバが提供するクラスの，パッケージ名称とクラス名称を次に示します。

パッケージ名称：**JP.co.Hitachi.soft.HiRDB.JDBC**

XADataSource クラス名称：**JdbhXADataSource**

XAConnection クラス名称：**JdbhXAConnection**

XAResource クラス名称：**JdbhXAResource**

Xid クラス名称：**JdbhXid**

なお，JDBC ドライバが提供する XADataSource クラスの接続情報の設定は，JDBC ドライバが提供する DataSource クラスの接続情報の設定と同じです。

16.5　JAR ファイルアクセス機能

Java ストアドルーチンを使用する場合，JAR ファイルを HiRDB へ登録する必要があります。このとき，JDBC ドライバ経由で処理がされます。

ここでは，JAR ファイルの登録，削除，及び再登録をするときの，クラス名及びメソッド名について説明します。

16.5.1　クラス名

クラス名を次に示します。

```
JP.co.Hitachi.soft.HiRDB.JDBC.Jdbh_JARAccess
```

16.5.2　メソッド名

(1)　JAR ファイルの HiRDB への登録

(a)　書式

```
  public  void  Jdbh_JARInstall(java.sql.Connection   con,
                                String                JarName)
```

(b)　引数

con：

JAR ファイルを登録するための Java.sql.Connection オブジェクトを指定します。

JarName：

JAR ファイルの名称を指定します。

絶対パス名又は相対パス名で指定します。ほかのサーバマシンのファイルは指定できません。また，ワイルドカードは指定できません。

(c)　戻り値

なし。

(d)　例外

SQLException：データベースへのアクセスエラーが発生した場合

(e) 機能

Java.sql.Connection オブジェクトを使用して，指定した JAR ファイルを HiRDB へ登録します。HiRDB に，既に同じ名称のファイルがある場合はエラーとなります。

(2) JAR ファイルの HiRDB からの削除

(a) 書式

```
public  void  Jdbh_JARUnInstall(java.sql.Connection   con,
                                String                JarName)
```

(b) 引数

con：

JAR ファイルを削除するための Java.sql.Connection オブジェクトを指定します。

JarName：

JAR ファイルの名称を指定します。

絶対パス名又は相対パス名では指定できません。また，ワイルドカードは指定できません。

(c) 戻り値

なし。

(d) 例外

SQLException：データベースへのアクセスエラーが発生した場合

(e) 機能

Java.sql.Connection オブジェクトを使用して，指定した JAR ファイルを HiRDB から削除します。

(3) JAR ファイルの HiRDB への再登録

(a) 書式

```
public  void  Jdbh_JARReInstall(java.sql.Connection   con,
                                String                JarName)
```

(b) 引数

con：

JAR ファイルを再登録するための Java.sql.Connection オブジェクトを指定します。

JarName：

JAR ファイルの名称を指定します。

絶対パス名又は相対パス名で指定します。ほかのサーバマシンのファイルは指定できません。ワイルドカードは指定できません。

(c) 戻り値

なし。

(d) 例外

SQLException：データベースへのアクセスエラーが発生した場合

(e) 機能

Java.sql.Connection オブジェクトを使用して，指定した JAR ファイルを HiRDB に再登録します。HiRDB に，既に同じ名称のファイルがある場合は上書きされます（エラーにはなりません）。

16.6　Array クラス

JDBC ドライバでは，Array クラスを使用して繰返し列をアクセスできます。各メソッドの注意事項を次に示します。

16.6.1　getArray

- MAP は使用できません。
- このメソッドで返すオブジェクトの型を次の表に示します。

表 16–13　getArray で返すオブジェクトの型

HiRDB のデータ型	オブジェクトの型
INTEGER	java.lang.Integer[]
SMALLINT	java.lang.Short[]
DECIMAL	java.math.BigDecimal[]
FLOAT，DOUBLE PRECISION	java.lang.Double[]
SMALLFLT，REAL	java.lang.Float[]
CHAR	java.lang.String[]
VARCHAR	java.lang.String[]
NCHAR	java.lang.String[]
NVARCHAR	java.lang.String[]
MCHAR	java.lang.String[]
MVARCHAR	java.lang.String[]
DATE	java.sql.Date[]
TIME	java.sql.Time[]
TIMESTAMP	java.sql.Timestamp[]

16.6.2　getResultSet

- MAP は使用できません。
- このメソッドで返す結果セットは，配列要素ごとに 1 行を含んでいて，各行には二つの列があります。二つ目の列には要素の値が格納され，一つ目の列には配列内の対応する要素のインデクスが格納されます（最初の配列要素のインデクスは 1）。行は，インデクスに基づいて昇順で並べられます。
- ステートメントがクローズされると，このメソッドで返した結果セットもクローズします。

- このメソッドで返す結果セットの属性値を次の表に示します。

表 16–14　getResultSet で返す結果セットの属性値

ResultSetMetaData クラスのメソッド名	メソッドが返す値	
	一つ目の列	二つ目の列
getCatalogName	null	null
getColumnClassName	java.lang.Integer	データベースの列の属性に依存
getColumnDisplaySize	10	データベースの列長に依存
getColumnLabel	JDBC_Array_Index	データベースの列名に依存
getColumnName		
getColumnType	java.sql.Types.INTEGER	データベースの列の属性に依存
getColumnTypeName	INTEGER	
getPrecision	10	データベースの列属性と列長に依存
getScale	0	
getSchemaName	null	null
getTableName		
isAutoIncrement	true	false
isCaseSensitive	false	データベースの列の属性に依存
isCurrency		false
isDefinitelyWritable		
isNullable	java.sql.ResultSetMetaData.columnNoNulls	データベースの列の属性に依存
isReadOnly	true	false
isSearchable	false	true
isSigned		データベースの列の属性に依存
isWritable		false

16.7　繰返し列を?パラメタにしたときの値の指定方法

繰返し列を?パラメタにしたときの値の指定方法について説明します。

?パラメタに値を指定する方法として，Array インタフェースを実装したクラスのオブジェクト，又は配列のオブジェクトを，setObject メソッドで指定します。

16.7.1　Array インタフェースを実装したクラスのオブジェクトの指定方法

- Array インタフェースを実装したクラスのオブジェクトを作成し，そのオブジェクトを setArray メソッド又は setObject メソッドで指定します。
- JDBC ドライバでは，Array.getBaseType メソッドを使用して，そのオブジェクトのデータ型を調べます。そのとき，データベースのデータ型とオブジェクトのデータ型が異なると，SQLException を投入します。データベースのデータ型とオブジェクトのデータ型については，「データ型」を参照してください。
- 実際のデータは，引数なしの Array.getArray()メソッドで取得します。そのときに返ってこなければならないオブジェクトの型を次の表に示します。返ってきたオブジェクトの型が次の表と異なる場合は，SQLException を投入します。

表 16–15　引数なしの Array.getArray()メソッド取得時に返ってくるオブジェクトの型

Array.getBaseType メソッドで返すデータ型	引数なしの Array.getArray()メソッド取得時に返ってくるオブジェクトの型
java.sql.Types.INTEGER	int[]，又は java.lang.Integer[]
java.sql.Types.SMALLINT	short[]，又は java.lang.Short[]
java.sql.Types.DECIMAL	java.math.BigDecimal[]
java.sql.Types.FLOAT	double[]，又は java.lang.Double[]
java.sql.Types.REAL	float[]，又は java.lang.Float[]
java.sql.Types.CHAR	java.lang.String[]
java.sql.Types.VARCHAR	java.lang.String[]
java.sql.Types.DATE	java.sql.Date[]
java.sql.Types.TIME	java.sql.Time[]
java.sql.Types.LONGVARBINARY	java.io.DataInputStream[]
java.sql.Types.TIMESTAMP	java.sql.Timestamp[]

16.7.2　配列オブジェクトを setObject メソッドで指定する方法

- データベースのデータ型と配列オブジェクトのデータ型が異なる場合は，SQLException を投入します。
- setObject メソッドで指定した SQL のデータ型と，配列オブジェクトのデータ型が，次の表と異なる場合は SQLException を投入します。

表 16–16　setObject メソッドで指定した SQL のデータ型と配列オブジェクトのデータ型

setObject メソッドで指定した SQL のデータ型	配列オブジェクトのデータ型
java.sql.Types.INTEGER	int[]又は java.lang.Integer[]
java.sql.Types.SMALLINT	short[]又は java.lang.Short[]
java.sql.Types.DECIMAL	java.math.BigDecimal[]
java.sql.Types.FLOAT	double[]又は java.lang.Double[]
java.sql.Types.REAL	float[]又は java.lang.Float[]
java.sql.Types.CHAR	java.lang.String[]
java.sql.Types.VARCHAR	java.lang.String[]
java.sql.Types.DATE	java.sql.Date[]
java.sql.Types.TIME	java.sql.Time[]
java.sql.Types.LONGVARBINARY	java.io.DataInputStream[]
java.sql.Types.TIMESTAMP	java.sql.Timestamp[]

16.7.3　繰返し列の要素と?パラメタに指定したオブジェクトとの関係

Array インタフェースを実装したクラスの，オブジェクトから Array.getArray()メソッドで得られた，配列オブジェクトの順序と繰返し列の順序は同じになります。そのため，配列オブジェクトの第 1 要素は繰返し列の第 1 要素となり，配列オブジェクトの第 2 要素は繰返し列の第 2 要素となります。

setObject メソッドで指定した配列オブジェクトも同じです。また，要素数 1 だけの配列オブジェクトも指定できます。

16.7.4　繰返し列の途中の要素にナル値を指定する方法

Array インタフェースを実装したクラスのオブジェクト，又は配列のオブジェクトに関係なく，要素の途中にナル値を指定する場合は，該当する配列の要素を null にします。そのため，繰返し列の第 2 要素をナル値にしたい場合は，Array インタフェースを実装したクラスのオブジェクトから，Array.getArray()メソッドで得られた配列オブジェクトの第 2 要素を null にします。

setObject メソッドで指定した配列オブジェクトも同じです。

16.8　HiRDB JDBC ドライバの提供機能

JDBC2.0 で規格されていない HiRDB JDBC ドライバだけの機能について説明します。

16.8.1　提供クラス

HiRDB JDBC ドライバだけで提供する機能を使用する場合，次のクラスを使用する必要があります。

インタフェース名称	主な機能	クラス名称
PreparedStatement	• ?パラメタ付きの SQL の実行 • ?パラメタに対する値の設定 • Statement の機能（Statement のサブクラスであるため，機能はすべて引き継ぎます）	JdbcDbpsvPreparedStatement

16.8.2　setBlockUpdate

(1)　機能

?パラメタを使用したデータベースの更新で，複数のパラメタセットを一度に処理するかどうかを設定します。

(2)　形式

```
public void setBlockUpdate(boolean Mode)
```

(3)　引数

boolean Mode：

複数のパラメタセットを一度に処理するかどうかの設定情報です。省略した場合，false が仮定されます。

true：

一度に処理します。

false：

パラメタセットを一つずつ分割して処理します。※

注※

DB 接続時，DriverManager クラスの getConnection メソッドの引数に BLOCK_UPDATE=TRUE を指定した場合，この機能のデフォルトは true となります。また，システムプロパティの HiRDB_for_Java_BLOCK_UPDATE=TRUE を指定した場合も，この機能のデフォルトは true となります。

(4)　戻り値

なし。

(5)　機能詳細

?パラメタを使用したデータベースの更新（INSERT，UPDATE，又は DELETE）で，複数のパラメタセットを一度に処理するかどうかを設定します。

なお，実際にパラメタセットが一度に処理されるかどうかは，配列を使用した機能の使用方法で決まります。配列を使用した機能の使用方法については，「配列を使用した機能」を参照してください。

(6)　発生する例外

なし。

(7)　注意事項

複数行の?パラメタの一括処理をする方法については，「Properties info の設定内容」，及び「バッチ更新」を参照してください。

16.8.3　getBlockUpdate

(1)　機能

?パラメタを使用したデータベースの更新で，複数のパラメタセットを一度に処理するかどうかを取得します。

(2)　形式

```
public boolean getBlockUpdate()
```

(3)　引数

なし。

(4)　戻り値

boolean：

複数のパラメタセットを一度に処理するかどうかの情報です。省略した場合，false が仮定されます。

true：

一度に処理します。

false：

パラメタセットを一つずつ分割して処理します。※

注※

DB 接続時，DriverManager クラスの getConnection メソッドの引数に BLOCK_UPDATE=TRUE を指定した場合，この機能のデフォルトは true となります。

(5) 機能詳細

?パラメタを使用したデータベースの更新（INSERT，UPDATE，又は DELETE）で，複数のパラメタセットを一度に処理するかどうかを取得します。

(6) 発生する例外

なし。

16.9　BLOB 型を使用する場合の注意事項

BLOB 型を使用する場合のメソッドの注意点などについて説明します。

16.9.1　各メソッドの処理内容と注意事項

各メソッドの処理内容と注意事項を次の表に示します。

表 16–17　各メソッドの処理内容と注意事項

Blob インタフェースクラスのメソッド名	処理内容と注意事項
getBinaryStream	JdbhInputStream を実装とした InputStream クラスを返します。取得できるデータ長は 2,147,483,639 までです。
getBytes(long pos, int length)	指定した pos 位置から，最大 length までのデータを byte[]オブジェクトで返します。データベースの内容がナル値の場合，指定した位置からデータが取得できない場合，又は 0 バイトデータの場合は，null を返します。ただし，length の値は 2,147,483,639 まで有効であるため，それを超えると SQLException を投入します。
length()	実データ長を返します。
position(Blob pattern,long start)	position(pattern.getBytes(1, (int)(pattern.length())), start)に置き換えて実行します。ただし，pattern に null を指定した場合は，NullPointerException を投入します。
position(byte[] pattern,long start)	指定した start 位置から，pattern に合った位置を返します。返す値は>=start となります。pattern に合った位置が見付からなかった場合，-1 を返します。ただし，pattern.length の値が 2,147,483,639 まで有効であるため，それを超えると SQLException を投入します。また，pattern に null を指定した場合は，NullPointerException を投入します。
setBinaryStream(long pos)	無条件に SQLException を投入します。
setBytes(long pos,byte[] bytes)	
setBytes(long pos,byte[] bytes,int offset,int len)	
truncate(long len)	

注

位置付け子機能を使用してデータを取得した場合，ResultSet.close()や Statement.close()などを行うと，データが取得できなくなります。

16.9.2　?パラメタでの指定方法

？パラメタに値を指定する場合，PreparedStatement. setBlob()メソッド，及びCallableStatement.setBlob()メソッドを使用する方法があります。これらを使用する場合の注意点を次に示します。

(1)　Blob インタフェースを実装したオブジェクトを使用する場合

setBlob()メソッドを使用する場合，Blob インタフェースを実装したオブジェクトを指定する必要があります。また，Blob インタフェースを実装したオブジェクトを UAP が作成する必要があります。

JDBC では，Blob.getBytes()メソッドで，設定する値を byte[]の形式で取得します。次に示す方法で，設定する値を取得します。

```
Blob.getBytes(1, (int)(Blob.length()))
```

UAP では，getBytes()メソッドと length()メソッドでは，正常な値を返す必要があります。JDBC では，これらのメソッドで返される値が正しいものとして動作します。

(2)　ResultSet.getBlob()メソッド，又は CallableStatement.getBlob()メソッドで取得した Blob オブジェクトを使用する場合

JDBC からの実行結果として，ResultSet.getBlob()メソッドや CallableStatement.getBlob()メソッドで取得した Blob オブジェクトをそのまま使用する場合，位置付け子機能を使用したアクセスでの取得かどうかで動作が異なります。

- 位置付け子機能を使用したアクセスでない場合

  ResultSet.getBlob()メソッドや CallableStatement.getBlob()メソッドで取得した時点でのデータを，?パラメタの値とします。
- 位置付け子機能を使用したアクセスの場合

  setBlob()メソッドを呼び出した場合に，内部で Blob.getBytes(1, (int)(Blob.length()))を実行します。?パラメタの値は，Blob.getBytes(1, (int)(Blob.length()))で取得されたデータとなります。

16.10　システムプロパティの設定

16.10.1　配列機能の設定

(1)　概要

プログラム実行時に，システムプロパティの HiRDB_for_Java_BLOCK_UPDATE を設定することで，?パラメタを使用したデータベースの更新（INSERT，UPDATE，又は DELETE）で，複数のパラメタセットを一度に処理するかどうかを指定できます。

(2)　設定方法

プログラム実行時に，java コマンドの-D オプションでシステムプロパティの HiRDB_for_Java_BLOCK_UPDATE を設定します。

(a)　機能

?パラメタを使用したデータベースの更新（INSERT，UPDATE，又は DELETE）で，複数のパラメタセットを一度に処理するかどうかを設定します。

(b)　形式

```
java -D<name>=<value> クラス名
```

(c)　説明

name：

HiRDB_for_Java_BLOCK_UPDATE

value：

TRUE：一度に処理します。

FALSE：パラメタセットを一つずつ分割して処理します。

上記以外：パラメタセットを一つずつ分割して処理します。

(d)　機能詳細

?パラメタを使用したデータベースの更新で，複数のパラメタセットを一度に処理するかどうかを設定します。

実際にパラメタセットが一度に処理されるかどうかは，配列を使用した機能の使用方法で決まります。配列を使用した機能の使用方法については，「配列を使用した機能」を参照してください。

(e)　注意事項

- -D<name>=<value> を指定する場合は，指定内容に空白を含めないでください。指定内容が次の例のどれかに該当する場合，正しく設定されません。△は空白を示します。
  - -D△<name>=<value>
  - -D<name>△=<value>
  - -D<name>=△<value>
- DB 接続時に BLOCK_UPDATE（データソース接続時は setBlockUpdate メソッド）を設定している場合は，BLOCK_UPDATE，又は setBlockUpdate メソッドの設定値が有効になります。
- PreparedStatement クラスの setBlockUpdate メソッドを使用した場合，DB 接続時の設定に関係なく，複数のパラメタセットを一度に処理するかどうかの設定を変更できます。
- 複数行の?パラメタの一括処理をする方法については，「Properties info の設定内容」，及び「バッチ更新」を参照してください。

(f)　設定例

システムプロパティの HiRDB_for_Java_BLOCK_UPDATE の設定例を次に示します。

```
java -DHiRDB_for_Java_BLOCK_UPDATE=TRUE TestUP
```

16.10.2　SQL の検索項目，又は?パラメタの最大数の設定

(1)　概要

プログラム実行時に，システムプロパティの HiRDB_for_Java_SQL_IN_NUM，又は HiRDB_for_Java_SQL_OUT_NUM を設定することで，SQL の前処理時に取得する検索項目，出力?パラメタ，入力?パラメタ，又は入出力?パラメタ情報の最大数を指定できます。

(2)　設定方法

プログラム実行時に，java コマンドの-D オプションで，システムプロパティの HiRDB_for_Java_SQL_OUT_NUM，HiRDB_for_Java_SQL_IN_NUM のどちらか，又は両方を設定します。

(a)　機能

SQL の前処理時に取得する検索項目，出力?パラメタ，入力?パラメタ，又は入出力?パラメタ情報の最大数を指定します。

(b)　形式

```
java -D<name>=<value> クラス名
```

(c) 説明

<name>，及び<value>に指定できる内容を次に示します。

<name>	<value>
HiRDB_for_Java_SQL_IN_NUM	実行する SQL の入力，又は入出力?パラメタの最大数を指定します。この指定は，SQL の前処理時に取得する入力，又は入出力?パラメタ情報の数となります。実際の入力，又は入出力?パラメタの数が，このプロパティの指定値よりも多い場合，SQL の前処理の後に入力，又は入出力?パラメタ情報を取得します。 指定値は，1～30,000 です（デフォルトは 64）。これ以外の値，又は数字以外を指定した場合は，DB 接続時にエラーとなります。
HiRDB_for_Java_SQL_OUT_NUM	実行する SQL の出力項目数の最大数を指定します。この指定は SQL の前処理時に取得する出力項目情報の数となります。実際の出力項目情報の数が，このプロパティの指定値よりも多い場合，SQL の前処理の後に出力項目情報を取得します。 指定値は，1～30,000 です（デフォルトは 64）。これ以外の値，又は数字以外を指定した場合は，DB 接続時にエラーとなります。

(d) 機能詳細

SQL の前処理時に取得する検索項目，出力?パラメタ，入力?パラメタ，又は入出力?パラメタ情報の最大数を指定します。この指定を十分な大きさにすることで，SQL の前処理と，検索項目，出力?パラメタ，入力?パラメタ，又は入出力?パラメタ情報を同時に取得でき，前処理後に情報を取得する場合に比べて性能が向上します。

(e) 注意事項

- -D<name>=<value> を指定する場合は，指定内容に空白を含めないでください。指定内容が次の例のどれかに該当する場合，正しく設定されません。△は空白を示します。
  - -D△<name>=<value>
  - -D<name>△=<value>
  - -D<name>=△<value>
- DB 接続時，HiRDB_for_Java_SQL_IN_NUM（データソース接続時は setSQLInNum メソッド）を設定している場合は，HiRDB_for_Java_SQL_IN_NUM，又は setSQLInNum メソッドの設定値が有効になります。
- DB 接続時，HiRDB_for_Java_SQL_OUT_NUM（データソース接続時は setSQLOutNum メソッド）を設定している場合は，HiRDB_for_Java_SQL_OUT_NUM，又は setSQLOutNum メソッドの設定値が有効になります。
- SQL の前処理と，検索項目，出力?パラメタ，入力?パラメタ，又は入出力?パラメタ情報の取得を同時実行するには，接続する HiRDB サーバがバージョン 07-02 以降でなければなりません。

(f)　設定例

システムプロパティの HiRDB_for_Java_SQL_IN_NUM，及び HiRDB_for_Java_SQL_OUT_NUM の設定例を次に示します。

```
java -DHiRDB_for_Java_SQL_IN_NUM=128 -DHiRDB_for_Java_SQL_OUT_NUM=128 TestUP
```

16.11　接続情報設定／取得インタフェース

JDBC ドライバで提供する，JdbhDataSource，JdbhConnectionPoolDataSource，及び JdbhXADataSource の各クラスでは，JDBC2.0 Optional Package 規格で定められたメソッドのほかに，DB 接続に必要となる接続情報設定／取得用のメソッドを提供します。

接続情報設定／取得メソッドの一覧を次の表に示します。

表 16–18　接続情報設定／取得メソッドの一覧

メソッド	機能
setDescription	接続する DB に必要な接続付加情報を設定します。
getDescription	接続する DB に必要な接続付加情報を取得します。
setDBHostName	接続する HiRDB のホスト名を設定します。
getDBHostName	接続する HiRDB のホスト名を取得します。
setEncodeLang	指定したエンコード文字コードを使用し，データ変換をします。
getEncodeLang	データ変換で使用するエンコード文字を返却します。
setUser	認可識別子を設定します。
getUser	認可識別子を取得します。
setPassword	パスワードを設定します。
getPassword	パスワードを取得します。
setXAOpenString※	XA_OPEN 文字列を設定します。
getXAOpenString※	XA_OPEN 文字列を取得します。
setXACloseString※	XA_CLOSE 文字列を設定します。
getXACloseString※	XA_CLOSE 文字列を取得します。
setRMID※	リソースマネジャの識別子を設定します。
getRMID※	リソースマネジャの識別子を取得します。
setXAThreadMode※	XA 使用時のスレッドモードを設定します。
getXAThreadMode※	XA 使用時のスレッドモードを取得します。
setCommit_Behavior	カーソルが COMMIT をわたって有効かどうかを設定します。
getCommit_Behavior	カーソルが COMMIT をわたって有効かどうかを取得します。
setBlockUpdate	複数のパラメタセットを，一度に処理するかどうかを指定します。
getBlockUpdate	複数のパラメタセットを，一度に処理するかどうかを取得します。
setLONGVARBINARY_Access	LONGVARBINARY（列属性が BLOB 又は BINARY）のデータベースのアクセス方法を指定します。

メソッド	機能
getLONGVARBINARY_Access	LONGVARBINARY（列属性が BLOB 又は BINARY）のデータベースのアクセス方法を取得します。
setSQLInNum	実行する SQL の入力/入出力?パラメタの最大数を指定します。
getSQLInNum	setSQLInNum で設定した，実行する SQL の入力，又は入出力?パラメタの最大数を取得します。
setSQLOutNum	実行する SQL の検索項目，又は出力若しくは入出力?パラメタの最大数を指定します。
getSQLOutNum	setSQLOutNum で設定した，実行する SQL の検索項目，又は出力若しくは入出力?パラメタの最大数を取得します。
setSQLWarningLevel	SQL 実行時に発生した警告保持レベルを指定します。
getSQLWarningLevel	setSQLWarningLevel で指定した警告保持レベルを取得します。
setClear_Env	DB 接続時に OS の環境変数として設定した HiRDB のクライアント環境定義を無効にするかどうかを指定します。
getClear_Env	setClear_Env で設定した環境変数無効指定を取得します。

注※

JdbhXADataSource クラスでだけ提供されるメソッドです。

16.11.1 setDescription

(1) 機能

接続する DB に必要な接続付加情報を設定します。

(2) 形式

```
public void setDescription ( String description )
```

(3) 引数

String description：

接続付加情報を指定します。

(4) 戻り値

なし。

(5) 機能詳細

接続する DB に必要な接続付加情報を設定します。設定する内容と要否を次に示します。

設定内容	設定する内容	設定の要否
HiRDB のポート番号	HiRDB のポート番号を文字列で設定します。	任意
HiRDB の環境変数グループ名	HiRDB の環境変数グループ名を「@HIRDBENVGRP=」に続けて文字列で設定します。環境変数グループ名に半角空白文字，及び半角@文字を含む場合は，半角引用符（"）で囲んで指定してください。環境変数グループ名を半角引用符で囲んだ場合，最後の半角引用符から文字終端までの文字は無視されます。半角引用符，及び半角コンマを含む環境変数グループ名は指定できません。	任意
HiRDB の環境変数グループ識別子	HiRDB の環境変数グループ識別子を英数字だけの 4 文字で設定します。	XA 接続時は必要

注 1

環境変数グループに登録された環境変数は，ユーザ環境変数や HiRDB.INI で登録した環境変数よりも優先されます。

注 2

指定例を次に示します。なお，指定例では，JdbhDataSource クラスのインスタンスの参照を持つ変数名を「ds」としています。

UNIX 版の場合：

例 1：HiRDB の環境変数グループ名のパスが「/HiRDB_P/Client/HiRDB.ini」の場合

```
ds.setDescription("@HIRDBENVGRP=/HiRDB_P/Client/HiRDB.ini");
```

Windows 版の場合：

例 1：HiRDB のポート番号を指定する場合

```
ds.setDescription("22200");
```

例 2：HiRDB クライアント環境変数登録ツールで登録した環境変数グループ名「HiRDB_ENV_GROUP」を指定する場合

```
ds.setDescription("@HIRDBENVGRP=HiRDB_ENV_GROUP");
```

例 3：HiRDB の環境変数グループ名のパスが「C:¥HiRDB_P¥Client¥HiRDB.ini」の場合

```
ds.setDescription("@HIRDBENVGRP=C:¥¥HiRDB_P¥¥Client¥¥HiRDB.ini");
```

例 4：HiRDB の環境変数グループ名のパスが「C:¥Program△Files¥HITACHI¥HiRDB¥HiRDB.ini」の場合（△は半角空白文字）

```
ds.setDescription("@HIRDBENVGRP=¥"C:¥¥Program△Files¥¥HITACHI¥¥HiRDB¥HiRDB.ini¥"");
```

例 5：HiRDB の環境変数グループ識別子が「HDB1」の場合

```
ds.setDescription(”HDB1”);
```

注 3

接続先のポート番号が 65535 である場合は，クライアント環境定義の PDNAMEPORT 又は環境変数グループを使用して接続先を指定してください。

(6) 発生する例外

XA 接続以外の接続で，@から始まる環境変数グループ名を指定した場合，@より後の指定内容に半角スペースがあるときは，SQLException を投入します。

16.11.2 getDescription

(1) 機能

setDescription メソッドで指定した，接続する DB に必要な接続付加情報を取得します。

(2) 形式

```
public String getDescription ()
```

(3) 引数

なし。

(4) 戻り値

String：

接続付加情報です。設定されていない場合，null を返却します。

(5) 発生する例外

なし。

16.11.3 setDBHostName

(1) 機能

接続する HiRDB のホスト名（クライアント環境定義 PDHOST に設定するホスト名）を設定します。

XA 接続以外の接続で，かつ接続付加情報に HiRDB クライアントの環境変数グループ名を指定している場合は，このメソッドで値を指定しても無効になります。

(2) 形式

```
public void setDBHostName ( String db_host_name )
```

(3) 引数

String db_host_name：

HiRDB のホスト名を設定します。

(4) 戻り値

なし。

(5) 発生する例外

なし。

16.11.4 getDBHostName

(1) 機能

setDBHostName メソッドで指定した，接続する HiRDB のホスト名を取得します。

(2) 形式

```
public String getDBHostName ()
```

(3) 引数

なし。

(4) 戻り値

String：

HiRDB のホスト名です。設定されていない場合，null を返却します。

(5) 発生する例外

なし。

16.11.5　setEncodeLang

(1)　機能

JDBC ドライバ内で文字コード変換に使用する文字セットを指定します。

(2)　形式

```
public void setEncodeLang ( String encode_lang )
```

(3)　引数

String encode_lang：

Java がサポートしている文字セット（MS932 など）を指定します。

このメソッドで"OFF"を指定した場合，又は指定をしなかった場合（Properties info の ENCODELANG での設定，及び URL の ENCODELANG での設定も含む），次に示す動作となります。

OFF：

JDBC ドライバが接続先 HiRDB の文字コード種別に対応する文字セットを決定します。接続先 HiRDB の文字コード種別と JDBC ドライバで使用する文字エンコードの対応を次に示します。

HiRDB の文字コード種別※	使用する文字エンコード
lang-c	8859_1
sjis	Java 仮想マシンの標準エンコード
ujis	EUCJIS
utf-8	UTF-8
chinese	GB2312
chinese-gb18030	GB18030

注※

UNIX 版の場合は pdsetup コマンドの-c オプション，Windows 版の場合は pdntenv コマンドの-c オプションの指定値です。pdntenv コマンドを実行しない場合の文字コード種別については，マニュアル「HiRDB システム導入・設計ガイド」を参照してください。

指定なし：

＜ UNIX 版の場合＞

JDBC ドライバが HiRDB の文字コード種別に対応した文字セットを決定します。

＜ Windows 版の場合＞

JDBC ドライバが次の規則で決定します。

Java 仮想マシンの標準エンコード	HiRDB の文字コード種別	
	SJIS	SJIS 以外
MS932	MS932	HiRDB の文字コード種別に対応した文字セット
MS932 以外	SJIS	

(4) 戻り値

なし。

(5) 機能詳細

Java プログラム内では，文字コードは Unicode で扱うため，HiRDB との文字データ処理時に，JDBC ドライバが HiRDB の文字データと Unicode との相互文字コード変換をします。この文字コード変換処理で，JDBC ドライバは Java 仮想マシンが提供するエンコーダ及びデコーダを利用します。このメソッドでは，Java 仮想マシンが提供するエンコーダ及びデコーダに対して，JDBC ドライバが指定する文字セット名称を指定します。

(6) 発生する例外

なし。

16.11.6 getEncodeLang

(1) 機能

setEncodeLang メソッドで指定した文字セットを取得します。

(2) 形式

```
public String getEncodeLang ()
```

(3) 引数

なし。

(4) 戻り値

String：

文字セットです。指定していない場合は，null を返却します。

(5)　発生する例外

なし。

16.11.7　setUser

(1)　機能

認可識別子を設定します。

(2)　形式

```
public void setUser ( String user )
```

(3)　引数

String user：

認可識別子を指定します。

(4)　戻り値

なし。

(5)　機能詳細

認可識別子を設定します。

認可識別子は，DataSource.getConnection メソッド，ConnectionPoolDataSource.getPooledConnection メソッド，又は XADataSource.getXAConnection メソッド（これらをまとめて DB 接続メソッドと呼びます）の引数でも指定できます。

このメソッドによって認可識別子が設定され，かつ認可識別子，パスワードを引数に持つ DB 接続メソッドが呼び出された場合は，DB 接続メソッドで指定した認可識別子の指定値が優先されます。

認可識別子の指定に関する注意事項については，「getConnection メソッドの引数の内容」を参照してください。

(6)　発生する例外

なし。

16.11.8　getUser

(1)　機能

認可識別子を取得します。

(2)　形式

```
public String getUser ()
```

(3)　引数

なし。

(4)　戻り値

String：

認可識別子です。設定されていない場合，null を返却します。

(5)　機能詳細

setUser メソッドで指定した，認可識別子を返却します。

OS ログインユーザの簡易認証機能を使用している場合は，簡易認証キーワードを返却します。

setUser メソッドによって認可識別子が設定され，かつ認可識別子，パスワードを引数に持つ DB 接続メソッド（DataSource.getConnection メソッド，ConnectionPoolDataSource.getPooledConnection メソッド，又は XADataSource.getXAConnection メソッド）が呼び出された場合は，DB 接続メソッドで指定した認可識別子の指定値を返却します。

(6)　発生する例外

なし。

16.11.9　setPassword

(1)　機能

パスワードを設定します。

(2) 形式

```
public void setPassword ( String password )
```

(3) 引数

String password：

パスワードを指定します。

(4) 戻り値

なし。

(5) 機能詳細

パスワードを設定します。

パスワードは，DataSource.getConnection メソッド，ConnectionPoolDataSource.getPooledConnection メソッド，又は XADataSource.getXAConnection メソッド（これらをまとめて DB 接続メソッドと呼びます）の引数でも指定できます。

このメソッドによってパスワードが設定され，かつ認可識別子，パスワードを引数に持つ DB 接続メソッドが呼び出された場合は，DB 接続メソッドで指定したパスワードの指定値が優先されます。

(6) 発生する例外

なし。

16.11.10 getPassword

(1) 機能

パスワードを取得します。

(2) 形式

```
public String getPassword ()
```

(3) 引数

なし。

(4) 戻り値

String：

パスワードです。設定されていない場合，null を返却します。

(5) 機能詳細

setPassword メソッドで指定した，パスワードを返却します。

setPassword メソッドによってパスワードが設定され，かつ認可識別子，パスワードを引数に持つ DB 接続メソッド（DataSource.getConnection メソッド，ConnectionPoolDataSource.getPooledConnection メソッド，又は XADataSource.getXAConnection メソッド）が呼び出された場合は，DB 接続メソッドで指定したパスワードの指定値を返却します。

(6) 発生する例外

なし。

16.11.11 setXAOpenString

(1) 機能

XA オープン文字列を設定します。

(2) 形式

```
public void setXAOpenString ( String xa_string )
```

(3) 引数

String xa_string：

XA オープン文字列を指定します。

(4) 戻り値

なし。

(5) 機能詳細

XA オープン文字列を設定します。

このメソッドは，JdbhDbpsvXADataSource クラスでだけ提供されます。

XA オープン文字列は，「HiRDB の環境変数グループ識別子＋ HiRDB の環境変数グループ名」の形式で指定します。HiRDB の環境変数グループ識別子は，setDescription メソッドでの設定と同じにする必要があります。設定例を次に示します。

例 1：

HiRDB クライアント環境変数登録ツールで登録した，環境変数グループ名「HiRDB_ENV_GROUP」を設定する場合

```
ds.setDescription(”HDB1”);
ds.setXAOpenString(”HDB1+HiRDB_ENV_GROUP”);
```

例 2：

HiRDB の環境変数グループ名のパスが「C:¥Program△Files¥HITACHI¥HiRDB¥HiRDB.ini」の場合（△は半角空白文字）

```
ds.setDescription(”HDB1”);
ds.setXAOpenString(”HDB1+C:¥¥Program△Files¥¥HITACHI¥¥HiRDB¥¥HiRDB.ini”);
```

(6) 発生する例外

なし。

16.11.12 getXAOpenString

(1) 機能

setXAOpenString メソッドで指定した，XA オープン文字列を取得します。このメソッドは，JdbhDbpsvXADataSource クラスでだけ提供されます。

(2) 形式

```
public String getXAOpenString ()
```

(3) 引数

なし。

(4) 戻り値

String：

XA オープン文字列です。設定されていない場合，null を返却します。

(5) 発生する例外

なし。

16.11.13 setXACloseString

(1) 機能

XA クローズ文字列を設定します。このメソッドは，JdbhDbpsvXADataSource クラスでだけ提供されます。

(2) 形式

```
public void setXACloseString ( String xa_string )
```

(3) 引数

String xa_string：

XA クローズ文字列を設定します。

(4) 戻り値

なし。

(5) 発生する例外

なし。

16.11.14 getXACloseString

(1) 機能

setXACloseString メソッドで指定した，XA クローズ文字列を取得します。このメソッドは，JdbhDbpsvXADataSource クラスでだけ提供されます。

(2) 形式

```
public String getXACloseString ()
```

(3) 引数

なし。

(4) 戻り値

String：

XA クローズ文字列です。設定されていない場合，null を返却します。

(5) 発生する例外

なし。

16.11.15 setRMID

(1) 機能

リソースマネジャに対する識別子を設定します。

(2) 形式

```
public void setRMID ( int rmid )
```

(3) 引数

int rmid：

リソースマネジャに対する識別子を指定します。

(4) 戻り値

なし。

(5) 機能詳細

リソースマネジャに対する識別子を，１以上の正の数値で設定します。

複数のリソースマネジャを同時に使用する場合，リソースマネジャごとに一意な識別子を設定する必要があります。

このメソッドが呼び出されない場合，デフォルトの識別子として 1 を使用します。

このメソッドは，JdbhDbpsvXADataSource クラスでだけ提供されます。

(6) 発生する例外

引数の内容が 1 より小さい場合，SQLException を投入します。

16.11.16 getRMID

(1) 機能

setRMID メソッドで指定した，リソースマネジャに対する識別子を取得します。このメソッドは，JdbhDbpsvXADataSource クラスでだけ提供されます。

(2) 形式

```
public int getRMID ()
```

(3) 引数

なし。

(4) 戻り値

int：

リソースマネジャに対する識別子です。設定されていない場合，1 を返却します。

(5) 発生する例外

なし。

16.11.17 setXAThreadMode

(1) 機能

XA 使用時のスレッドモードを設定します。

(2) 形式

```
public void setXAThreadMode ( boolean mode )
```

(3) 引数

boolean mode：

XA 使用時のスレッドモードを指定します。

true：マルチスレッドモード

false：シングルスレッドモード

(4) 戻り値

なし。

(5) 機能詳細

XA 使用時のスレッドモードを設定します。このメソッドが呼び出されない場合，デフォルト値は false（シングルスレッドモード）となります。

このメソッドは，JdbhDbpsvXADataSource クラスでだけ提供されます。

RM（リソースマネジャ）が提供する XA ライブラリがマルチスレッド対応で，かつアプリケーションがマルチスレッドで動作する場合，このメソッドを true（マルチスレッドモード）で呼び出す必要があります。

(6) 発生する例外

なし。

16.11.18 getXAThreadMode

(1) 機能

setXAThreadMode メソッドで指定した，XA 使用時のスレッドモードを取得します。このメソッドは，JdbhDbpsvXADataSource クラスでだけ提供されます。

(2) 形式

```
public boolean getXAThreadMode ()
```

(3) 引数

なし。

(4) 戻り値

boolean：

XA 使用時のスレッドモードです。

true：マルチスレッドモード

false：シングルスレッドモード

(5) 発生する例外

なし。

16.11.19 setCommit_Behavior

(1) 機能

HiRDB がコミットをした場合に，次に示すクラスをコミット実行後も有効とするかどうかを設定します。

- ResultSet クラス
- Statement クラス，PreparedStatement クラス，及び CallableStatement クラス

(2) 形式

```
public void setCommit_Behavior (String type)
```

(3) 引数

String type：

Statement クラス，PreparedStatement クラス，及び CallableStatement クラスのオブジェクト，並びに ResultSet クラスのオブジェクトが，トランザクションの終了をわたって有効かどうかを設定します。

指定値	ResultSet クラス	Statement クラス，PreparedStatement クラス，及び CallableStatement クラス
"DELETE"（デフォルト値）	無効※1	無効※2
"CLOSE"	無効※1	有効
"PRESERVE"	有効※3	有効※3

注※1

コミット実行後に ResultSet クラスのオブジェクトを無効とする条件は，ResultSet クラスの次に示すメソッドを実行したことによって ResultSet クラスの getXXX メソッドが実行できる場合です。

- next メソッド
- first メソッド
- last メソッド
- absolute メソッド
- relative メソッド

無効とした ResultSet クラスのオブジェクトでメソッドを実行した場合の動作については保証しません。

注※2

コミット実行後に無効となるものを次に示します。

- Connection.prepareStatement メソッドでプリコンパイルした SQL 文
- Connection.prepareCall メソッドでプリコンパイルした SQL 文
- Statement クラス，PreparedStatement クラス，及び CallableStatement クラスの executeQuery メソッドで取得した ResultSet クラスのオブジェクト

注※3

接続先の HiRDB のバージョンが 07-01 以前の場合，LOCK TABLE による表の排他が必要です。

(4) 戻り値

なし。

(5) 機能詳細

Statement クラス，PreparedStatement クラス，及び CallableStatement クラスのオブジェクト，並びに ResultSet クラスのオブジェクトが，トランザクションの終了をわたって有効かどうかを設定します。このメソッドが呼び出されない場合，デフォルトは DELETE となります。

このメソッドの実行は，DriverManager を使用して DB 接続するときに行う，プロパティ COMMIT_BEHAVIOR の設定と同等になります。

(6) 発生する例外

XADataSource を使用した接続の場合，指定値に関係なく"DELETE"となります。ただし，getCommit_Behavior の戻り値は引数 type で指定したものとなります。

(7)　注意事項

注意事項については，「Properties info の設定内容」の「COMMIT_BEHAVIOR についての注意事項」を参照してください。

16.11.20　getCommit_Behavior

(1)　機能

Statement クラス，PreparedStatement クラス，及び CallableStatement クラスのオブジェクト，並びに ResultSet クラスのオブジェクトが，トランザクションの終了をわたって有効かどうかを取得します。

(2)　形式

```
public String getCommit_Behavior ()
```

(3)　引数

なし。

(4)　戻り値

String：

Statement クラス，PreparedStatement クラス，及び CallableStatement クラスのオブジェクト，並びに ResultSet クラスのオブジェクトが，トランザクションの終了をわたって有効かどうかの種別が設定されていない場合，DELETE を返却します。

(5)　機能詳細

setCommit_Behavior メソッドで指定した情報が返却されます。

(6)　発生する例外

なし。

16.11.21　setBlockUpdate

(1)　機能

?パラメタを使用したデータベースの更新（INSERT，UPDATE，及び DELETE）で，複数のパラメタセットを一度に処理するかどうかを設定します。

なお，実際にパラメタセットが一度に処理されるかどうかは，配列を使用した機能の使用方法で決まります。配列を使用した機能の使用方法については，「配列を使用した機能」を参照してください。

(2) 形式

```
public void setBlockUpdate(boolean Mode)
```

(3) 引数

boolean Mode：

複数のパラメタセットを一度に処理するかどうかの設定情報です。省略した場合，false が仮定されます。

true：

一度に処理します。

false：

パラメタセットを一つずつ分割して処理します。

(4) 戻り値

なし。

(5) 発生する例外

なし。

(6) 注意事項

複数行の?パラメタの一括処理をする方法については，「Properties info の設定内容」，及び「バッチ更新」を参照してください。

この機能は，システムプロパティの HiRDB_for_Java_BLOCK_UPDATE でも指定できます。ただし，setBlockUpdate メソッドを設定した場合は，システムプロパティの HiRDB_for_Java_BLOCK_UPDATE の設定は無効となります。

16.11.22 getBlockUpdate

(1) 機能

?パラメタを使用したデータベースの更新（INSERT，UPDATE，及び DELETE）で，複数のパラメタセットを一度に処理するかどうかを取得します。

(2) 形式

```
public boolean getBlockUpdate()
```

(3) 引数

なし。

(4) 戻り値

boolean：

複数のパラメタセットを一度に処理するかどうかの情報です。省略した場合，false が仮定されます。

true：

一度に処理します。

false：

パラメタセットを一つずつ分割して処理します。

(5) 発生する例外

なし。

(6) 注意事項

なし。

16.11.23 setLONGVARBINARY_Access

(1) 機能

LONGVARBINARY（列属性が BLOB 又は BINARY）のデータベースアクセス方法を指定します。

(2) 形式

```
public void setLONGVARBINARY_Access(String Mode)
```

(3) 引数

String Mode：

LONGVARBINARY（列属性が BLOB 又は BINARY）のデータベースアクセス方法の設定情報です。省略した場合，"REAL"が仮定されます。

"REAL"：

実データでアクセスします。

"LOCATOR"：

HiRDB の位置付け子機能を使用してアクセスします。

上記以外：

"REAL"が指定されたとみなされます。

(4) 戻り値

なし。

(5) 発生する例外

なし。

16.11.24 getLONGVARBINARY_Access

(1) 機能

LONGVARBINARY（列属性が BLOB 又は BINARY）のデータベースアクセス方法を取得します。

(2) 形式

```
public String getLONGVARBINARY_Access()
```

(3) 引数

なし。

(4) 戻り値

String：

LONGVARBINARY（列属性が BLOB 又は BINARY）のデータベースアクセス方法の設定情報です。何も設定されていない場合，"REAL"が返されます。

"REAL"：

実データでアクセスします。

"LOCATOR"：

HiRDB の位置付け子機能を使用してアクセスします。

(5) 機能詳細

setLONGVARBINARY_Access メソッドで指定された情報を返却します。

(6) 発生する例外

なし。

16.11.25 setSQLInNum

(1) 機能

実行する SQL の入力，又は入出力?パラメタの最大数を指定します。

(2) 形式

```
public  void  setSQLInNum(int inNum)
```

(3) 引数

int inNum：

実行する SQL の入力，又は入出力?パラメタの最大数を指定します。指定値は，1～30,000（デフォルトは 64）です。

(4) 戻り値

なし。

(5) 機能詳細

SQL の前処理時に取得する，入力，又は入出力?パラメタ情報の数を指定します。

実際の?パラメタの数が，このプロパティの指定値よりも多い場合，SQL の前処理の後に入力，又は入出力?パラメタ情報を取得します。

このメソッドで指定した値は，DB 接続時にプロパティの HiRDB_for_Java_SQL_IN_NUM の値となります。

(6) 発生する例外

引数が指定値の範囲外の場合，SQLException を投入します。

(7) 注意事項

- この機能は，システムプロパティの HiRDB_for_Java_SQL_IN_NUM でも指定できます。ただし，setSQLInNum メソッドを設定した場合は，システムプロパティの HiRDB_for_Java_SQL_IN_NUM の設定は無効となります。
- 入力，又は入出力?パラメタのある SQL 文を実行しない場合は，1 を指定することをお勧めします。

16.11.26 getSQLInNum

(1) 機能

setSQLInNum で設定した，実行する SQL の入力，又は入出力?パラメタの最大数を取得します。

(2) 形式

```
public  int  getSQLInNum()
```

(3) 引数

なし。

(4) 戻り値

int：

setSQLInNum で設定した，実行する SQL の入力，又は入出力?パラメタの最大数です。設定されていない場合は，デフォルト値（64）が返されます。

(5) 発生する例外

なし。

16.11.27 setSQLOutNum

(1) 機能

実行する SQL の検索項目，又は出力若しくは入出力?パラメタの最大数を指定します。

(2) 形式

```
public  void  setSQLOutNum(int outNum)
```

(3) 引数

int outNum：

実行する SQL の検索項目，又は出力若しくは入出力?パラメタの最大数を指定します。指定値は，1〜30,000（デフォルトは 64）です。

(4) 戻り値

なし。

(5) 機能詳細

実行する SQL の検索項目，又は出力若しくは入出力?パラメタの最大数を指定します。

この指定は，SQL の前処理時に取得する出力項目の数となります。

実際の出力項目数が，このプロパティの指定値よりも多い場合，SQL の前処理の後に出力項目情報を取得します。

このメソッドで指定した値は，DB 接続時にプロパティの HiRDB_for_Java_SQL_OUT_NUM の値となります。

(6) 発生する例外

引数が指定値の範囲外の場合，SQLException を投入します。

(7) 注意事項

- この機能は，システムプロパティの HiRDB_for_Java_SQL_OUT_NUM でも指定できます。ただし，setSQLOutNum メソッドを設定した場合は，システムプロパティの HiRDB_for_Java_SQL_OUT_NUM の設定は無効となります。
- 検索項目，又は出力若しくは入出力?パラメタがない場合は，1 を指定することをお勧めします。

16.11.28 getSQLOutNum

(1) 機能

setSQLOutNum で設定した，実行する SQL の検索項目，又は出力若しくは入出力?パラメタの最大数を取得します。

(2) 形式

```
public  int  getSQLOutNum()
```

(3) 引数

なし。

(4) 戻り値

int：

setSQLOutNum で設定した，実行する SQL の検索項目，又は出力若しくは入出力?パラメタの最大数です。設定されていない場合，デフォルト値（64）が返されます。

(5) 発生する例外

なし。

16.11.29 setSQLWarningLevel

(1) 機能

SQL 実行時に発生した警告保持レベルを指定します。このメソッドで指定した値は，DB 接続時にプロパティの HiRDB_for_Java_SQLWARNING_LEVEL の値となります。

(2) 形式

```
public  void  setSQLWarningLevel (String warningLevel)
```

(3) 引数

String warningLevel：

SQL 実行時に発生した警告情報の保持レベルを指定します。次の値を指定できます。なお，指定値と，保持する警告の関係については，「SQLWarning クラス」を参照してください。

- IGNORE
- SQLWARN（デフォルト）
- ALLWARN

なお，このメソッドでは，引数に指定した内容の大文字，小文字は区別されません。

(4) 戻り値

なし。

(5) 発生する例外

引数が指定値以外の場合，SQLException を投入します。

16.11.30 getSQLWarningLevel

(1) 機能

setSQLWarningLevel で指定した警告保持レベルを取得します。

(2) 形式

```
public String getSQLWarningLevel ()
```

(3) 引数

なし。

(4) 戻り値

String：

setSQLWarningLevel で指定した警告保持レベル（IGNORE，SQLWARN，又は ALLWARN）が返されます。指定していない場合は，デフォルト値（SQLWARN）が返されます。返却値と，保持する警告の関係については，「SQLWarning クラス」を参照してください。

(5) 発生する例外

なし。

16.11.31 setClear_Env

(1) 機能

DB 接続時に OS の環境変数として設定した HiRDB のクライアント環境定義を無効にするかどうかを指定します。このメソッドで指定した値は，DB 接続時にプロパティの HiRDB_for_Java_CLEAR_ENV の設定と同等になります。

(2) 形式

```
public  void  setClear_Env(boolean Mode)
```

(3) 引数

boolean Mode：

HiRDB のクライアント環境定義を無効にするかどうかを指定します。

true：無効にします。

false：無効にしません。

(4) 戻り値

なし。

(5) 発生する例外

なし。

(6) 注意事項

注意事項については，表「Properties info の設定内容」の HiRDB_for_Java_CLEAR_ENV を参照してください。

16.11.32 getClear_Env

(1) 機能

setClear_Env で指定した環境変数無効指定を取得します。

(2) 形式

```
public  boolean  getClear_Env()
```

(3) 引数

なし。

(4) 戻り値

String：

setClear_Env で指定した環境変数無効指定の内容が返されます。指定していない場合は，デフォルト値（false）が返されます。

true：

DB 接続時に OS の環境変数として設定した HiRDB のクライアント環境定義を無効にします。

false：

DB 接続時に OS の環境変数として設定した HiRDB のクライアント環境定義を無効にしません。

(5) 発生する例外

なし。

16.12　データ型，文字コード

16.12.1　データ型

JDBC の SQL データ型と，HiRDB のクライアントライブラリを経由して接続する SQL データ型は，完全には一致しません。JDBC ドライバでは，JDBC の SQL データ型と HiRDB の SQL データ型とのマッピングをします。マッピングできない SQL データ型を使用してアクセスした場合，SQLException を投入します。

SQL データ型のマッピングは，ResultSet，PreparedStatement，及び CallableStatement の各クラスの，getXXX メソッド及び setXXX メソッドでします。なお，SQL データ型と getXXX メソッド，setXXX メソッドのマッピング規則については，JDBC1.0 規格のドキュメントを参照してください。

HiRDB と JDBC の SQL データ型の対応を次の表に示します。

表 16–19　HiRDB と JDBC の SQL データ型の対応

HiRDB の SQL データ型	JDBC の SQL データ型
INTEGER	INTEGER
SMALLINT	SMALLINT
DECIMAL	DECIMAL
FLOAT，DOUBLE PRECISION	FLOAT
SMALLFLT，REAL	REAL
CHAR	CHAR
VARCHAR	VARCHAR
NCHAR	CHAR
NVARCHAR	VARCHAR
MCHAR	CHAR
MVARCHAR	VARCHAR
DATE	DATE
TIME	TIME
BLOB	LONGVARBINARY
TIMESTAMP	TIMESTAMP
BINARY※	LONGVARBINARY

注※

データの扱いは，BLOB と同じです。

16.12.2　文字コード変換機能

Java プログラム内では，文字コードは Unicode で扱うため，JDBC ドライバが HiRDB の文字データと Unicode との相互文字コード変換をします。この文字コード変換処理で，JDBC ドライバは Java 仮想マシンが提供するエンコーダ及びデコーダを利用します。このとき，Properties info の ENCODELANG では，Java 仮想マシンが提供するエンコーダ及びデコーダに対して，JDBC ドライバが指定する文字セット名称を指定します。

HiRDB の文字コードと Java の文字セットの対応（UNIX 版の場合）を次の表に示します。

表 16–20　HiRDB の文字コードと Java の文字セットの対応（UNIX 版の場合）

HiRDB の文字コード	文字セット	備考
sjis （シフト JIS 漢字）	"SJIS"	全角文字に外字を含みます。
ujis （EUC 日本語漢字）	"EUC_JP" (Japanese EUC)	全角文字に外字を含みません。※
chinese （EUC 中国語漢字）	"EUC_CN" (Simplified Chinese)	全角文字に外字を含みません。※
lang-c （8 ビットコード）	"ISO-8859-1" (ISO Latin-1)	US ASCII 及び 8 ビットコードの場合に使用できます。
UTF-8	UTF-8	なし。
chinese-gb18030 （中国語漢字コード（GB18030））	GB18030	なし。

注

次に示す方法で Properties info の ENCODELANG を設定している場合，これを優先してエンコーディングします。

- DriverManager.getConnection メソッドの引数として渡す Properties info で設定
- JdbhDataSource.setEncodeLang メソッド，JdbhConnectionPoolDataSource.setEncodeLang メソッド，又は JdbhXADataSource メソッドで設定

上記の方法で ENCODELANG を設定していない場合の動作，及び"OFF"を設定した場合の動作については，「setEncodeLang」を参照してください。

注※

EUC コードセット 3（$(8F)_{16}(XXXX)_{16}$ の 3 バイトで表現される文字コード）に割り当てられた外字コードは使用できません。

HiRDB の文字コードと Java の文字セットの対応（Windows 版の場合）を次の表に示します。

表 16–21　HiRDB の文字コードと Java の文字セットの対応（Windows 版の場合）

HiRDB の文字コード	文字セット	備考
sjis （シフト JIS 漢字）	Java 仮想マシンの標準エンコードが"MS932"の場合は"MS932" "MS932"以外の場合は"SJIS"	全角文字に外字を含みます。
UTF-8	UTF-8	なし。

注

次に示す方法で Properties info の ENCODELANG を設定している場合，これを優先してエンコーディングします。

- DriverManager.getConnection メソッドの引数として渡す Properties info で設定
- JdbhDataSource.setEncodeLang メソッド，JdbhConnectionPoolDataSource.setEncodeLang メソッド，又は JdbhXADataSource メソッドで設定

上記の方法で ENCODELANG を設定していない場合の動作，及び"OFF"を設定した場合の動作については，「setEncodeLang」を参照してください。

16.13　制限事項があるクラスとメソッド

ここでは，JDBC1.0 規格で定義されているクラスについて説明します。

なお，JDBC2.0 基本規格で定義されている次のクラスは使用できません。

- Clob クラス
- Struct クラス
- Ref クラス
- SQLData クラス
- SQLInput クラス
- SQLOutput クラス

16.13.1　Driver クラス

制限事項はありません。

16.13.2　Connection クラス

Connection クラスの JDBC1.0 規格で定義されているメソッドの制限事項を次の表に示します。

表 16–22　Connection クラスの JDBC1.0 規格で定義されているメソッドの制限事項

JDBC1.0 規格で定義されているメソッド	制限事項
setReadOnly	使用できません。
isReadOnly	無条件に false を返却します。
setCatalog	使用できません。
getCatalog	無条件に null を返却します。
setTransactionIsolation	使用できません。
getTransactionIsolation	無条件に TRANSACTION_REPEATABLE_READ を返却します。

Connection クラスの JDBC2.0 基本規格で追加されたメソッドの制限事項を次の表に示します。

表 16–23　Connection クラスの JDBC2.0 基本規格で追加されたメソッドの制限事項

JDBC2.0 基本規格で追加されたメソッド	制限事項
createStatement	更新結果を反映する結果セットが使用できません。
prepareStatement	

JDBC2.0 基本規格で追加されたメソッド	制限事項
prepareCall	そのため，結果セットタイプに TYPE_SCROLL_SENSITIVE を指定した場合は，TYPE_SCROLL_INSENSITIVE に切り替え，SQLWarning を設定します。 ?パラメタに NULL 述語を指定した SQL（? IS 〔NOT〕 NULL）を指定した場合，SQLException を投入します。
getTypeMap	ユーザ定義型が使用できないため，無条件に SQLException を投入します。
setTypeMap	

16.13.3 Statement クラス

Statement クラスの JDBC1.0 規格で定義されているメソッドの制限事項を次の表に示します。

表 16–24 Statement クラスの JDBC1.0 規格で定義されているメソッドの制限事項

JDBC1.0 規格で定義されているメソッド	制限事項
setCursorName	使用できません（位置決めされた更新及び削除が使用できないため）。
getMaxFieldSize	setMaxFieldSize で指定した値を返却します。
getMoreResults	無条件に false を返却します。
setMaxRows	使用できません。
setQueryTimeout	

Statement クラスの JDBC2.0 基本規格で追加されたメソッドの制限事項を次の表に示します。

表 16–25 Statement クラスの JDBC2.0 基本規格で追加されたメソッドの制限事項

JDBC2.0 基本規格で追加されたメソッド	制限事項
setFetchDirection	FETCH_FORWARD 以外を指定した場合，SQLException を投入します。
getFetchSize	setFetchSize メソッドで指定された値を返却します。

16.13.4 PreparedStatement クラス

PreparedStatement クラスの JDBC2.0 基本規格で追加されたメソッドの制限事項を次の表に示します。

表 16–26 PreparedStatement クラスの JDBC2.0 基本規格で追加されたメソッドの制限事項

JDBC2.0 基本規格で追加されたメソッド	制限事項
setBlob	JDBC ドライバでは，JDBC SQL タイプを LONGVARBINARY として扱います。

JDBC2.0 基本規格で追加されたメソッド	制限事項
setClob	SQL CLOB 型が使用できないため，無条件に SQLException を投入します。
setRef	SQL 構造化型が使用できないため，無条件に SQLException を投入します。
setNull	SQL 構造化型，SQL 配列型などが使用できないため，SQL ユーザ定義型の完全指定の名前を指定した場合は，無条件に SQLException を投入します。
setObject	scale の指定は無視し，指定した実際の値から scale 値を求めます。

16.13.5 CallableStatement クラス

CallableStatement クラスの JDBC2.0 基本規格で追加されたメソッドの制限事項を次の表に示します。

表 16–27 CallableStatement クラスの JDBC2.0 基本規格で追加されたメソッドの制限事項

JDBC2.0 基本規格で追加されたメソッド	制限事項
getObject	Map 指定が使用できないため，Map 指定時は SQLException を投入します。
getBlob	JDBC ドライバでは，JDBC SQL タイプを LONGVARBINARY として扱います。
getClob	SQL CLOB 型が使用できないため，無条件に SQLException を投入します。
getRef	SQL 構造化型が使用できないため，無条件に SQLException を投入します。

16.13.6 ResultSet クラス

ResultSet クラスの JDBC2.0 基本規格で追加されたメソッドの制限事項を次の表に示します。

表 16–28 ResultSet クラスの JDBC2.0 基本規格で追加されたメソッドの制限事項

JDBC2.0 基本規格で追加されたメソッド	制限事項
setFetchDirection	FETCH_FORWARD 以外を指定した場合，SQLException を投入します。
rowUpdated	更新可能型結果セットが使用できないため，無条件に SQLException を投入します。
rowInserted	
rowDeleted	
updateNull	

JDBC2.0 基本規格で追加されたメソッド	制限事項
updateBoolean	
updateByte	
updateShort	
updateInt	
updateLong	
updateFloat	
updateDouble	
updateBigDecimal	
updateString	
updateBytes	
updateDate	
updateTime	
updateTimestamp	
updateAsciiStream	
updateBinaryStream	
updateCharacterStream	
updateObject	
insertRow	
updateRow	
deleteRow	
refreshRow	
cancelRowUpdates	
moveToInsertRow	
moveToCurrentRow	
getObject	Map 指定が使用できないため，Map 指定時は SQLException を投入します。
getBlob	JDBC ドライバでは，JDBC SQL タイプを LONGVARBINARY として扱います。
getClob	SQL CLOB 型が使用できないため，無条件に SQLException を投入します。
getRef	SQL 構造化型が使用できないため，無条件に SQLException を投入します。

16.13.7　ResultSetMetaData クラス

ResultSetMetaData クラスの JDBC1.0 規格で定義されているメソッドの制限事項を次の表に示します。ただし，Array クラスの getResultSet メソッドで生成された，結果セットから取得した ResultSetMetaData クラスの各メソッドの戻り値については，表「getResultSet で返す結果セットの属性値」を参照してください。

表 16–29　ResultSetMetaData クラスの JDBC1.0 規格で定義されているメソッドの制限事項

JDBC1.0 規格で定義されているメソッド	制限事項
isAutoIncrement	無条件に false を返却します。
isCaseSensitive	無条件に true を返却します。
isCurrency	無条件に false を返却します。
getColumnLabel	列のラベル（列ヘッダ）は使用できないため，列名を返却します。
getSchemaName	無条件に null を返却します。
getTableName	
getCatalogName	
isReadOnly	無条件に false を返却します。
isWritable	
isDefinitelyWritable	

16.13.8　DatabaseMetaData クラス

DatabaseMetaData クラスの JDBC1.0 規格で定義されているメソッドの制限事項又は返却内容を次の表に示します。

なお，各メソッドが返却する値は，常に JDBC ドライバのバージョンと同一のバージョンの HiRDB サーバに関する情報です。

表 16–30　DatabaseMetaData クラスの JDBC1.0 規格で定義されているメソッドの制限事項

JDBC1.0 規格で定義されているメソッド	制限事項又は返却内容
allProceduresAreCallable	false を返却します。
allTablesAreSelectable	false を返却します。
getURL	接続しているデータベースの JDBC URL を返却します。
getUserName	データベースに接続する際に使用した認可識別子を返却します。
isReadOnly	アクセスモードを変更できないため，無条件に false を返却します。
nullsAreSortedHigh	true を返却します。

JDBC1.0 規格で定義されているメソッド	制限事項又は返却内容
nullsAreSortedLow	false を返却します。
nullsAreSortedAtStart	false を返却します。
nullsAreSortedAtEnd	無条件に false を返却します。
getDatabaseProductName	'HiRDB'を返却します。
getDatabaseProductVersion	null を返却します。
getDriverName	'HiRDB_for_JDBC'を返却します。
getDriverVersion	08.05.0000 を返却します。
getDriverMajorVersion	8 です。
getDriverMinorVersion	5 です。
usesLocalFiles	無条件に false を返却します。
usesLocalFilePerTable	無条件に false を返却します。
supportsMixedCaseIdentifiers	無条件に false を返却します。
storesUpperCaseIdentifiers	true を返却します。
storesLowerCaseIdentifiers	無条件に false を返却します。
storesMixedCaseIdentifiers	false を返却します。
supportsMixedCaseQuotedIdentifiers	true を返却します。
storesUpperCaseQuotedIdentifiers	false を返却します。
storesLowerCaseQuotedIdentifiers	無条件に false を返却します。
storesMixedCaseQuotedIdentifiers	true を返却します。
getIdentifierQuoteString	無条件に引用符を返却します。
getSQLKeywords	HiRDB 固有の SQL キーワードを返却します。
getNumericFunctions	数学関数のリストを返却します。
getStringFunctions	文字列関数のリストを返却します。
getSystemFunctions	システム関数のリストを返却します。
getTimeDateFunctions	時間関数と日付関数のリストを返却します。
getSearchStringEscape	'¥'を返却します。
getExtraNameCharacters	SQL 識別名に使用できる特殊文字を返却します。
supportsAlterTableWithAddColumn	true を返却します。
supportsAlterTableWithDropColumn	
supportsColumnAliasing	
nullPlusNonNullIsNull	

JDBC1.0 規格で定義されているメソッド	制限事項又は返却内容
supportsConvert（引数なし）	true を返却します。
supportsConvert（引数あり）	引数で指定したデータ型の組み合わせによって，true 又は false のどちらかを返却します。
supportsTableCorrelationNames	true を返却します。
supportsDifferentTableCorrelationNames	
supportsExpressionsInOrderBy	false を返却します。
supportsOrderByUnrelated	true を返却します。
supportsGroupBy	
supportsGroupByUnrelated	
supportsGroupByBeyondSelect	
supportsLikeEscapeClause	
supportsMultipleResultSets	無条件に true を返却します。
supportsMultipleTransactions	
supportsNonNullableColumns	true を返却します。
supportsMinimumSQLGrammar	無条件に true を返却します。
supportsCoreSQLGrammar	
supportsExtendedSQLGrammar	false を返却します。
supportsANSI92EntryLevelSQL	無条件に true を返却します。
supportsANSI92IntermediateSQL	無条件に false を返却します。
supportsANSI92FullSQL	
supportsIntegrityEnhancementFacility	false を返却します。
supportsOuterJoins	true を返却します。
supportsFullOuterJoins	false を返却します。
supportsLimitedOuterJoins	true を返却します。
getSchemaTerm	'schema'を返却します。
getProcedureTerm	'procedure'を返却します。
getCatalogTerm	null を返却します。
isCatalogAtStart	false を返却します。
getCatalogSeparator	null を返却します。
supportsSchemasInDataManipulation	無条件に true を返却します。
supportsSchemasInProcedureCalls	true を返却します。

JDBC1.0 規格で定義されているメソッド	制限事項又は返却内容
supportsSchemasInTableDefinitions	
supportsSchemasInIndexDefinitions	
supportsSchemasInPrivilegeDefinitions	
supportsCatalogsInDataManipulation	false を返却します。
supportsCatalogsInProcedureCalls	
supportsCatalogsInTableDefinitions	
supportsCatalogsInIndexDefinitions	無条件に false を返却します。
supportsCatalogsInPrivilegeDefinitions	
supportsPositionedDelete	
supportsPositionedUpdate	
supportsSelectForUpdate	
supportsStoredProcedures	true を返却します。
supportsSubqueriesInComparisons	
supportsSubqueriesInExists	
supportsSubqueriesInIns	
supportsSubqueriesInQuantifieds	
supportsCorrelatedSubqueries	
supportsUnion	
supportsUnionAll	
supportsOpenCursorsAcrossCommit	次のどれかの値が PRESERVE の場合，true を返却します。 • URL での COMMIT_BEHAVIOR の設定 • Properties info での COMMIT_BEHAVIOR の設定 • setCommit_Behavior メソッド実行時の引数
supportsOpenCursorsAcrossRollback	無条件に false を返却します。
supportsOpenStatementsAcrossCommit	次のどれかの値が PRESERVE 又は CLOSE の場合，true を返却します。 • URL での COMMIT_BEHAVIOR の設定 • Properties info での COMMIT_BEHAVIOR の設定 • setCommit_Behavior メソッド実行時の引数
supportsOpenStatementsAcrossRollback	無条件に false を返却します。
getMaxBinaryLiteralLength	64000 を返却します。
getMaxCharLiteralLength	32000 を返却します。
getMaxColumnNameLength	30 を返却します。

JDBC1.0 規格で定義されているメソッド	制限事項又は返却内容
getMaxColumnsInGroupBy	255 を返却します。
getMaxColumnsInIndex	16 を返却します。
getMaxColumnsInOrderBy	255 を返却します。
getMaxColumnsInSelect	30000 を返却します。
getMaxColumnsInTable	
getMaxConnections	0 を返却します。
getMaxCursorNameLength	30 を返却します。
getMaxIndexLength	4036 を返却します。
getMaxSchemaNameLength	8 を返却します。
getMaxProcedureNameLength	30 を返却します。
getMaxCatalogNameLength	0 を返却します。
getMaxRowSize	
doesMaxRowSizeIncludeBlobs	false を返却します。
getMaxStatementLength	2000000 を返却します。
getMaxStatements	64 を返却します。
getMaxTableNameLength	30 を返却します。
getMaxTablesInSelect	64 を返却します。
getMaxUserNameLength	8 を返却します。
getDefaultTransactionIsolation	無条件に TRANSACTION_REPEATABLE_READ を返却します。
supportsTransactions	無条件に true を返却します。
supportsTransactionIsolationLevel	与えられたトランザクションアイソレーションレベルが次のどれかの場合，true を返却します。 • TRANSACTION_READ_COMMITTED • TRANSACTION_READ_UNCOMMITTED • TRANSACTION_REPEATABLE_READ
supportsDataDefinitionAndDataManipulationTransactions	false を返却します。
supportsDataManipulationTransactionsOnly	false を返却します。
dataDefinitionCausesTransactionCommit	true を返却します。
dataDefinitionIgnoredInTransactions	無条件に false を返却します。
getProcedures	Java ストアドルーチン関する記述を返却します。
getProcedureColumns	Java ストアドルーチンのパラメタに関する記述を返却します。

JDBC1.0 規格で定義されているメソッド	制限事項又は返却内容
getTables	表に関する記述を返却します。組み込む表の型のリスト（types）に指定できるのは，getTableTypes で返却する表の型だけです。
getSchemas	スキーマ名称に関する記述を返却します。
getCatalogs	返却する結果は常に 0 件です。
getTableTypes	表の型に関する記述を返却します。返却する内容を次に示します。 "SYSTEM TABLE"：システム表 "BASE TABLE"：実表 "VIEW"：ビュー表 "READ ONLY VIEW"：読み込み専用ビュー表
getColumns	列に関する記述を返却します。
getColumnPrivileges	列の権限に関する記述を返却します。
getTablePrivileges	表の権限に関する記述を返却します。
getBestRowIdentifier	返却する結果は常に 0 件です。
getVersionColumns	
getPrimaryKeys	主キーの列に関する記述を返却します（返却する結果は常に 0 件です）。
getImportedKeys	返却する結果は常に 0 件です。
getExportedKeys	主キーの列を参照する外部キーの列に関する記述を返却します（返却する結果は常に 0 件です）。
getCrossReference	主キーの表の主キー列を参照する，外部キーの表の外部キー列に関する記述を返却します（返却する結果は常に 0 件です）。
getTypeInfo	データベースで使用できる標準 SQL タイプに関する記述を返却します。
getIndexInfo	インデクスに関する記述を返却します。

DatabaseMetaData クラスの JDBC2.0 基本規格で追加されたメソッドの制限事項又は返却内容を次の表に示します。

なお，各メソッドが返却する値は，常に JDBC ドライバのバージョンと同一のバージョンの HiRDB サーバに関する情報です。

表 16–31　DatabaseMetaData クラスの JDBC2.0 基本規格で追加されたメソッドの制限事項

JDBC2.0 基本規格で追加されたメソッド	制限事項又は返却内容
supportsResultSetType	結果セットタイプが TYPE_FORWARD_ONLY 又は TYPE_SCROLL_INSENSITIVE の場合，true を返却します。

JDBC2.0 基本規格で追加されたメソッド	制限事項又は返却内容
supportsResultSetConcurrency	結果セットタイプが TYPE_FORWARD_ONLY 又は TYPE_SCROLL_INSENSITIVE で，並行処理タイプが CONCUR_READ_ONLY の場合，true を返却します。
ownUpdatesAreVisible	無条件に false を返却します。
ownDeletesAreVisible	
ownInsertsAreVisible	
othersUpdatesAreVisible	
othersDeletesAreVisible	
othersInsertsAreVisible	
updatesAreDetected	
deletesAreDetected	
insertsAreDetected	
supportsBatchUpdates	無条件に true を返却します。
getUDTs	返却する結果は常に 0 件です。
getConnection	DatabaseMetaData インスタンスの生成元である Connection インスタンスを返却します。

16.13.9　Blob クラス

Blob クラスの JDBC2.0 基本規格で追加されたメソッドの制限事項を次の表に示します。

表 16–32　Blob クラスの JDBC2.0 基本規格で追加されたメソッドの制限事項

JDBC2.0 基本規格で追加されたメソッド	制限事項
setBinaryStream	JDBC1.4 のメソッドのため使用できません。無条件に SQLException を投入します。
setBytes	
truncate	

16.13.10　Array クラス

Array クラスの JDBC2.0 基本規格で追加されたメソッドの制限事項を次の表に示します。

表 16–33　Array クラスの JDBC2.0 基本規格で追加されたメソッドの制限事項

JDBC2.0 基本規格で追加されたメソッド	制限事項
getArray	MAP が使用できないため，引数に MAP を指定した場合，SQLException を投入します。
getResultSet	

17 Type4 JDBC ドライバ

この章では，Type4 JDBC ドライバのインストール，環境設定，JDBC の機能などについて説明します。

以降，この章では，Type4 JDBC ドライバを単に「JDBC ドライバ」と表記します。

17.1　インストールと環境設定

17.1.1　前提条件

(1)　接続先 HiRDB サーバ

08-00 以降の HiRDB に対して接続できます。プラットフォーム，シングルサーバ，パラレルサーバの条件は問いません。

(2)　HiRDB クライアント

JDBC ドライバが動作する前提条件を次の表に示します。

表 17–1　JDBC ドライバが動作する前提条件

対象	前提条件
OS	なし
Java Runtime Environment のバージョン	JDBC2.0：6 以上 JDBC4.0：6 以上 JDBC4.3：11 以上

17.1.2　インストール

JDBC ドライバは，HiRDB をインストールすると同時にインストールされます。インストール後は，次のファイル構成となります。

UNIX の場合

```
HiRDB/client/lib/pdjdbc2.jar
HiRDB/client/lib/pdjdbc4.jar
HiRDB/client/lib/pdjdbc43.jar
```

Windows（サーバ製品）の場合

```
HiRDB¥client¥utl¥pdjdbc2.jar
HiRDB¥client¥utl¥pdjdbc4.jar
HiRDB¥client¥utl¥pdjdbc43.jar
```

Windows（クライアント製品）の場合

```
HiRDB¥utl¥pdjdbc2.jar
HiRDB¥utl¥pdjdbc4.jar
HiRDB¥utl¥pdjdbc43.jar
```

注

下線で示す部分は，HiRDB のインストールディレクトリを指定します。

17.1.3　環境設定

JDBC ドライバを使用して UAP を実行する前に，次の表に示すファイルを OS の環境変数 CLASSPATH に指定してください。

表 17–2　環境変数 CLASSPATH に指定するファイル

JDBC ドライバのバージョン	環境変数 CLASSPATH に指定するファイル
JDBC2.0	pdjdbc2.jar
JDBC4.0	pdjdbc4.jar
JDBC4.3	pdjdbc43.jar

また，JDBC ドライバが提供する JDBC 規格外のメソッドなど，JDBC ドライバが提供するクラスを直接操作する場合には，UAP をコンパイルする前に環境変数 CLASSPATH を設定する必要があります。

なお，Cosminexus などのアプリケーションサーバから JDBC ドライバを利用する場合，アプリケーションサーバの環境設定に依存します。詳細については，各アプリケーションサーバのマニュアルを参照し，仕様を確認してください。

JDBC4.0 の JDBC ドライバを使用する場合の設定例を次に示します。

(1)　UNIX 環境の場合

(a)　ボーンシェル

```
CLASSPATH=${CLASSPATH}:/HiRDB/client/lib/pdjdbc4.jar
export CLASSPATH
```

注

下線で示す部分は，HiRDB のインストールディレクトリを指定します。

(b)　Cシェル

```
setenv CLASSPATH ${CLASSPATH}:/HiRDB/client/lib/pdjdbc4.jar
```

注

下線で示す部分は，HiRDB のインストールディレクトリを指定します。

(2) Windows 環境の場合（コマンドプロンプトからプログラムを実行）

```
set CLASSPATH=%CLASSPATH%;C:¥Program Files¥HITACHI¥HiRDB¥client¥utl¥pdjdbc4.jar
```

注

下線で示す部分は，HiRDB のインストールディレクトリを指定します。

17.1.4 メソッドの略記について

- 先頭に「get」が付くメソッドをまとめて表す場合，getXXX メソッドと表記します。
- 先頭に「set」が付くメソッドをまとめて表す場合，setXXX メソッドと表記します。
- 先頭に「execute」が付くメソッドをまとめて表す場合，executeXXX メソッドと表記します。
- 次に示すインタフェースをまとめて表す場合，DataSource 系インタフェースと表記します。
  - DataSource インタフェース
  - ConnectionPoolDataSource インタフェース
  - XADataSource インタフェース

17.2　DriverManager クラスによる DB 接続

DriverManager クラスから HiRDB に接続して，Connection クラスのインスタンスを生成するには，次の手順で接続します。

1. JDBC2.0 を使用する場合，Driver クラスを Java 仮想マシンに登録します。
   JDBC4.0 以降を使用する場合，自動で Driver クラスが登録されるため，Driver クラスを登録する必要はありません。
2. 接続情報を引数にして，DriverManager クラスの getConnection メソッドによって HiRDB に接続します。

17.2.1　Driver クラスの登録

Java 仮想マシンに JDBC ドライバを登録する方法を次に示します。

なお，Java 仮想マシンに Driver クラスを登録する際に必要なドライバ名称は，「パッケージ名称.クラス名称」です。JDBC ドライバのパッケージ名称とクラス名称は次のとおりです。

パッケージ名称：**JP.co.Hitachi.soft.HiRDB.JDBC**

クラス名称：**HiRDBDriver**

(1)　Class クラスの forName メソッドによる登録

アプリケーション内で，次のように Class クラスの forName メソッドを呼び出します。

```
Class.forName("JP.co.Hitachi.soft.HiRDB.JDBC.HiRDBDriver");
```

(2)　システムプロパティへの登録

Java 仮想マシンのシステムプロパティ「jdbc.drivers」に次の値を設定します。

```
System.setProperty("jdbc.drivers", "JP.co.Hitachi.soft.HiRDB.JDBC.HiRDBDriver");
```

(3)　Java 仮想マシンの動作設定ファイルへの登録（Applet の場合）

[JAVA_HOME]¥.hotjava¥properties ファイルに，次の内容を記述します（[JAVA_HOME]は，Java 実行環境によって異なります）。複数の JDBC ドライバを登録する場合には，コロン（:）で区切って記述してください。

```
jdbc.drivers="JP.co.Hitachi.soft.HiRDB.JDBC.HiRDBDriver"
```

17.2.2　getConnection メソッドによる HiRDB への接続

DriverManager クラスの getConnection メソッドには，引数が異なる次の三つの形式があります。

- public static Connection getConnection(String url)
- public static Connection getConnection(String url, String user, String password)
- public static Connection getConnection(String url, Properties info)

これらのメソッドの引数（url，user，password，及び info）には，HiRDB への接続に必要な接続情報を指定します。

正常に HiRDB に接続した場合，JDBC ドライバはこれらのメソッドを呼び出した結果として，Connection クラスのインスタンスの参照を返却します。ただし，次の場合には，これらのメソッドは SQLException を投入します。

- 必要な接続情報が各引数に指定されていない場合
- 接続情報の指定内容が不正な場合
- 接続に失敗した場合（接続先の HiRDB が開始していないときなど）

getConnection メソッドの引数の指定内容を次の表に示します。

表 17–3　getConnection メソッドの引数の指定内容

引数	指定内容	外部ドライバ※1	内部ドライバ※2
String url	URL を指定します。詳細については，「URL の構文」を参照してください。	○	○
String user	認可識別子を指定します。 指定値がナル値の場合は，認可識別子の指定がないものとみなされます。長さ 0 の文字列の場合は，SQLException を投入し，KFPJ20212-E メッセージの埋字となる"aa....aa"には"user"が設定されます。 指定の優先順位については，「接続情報の優先順位」を参照してください。	○	×
String password	パスワードを指定します。指定の優先順位については，「接続情報の優先順位」を参照してください。 指定値がナル値，又は長さ 0 の文字列の場合は，パスワードの指定がないものとみなされます。	○	×

引数	指定内容	外部ドライバ※1	内部ドライバ※2
Properties info	各種接続情報を指定します。詳細については，「ユーザプロパティ」を参照してください。	○	○

（凡例）

○：指定は有効になります。

×：指定は無効になります。

注※1

Java アプリケーションで使用する JDBC ドライバのことです。

注※2

Java ストアドプロシジャで使用する JDBC ドライバのことです。

(1)　URL の構文

JDBC ドライバで指定できる URL の構文について説明します。

なお，URL 内の各項目及び項目間にはスペースを入れないでください。また，各項目名称の大文字と小文字を区別するため，注意してください。

(a)　URL の構文

```
jdbc:hitachi:hirdb[://[DBID=接続付加情報]
                  [,DBHOST=DBホスト名称]
                  [,ENCODELANG=変換文字セット]
                  [,HIRDB_CURSOR=カーソル動作モード]
                  [,STATEMENT_COMMIT_BEHAVIOR=ステートメントのコミット実行後の状態]
                  [,JDBC_IF=JDBCインタフェースメソッドトレースの取得の有無]
                  [,TRC_NO=トレースのエントリ数]
                  [,SQLWARNING_IGNORE=警告情報を保持するかどうか]
                  [,LONGVARBINARY_ACCESS=BLOB型，BINARY型のアクセス方法]
                  [,SQL_IN_NUM=？パラメタ数]
                  [,SQL_OUT_NUM=出力パラメタ数]
                  [,SQLWARNING_LEVEL=警告保持レベル]
                  [,LONGVARBINARY_ACCESS_SIZE=LONGVARBINARYデータアクセスサイズ]
                  [,MAXBINARYSIZE=LONGVARBINARYデータの最大長]
                  [,LONGVARBINARY_TRUNCERROR=例外を投入するかどうか]
                  [,STATEMENT_CLOSE_BEHAVIOR=前処理結果を無効にするかどうか]
                  [,HiRDB_INI=HiRDB.iniファイルのディレクトリパス]
                  [,USER=ユーザ名称]
                  [,PASSWORD=パスワード]
                  [,UAPNAME=アプリケーション名称]
                  [,BATCHEXCEPTION_BEHAVIOR=BatchExceptionの更新カウントがJDBC規格に
                  準拠するかどうか]
                  [,UPDATECOUNT_BEHAVIOR=ステートメントの更新カウントがJDBC規格に準拠
                  するかどうか]
                  ]
```

(b)　URL の各項目の説明

jdbc:hitachi:hirdb

プロトコル名称，サブプロトコル名称，及びサブネームです。必ず指定してください。また，大文字と小文字の区別をするため，注意してください。

DBID=接続付加情報

HiRDB サーバのポート番号を指定します（クライアント環境定義の PDNAMEPORT に相当します）。又は，HiRDB の環境変数グループを指定します。

HiRDB サーバのポート番号を指定しなかった場合，そのほかの方法で設定した値が有効となります。HiRDB のポート番号の設定方法，及び優先順位については，「接続情報の優先順位」を参照してください。

どちらも指定されていない場合は，getConnection メソッド実行時に SQLException を投入します。

なお，内部ドライバの場合，HiRDB のポート番号指定は無効です。

＜注意事項＞

接続付加情報に HiRDB の環境変数グループを指定する場合，次の点に注意してください。

- HiRDB の環境変数グループ名を指定する場合は，「@HIRDBENVGRP=」に続けて絶対パス名を指定します。「@HIRDBENVGRP=,」のように，"="の後に何も指定しないと，この項目による指定がないものとみなされます。
- 環境変数グループ名は大文字と小文字を区別するため，注意してください。なお，環境変数グループ名は OS に依存します。
- 環境変数グループ名に半角スペース，又は半角@文字を含む場合，半角引用符（"）で囲んでください。環境変数グループ名を半角引用符で囲んだ場合，最後の半角引用符から次の設定項目，又は文字終端までの文字は無視されます。また，半角引用符，及び半角コンマを含む環境変数グループ名は指定できません。

  エラーになる指定例を次に示します。

```
@△HIRDBENVGRP=/HiRDB_P/Client/HiRDB.ini
@HIRDBENVGRP△=/HiRDB_P/Client/HiRDB.ini
@HIRDBENVGRP=△/HiRDB_P/Client/HiRDB.ini
@HIRDBENVGRP=/HiRDB_P/Client/HiRDB.ini△
```

  注　△は半角スペースを示します。

DBHOST=DB ホスト名称

HiRDB のホスト名を指定します。

この指定を省略した場合，そのほかの方法で設定した値が有効となります。HiRDB のホスト名称の設定方法，及び優先順位については，「接続情報の優先順位」を参照してください。

どちらも指定されていない場合は，getConnection メソッド実行時に SQLException を投入します。

なお，内部ドライバの場合は，この指定は無効です。

ENCODELANG=変換文字セット

String クラスで HiRDB とのデータ受け渡しをする場合に，接続先の HiRDB の文字コードに対応する変換文字セットを指定します。指定できる変換文字セットは，「Java 2 SDK, Standard Edition ドキュメント」の「国際化」で示されるエンコーディング一覧から選択して指定してください。なお，Java 仮想マシンがサポートしていない変換文字セット名称を指定した場合は，HiRDB サーバとの接続時に SQLException を投入します。

この指定を省略した場合，JDBC ドライバは HiRDB の文字コードと対応する変換文字セットで文字の変換をします。HiRDB の文字コードと対応する変換文字セットを次の表に示します。

表 17–4　HiRDB の文字コードと対応する変換文字セット

HiRDB の文字コード （pdntenv 又は pdsetup コマンドで設定した文字コード）	JDBC ドライバが使用する変換文字セット
lang-c	ISO8859_1
sjis	SJIS 又は MS932※
ujis	EUC_JP
utf-8	UTF-8
utf-8_ivs	UTF-8
chinese	EUC_CN
chinese-gb18030	GB18030

注※

SJIS か MS932 の指定は，アプリケーションでの Windows 特殊文字の扱いによります。

"OFF"を指定した場合も，HiRDB の文字コードに対して表「HiRDB の文字コードと対応する変換文字セット」の変換文字セットが指定されたものとして動作します。HiRDB の文字コードが sjis の場合は，JDBC ドライバが動作する OS によって変換文字セットは次のようになります。

UNIX の場合：**SJIS**

Windows の場合：**MS932**

なお，"OFF"以外の指定値は大文字と小文字を区別するため，注意してください。

また，次のものについては，Java 仮想マシンのデフォルトの変換文字セットで変換します。

- UAP 名称（プロパティの UAPNAME などで設定）の指定値
- 認可識別子又はパスワード（getConnection メソッドなどで指定）
- EnvironmentVariables で指定したクライアント環境定義の指定値
- HiRDB クライアントの環境変数グループ名で指定した各環境変数の指定値

HIRDB_CURSOR=カーソル動作モード

HiRDB がコミットした場合に，ResultSet クラスのオブジェクトを有効とするかどうかを指定します。

TRUE：コミットした場合でも，ResultSet クラスのオブジェクトを有効とします。

FALSE：コミットした場合，ResultSet クラスのオブジェクトを無効とします。

この指定を省略した場合，FALSE が仮定されます。

TRUE 又は FALSE 以外の値が指定された場合は，SQLException を投入します。

また，無効となった ResultSet オブジェクトで close メソッド呼び出し以外の操作をした場合は，SQLException を投入します。

＜注意事項＞

HIRDB_CURSOR 指定時の注意事項については，「HIRDB_CURSOR 及び STATEMENT_COMMIT_BEHAVIOR 指定時の注意事項」を参照してください。

STATEMENT_COMMIT_BEHAVIOR=ステートメントのコミット実行後の状態

HiRDB がコミットした場合に，Statement クラス，PreparedStatement クラス，及び CallableStatement クラス（以降，この三つのクラスをまとめてステートメントと表記します）のオブジェクトを，コミット実行後も有効とするかどうかを指定します。

TRUE：コミットした場合でも，ステートメントのオブジェクトを有効とします。

FALSE：コミットした場合，ステートメントのオブジェクトを無効とします。

なお，コミット実行後に無効とするのは，Connection クラスの prepareStatement メソッド，若しくは prepareCall メソッドでプリコンパイルした SQL 文，又はステートメントの executeQuery メソッドで取得した ResultSet クラスのオブジェクトです。

この指定を省略した場合，TRUE が仮定されます。

＜注意事項＞

STATEMENT_COMMIT_BEHAVIOR 指定時の注意事項については，「HIRDB_CURSOR 及び STATEMENT_COMMIT_BEHAVIOR 指定時の注意事項」を参照してください。

JDBC_IF=JDBC インタフェースメソッドトレースの取得の有無

JDBC インタフェースメソッドトレースの取得の有無を指定します。

ON：取得します。

OFF：取得しません。

この指定を省略した場合，OFF が仮定されます。上記以外の値を指定した場合は，SQLException を投入します。

この指定の詳細については，「JDBC_IF」を参照してください。

TRC_NO=トレースのエントリ数

JDBC インタフェースメソッドトレースのエントリ数を指定します。

この指定の詳細については，「TRC_NO」を参照してください。

SQLWARNING_IGNORE=警告情報を保持するかどうか

データベースから返される警告を Connection オブジェクトで保持するかどうかを指定します。

TRUE：警告を保持しません。

FALSE：警告を保持します。

この指定を省略した場合，FALSE が仮定されます。

上記以外の値を指定した場合は，SQLException を投入します。

この指定の詳細については，「SQLWARNING_IGNORE」を参照してください。

LONGVARBINARY_ACCESS=BLOB 型，BINARY 型のアクセス方法

JDBC の SQL データ型 LONGVARBINARY 型（HiRDB の BLOB 型，BINARY 型）のデータベースのアクセス方法を指定します。

REAL：実データでアクセスします。

LOCATOR：HiRDB の位置付け子機能を使用してアクセスします。

上記以外の場合は SQLException を投入します。

この指定の詳細については，「LONGVARBINARY_ACCESS」を参照してください。

SQL_IN_NUM=？パラメタ数

実行する SQL の入力？パラメタの最大数を指定します。

この指定の詳細については，「HiRDB_for_Java_SQL_IN_NUM」を参照してください。

SQL_OUT_NUM=出力パラメタ数

実行する SQL の出力項目数の最大数を指定します。

この指定の詳細については，「HiRDB_for_Java_SQL_OUT_NUM」を参照してください。

SQLWARNING_LEVEL=警告保持レベル

SQL 実行時に発生した警告情報の保持レベルを指定します。警告情報の保持レベルの詳細については，「SQLWarning オブジェクトの生成条件」を参照してください。

IGNORE：警告情報を IGNORE レベルで保持します。

SQLWARN：警告情報を SQLWARN レベルで保持します。

ALLWARN：警告情報を ALLWARN レベルで保持します。

この指定を省略した場合，SQLWARN が仮定されます。

指定値が不正な場合は，SQLException を投入します。

LONGVARBINARY_ACCESS_SIZE=LONGVARBINARY データアクセスサイズ

HiRDB サーバに対して一度に要求する JDBC の SQL データ型 LONGVARBINARY のデータ長を指定します（単位：キロバイト）。

この指定の詳細については，「HiRDB_for_Java_LONGVARBINARY_ACCESS_SIZE」を参照してください。

MAXBINARYSIZE=LONGVARBINARY データの最大長

JDBC の SQL データ型 LONGVARBINARY 型データ取得時のデータサイズの上限を設定します（単位：バイト）。

この指定の詳細については，「HiRDB_for_Java_MAXBINARYSIZE」を参照してください。

LONGVARBINARY_TRUNCERROR=例外を投入するかどうか

JDBC の SQL データ型 LONGVARBINARY 型データ取得時に切り捨てが発生した場合に，例外を投入するかどうかを指定します。

TRUE：例外を投入します。

FALSE：例外を投入しません。

この指定の詳細については，「HiRDB_for_Java_LONGVARBINARY_TRUNCERROR」を参照してください。

STATEMENT_CLOSE_BEHAVIOR=前処理結果を無効にするかどうか

ステートメント（Statement クラス，PreparedStatement クラス，及び CallableStatement クラス）の close メソッド実行時に前処理結果を無効にするかどうかを指定します。

TRUE：前処理結果を無効にします。

FALSE：前処理結果を無効にしません。

上記以外の値を指定した場合は，SQLException を投入します。

この指定の詳細については，「HiRDB_for_Java_STATEMENT_CLOSE_BEHAVIOR」を参照してください。

HiRDB_INI=HiRDB.ini ファイルのディレクトリパス

HiRDB.ini ファイル内に記述されている HiRDB クライアント環境変数を有効にする場合に，HiRDB.ini ファイルが存在するディレクトリの絶対パスを指定します。ここで指定したディレクトリの HiRDB.ini ファイルの HiRDB クライアント環境変数が有効になります。指定したディレクトリに HiRDB.ini ファイルがない場合，及び内部ドライバの場合，この指定は無効になります。

また，この指定を省略した場合，HiRDB.ini ファイルの内容は無視されます。

USER=ユーザ名称

ユーザ名称を指定します。

この指定の詳細については，「user」を参照してください。

PASSWORD=パスワード

パスワードを指定します。

この指定の詳細については，「password」を参照してください。

UAPNAME=アプリケーション名称

HiRDB サーバに対してアクセスする UAP の識別情報（UAP 識別子）を指定します。

この指定の詳細については，「UAPNAME」を参照してください。

BATCHEXCEPTION_BEHAVIOR=BatchException の更新カウントが JDBC 規格に準拠するかどうか

java.sql.BatchUpdateException の getUpdateCounts メソッドの戻り値に，JDBC 規格に準拠した更新カウントを設定するかどうかを指定します。

TRUE：JDBC 規格に準拠した更新カウントを設定します。

FALSE：HiRDB 独自の更新カウントを設定します。

この指定の詳細については，「HiRDB_for_Java_BATCHEXCEPTION_BEHAVIOR」を参照してください。

UPDATECOUNT_BEHAVIOR=ステートメントの更新カウントが JDBC 規格に準拠するかどうか

更新系 SQL 発行時の更新行数が 0 の場合，java.sql.Statement の getUpdateCount メソッドの戻り値として，-1 を返却するかどうかを指定します。

TRUE：更新系 SQL 発行時の更新行数が 0 の場合，-1 を返却します（JDBC 規格）。

FALSE：更新系 SQL 発行時の更新行数が 0 の場合，0 を返却します。

この指定の詳細については，「HiRDB_for_Java_UPDATECOUNT_BEHAVIOR」を参照してください。

(c) HIRDB_CURSOR 及び STATEMENT_COMMIT_BEHAVIOR 指定時の注意事項

HIRDB_CURSOR 及び STATEMENT_COMMIT_BEHAVIOR 指定時の注意事項を次に示します。

＜ HIRDB_CURSOR 又は STATEMENT_COMMIT_BEHAVIOR に TRUE を指定した場合＞

- クライアント環境定義 PDDDLDEAPRPEXE が NO，かつ PDDDLDEAPRP が NO の場合，SELECT 文，INSERT 文，DELETE 文，UPDATE 文，PURGE TABLE 文，又は CALL 文のどれかでアクセスするスキーマ資源（表やインデクスなど）に対して，ほかのユーザが定義系 SQL を実行すると，スキーマ資源にアクセスしていたコネクションを切断するまでの間，又はスキーマ資源にアクセスしていたステートメントオブジェクトをクローズしコミットするまでの間（STATEMENT_CLOSE_BEHAVIOR に TRUE を設定している場合），定義系 SQL は排他待ちの状態になります。
- クライアント環境定義 PDDDLDEAPRPEXE が YES，又は PDDDLDEAPRP が YES の場合，SELECT 文，INSERT 文，DELETE 文，UPDATE 文，PURGE TABLE 文，又は CALL 文のどれかでアクセスするスキーマ資源（表やインデクスなど）に対して，ほかのユーザが定義系 SQL を実行すると，SELECT 文，INSERT 文，DELETE 文，UPDATE 文，PURGE TABLE 文，又は CALL 文の前処理結果は無効になります。前処理結果が無効となった SQL を実行すると，SQLException 例外（getErrorCode メソッドで取得できる値は-1542）が発生します。
- HIRDB_CURSOR 又は STATEMENT_COMMIT_BEHAVIOR に TRUE を指定※1 することで，コミット※2 後もプリコンパイルした SQL 文※3 が有効になるのは，SELECT 文，INSERT 文，DELETE 文，UPDATE 文，PURGE TABLE 文，及び CALL 文だけです。

  注※1

  次のどちらかの指定の場合も該当します。

  ・getConnection メソッドで指定するプロパティ中の次の項目に TRUE を設定した。

  　・HIRDB_CURSOR

  　・HiRDB_for_Java_STATEMENT_COMMIT_BEHAVIOR

  ・PrdbDataSource クラス，PrdbConnectionPoolDataSource クラス，又は PrdbXADataSource クラスの次のメソッドで true を指定した。

  　・setHiRDBCursorMode

  　・setStatementCommitBehavior

  注※2

  commit メソッドによる明示的なコミットのほかに，次の場合も該当します。

  ・自動コミットによる暗黙的なコミット

  ・定義系 SQL の実行

  ・PURGE TABLE 文の実行

  注※3

  Connection クラスの prepareStatement メソッド又は prepareCall メソッドを実行することで SQL 文をプリコンパイルできます。

SELECT 文，INSERT 文，DELETE 文，UPDATE 文，PURGE TABLE 文，及び CALL 文以外の SQL 文では，コミット時にプリコンパイル済みの SQL 文は無効になります。

無効になったプリコンパイルした SQL 文を格納した PreparedStatement クラス又は CallableStatement クラスのオブジェクトで SQL 文を実行するとエラーになります。エラーとなる例を次に示します。

```
PreparedStatement pstmt1 = con.prepareStatement("lock table tb1");
PreparedStatement pstmt2 = con.prepareStatement("lock table tb2");
pstmt1.execute();  //エラーになりません。
con.commit();
pstmt2.execute();  //エラーになります。
pstmt1.close();
pstmt2.close();
```

この例では，実行する SQL 文が LOCK 文であるため，STATEMENT_COMMIT_BEHAVIOR に TRUE を指定していても，コミット後は PreparedStatement が無効となって，エラーが発生します。

- HIRDB_CURSOR に TRUE を指定した場合，JDBC ドライバは HiRDB のホールダブルカーソル機能を使用します。
- HIRDB_CURSOR に TRUE を指定した場合，結果集合返却機能を使用してプロシジャが返却する結果集合のカーソル動作モードは，プロシジャの定義によって決まります。ただし，JDBC ドライバではプロシジャのカーソル動作モードを検知できないため，HIRDB_CURSOR に指定された動作モードと仮定して動作します。そのため，HIRDB_CURSOR が TRUE であっても，プロシジャが返却する結果集合のカーソル動作モードがホールダブルカーソルでない場合，コミット後のデータベースアクセスは SQLException となります。コミット後の ResultSet オブジェクトの状態を次に示します。

プロシジャが返却する結果集合	HIRDB_CURSOR の設定値	
	TRUE	FALSE
ホールダブルカーソルである	コミット後も使用可能	コミット後の操作でデータベースアクセスを伴う場合はエラーとなる※
ホールダブルカーソルでない	コミット後の操作でデータベースアクセスを伴う場合はエラーとなる	コミット後の操作でデータベースアクセスを伴う場合はエラーとなる

注※

コミット時にカーソルがクローズされないため，ResultSet オブジェクトのクローズ，若しくは ResultSet オブジェクトを生成した CallableStatement オブジェクト又は Connection オブジェクトをクローズするまでカーソルは開いたままとなります。

＜ HIRDB_CURSOR 及び STATEMENT_COMMIT_BEHAVIOR の組み合わせ＞

コミット実行後に ResultSet オブジェクト及びステートメントオブジェクトが有効になるかどうかを，HIRDB_CURSOR，STATEMENT_COMMIT_BEHAVIOR の組み合わせごとに次の表に示します。

表 17–5　コミット実行後の ResultSet オブジェクト及びステートメントオブジェクトの状態

STATEMENT_COMMIT_BEHAVIOR の指定値	HIRDB_CURSOR の指定値	
	TRUE	FALSE
TRUE	ResultSet オブジェクト： 有効 ステートメントオブジェクト： 有効	ResultSet オブジェクト： 無効 ステートメントオブジェクト： 有効
FALSE		ResultSet オブジェクト： 無効 ステートメントオブジェクト： 無効

また，DatabaseMetaData のメソッドの戻り値を，HIRDB_CURSOR，STATEMENT_COMMIT_BEHAVIOR の組み合わせごとに次の表に示します。

表 17–6　DatabaseMetaData のメソッドの戻り値

STATEMENT_COMMIT_BEHAVIOR の指定値	HIRDB_CURSOR の指定値	
	TRUE	FALSE
TRUE	supportsOpenStatementsAcrossCommit： true supportsOpenCursorsAcrossCommit： true	supportsOpenStatementsAcrossCommit： true supportsOpenCursorsAcrossCommit： false
FALSE		supportsOpenStatementsAcrossCommit： false supportsOpenCursorsAcrossCommit： false

＜コミット実行時の JDBC ドライバの動作例＞

コミット実行時の JDBC ドライバの動作は，HIRDB_CURSOR 及び STATEMENT_COMMIT_BEHAVIOR の指定によって異なります。

指定例

```
[A]
pstmt1=con.prepareStatement("select c1 from tb1");  [1]
[B]
rs1=pstmt1.executeQuery();                          [2]
[C]
rs1.next()                                          [3]
[D]
v1=rs1.getInt(1)                                    [4]
[E]
rs1.next()                                          [5]
[F]
v1=rs1.getInt(1)                                    [6]
```

```
[G]
rs1.close()                                    [7]
```

コミット実行時の動作

コミットのタイミング	H=T，S=T※1	H=F，S=T※2	H=F，S=F※3
[A]	[1]～[7]は正常に動作します。		
[B]	[1]～[7]は正常に動作します。		[1]，[2]，[7]：正常に動作します。 [3]～[6]：SQLException を投入します。
[C]	[1]～[7]は正常に動作します。	[1]，[2]，[7]：正常に動作します。 [3]～[6]：SQLException を投入します。	
[D]	[1]～[7]は正常に動作します。	[1]～[3]，[7]：正常に動作します。 [4]～[6]：SQLException を投入します。	
[E]	[1]～[7]は正常に動作します。	[1]～[4]，[7]：正常に動作します。 [5]，[6]：SQLException を投入します。	
[F]	[1]～[7]は正常に動作します。	[1]～[5]，[7]：正常に動作します。 [6]：SQLException を投入します。	
[G]	[1]～[7]は正常に動作します。		

注※1　HIRDB_CURSOR に TRUE，STATEMENT_COMMIT_BEHAVIOR に TRUE を指定した場合を示します。

注※2　HIRDB_CURSOR に FALSE，STATEMENT_COMMIT_BEHAVIOR に TRUE を指定した場合を示します。

注※3　HIRDB_CURSOR に FALSE，STATEMENT_COMMIT_BEHAVIOR に FALSE を指定した場合を示します。

＜そのほかの注意事項＞

クライアント環境定義 PDDDLDEAPRP についての注意事項は，「クライアント環境定義の設定内容」を参照してください。

また，DECLARE CURSOR のホールダブルカーソルについての規則は，マニュアル「HiRDB SQL リファレンス」を参照してください。

(2)　ユーザプロパティ

DriverManager クラスの getConnection メソッドに指定できるプロパティを次の表に示します。なお，プロパティの指定値がナル値の場合，指定を省略したものとして扱われます。

表 17–7　getConnection メソッドに指定できるプロパティ

プロパティ	指定内容
user	認可識別子
password	パスワード
UAPNAME	UAP 識別子
JDBC_IF	JDBC インタフェースメソッドトレースの取得の有無
TRC_NO	JDBC インタフェースメソッドトレースのエントリ数
ENCODELANG	接続先の HiRDB の文字コードに対応する変換文字セット
SQLWARNING_IGNORE	データベースから返される警告を Connection オブジェクトで保持するかどうか
HIRDB_CURSOR	カーソル動作モード
LONGVARBINARY_ACCESS	JDBC SQL タイプ LONGVARBINARY（HiRDB のデータ型である BLOB 型，及び BINARY 型）のデータへのアクセス方法
HiRDB_for_Java_SQL_IN_NUM	実行する SQL の入力？パラメタの最大数
HiRDB_for_Java_SQL_OUT_NUM	実行する SQL の出力項目の最大数
HiRDB_for_Java_SQLWARNING_LEVEL	SQL 実行時に発生した警告情報の保持レベル
HiRDB_for_Java_ENV_VARIABLES	HiRDB クライアント環境変数
HiRDB_for_Java_STATEMENT_COMMIT_BEHAVIOR	ステートメントのコミット実行後の状態
HiRDB_for_Java_LONGVARBINARY_ACCESS_SIZE	HiRDB サーバに対して一度に要求する JDBC SQL タイプ LONGVARBINARY 型データの長さ
HiRDB_for_Java_MAXBINARYSIZE	JDBC SQL タイプ LONGVARBINARY 型データ取得時のデータサイズの上限
HiRDB_for_Java_LONGVARBINARY_TRUNCERROR	JDBC SQL タイプ LONGVARBINARY 型データ取得時に切り捨てが発生した場合に，例外を投入するかどうか
HiRDB_for_Java_STATEMENT_CLOSE_BEHAVIOR	ステートメント（Statement クラス，PreparedStatement クラス，及び CallableStatement クラス）の close メソッド実行時に前処理結果を無効にするかどうか
HiRDB_for_Java_DBID	接続付加情報
HiRDB_for_Java_DBHOST	ホスト名称
HiRDB_for_Java_HiRDB_INI	HiRDB.ini ファイルのディレクトリパス
HiRDB_for_Java_BATCHEXCEPTION_BEHAVIOR	JDBC 規格に準拠した更新カウントを設定するかどうか

プロパティ	指定内容
HiRDB_for_Java_UPDATECOUNT_BEHAVIOR	ステートメントの更新カウントが JDBC 規格に準拠するかどうか

(a) user

認可識別子を指定します。

指定値がナル値の場合は，認可識別子の指定がないものとみなされます。長さ 0 の文字列の場合は，SQLException を投入します。

この指定を省略した場合，getConnection メソッドの引数 Properties 中の，HiRDB_for_Java_ENV_VARIABLES で指定した HiRDB クライアント環境定義の PDNAMEPORT，又は URL 中の DBID に指定した HiRDB 環境変数グループ内の PDUSER の指定値が有効となります。指定の優先順位については，「接続情報の優先順位」を参照してください。

どちらも指定されていない場合は，getConnection メソッド実行時に SQLException を投入します。

なお，内部ドライバの場合は，このプロパティの指定は無効です。

(b) password

パスワードを指定します。

指定値がナル値の場合又は長さが 0 の場合は，パスワードの指定がないものとみなされます。

この指定を省略した場合については，「接続情報の優先順位」を参照してください。

なお，内部ドライバの場合は，このプロパティの指定は無効です。

(c) UAPNAME

HiRDB サーバに対してアクセスする，UAP の識別情報（UAP 識別子）を指定します。

次の場合は，認可識別子の指定がないものとみなされます。

- ナル値を指定した場合
- 長さ 0 の文字列，又は半角スペースだけの文字列を指定した場合

指定できる文字列の詳細については，「クライアント環境定義の設定内容」のクライアント環境定義 PDCLTAPNAME を参照してください。

この指定を省略した場合については，「接続情報の優先順位」を参照してください。

なお，内部ドライバの場合は，このプロパティの指定は無効です。

《注意事項》

このプロパティで指定した UAP は，ENCODELANG で指定された変換文字セットでエンコードされ，エンコード後の UAP 識別子の先頭から 30 バイトが HiRDB サーバに転送されます（30 バイト目が文字の途中であっても，30 バイトまでで打ち切られます）。そのため，HiRDB サーバで取得できる UAP 識別子は，エンコード後の先頭 30 バイトまでです。

(d) JDBC_IF

JDBC インタフェースメソッドトレースの取得の有無を指定します。

ON：JDBC インタフェースメソッドトレースを取得します。

OFF：JDBC インタフェースメソッドトレースを取得しません。

この指定を省略した場合，OFF が仮定されます。

また，これらの値以外を指定すると，SQLException を投入します。

なお，次に示す場合，このプロパティの指定は無効です。

- setLogWriter メソッドで有効なログライタを指定していない場合
- 内部ドライバの場合

JDBC インタフェースメソッドトレースの詳細は，「JDBC インタフェースメソッドトレース」を参照してください。

(e) TRC_NO

～＜符号なし整数＞((10～1000))《500》

JDBC インタフェースメソッドトレースのエントリ数を指定します。

このプロパティは，次の条件をすべて満たしている場合に有効になります。

- setLogWriter メソッドで有効なログライタを設定している。
- JDBC_IF に ON を指定している。

なお，内部ドライバの場合は，このプロパティの指定は無効です。

プロパティの指定が有効な状態で，かつ指定値が不正な場合は，SQLException を投入します。

JDBC インタフェースメソッドトレースの詳細は，「JDBC インタフェースメソッドトレース」を参照してください。

(f) ENCODELANG

String クラスで HiRDB とのデータ受け渡しをする場合に，接続先の HiRDB の文字コードに対応する変換文字セットを指定します。

指定できる変換文字セットは，「Java 2 SDK, Standard Edition ドキュメント」の「国際化」で示されるエンコーディング一覧から選択してください。

HiRDB の文字コードと対応する変換文字セットは，「HiRDB の文字コードと対応する変換文字セット」を参照してください。

OFF を指定すると，HiRDB の文字コードに対して「HiRDB の文字コードと対応する変換文字セット」の変換文字セットが指定されたものとして動作します。なお，HiRDB の文字コードが sjis の場合，JDBC ドライバが動作する OS によって変換文字セットは次のようになります。

UNIX の場合：**SJIS**

Windows の場合：**MS932**

また，OFF 以外の指定値は大文字と小文字を区別するため，注意してください。

Java 仮想マシンがサポートしていない変換文字セット名称を指定した場合は，HiRDB サーバとの接続時に SQLException を投入します。

この指定を省略した場合，JDBC ドライバは URL 中の ENCODELANG で指定した変換文字セットで文字の変換をします。

(g) SQLWARNING_IGNORE

データベースから返される警告を Connection オブジェクトで保持するかどうかを指定します。

TRUE：警告を保持しません。

FALSE：警告を保持します。

この指定を省略した場合，URL 中の SQLWARNING_IGNORE で指定した値が有効になります。TRUE 又は FALSE 以外を指定すると，SQLException を投入します。

Connection オブジェクトの警告保持は，このプロパティと，HiRDB_for_Java_SQLWARNING_LEVEL の指定値で決定します。警告保持レベルについては，「SQLWarning オブジェクトの生成条件」を参照してください。なお，このプロパティでは，指定した内容について大文字と小文字を区別しません。

(h) HIRDB_CURSOR

HiRDB がコミットした場合に，ResultSet クラスのオブジェクトを有効とするかどうか（カーソル動作モード）を指定します。

TRUE：コミットした場合でも，ResultSet クラスのオブジェクトを有効とします。

FALSE：コミットした場合，ResultSet クラスのオブジェクトを無効とします。

この指定を省略した場合，URL 中の HIRDB_CURSOR で指定した値が有効になります。TRUE 又は FALSE 以外を指定すると，SQLException を投入します。

また，無効となった ResultSet オブジェクトで close メソッド呼び出し以外の操作をした場合，SQLException を投入します。

《注意事項》

このプロパティを指定するときの注意事項については，「HIRDB_CURSOR 及び STATEMENT_COMMIT_BEHAVIOR 指定時の注意事項」を参照してください。

(i) LONGVARBINARY_ACCESS

JDBC SQL タイプ LONGVARBINARY（HiRDB のデータ型である BLOB 型，及び BINARY 型）のデータへのアクセス方法を指定します。

REAL：実データでアクセスします。

LOCATOR：HiRDB の位置付け子機能を使用してアクセスします。

ただし，定義長が 1,024 バイト以下の BINARY 列へのアクセス時は，実データでアクセスします。

この指定を省略した場合，REAL が仮定されます。

また，これらの値以外を指定すると，SQLException を投入します。

《注意事項》

LONGVARBINARY_ACCESS 指定時の注意事項を次に示します。

＜ HiRDB_for_Java_LONGVARBINARY_ACCESS_SIZE と併せて指定した場合の注意事項＞

HiRDB_for_Java_LONGVARBINARY_ACCESS_SIZE，及び LONGVARBINARY_ACCESS の指定による，BLOB 型又は BINARY 型データ（HiRDB のデータ型）の取得方法の違いを次の表に示します。

表 17–8　BLOB 型又は BINARY 型データ（HiRDB のデータ型）の取得方法の違い

実行メソッド	LONGVARBINARY_ACCESS の指定値※	
	REAL	LOCATOR
CallableStatement.execute ResultSet.next	BLOB 型又は BINARY 型データの全体を，接続先 DB から取得します。	BLOB 型又は BINARY 型データの全体ではなく，接続先 DB 内の BLOB 型又は BINARY 型データを示す位置付け子を取得します。
CallableStatement.getBytes CallableStatement.getString CallableStatement.getObject ResultSet.getBytes ResultSet.getString ResultSet.getObject	ResultSet.next で取得した BLOB 型又は BINARY 型データを使用します。	BLOB 型又は BINARY 型データの全体を，接続先 DB から ACCESSSIZE×1024 バイト単位に分割して取得します。
Blob.getBytes	ResultSet.next で取得した BLOB 型又は BINARY 型データから，引数で指定された範囲を切り出して取得します。	引数で指定された範囲の BLOB 型又は BINARY 型データを，接続先 DB から ACCESSSIZE×1024 バイト単位に分割して取得します。

実行メソッド		LONGVARBINARY_ACCESS の指定値※	
		REAL	LOCATOR
CallableStatement.getBinaryStream ResultSet.getBinaryStream ResultSet.getAsciiStream ResultSet.getUnicodeStream Blob.getBinaryStream		実行メソッドによって取得した InputStream の read メソッドを実行した場合，ResultSet.next で取得した BLOB 型又は BINARY 型データから抽出して取得します。	実行メソッドによって取得した InputStream の read メソッドを実行した場合，接続先 DB からデータを取得します。
Blob.length		ResultSet.next で取得した BLOB 型又は BINARY 型データからデータ長を取得します。	接続先 DB からデータ長を取得します。
Blob.position		ResultSet.next で取得した BLOB 型又は BINARY 型データから，検索パターンに一致する位置を取得します。	接続先 DB から，検索パターンに一致する位置を取得します。
CallableStatement.getBinaryStream，ResultSet.getBinaryStream，Blob.getBinaryStream で取得した InputStream	InputStream.available	位置付け子が示す実データの長さ以下の値を返却します。	ACCESSSIZE×1024 バイト以下の値を返却します。
	InputStream.skip	位置付け子が示す実データの長さ以下までの範囲で読み飛ばします。	最大 ACCESSSIZE×1024 バイト以下で読み飛ばします。
CallableStatement.getCharacterStream ResultSet.getCharacterStream		getCharacterStream によって取得した Reader の read メソッドを実行した場合，ResultSet.next で取得した BLOB 型又は BINARY 型データから抽出して取得します。	getCharacterStream によって取得した Reader の read メソッドを実行した場合，接続先 DB からデータを取得します。

（凡例）

ACCESSSIZE：HiRDB_for_Java_LONGVARBINARY_ACCESS_SIZE の指定値を示します。

InputStream 及び Reader：JDBC ドライバの getBinaryStream，getAsciiStream，getCharacterStream が返したオブジェクトのクラスを示します。

注※　定義長が 1,024 バイト以下の BINARY 列へのアクセス時は，"REAL"が指定されたものとして動作します。

＜実行性能に関する注意事項＞

LONGVARBINARY_ACCESS に"LOCATOR"を指定すると，"REAL"を指定した場合に比べて性能が低下するおそれがあります。

"REAL"を指定した場合，ResultSet.next 時又は CallableStatement.execute 時に位置付け子を取得するために 1 回接続先 DB にアクセスします。それに対して"LOCATOR"を指定すると，ResultSet.next 時又は CallableStatement.execute 時のアクセス 1 回に加えて，getBytes などのデータを取得するメソッド実行時などにデータ長取得のために 1 回，データ取得のために 1 回以上接続先 DB にアクセスします。

＜ AUTO コミットが有効な場合の注意事項＞

AUTO コミットが有効な場合でも，次のタイミングではコミットを実行しません。

- LONGVARBINARY_ACCESS に"LOCATOR"を指定し，次のどちらかに該当する出力パラメタを指定しているストアドプロシジャの実行
  ・出力パラメタが 1,024 バイトより大きいサイズの BINARY 型
  ・出力パラメタが BLOB 型
- 結果集合返却機能を使用したストアドプロシジャの実行

＜トランザクション終了後のデータ操作に間する注意事項＞

LONGVARBINARY_ACCESS に"LOCATOR"を指定した場合，SQL の実行結果取得（ResultSet.next，又は CallableStatement.execute）からデータ操作（Blob.getBytes や InputStream.read など）までにトランザクションが終了すると，データ操作ができません。また，HIRDB_CURSOR の指定が"TRUE"であっても，トランザクション終了後のデータ操作は実行できません。

そのため，データ操作はトランザクション終了前に実行してください。

(j) HiRDB_for_Java_SQL_IN_NUM

～＜符号なし整数＞((1～30000))《300》

実行する SQL の入力？パラメタの最大数を指定します。

この指定は，SQL の前処理時に取得する入力？パラメタ情報の数となります。実際の入力？パラメタの数がこのプロパティの指定値よりも多い場合，JDBC ドライバは SQL の前処理の後に HiRDB サーバから入力？パラメタ情報を取得します。

指定値が不正な場合は，SQLException を投入します。

《注意事項》

入力？パラメタのある SQL 文を実行しない場合は，1 を指定することを推奨します。

(k) HiRDB_for_Java_SQL_OUT_NUM

～＜符号なし整数＞((1～30000))《300》

実行する SQL の出力項目の最大数を指定します。

この指定は，SQL の前処理時に取得する出力項目情報の数となります。実際の出力項目情報の数がこのプロパティの指定値よりも多い場合，JDBC ドライバは SQL の前処理の後に HiRDB サーバから出力項目情報を取得します。

指定値が不正な場合は，SQLException を投入します。

《注意事項》

出力項目のある SQL 文を実行しない場合は，1 を指定することを推奨します。

(l)　HiRDB_for_Java_SQLWARNING_LEVEL

SQL 実行時に発生した警告情報の保持レベルを指定します。警告情報の保持レベルの詳細については，「SQLWarning オブジェクトの生成条件」を参照してください。

IGNORE：警告情報を IGNORE レベルで保持します。

SQLWARN：警告情報を SQLWARN レベルで保持します。

ALLWARN：警告情報を ALLWARN レベルで保持します。

この指定を省略した場合，"SQLWARN"が仮定されます。

指定値が不正な場合は，SQLException を投入します。

(m)　HiRDB_for_Java_ENV_VARIABLES

HiRDB クライアント環境定義を指定します。次のように指定してください。

```
変数名＝値;変数名＝値; ..〈省略〉..  ;変数名＝値
```

JDBC ドライバで指定できるクライアント環境定義は，「指定できるクライアント環境定義」を参照してください。変数名に JDBC ドライバで指定できないクライアント環境定義が指定された場合，指定を無視します。なお，変数名は大文字と小文字を区別するため，注意してください。

複数の設定方法を持つ接続情報の優先順位については，「接続情報の優先順位」を参照してください。

《指定例》

```
java.util.Properties prop;
prop=new java.util.Properties();
prop.setProperty(”HiRDB_for_Java_ENV_VARIABLES”,
 ”PDFESHOST=FES1;PDCWAITTIME=0”);
```

(n)　HiRDB_for_Java_STATEMENT_COMMIT_BEHAVIOR

HiRDB がコミットした場合に，ステートメントのオブジェクトをコミット実行後も有効とするかどうかを指定します。

TRUE：コミットした場合でも，ステートメントのオブジェクトを有効とします。

FALSE：コミットした場合，ステートメントのオブジェクトを無効とします。

なお，コミット実行後に無効とするのは，Connection クラスの prepareStatement メソッドでプリコンパイルした SQL 文，Connection クラスの prepareCall メソッドでプリコンパイルした SQL 文，又はステートメントの executeQuery メソッドで取得した ResultSet クラスのオブジェクトです。

この指定を省略した場合，URL 中の STATEMENT_COMMIT_BEHAVIOR で指定した値が有効になります。

《注意事項》

このプロパティを指定するときの注意事項については，「HIRDB_CURSOR 及び STATEMENT_COMMIT_BEHAVIOR 指定時の注意事項」を参照してください。

(o) HiRDB_for_Java_LONGVARBINARY_ACCESS_SIZE

～<符号なし整数>((0～2097151))《0》（単位：キロバイト）

HiRDB サーバに対して一度に要求する JDBC SQL タイプ LONGVARBINARY 型データの長さを指定します。LONGVARBINARY_ACCESS で LOCATOR 以外を指定している場合，この指定は無効です。

例えば，このプロパティに 20 を指定した場合，データベースに格納している 100 キロバイトの JDBC SQL タイプ LONGVARBINARY 型データを ResultSet の getBytes メソッドで取得しようとすると，JDBC ドライバはデータを 20 キロバイトずつ 5 回に分けて取得し，返却します。0 の場合はデータ全体を一度に要求します。

指定値が不正な場合は，SQLException を投入します。

《注意事項》

このプロパティを指定するときの注意事項については，「LONGVARBINARY_ACCESS」を参照してください。

(p) HiRDB_for_Java_MAXBINARYSIZE

～<符号なし整数>((0～2147483647))（単位：バイト）

JDBC SQL タイプ LONGVARBINARY 型データ取得時のデータサイズの上限を指定します。

JDBC SQL タイプ LONGVARBINARY 型データを取得する際，JDBC ドライバはデータを取得するまで実際のデータ長を認識できないため，定義長分のメモリを確保します。そのため，定義長に HiRDB のデータ型である BINARY 型又は BLOB 型の最大長である 2,147,483,647 バイトのように長大なサイズを指定した列の値を取得する場合は，その定義長である 2,147,483,647 バイトのメモリを確保しようとします。そのため，実行環境によってはメモリ不足が発生することがあります。

したがって，このプロパティで，実際に格納されているデータの最大長を指定してください。取得対象となる HiRDB のデータ型である BINARY 型又は BLOB 型データの定義長が，このプロパティで指定したサイズよりも大きい場合，取得データをこのプロパティで指定したサイズに切り捨てます。実際にデータを切り捨てた場合，ResultSet の next メソッド実行時に，JDBC ドライバは HiRDB サーバから警告を受け取ります。受け取った警告に対しては，setLONGVARBINARY_TruncError の指定値に従って SQLException の投入，SQLWarning の生成（又は無視）をします。

このプロパティで上限を設定していない場合は，取得対象データの定義長を上限とします。

指定値が不正な場合は，SQLException を投入します。

《注意事項》

LONGVARBINARY_ACCESS に"LOCATOR"を指定し，BLOB 列又は定義長が 1,024 バイトより大きい BINARY 列にアクセスする場合は，このプロパティの指定値は無効です。実際のデータ長に基づいて領域を確保し，全データを取得します。

(q) HiRDB_for_Java_LONGVARBINARY_TRUNCERROR

JDBC SQL タイプ LONGVARBINARY 型データ取得時に切り捨てが発生した場合に，例外を投入するかどうかを指定します。

TRUE：切り捨てが発生した場合に例外を投入します。

FALSE：切り捨てが発生した場合に例外を投入しません。

この指定を省略した場合，TRUE が仮定されます。

HiRDB_for_Java_SQLWARNING_LEVEL に IGNORE を指定している場合は，FALSE が指定されたものとして動作します。

なお，JDBC SQL タイプ LONGVARBINARY 型データ取得時に発生する切り捨てとは，次の条件を満たしている場合を指します。

```
SQLの実行で得られるJDBC SQLタイプLONGVARBINARY型データの実際の長さ ＞ HiRDB_for_Java_MAXBINARYSIZEで指定したデータ長
```

(r) HiRDB_for_Java_STATEMENT_CLOSE_BEHAVIOR

ステートメントオブジェクト又は ResultSet オブジェクトを，コミット後も有効にする指定をしている場合※1，ステートメント（Statement クラス，PreparedStatement クラス，及び CallableStatement クラス）の close メソッド実行時に前処理結果を無効にするかどうかを指定します。

なお，システムプロパティの HiRDB_for_Java_STATEMENT_CLOSE_BEHAVIOR でも指定できます。

サーバが XDM/RD E2 の場合，この指定は無効になります。

TRUE：前処理結果を無効にします。

FALSE：前処理結果を無効にしません。

指定値が不正な場合は，SQLException を投入します。

このプロパティは，指定した内容について大文字と小文字を区別しません。

また，この指定を省略した場合，URL 中の項目 STATEMENT_CLOSE_BEHAVIOR の指定値が有効になります。URL 中に指定がない場合は，FALSE が指定されたものとして動作します。

クライアント環境変数 PDDDLDEAPRP，及び PDDDLDEAPRPEXE に NO を指定した場合，SQL（SELECT 文，INSERT 文，DELETE 文，UPDATE 文，PURGE TABLE 文，CALL 文）を実行したとき，SQL がアクセ

スするスキーマ資源（表やインデクスなど）に対して，ほかのユーザが定義系 SQL を実行すると，定義系 SQL は排他待ちの状態になります。この排他待ち状態を解除する契機を次に示します。なお，TRUE を指定すると，コミット実行時に解除できます。

- ステートメントオブジェクト又は ResultSet オブジェクトを，コミット後も有効にする指定をしている場合
  - TRUE を指定したとき

    ステートメント close 後のコミット実行時※2，又はコネクション切断時
  - FALSE を指定したとき

    コネクション切断時
- その他の場合

  コミット実行時※2，又はコネクション切断時

注※1

次のどれかになります。

1. getConnection メソッドで指定するプロパティ中の次の項目に TRUE を設定している。

   HIRDB_CURSOR

   HiRDB_for_Java_STATEMENT_COMMIT_BEHAVIOR
2. getConnection メソッドで指定する URL 中の次の項目に TRUE を設定している。

   HIRDB_CURSOR

   STATEMENT_COMMIT_BEHAVIOR
3. PrdbDataSource クラス，PrdbConnectionPoolDataSource クラス，又は PrdbXADataSource クラスの次のメソッドで true を設定している。

   setHiRDBCursorMode

   setStatementCommitBehavior
4. Connection.setHoldability（引数に ResultSet.HOLD_CURSORS_OVER_COMMIT 指定）メソッドを実行している。
5. Connection クラスの次のメソッド（引数 resultSetHoldability に ResultSet.HOLD_CURSORS_OVER_COMMIT を指定）を実行している。

   createStatement(int resultSetType, int resultSetConcurrency, int resultSetHoldability)

   prepareCall(String sql, int resultSetType, int resultSetConcurrency, int resultSetHoldability)

   prepareStatement(String sql, int resultSetType, int resultSetConcurrency, int resultSetHoldability)

注※2

commit メソッドによる明示的なコミットのほかに，次の場合も該当します。

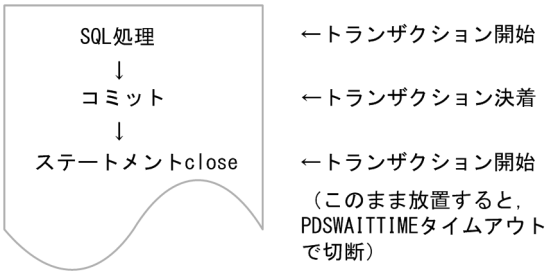
- 自動コミットによる暗黙的なコミット
- 定義系 SQL の実行

- PURGE TABLE 文の実行
- rollback メソッドによる明示的なロールバック
- SQL 実行エラーによる暗黙的なロールバック

TRUE を指定，かつステートメントオブジェクト又は ResultSet オブジェクトを，コミット後も有効にする指定をしている場合，ステートメントの close 時，HiRDB はトランザクションを生成して DEALLOCATE PREPARE 処理を実行します。このため，クライアント環境変数 PDSWAITTIME を指定している場合，PDSWAITTIME で指定した時間監視が開始されます。自動コミットを無効にしている場合，ステートメントの close 後，ほかのステートメントでの SQL 実行や，コミット又はコネクション切断を長時間実行しないと，時間監視によって HiRDB は接続を切断します。

DEALLOCATE PREPARE 処理とトランザクションの関係を次の図に示します。

図 17–1　DEALLOCATE PREPARE 処理とトランザクションの関係



Java EE アプリケーションサーバのコネクションプーリング機能を使用している場合，アプリケーションからコネクションの close メソッドを呼び出しても，コネクションはプールに戻り，トランザクションは残ります。このため，TRUE を指定した場合は，ステートメントの close 後にコミットを実行する必要があります。

また，ステートメントプーリング機能を使用している場合，この指定を省略するか，FALSE を指定してください。

(s)　HiRDB_for_Java_DBID

HiRDB のポート番号（PDNAMEPORT に当たる情報），又は HiRDB クライアントの環境変数グループファイル名を指定します。この指定の詳細については，「URL の各項目の説明」の DBID の説明を参照してください。

(t)　HiRDB_for_Java_DBHOST

HiRDB のホスト名称を指定します。この指定の詳細については，「URL の各項目の説明」の DBHOST の説明を参照してください。

(u)　HiRDB_for_Java_HiRDB_INI

HiRDB.ini ファイル内に記述されている HiRDB クライアント環境変数を有効にする場合に，HiRDB.ini ファイルが存在するディレクトリの絶対パスを指定します。ここで指定したディレクトリの HiRDB.ini ファイ

ルの HiRDB クライアント環境変数が有効になります。指定したディレクトリに HiRDB.ini ファイルがない場合，及び内部ドライバの場合，この指定は無効になります。

また，この指定を省略した場合，URL 中の項目 HiRDB_INI の指定値が仮定されます。URL 中に指定がない場合は，このファイルの内容を無視します。

(v) HiRDB_for_Java_BATCHEXCEPTION_BEHAVIOR

java.sql.BatchUpdateException の getUpdateCounts メソッドの戻り値に，JDBC 規格に準拠した更新カウントを設定するかどうかを指定します。

なお，このプロパティの指定は，接続先がバージョン 08-02 以降の HiRDB の場合に有効です。

TRUE：JDBC 規格に準拠した更新カウントを設定します。

FALSE：HiRDB 独自の更新カウントを設定します。

このプロパティは，指定した内容について大文字と小文字を区別しません。

また，この指定を省略した場合，URL 中の項目 BATCHEXCEPTION_BEHAVIOR の指定値が有効になります。URL 中に指定がない場合は，TRUE が指定されたものとして動作します。

(w) HiRDB_for_Java_UPDATECOUNT_BEHAVIOR

更新系 SQL 発行時の更新行数が 0 の場合，java.sql.Statement の getUpdateCount メソッドの戻り値として-1 を返却するかどうかを指定します。

TRUE：更新系 SQL 発行時の更新行数が 0 の場合，-1 を返却します（JDBC 規格）。

FALSE：更新系 SQL 発行時の更新行数が 0 の場合，0 を返却します。

このプロパティは，指定した内容について大文字と小文字を区別しません。

また，この指定を省略した場合，ほかの設定方法で指定した値が有効になります。ほかの設定方法については，「接続情報の優先順位」を参照してください。値が指定されていない場合は，FALSE が指定されたものとして動作します。

17.3　DataSource と JNDI を使用した DB 接続

17.3.1　DataSource と JNDI を使用した DB 接続の手順

DataSource と JNDI を使用した DB 接続は，JDBC2.0 Optional Package で使用できるようになりました。

必ずしも JNDI を使用する必要はありませんが，JNDI を使用することで接続情報の設定が 1 回で済むというメリットがあります。DataSource クラスのインタフェース定義，及び JNDI は，JDK に標準で含まれていないため，AP 開発をする場合には，JavaSoft の Web サイトから入手する必要があります。

DataSource と JNDI を使用した DB 接続の手順を次に示します。

1. DataSource オブジェクトの生成
2. 接続情報の設定
3. JNDI への DataSource の登録
4. JNDI からの DataSource の取得
5. DB 接続

JNDI を使用しない場合は，3 及び 4 の操作をする必要はありません。

JNDI を使用する場合，1～3 の操作は 1 回だけ実行します。その後，4 及び 5 の操作をするだけで，DB 接続ができます。また，4 の操作の後，必要に応じて接続情報を変更できます。

(1)　DataSource オブジェクトの生成

JDBC ドライバが提供する，DataSource クラスのオブジェクトを生成します。

DataSource クラスのオブジェクト生成で必要となる，JDBC ドライバの DataSource クラス名は **PrdbDataSource** となります。

DataSource クラスのオブジェクトの生成例を次に示します。

```
JP.co.Hitachi.soft.HiRDB.JDBC.PrdbDataSource  ds = null ;
ds = new JP.co.Hitachi.soft.HiRDB.JDBC.PrdbDataSource() ;
```

(2)　接続情報の設定

DataSource オブジェクトに対して，接続情報設定用メソッドを呼び出し，接続情報の設定をします。接続情報取得用のメソッドも使用できるため，現在の接続情報の確認もできます。接続情報設定／取得メソッドについては，「接続情報設定／取得インタフェース」を参照してください。

(3) JNDI への DataSource の登録

DataSource オブジェクトを JNDI に登録します。

JNDI は，その実行環境によって幾つかのサービスプロバイダを選択できます。

DataSource オブジェクトの JNDI への登録例を次に示します（Windows の場合の例です）。なお，登録例では，サービスプロバイダの一つである，File System サービスプロバイダを使用しています。ほかのサービスプロバイダについては，JNDI のドキュメントを参照してください。

```
// JDBCドライバが提供するDataSourceクラスのオブジェクトを生成する
JP.co.Hitachi.soft.HiRDB.JDBC.PrdbDataSource  ds;
ds = new JP.co.Hitachi.soft.HiRDB.JDBC.PrdbDataSource();

// 接続情報を設定する
        :
// システムプロパティを取得する
Properties sys_prop = System.getProperties() ;

// File Systemサービスプロバイダのプロパティを設定する
sys_prop.put(Context.INITIAL_CONTEXT_FACTORY,
             ”com.sun.jndi.fscontext.RefFSContextFactory”);
// File Systemサービスプロバイダで使用するディレクトリを設定する
//（この場合，c:¥JNDI_DIRの下に登録される）
sys_prop.put(Context.PROVIDER_URL, ”file:c:¥¥” + ”JNDI_DIR”);

// システムプロパティを更新する
System.setProperties(sys_prop) ;
// JNDIを初期化する
Context ctx = new InitialContext();

// HiRDBドライバが提供するDataSourceクラスのオブジェクトを，
// jdbc/TestDataSourceという論理名称でJNDIに登録する
ctx.bind(”jdbc” + ”¥¥” + ”TestDataSource”, ds);
                       :
```

なお，JDBC2.0 規格では，JNDI に登録する論理名称は，"jdbc"というサブコンテキスト下（登録例では jdbc/TestDataSource）に登録するように推奨されています。

(4) JNDI からの DataSource の取得

JNDI から DataSource オブジェクトを取得します。

JNDI からの DataSource オブジェクトの登録例を次に示します（Windows の場合の例です）。なお，登録例では，サービスプロバイダの一つである，File System サービスプロバイダを使用しています。ほかのサービスプロバイダについては，JNDI のドキュメントを参照してください。

```
// システムプロパティを取得する
Properties sys_prop = System.getProperties() ;

// File Systemサービスプロバイダのプロパティを設定する
sys_prop.put(Context.INITIAL_CONTEXT_FACTORY,
```

```
            "com.sun.jndi.fscontext.RefFSContextFactory");
// File Systemサービスプロバイダで使用するディレクトリを設定する
//（この場合，c:¥JNDI_DIRの下に登録されている）
sys_prop.put(Context.PROVIDER_URL, "file:c:¥¥" + "JNDI_DIR");
// システムプロパティを更新する
System.setProperties(sys_prop) ;

// JNDIを初期化する
Context ctx = new InitialContext();
// jdbc/TestDataSourceという論理名称のオブジェクトをJNDIから取得する
Object obj = ctx.lookup("jdbc" + "¥¥" + "TestDataSource") ;

// 取り出したオブジェクトを，DataSourceクラスの型にキャストする
DataSource ds = (DataSource)obj;
              :
```

(5)　DB 接続

DataSource オブジェクトに対して，getConnection メソッドを呼び出します。

getConnection メソッドの呼び出し例を次に示します。

```
DataSource ds

// JNDIからDataSourceオブジェクトを取得する
                    :
// getConnectionメソッドを発行する
Connection con = ds.getConnection();
  又は
Connection con = ds.getConnection("USERID", "PASSWORD");※
```

注※

メソッドの引数（認可識別子，パスワード）は，DataSource オブジェクトに設定した接続情報よりも優先されます。必要な接続情報が DataSource オブジェクトに設定されていない場合，接続情報の内容が不正な場合，又は HiRDB サーバとの接続に失敗した場合，getConnection メソッドは SQLException を投入します。

JNDI から DataSource オブジェクトを取得後，必要に応じて接続情報を再度設定できます。この場合，DataSource オブジェクトを，JDBC ドライバが提供する DataSource クラスの型にキャストしてから設定する必要があります。例を次に示します。

```
DataSource ds
JP.co.Hitachi.soft.HiRDB.JDBC.PrdbDataSource  hirdb_ds;

// JNDIからDataSourceオブジェクトを取得する
                    :
// DataSourceオブジェクトを，JDBCドライバが提供する
// DataSourceクラスの型にキャストする
dbp_ds = (JP.co.Hitachi.soft.HiRDB.JDBC.PrdbDataSource)ds;
```

```
// 接続情報を再設定する
        :
```

17.4　JDBC1.2 コア API

17.4.1　Driver インタフェース

(1)　概要

Driver インタフェースでは，主に次の機能が提供されます。

- DB 接続
- 指定した URL の妥当性チェック
- DriverManager.getConnection メソッドで指定する接続プロパティの情報取得
- ドライババージョンの返却

(2)　メソッド

Driver インタフェースのメソッド一覧を次の表に示します。なお，表に記載されていないメソッドはサポートしていません。サポートしていないメソッドを指定すると，SQLException を投入します。

表 17–9　Driver インタフェースのメソッド一覧

メソッド	提供ドライバ			機能
	JDBC2.0	JDBC4.0	JDBC4.3	
acceptsURL(String url)	○	○	○	ドライバが指定された URL に接続できるかどうかを確認します。
connect(String url, Properties info)	○	○	○	指定された URL にデータベース接続を試みます。
getMajorVersion()	○	○	○	ドライバのメジャーバージョンを取得します。
getMinorVersion()	○	○	○	ドライバのマイナーバージョンを取得します。
getPropertyInfo(String url, Properties info)	○	○	○	ドライバの有効なプロパティについての情報を取得します。
jdbcCompliant()	○	○	○	ドライバが JDBC Compliant™ であるかどうかを通知します。

（凡例）

○：提供されます。

(a) acceptsURL(String url)

【機能】

ドライバが指定された URL に接続できるかどうかを確認します。

【形式】

```
public boolean acceptsURL(String url) throws SQLException
```

【引数】

String url：

データベースの URL

【戻り値】

ドライバが指定された URL を認識する場合は true，そうでない場合は false を返します。

【発生する例外】

なし。

(b) connect(String url, Properties info)

【機能】

指定された URL にデータベース接続を試みます。

【形式】

```
public Connection connect(String url, Properties info) throws SQLException
```

【引数】

String url：

接続先のデータベースの URL。詳細については，「URL の構文」を参照してください。

Properties info：

接続引数としてのプロパティ名称，及び値のペアのリスト。詳細については，「ユーザプロパティ」を参照してください。

【戻り値】

URL への接続を表す Connection オブジェクト

【機能詳細】

URL で示すデータベースと接続します。

このメソッドは，DriverManager.getLoginTimeout が返す値を，HiRDB サーバとの接続時に行う通信の最大待ち時間として使用します。

getLoginTimeout が 0 を返す場合，クライアント環境定義 PDCONNECTWAITTIME で指定した値が，最大待ち時間になります。

なお，待ち時間は DriverManager.setLoginTimeout で指定できます。

【発生する例外】

次の場合，SQLException を投入します。

- データベースアクセスエラーが発生した場合
- 指定した接続情報が不正である場合
- DriverManager.getLoginTimeout の返却値が 0〜300 の範囲外となる場合

(c) getMajorVersion()

【機能】

ドライバのメジャーバージョンを取得します。

【形式】

```
public synchronized int getMajorVersion()
```

【引数】

なし。

【戻り値】

このドライバのメジャーバージョン番号

【発生する例外】

なし。

(d) getMinorVersion()

【機能】

ドライバのマイナーバージョンを取得します。

【形式】

```
public synchronized int getMinorVersion()
```

【引数】

なし。

【戻り値】

このドライバのマイナーバージョン番号

【発生する例外】

なし。

(e) getPropertyInfo(String url, Properties info)

【機能】

このドライバの有効なプロパティについての情報を取得します。

【形式】

```
public synchronized DriverPropertyInfo[] getPropertyInfo(String url, Properties info) thr
ows SQLException
```

【引数】

String url：

接続先のデータベースの URL

Properties info：

接続引数としてのプロパティ名称，及び値のペアのリスト

【戻り値】

有効なプロパティを記述する DriverPropertyInfo オブジェクトの配列（プロパティが必要ない場合は，この配列は空になることもあります）

DriverPropertyInfo の各フィールドの設定値を次の表に示します。

表 17–10　DriverPropertyInfo の各フィールドの設定値

プロパティ名	DriverPropertyInfo フィールド				
	name	value	description	required	choices
user	プロパティ名と同じ	null	"UserID"	false	null
password	同上	""	"Password"	false	null
UAPNAME	同上	""	"UAPNAME"	false	null
JDBC_IF	同上	"OFF"	"JDBC Interface Trace"	false	{"ON","OFF"}
TRC_NO	同上	"500"	"Trace Entry Number"	false	null
ENCODELANG	同上	null	"Encode Lang"	false	null
HIRDB_CURSOR	同上	"FALSE"	"HiRDB Cursor across commit"	false	null
LONGVARBINARY_ACCESS	同上	"REAL"	"Longvarbinary locator access"	false	null
HiRDB_for_Java_SQL_IN_NUM	同上	"300"	"SQL In Number"	false	null
HiRDB_for_Java_SQL_OUT_NUM	同上	"300"	"SQL Out Number"	false	null
HiRDB_for_Java_SQLWARNING_LEVEL	同上	"SQLWARN"	"SQL Warning Level"	false	null
HiRDB_for_Java_ENV_VARIABLES	同上	null	"HiRDB Environment Variables"	false	null

プロパティ名	DriverPropertyInfo フィールド				
	name	value	description	required	choices
HiRDB_for_Java_STATEMENT_COMMIT_BEHAVIOR	同上	"TRUE"	"HiRDB Statement across commit"	false	{"TRUE","FALSE"}
HiRDB_for_Java_LONGVARBINARY_ACCESS_SIZE	同上	0	"Longvarbinary locator access size"	false	null
HiRDB_for_Java_MAXBINARYSIZE	同上	null	"Longvarbinary maximum binary size"	false	null
HiRDB_for_Java_LONGVARBINARY_TRUNCERROR	同上	"TRUE"	"Longvarbinary truncate error"	false	{"TRUE","FALSE"}
HiRDB_for_Java_DBID	同上	null	"Port number of HiRDB server or Environment variable group of HiRDB"	false	null
HiRDB_for_Java_DBHOST	同上	null	"Host name with HiRDB"	false	null
HiRDB_for_Java_HiRDB_INI	同上	null	"HiRDB.ini file "	false	null
HiRDB_for_Java_BATCHEXCEPTION_BEHAVIOR	同上	"TRUE"	"BatchUpdateException UpdateCounts that conforms to JDBC standard"	false	{"TRUE","FALSE"}
SQLWARNING_IGNORE	同上	"FALSE"	"Warning generated by the Connection object is not maintained with the Connection object"	false	{"TRUE","FALSE"}
HiRDB_for_Java_STATEMENT_CLOSE_BEHAVIOR	同上	"FALSE"	"HiRDB Statement close behavior"	false	{"TRUE","FALSE"}
HiRDB_for_Java_UPDATECOUNT_BEHAVIOR	同上	"FALSE"	"Statement UpdateCounts that conforms to JDBC standard"	false	{"TRUE","FALSE"}

【機能詳細】

引数 url，info に指定された情報を解析し，データベースに接続するための情報を返します。

なお，acceptsURL(String url)が false となる場合，このメソッドは null を返します。

【発生する例外】

なし。

(f)　jdbcCompliant()

【機能】

このドライバが JDBC Compliant™ であるかどうかを通知します。

【形式】

```
public synchronized boolean jdbcCompliant()
```

【引数】

なし。

【戻り値】

ドライバが JDBC Compliant の場合は true，そうでない場合は false を返します。

【発生する例外】

なし。

(3)　パッケージ名称及びクラス名称

このインタフェースを実装するパッケージ名称とクラス名称を次に示します。

パッケージ名称：**JP.co.Hitachi.soft.HiRDB.JDBC**

クラス名称：**HiRDBDriver**

(4)　エスケープ句

SQL 文中で｛｝で囲まれた部分をエスケープ句と呼びます。エスケープ句は一つのキーワードと複数のパラメタで構成されます。キーワードの大文字と小文字は区別しません。

エスケープ句の一覧を次の表に示します。

表 17–11　エスケープ句の一覧

エスケープ句の種別	キーワード
日付，時刻，時刻印	d，t，ts
LIKE エスケープ文字	escape
外部結合	oj
プロシジャ呼び出し	call
スカラ関数	fn
代入	set

エスケープ句で指定できるスカラ関数については，「エスケープ句で指定できるスカラ関数」を参照してください。

＜エスケープ構文の解析＞

エスケープ構文の解析を有効にするかどうかは，Statement クラスの setEscapeProcessing メソッドで指定します。指定がない場合は，有効となります。エスケープ構文の解析が有効な場合，JDBC ドライバは SQL 文内にエスケープ句がないか解析します。SQL 文内にエスケープ句があった場合は，HiRDB が実行可能な SQL 文に変換します。

17.4.2　Connection インタフェース

(1)　概要

Connection クラスでは，主に次の機能が提供されます。

- Statement クラス，PreparedStatement クラス，及び CallableStatement クラスのオブジェクト生成
- トランザクションの決着（コミット又はロールバック）
- AUTO コミットモードの設定

(2)　メソッド

Connection インタフェースのメソッド一覧を次の表に示します。なお，表に記載されていないメソッドはサポートしていません。サポートしていないメソッドを指定すると，SQLException を投入します。

表 17–12　Connection インタフェースのメソッド一覧

メソッド	提供ドライバ			機能
	JDBC2.0	JDBC4.0	JDBC4.3	
clearWarnings()	○	○	○	この Connection オブジェクトに関して通知されたすべての警告をクリアします。
close()	○	○	○	HiRDB との接続を切断します。
commit()	○	○	○	直前のコミット又はロールバック以降に行われた変更を，すべて有効とします。
createStatement()	○	○	○	SQL 文をデータベースに送るための Statement オブジェクトを生成します。
createStatement(int resultSetType, int resultSetConcurrency)	○	○	○	SQL 文をデータベースに送るための Statement オブジェクトを生成します。
createStatement(int resultSetType, int resultSetConcurrency, int resultSetHoldability)	○	○	○	SQL 文をデータベースに送るための Statement オブジェクトを生成します。

メソッド	提供ドライバ			機能
	JDBC2.0	JDBC4.0	JDBC4.3	
getAutoCommit()	○	○	○	この Connection オブジェクトでの現在の自動コミットモードを取得します。
getCatalog()	○	○	○	この Connection オブジェクトの現在のカタログ名を取得します。
getHoldability()	○	○	○	この Connection オブジェクトを使用して生成される ResultSet オブジェクトの，現在の保持機能を取得します。
getMetaData()	○	○	○	DatabaseMetaData オブジェクトを生成します。
getTransactionIsolation()	○	○	○	この Connection オブジェクトの現在のトランザクション遮断レベルを取得します。
getTypeMap()	○	○	○	この Connection に関連した Map オブジェクトを取得します。
getWarnings()	○	○	○	この Connection オブジェクトに関する呼び出しによって報告される警告を，SQLWarning オブジェクトとして取得します。
isClosed()	○	○	○	この Connection オブジェクトがクローズされているかどうかを取得します。
isReadOnly()	○	○	○	この Connection オブジェクトが読み込み専用モードかどうかを取得します。
nativeSQL(String sql)	○	○	○	指定された SQL 文内のエスケープ句を，HiRDB が実行できる形式に変換して返します。
prepareCall(String sql)	○	○	○	CALL 文を実行するための CallableStatement オブジェクトを生成します。
prepareCall(String sql, int resultSetType, int resultSetConcurrency)	○	○	○	CALL 文を実行するための CallableStatement オブジェクトを生成します。
prepareCall(String sql, int resultSetType, int resultSetConcurrency, int resultSetHoldability)	○	○	○	CALL 文を実行するための CallableStatement オブジェクトを生成します。
prepareStatement(String sql)	○	○	○	パラメタ付き SQL 文をデータベースに送るための PreparedStatement オブジェクトを生成します。

メソッド	提供ドライバ			機能
	JDBC2.0	JDBC4.0	JDBC4.3	
prepareStatement(String sql, int resultSetType, int resultSetConcurrency)	○	○	○	パラメタ付き SQL 文をデータベースに送るための PreparedStatement オブジェクトを生成します。
prepareStatement(String sql, int resultSetType, int resultSetConcurrency, int resultSetHoldability)	○	○	○	パラメタ付き SQL 文をデータベースに送るための PreparedStatement オブジェクトを生成します。
rollback()	○	○	○	現在のトランザクションで行われた変更をすべて元に戻し，この Connection オブジェクトが現在保持するデータベースロックをすべて解除します。
setAutoCommit(boolean autoCommit)	○	○	○	この接続の自動コミットモードを，指定された状態に設定します。
setCatalog(String catalog)	○	○	○	HiRDB にはカタログが存在しないため，このドライバはメソッドの指定値を無視します。
setHoldability(int holdability)	○	○	○	この Connection オブジェクトを使用して生成される ResultSet オブジェクトの保持機能を，指定された保持機能へ変更します。
setReadOnly(boolean readOnly)	○	○	○	HiRDB には専用モードがないため，メソッドの指定値を無視します。
setTransactionIsolation(int level)	○	○	○	HiRDB では常に TRANSACTION_REPEATABLE_READ のため，このメソッドの指定値を無視します。
checkSession(int waittime)	○	○	○	現在の接続状態を確認します。
setHiRDB_Audit_Info(int pos，String userinf)	○	○	○	ユーザ任意の接続情報（ユーザ付加情報）を設定します。
isValid(int timeout)	×	○	○	現在の接続状態を確認します。
setSchema(String schema)	×	×	○	SQL 実行時，認可識別子を省略した場合に仮定する認可識別子を設定します。
getSchema()	×	×	○	この Connection オブジェクトで現在有効となっている SQL 実行時に認可識別子を省略した場合に仮定する認可識別子を返却します。
beginRequest	×	×	○	コネクションの使用開始を JDBC ドライバに通知します。

メソッド	提供ドライバ			機能
	JDBC2.0	JDBC4.0	JDBC4.3	
endRequest	×	×	○	コネクションの使用終了を JDBC ドライバに通知します。

（凡例）

○：提供されます。

×：提供されません。

(a)　clearWarnings()

【機能】

この Connection オブジェクトに関して通知されたすべての警告をクリアします。

このメソッドを呼び出した後，この Connection オブジェクトに対する新しい警告が報告されるまで，getWarnings()は null を返します。

【形式】

```
public synchronized void clearWarnings() throws SQLException
```

【引数】

なし。

【戻り値】

なし。

【発生する例外】

この Connection オブジェクトが close されている場合，SQLException を投入します。

(b)　close()

【機能】

HiRDB との接続を切断します。

【形式】

```
public synchronized void close() throws SQLException
```

【引数】

なし。

【戻り値】

なし。

【機能詳細】

通常接続時は，HiRDB との接続を切断するとともに，該当するオブジェクトを無効にし，不要なリソースを解放します。

プーリング環境下，及び XA 環境下では，物理的な接続の切断をしません。この場合，PooledConnection.close()で物理的な接続の切断をします。

Conneciton.close()の実行でエラーが発生した場合は，SQLException を throw しません。

プーリング環境下，及び XA 環境下での Connection.close()の実行で，致命的なエラーが発生しコネクションプーリングが使用できなくなった場合，ConnectionEventListener.connectionErrorOccurred()は発生しません。

既にクローズされた Connection オブジェクトで close メソッドを呼び出すと，このメソッドは何もしません。

【発生する例外】

なし。

(c) commit()

【機能】

直前のコミット又はロールバック以降に行われた変更を，すべて有効とします。

自動コミットモードを有効にしている状態でこのメソッドを呼び出しても，例外を投入しないで commit 処理を行います。

【形式】

```
public synchronized void commit() throws SQLException
```

【引数】

なし。

【戻り値】

なし。

【発生する例外】

次の場合，SQLException を投入します。

- データベースアクセスエラーが発生した場合
- Connection オブジェクトに対して close()が既に発行されている場合

(d) createStatement()

【機能】

SQL 文をデータベースに送るための Statement オブジェクトを生成します。

【形式】

```
public synchronized Statement createStatement() throws SQLException
```

【引数】

なし。

【戻り値】

Statement オブジェクト

【機能詳細】

SQL 文をデータベースに送るための Statement オブジェクトを生成します。

このメソッドで生成した Statement オブジェクトから生成される ResultSet の保持機能は，Connection.setHoldability で指定されている値になります。Connection.setHoldability を一度も実行していない場合は，プロパティ HIRDB_CURSOR の設定値になります。

【発生する例外】

次の場合，SQLException を投入します。

- Connection オブジェクトに対して close()が既に発行されている場合
- Statement オブジェクト生成でエラーが発生した場合

(e) createStatement(int resultSetType, int resultSetConcurrency)

【機能】

SQL 文をデータベースに送るための Statement オブジェクトを生成します。

【形式】

```
public synchronized Statement createStatement(int resultSetType, int resultSetConcurrency
) throws SQLException
```

【引数】

int resultSetType：

結果集合の型

int resultSetConcurrency：

並行処理モード

【戻り値】

Statement オブジェクト

【機能詳細】

SQL 文をデータベースに送るための Statement オブジェクトを生成します。

結果集合の型に TYPE_SCROLL_SENSITIVE を指定した場合，このドライバは TYPE_SCROLL_INSENSITIVE に切り替え，SQLWarning を設定します。

並行処理モードは CONCUR_READ_ONLY だけサポートします。CONCURE_UPDATABLE を指定した場合，CONCUR_READ_ONLY に切り替え，SQLWarning を設定します。

このメソッドで生成した Statement オブジェクトから生成される ResultSet の保持機能は，Connection.setHoldability で指定されている値になります。Connection.setHoldability を一度も実行していない場合は，プロパティ HIRDB_CURSOR の設定値になります。

【発生する例外】

次の場合，SQLException を投入します。

- Connection オブジェクトに対して close()が既に発行されている場合
- Statement オブジェクト生成でエラーが発生した場合
- 結果集合の型に ResultSet 定数以外を指定した場合
- 並行処理モードに ResultSet 定数以外を指定した場合

(f) createStatement(int resultSetType, int resultSetConcurrency, int resultSetHoldability)

【機能】

SQL 文をデータベースに送るための Statement オブジェクトを生成します。

【形式】

```
public synchronized Statement createStatement(int resultSetType, int resultSetConcurrency
, int resultSetHoldability) throws SQLException
```

【引数】

int resultSetType：

結果集合の型

int resultSetConcurrency：

並行処理モード

int resultSetHoldability：

ResultSet の保持機能

【戻り値】

Statement オブジェクト

【機能詳細】

SQL 文をデータベースに送るための Statement オブジェクトを生成します。

結果集合の型に TYPE_SCROLL_SENSITIVE を指定した場合，このドライバは TYPE_SCROLL_INSENSITIVE に切り替え，SQLWarning を設定します。

並行処理モードは CONCUR_READ_ONLY だけサポートします。CONCURE_UPDATABLE を指定した場合，CONCUR_READ_ONLY に切り替え，SQLWarning を設定します。

【発生する例外】

次の場合，SQLException を投入します。

- Connection オブジェクトに対して close()が既に発行されている場合
- Statement オブジェクト生成でエラーが発生した場合
- 結果集合の型に ResultSet 定数以外を指定した場合
- 並行処理モードに ResultSet 定数以外を指定した場合
- ResultSet の保持機能に ResultSet 定数以外を指定した場合

(g) getAutoCommit()

【機能】

この Connection オブジェクトでの現在の自動コミットモードを取得します。

【形式】

```
public synchronized boolean getAutoCommit() throws SQLException
```

【引数】

なし。

【戻り値】

Connection オブジェクトでの現在の自動コミットモードの状態

【発生する例外】

この Connection オブジェクトに対して close()が既に発行されている場合，SQLException を投入します。

(h) getCatalog()

【機能】

この Connection オブジェクトの現在のカタログ名を取得します。

このメソッドの戻り値は，常に null です。

【形式】

```
public synchronized String getCatalog() throws SQLException
```

【引数】

なし。

【戻り値】

HiRDB にはカタログが存在しないため，常に null を返します。

【発生する例外】

この Connection オブジェクトに対して close()が既に発行されている場合，SQLException を投入します。

(i) getHoldability()

【機能】

この Connection オブジェクトを使用して生成される ResultSet オブジェクトの，現在の保持機能を取得します。

このメソッドの戻り値が，Statement（PreparedStatement）オブジェクト生成時に保持機能を指定しなかった場合の，ResultSet オブジェクトの保持機能になります。

【形式】

```
public synchronized int getHoldability() throws SQLException
```

【引数】

なし。

【戻り値】

ResultSet の型。次のどちらかです。

- ResultSet.HOLD_CURSORS_OVER_COMMIT
- ResultSet.CLOSE_CURSORS_AT_COMMIT

【発生する例外】

この Connection オブジェクトに対して close()が既に発行されている場合，SQLException を投入します。

(j)　getMetaData()

【機能】

DatabaseMetaData オブジェクトを生成します。

【形式】

```
public synchronized DatabaseMetaData getMetaData() throws SQLException
```

【引数】

なし。

【戻り値】

DatabaseMetaData オブジェクト

【発生する例外】

この Connection オブジェクトに対して close()が既に発行されている場合，SQLException を投入します。

(k)　getTransactionIsolation()

【機能】

この Connection オブジェクトの現在のトランザクション遮断レベルを取得します。

常に TRANSACTION_REPEATABLE_READ を返します。

【形式】

```
public synchronized int getTransactionIsolation() throws SQLException
```

【引数】

なし。

【戻り値】

常に TRANSACTION_REPEATABLE_READ を返します。

【発生する例外】

この Connection オブジェクトに対して close()が既に発行されている場合，SQLException を投入します。

(l)　getTypeMap()

【機能】

この Connection に関連した Map オブジェクトを取得します。

このドライバは，情報を格納していない空の java.util.HashMap オブジェクトを返します。

【形式】

```
public synchronized java.util.Map getTypeMap() throws SQLException
```

【引数】

なし。

【戻り値】

空の java.util.HashMap オブジェクト

【発生する例外】

この Connection オブジェクトに対して close()が既に発行されている場合，SQLException を投入します。

(m)　getWarnings()

【機能】

この Connection オブジェクトに関する呼び出しによって報告される警告を，SQLWarning オブジェクトとして取得します。

【形式】

```
public synchronized SQLWarning getWarnings() throws SQLException
```

【引数】

なし。

【戻り値】

最初の SQLWarning オブジェクト（ない場合は null を返します）

【機能詳細】

該当する Connection オブジェクトが保持する SQLWarning オブジェクトを取得します。

取得した SQLWarning オブジェクトの getNextWarning メソッドを実行することによって，二つ目以降の警告を取得できます。

【発生する例外】

この Connection オブジェクトに対して close()が既に発行されている場合，SQLException を投入します。

(n) isClosed()

【機能】

この Connection オブジェクトがクローズされているかどうかを取得します。

【形式】

```
public synchronized boolean isClosed()
```

【引数】

なし。

【戻り値】

この Connection オブジェクトがクローズされている場合は true，まだクローズされていない場合は false を返します。

【機能詳細】

この Connection オブジェクトがクローズされているかどうかを取得します。データベースへの接続は，close メソッドが呼び出されるか，特定の致命的エラーが発生した場合に切断します。
Connection.close メソッドが呼び出された後に，このメソッドが呼び出された場合だけ，true を返すことを保証します。データベースへの接続が，有効か無効かを判定するために使用することはできません。

【発生する例外】

なし。

(o) isReadOnly()

【機能】

この Connection オブジェクトが読み込み専用モードかどうかを取得します。

このドライバでは常に false を返します。

【形式】

```
public synchronized boolean isReadOnly() throws SQLException
```

【引数】

なし。

【戻り値】

常に false を返します。

【発生する例外】

この Connection オブジェクトに対して close()が既に発行されている場合，SQLException を投入します。

(p)　nativeSQL(String sql)

【機能】

指定された SQL 文内のエスケープ句を，HiRDB が実行できる形式に変換して返します。

【形式】

```
public synchronized String nativeSQL(String sql) throws SQLException
```

【引数】

String sql：

変換前の SQL 文

【戻り値】

HiRDB が実行できる SQL 文（引数 sql が NULL の場合，NULL を返します。引数 sql が空文字の場合，空文字を返します）

【機能詳細】

指定された SQL 文内のエスケープ句を，HiRDB が実行できる形式に変換して返します。

エスケープ句の構文規則を次に示します。

```
エスケープ句 ::= 日付・時刻・時刻印のエスケープシーケンス
               | LIKEのエスケープシーケンス
               | 外部結合のエスケープシーケンス
               | プロシジャ呼び出しのエスケープシーケンス
               | スカラ関数のエスケープシーケンス
               | 代入のエスケープシーケンス

日付・時刻・時刻印のエスケープシーケンス ::= 日付のエスケープシーケンス
                                            | 時刻のエスケープシーケンス
                                            | 時刻印のエスケープシーケンス

日付のエスケープシーケンス ::=
          エスケープ開始子 d 日付データの既定の文字列表現 エスケープ終了子

時刻のエスケープシーケンス ::=
          エスケープ開始子 t 時刻データの既定の文字列表現 エスケープ終了子

時刻印のエスケープシーケンス ::=
          エスケープ開始子 ts 時刻印データの既定の文字列表現 エスケープ終了子

LIKEのエスケープシーケンス ::=
          エスケープ開始子 escape エスケープ文字 エスケープ終了子

外部結合のエスケープシーケンス ::= エスケープ開始子 oj 結合表 エスケープ終了子

プロシジャ呼び出しのエスケープシーケンス ::=
          エスケープ開始子 call 〔認可識別子.〕ルーチン識別子
```

〔（〔引数〔，引数〕…〕）〕 エスケープ終了子

代入のエスケープシーケンス ::= エスケープ開始子 set 代入文 エスケープ終了子

スカラ関数のエスケープシーケンス ::= エスケープ開始子 fn スカラ関数 エスケープ終了子
スカラ関数 ::= 標準形式のスカラ関数※

エスケープ開始子 ::= '{'
エスケープ終了子 ::= '}'

注※

標準形式のスカラ関数については，「エスケープ句で指定できるスカラ関数」を参照してください。

下線部については，マニュアル「HiRDB SQL リファレンス」を参照してください。なお，下線部には，エスケープ句を指定できません。また，JDBC ドライバでは構文解析をしないで，変換後もそのままとし，HiRDB サーバの構文解析に任せます。

エスケープシーケンスのキーワードを次に示します。キーワードについては，大文字と小文字を区別しません。

1. 日付のエスケープシーケンス内の"d"
2. 時刻のエスケープシーケンス内の"t"
3. 時刻印のエスケープシーケンス内の"ts"
4. LIKE のエスケープシーケンス内の"escape"
5. 外部結合のエスケープシーケンス内の"oj"
6. プロシジャ呼び出しのエスケープシーケンス内の"call"
7. スカラ関数のエスケープシーケンス内の"fn"
8. 代入のエスケープシーケンス内の"set"

エスケープ句の入力規則を次に示します。

- エスケープ句内の区切り文字は，半角空白が指定できます。
- 区切り文字は，エスケープ開始子の後ろ，キーワードの後ろ，及びエスケープ終了子の前に挿入できます。
- 区切り文字は，プロシジャ呼び出しのエスケープシーケンス内にある"call"の後ろに挿入してください。
- 一つの SQL 文中に，複数のエスケープ句を指定できます。
- 注釈（「/*」から「*/」までの文字）内の{}，アポストロフィ（'），又は引用符（"）で囲まれた{}は，エスケープ句と見なされません。
- ドライバは指定された SQL 文内のエスケープ句を，HiRDB が実行できる形式に変換します。変換するのは，{}で囲まれたエスケープ句内だけです。エスケープ句外は，何も変換しません。

エスケープ句の変換規則を次の表に示します。

表 17–13　エスケープ句の変換規則

対象エスケープ句	変換前	変換後
日付	エスケープ開始子 d 日付データの既定の文字列表現 エスケープ終了子	日付データの既定の文字列表現
時刻	エスケープ開始子 t 時刻データの既定の文字列表現 エスケープ終了子	時刻データの既定の文字列表現
時刻印	エスケープ開始子 ts 時刻印データの既定の文字列表現 エスケープ終了子	時刻印データの既定の文字列表現
LIKE	エスケープ開始子 escape エスケープ文字 エスケープ終了子	escape エスケープ文字
外部結合	エスケープ開始子 oj 結合表 エスケープ終了子	結合表
プロシジャ呼び出し	エスケープ開始子 call 〔認可識別子.〕ルーチン識別子〔(〔引数〔，引数〕…〕)〕 エスケープ終了子	HiRDB 接続の場合 call〔認可識別子.〕ルーチン識別子（〔引数〔，引数〕…〕）※1 XDM/RD E2 接続の場合 call〔認可識別子.〕ルーチン識別子〔(〔引数〔，引数〕…〕)〕
スカラ関数	エスケープ開始子 fn スカラ関数 エスケープ終了子	HiRDB 形式のスカラ関数※2
代入	エスケープ開始子 set 代入文 エスケープ終了子	set 代入文

注※1

ルーチン識別子の後ろの()が省略されている場合，()を付加します。

注※2

変換処理の詳細は，次のスカラ関数のエスケープ句の変換処理を参照してください。

標準形式のスカラ関数を，HiRDB 形式に変換します。XDM/RD E2 接続の場合は，XDM/RD E2 形式に変換します。

標準形式に対応する HiRDB のスカラ関数が，HiRDB のシステム定義スカラ関数の場合，関数名の先頭に"MASTER."を付加します。

基本的に，スカラ関数の引数の個数チェックはしません。ただし，スカラ関数名が LOCATE の場合は，引数の区切りを示すコンマを IN，FROM に変換するため，引数の個数チェックをします。

標準形式と HiRDB 形式が異なるスカラ関数の変換内容を次の表に示します。

表 17–14　標準形式と HiRDB 形式が異なるスカラ関数の変換内容一覧

スカラ関数	変換前の形式	変換後の形式（HiRDB 形式）
数学関数	CEILING(number)	MASTER.CEIL(number)
	LOG(float)	MASTER.LN(float)
	TRUNCATE(number, places)	MASTER.TRUNC(number, places)

スカラ関数	変換前の形式	変換後の形式（HiRDB 形式）
文字列関数	CHAR(code)	MASTER.CHR(code)
	INSERT(string1, start, length, string2)	MASTER.INSERTSTR(string1, start, length, string2)
	LCASE(string)	LOWER(string)
	LEFT(string, count)	MASTER.LEFTSTR(string, count)
	LOCATE(string1, string2[, start])	POSITION(string1 IN string2 [FROM start])
	RIGHT(string, count)	MASTER.RIGHTSTR(string, count)
	SUBSTRING(string, start, length)	SUBSTR(string, start, length)
	UCASE(string)	UPPER(string)
時刻と日付の関数	CURDATE()	CURRENT DATE
	CURRENT_DATE()	CURRENT DATE
	CURTIME()	CURRENT TIME
	CURRENT_TIME()	CURRENT TIME
	CURRENT_TIME(time-precision)	CURRENT TIME
	NOW()	CURRENT TIMESTAMP(6)
システム関数	USER()	USER

標準形式と XDM/RD E2 形式が異なるスカラ関数の変換内容を次の表に示します。

表 17–15　標準形式と XDM/RD E2 形式が異なるスカラ関数の変換内容一覧

スカラ関数	変換前の形式	変換後の形式（XDM/RD E2 形式）
数学関数	LOG(float)	LN(float)
文字列関数	LCASE(string)	LOWER(string)
	LOCATE(string1, string2[, start])	POSITION(string1 IN string2 [FROM start])
	LTRIM(string)	TRIM(LEADING FROM string)
	RTRIM(string)	TRIM(TRAILING FROM string)
	SUBSTRING(string, start, length)	SUBSTR(string, start, length)
	UCASE(string)	UPPER(string)
時刻と日付の関数	CURDATE()	CURRENT DATE
	CURRENT_DATE()	CURRENT DATE
	CURTIME()	CURRENT TIME
	CURRENT_TIME()	CURRENT TIME
	NOW()	CURRENT TIMESTAMP(6)

スカラ関数	変換前の形式	変換後の形式（XDM/RD E2 形式）
システム関数	USER()	USER

HiRDB 接続の場合の変換例を次に示します。

スカラ関数		変換前	変換後
標準形式と HiRDB 形式が同じスカラ関数	システム組み込みスカラ関数	**{fn** ABS(number)**}**	ABS(number)
	システム定義スカラ関数	**{fn** ASCII(string)**}**	**MASTER.**ASCII(string)
標準形式と HiRDB 形式が異なるスカラ関数	システム組み込みスカラ関数	**{fn** UCASE(string)**}**	**UPPER**(string)
	システム定義スカラ関数	**{fn** CEILING(number)**}**	**MASTER.CEIL**(number)

【発生する例外】

次の場合，SQLException を投入します。

- この Connection オブジェクトに対して close()が既に発行されている場合
- 指定された SQL のエスケープ句の形式が不正である場合
  - "{"及びキーワードはあるが，"}"がない。
  - {call の後ろにプロシジャ名称がない。
  - {call とプロシジャ名称の間に空白がない。
  - {call プロシジャ名"("の後ろに")"がない。
  - {? = call を指定している。
  - エスケープ句に LOCATE 関数指定時，引数の指定がない。
  - エスケープ句に LOCATE 関数指定時，()内の引数の数が不正である。
  - XDM/RD E2 接続の場合，エスケープ句に LTRIM 又は RTRIM 関数を指定したとき，引数の指定がない。

(q)　prepareCall(String sql)

【機能】

CALL 文を実行するための CallableStatement オブジェクトを生成します。

【形式】

```
public CallableStatement prepareCall(String sql) throws SQLException
```

【引数】

String sql：

実行する SQL 文（CALL 文以外も指定できます）

【戻り値】

CallableStatement オブジェクト

【機能詳細】

CALL 文を実行するための CallableStatement オブジェクトを生成します。

このメソッドで生成した CallableStatement オブジェクトから生成される ResultSet の保持機能は，Connection.setHoldability で指定されている値になります。Connection.setHoldability を一度も実行していない場合は，プロパティ HIRDB_CURSOR の設定値になります。

【発生する例外】

次の場合，SQLException を投入します。

- この Connection オブジェクトに対して close()が既に発行されている場合
- CallableStatement オブジェクト生成でエラーが発生した場合
- HiRDB SQL Tuning Advisor 用アクセスパス情報ファイルへの出力でエラーが発生した場合（JDBC4.0 以降のときだけ）

(r)　prepareCall(String sql, int resultSetType, int resultSetConcurrency)

【機能】

CALL 文を実行するための CallableStatement オブジェクトを生成します。

【形式】

```
public CallableStatement prepareCall(String sql, int resultSetType, int resultSetConcurrency) throws SQLException
```

【引数】

String sql：

実行する SQL 文（CALL 文以外も指定できます）

int resultSetType：

結果集合の型

int resultSetConcurrency：

並行処理モード

【戻り値】

CallableStatement オブジェクト

【機能詳細】

CALL 文を実行するための CallableStatement オブジェクトを生成します。

結果集合の型に TYPE_SCROLL_SENSITIVE を指定した場合，このドライバは TYPE_SCROLL_INSENSITIVE に切り替え，SQLWarning を設定します。

並行処理モードは CONCUR_READ_ONLY だけサポートします。CONCURE_UPDATABLE を指定した場合，CONCUR_READ_ONLY に切り替え，SQLWarning を設定します。

このメソッドで生成した CallableStatement オブジェクトから生成される ResultSet の保持機能は，Connection.setHoldability で指定されている値になります。Connection.setHoldability を一度も実行していない場合は，プロパティ HIRDB_CURSOR の設定値になります。

【発生する例外】

次の場合，SQLException を投入します。

- この Connection オブジェクトに対して close()が既に発行されている場合
- CallableStatement オブジェクト生成でエラーが発生した場合
- 結果集合の型に ResultSet 定数以外を指定した場合
- 並行処理モードに ResultSet 定数以外を指定した場合
- HiRDB SQL Tuning Advisor 用アクセスパス情報ファイルへの出力でエラーが発生した場合（JDBC4.0 以降のときだけ）

(s) prepareCall(String sql, int resultSetType, int resultSetConcurrency, int resultSetHoldability)

【機能】

CALL 文を実行するための CallableStatement オブジェクトを生成します。

【形式】

```
public CallableStatement prepareCall(String sql, int resultSetType, int resultSetConcurrency, int resultSetHoldability) throws SQLException
```

【引数】

String sql：

実行する SQL 文（CALL 文以外も指定できます）

int resultSetType：

結果集合の型

int resultSetConcurrency：

並行処理モード

int resultSetHoldability：

ResultSet の保持機能

【戻り値】

CallableStatement オブジェクト

【機能詳細】

CALL 文を実行するための CallableStatement オブジェクトを生成します。

結果集合の型に TYPE_SCROLL_SENSITIVE を指定した場合，このドライバは TYPE_SCROLL_INSENSITIVE に切り替え，SQLWarning を設定します。

並行処理モードは CONCUR_READ_ONLY だけサポートします。CONCURE_UPDATABLE を指定した場合，CONCUR_READ_ONLY に切り替え，SQLWarning を設定します。

【発生する例外】

次の場合，SQLException を投入します。

- この Connection オブジェクトに対して close()が既に発行されている場合
- CallableStatement オブジェクト生成でエラーが発生した場合
- 結果集合の型に ResultSet 定数以外を指定した場合
- 並行処理モードに ResultSet 定数以外を指定した場合
- ResultSet の保持機能に ResultSet 定数以外を指定した場合
- HiRDB SQL Tuning Advisor 用アクセスパス情報ファイルへの出力でエラーが発生した場合（JDBC4.0 以降のときだけ）

(t) prepareStatement(String sql)

【機能】

パラメタ付き SQL 文をデータベースに送るための PreparedStatement オブジェクトを生成します。

【形式】

```
public PreparedStatement prepareStatement(String sql) throws SQLException
```

【引数】

String sql：

実行する SQL 文

【戻り値】

PreparedStatement オブジェクト

【機能詳細】

パラメタ付き SQL 文をデータベースに送るための PreparedStatement オブジェクトを生成します。

このメソッドで生成した PreparedStatement オブジェクトから生成される ResultSet の保持機能は，Connection.setHoldability で指定されている値になります。Connection.setHoldability を一度も実行していない場合は，プロパティ HIRDB_CURSOR の設定値になります。

【発生する例外】

次の場合，SQLException を投入します。

- この Connection オブジェクトに対して close()が既に発行されている場合
- PreparedStatement オブジェクト生成でエラーが発生した場合
- HiRDB SQL Tuning Advisor 用アクセスパス情報ファイルへの出力でエラーが発生した場合（JDBC4.0 以降のときだけ）

(u) prepareStatement(String sql, int resultSetType, int resultSetConcurrency)

【機能】

パラメタ付き SQL 文をデータベースに送るための PreparedStatement オブジェクトを生成します。

【形式】

```
public PreparedStatement prepareStatement(String sql, int resultSetType, int resultSetCon
currency) throws SQLException
```

【引数】

String sql：

実行する SQL 文（CALL 文以外も指定できます）

int resultSetType：

結果集合の型

int resultSetConcurrency：

並行処理モード

【戻り値】

PreparedStatement オブジェクト

【機能詳細】

パラメタ付き SQL 文をデータベースに送るための PreparedStatement オブジェクトを生成します。

結果集合の型に TYPE_SCROLL_SENSITIVE を指定した場合，このドライバは TYPE_SCROLL_INSENSITIVE に切り替え，SQLWarning を設定します。

並行処理モードは CONCUR_READ_ONLY だけサポートします。CONCURE_UPDATABLE を指定した場合，CONCUR_READ_ONLY に切り替え，SQLWarning を設定します。

このメソッドで生成した PreparedStatement オブジェクトから生成される ResultSet の保持機能は，Connection.setHoldability で指定されている値になります。Connection.setHoldability を一度も実行していない場合は，プロパティ HIRDB_CURSOR の設定値になります。

【発生する例外】

次の場合，SQLException を投入します。

- この Connection オブジェクトに対して close()が既に発行されている場合
- PreparedStatement オブジェクト生成でエラーが発生した場合
- 結果集合の型に ResultSet 定数以外を指定した場合
- 並行処理モードに ResultSet 定数以外を指定した場合
- HiRDB SQL Tuning Advisor 用アクセスパス情報ファイルへの出力でエラーが発生した場合（JDBC4.0 以降のときだけ）

(v) prepareStatement(String sql, int resultSetType, int resultSetConcurrency, int resultSetHoldability)

【機能】

パラメタ付き SQL 文をデータベースに送るための PreparedStatement オブジェクトを生成します。

【形式】

```
public PreparedStatement prepareStatement(String sql, int resultSetType, int resultSetCon
currency, int resultSetHoldability) throws SQLException
```

【引数】

String sql：

実行する SQL 文（CALL 文以外も指定できます）

int resultSetType：

結果集合の型

int resultSetConcurrency：

並行処理モード

int resultSetHoldability：

ResultSet の保持機能

【戻り値】

PreparedStatement オブジェクト

【機能詳細】

パラメタ付き SQL 文をデータベースに送るための PreparedStatement オブジェクトを生成します。

結果集合の型に TYPE_SCROLL_SENSITIVE を指定した場合は，このドライバは TYPE_SCROLL_INSENSITIVE に切り替え，SQLWarning を設定します。

並行処理モードは CONCUR_READ_ONLY だけサポートします。CONCURE_UPDATABLE を指定した場合，CONCUR_READ_ONLY に切り替え，SQLWarning を設定します。

【発生する例外】

次の場合，SQLException を投入します。

- この Connection オブジェクトに対して close()が既に発行されている場合
- PreparedStatement オブジェクト生成でエラーが発生した場合
- 結果集合の型に ResultSet 定数以外を指定した場合
- 並行処理モードに ResultSet 定数以外を指定した場合
- ResultSet の保持機能に ResultSet 定数以外を指定した場合
- HiRDB SQL Tuning Advisor 用アクセスパス情報ファイルへの出力でエラーが発生した場合（JDBC4.0 以降のときだけ）

(w)　rollback()

【機能】

現在のトランザクションで行われた変更をすべて元に戻し，この Connection オブジェクトが現在保持するデータベースロックをすべて解除します。

自動コミットモードを有効にしている状態で，このメソッドを呼び出しても例外を投入しないで，rollback 処理を行います。

【形式】

```
public synchronized void rollback() throws SQLException
```

【引数】

なし。

【戻り値】

なし。

【発生する例外】

次の場合，SQLException を投入します。

- データベースアクセスエラーが発生した場合
- Connection オブジェクトに対して close()が既に発行されている場合

(x) setAutoCommit(boolean autoCommit)

【機能】

この接続の自動コミットモードを，指定された状態に設定します。

【形式】

```
public synchronized void setAutoCommit(boolean autoCommit) throws SQLException
```

【引数】

boolean autoCommit：

自動コミットモードを有効にする場合は true，無効にする場合は false を指定します。

【戻り値】

なし。

【機能詳細】

この接続の自動コミットモードを指定された状態に設定します。接続が自動コミットモードの場合，そのすべての SQL 文は，実行後に個別のトランザクションとしてコミットされます。そうでない場合，その SQL 文は，commit メソッド又は rollback メソッドへの呼び出しによって終了されるトランザクションにグループ化されます。デフォルトでは，新しい接続は自動コミットモードです。

自動コミットは，SQL 文の完了で発生します。SQL 文が ResultSet オブジェクトを返す場合，ResultSet オブジェクトがクローズされたときに，SQL 文は完了します。

トランザクションの途中でこのメソッドが呼び出されても，そのトランザクションはコミットされません。

【発生する例外】

次の場合，SQLException を投入します。

- この Connection オブジェクトが分散トランザクション下で動作している場合
- この Connection オブジェクトに対して close()が既に発行されている場合

(y)　setCatalog(String catalog)

【機能】

HiRDB にはカタログが存在しないため，このドライバはメソッドの指定値を無視します。

【形式】

```
public synchronized void setCatalog(String catalog) throws SQLException
```

【引数】

String catalog：

HiRDB にはカタログがないため，指定値を無視します。

【戻り値】

常に false を返します。

【発生する例外】

この Connection オブジェクトに対して close()が既に発行されている場合，SQLException を投入します。

(z)　setHoldability(int holdability)

【機能】

この Connection オブジェクトを使用して生成される ResultSet オブジェクトの保持機能を，指定された保持機能へ変更します。

【形式】

```
public synchronized void setHoldability(int holdability) throws SQLException
```

【引数】

int holdability：

ResultSet の保持機能定数（ResultSet.HOLD_CURSORS_OVER_COMMIT 又は ResultSet.CLOSE_CURSORS_AT_COMMIT）

【戻り値】

なし。

【機能詳細】

この Connection オブジェクトを使用して生成される ResultSet オブジェクトの保持機能を，指定された保持機能へ変更します。

このメソッドの値は，createStatement（prepareStatement，prepareCall）を保持機能の指定なしで呼び出したときの設定値であり，生成済みの Statement 及びその Statement から生成される（生成された）ResultSet には影響しません。

保持機能のデフォルト値を次の表に示します。

表 17–16　保持機能のデフォルト値

プロパティ HiRDB_CURSOR の指定値	デフォルト値
TRUE	ResultSet.HOLD_CURSORS_OVER_COMMIT
FALSE	ResultSet.CLOSE_CURSORS_AT_COMMIT
指定なし	ResultSet.CLOSE_CURSORS_AT_COMMIT

【発生する例外】

次の場合，SQLException を投入します。

- この Connection オブジェクトに対して close()が既に発行されている場合
- 引数 holdability に ResultSet.HOLD_CURSORS_OVER_COMMIT 又は ResultSet.CLOSE_CURSORS_AT_COMMIT 以外が指定された場合

(aa)　setReadOnly(boolean readOnly)

【機能】

HiRDB には専用モードがないため，メソッドの指定値を無視します。

【形式】

```
public synchronized void setReadOnly(boolean readOnly) throws SQLException
```

【引数】

boolean readOnly：

HiRDB には専用モードがないため，指定値を無視します。

【戻り値】

なし。

【発生する例外】

この Connection オブジェクトに対して close()が既に発行されている場合，SQLException を投入します。

(ab)　setTransactionIsolation(int level)

【機能】

HiRDB では常に TRANSACTION_REPEATABLE_READ のため，このメソッドの指定値を無視します。

【形式】

```
public void setTransactionIsolation(int level) throws SQLException
```

【引数】

int level：

トランザクション遮断レベル

【戻り値】

なし。

【発生する例外】

この Connection オブジェクトに対して close()が既に発行されている場合，SQLException を投入します。

(ac)　checkSession(int waittime)

【機能】

現在の接続状態を確認します。

【形式】

```
public int checkSession (int waittime) throws SQLException
```

【引数】

int waittime：

待ち時間（単位：秒）。0 の場合，クライアント環境定義 PDCWAITTIME で指定した時間まで待ちます。

【戻り値】

PrdbConnection.SESSION_ALIVE：

接続中であることを確認できました。

PrdbConnection.SESSION_NOT_ALIVE：

引数で指定された時間でのタイムアウト以外の原因によって，接続中であることを確認できませんでした。

PrdbConnection.SESSION_CHECK_TIMEOUT：

引数で指定された時間でのタイムアウトによって，接続中であることを確認できませんでした。

【発生する例外】

引数 waittime がマイナスの場合，例外 SQLException を投入します。

(ad)　setHiRDB_Audit_Info(int pos，String userinf)

【機能】

ユーザ任意の接続情報（ユーザ付加情報）を設定します。

【形式】

```
public synchronized void setHiRDB_Audit_Info(int pos, String userinf) throws SQLException
```

【引数】

int pos：

ユーザ付加情報を設定する番号

1：ユーザ付加情報 1

2：ユーザ付加情報 2

3：ユーザ付加情報 3

String userinf：

ユーザ付加情報。ユーザ付加情報に設定されたデータは，HiRDB との文字データ処理で，接続時のプロパティ ENCODELANG や setEncodeLang メソッドに指定された文字セット，又は HiRDB サーバの文字コードに変換されます。コード変換後のバイト数が 100 バイト以内となるデータを設定してください。設定情報を解除する場合は NULL 値を設定します。

【戻り値】

なし。

【機能詳細】

HiRDB サーバにアクセスするアプリケーションのアカウント情報などのユーザ付加情報を設定します。設定したユーザ付加情報は解除するまで有効となります。該当する Connection オブジェクトを使用して生成した Statement オブジェクト，PreparedStatement オブジェクト，及び CallableStatement オブジェクトを使用した SQL 実行時に，設定したユーザ付加情報が監査証跡に出力されます。

なお，このメソッドは，埋込み SQL でのユーザ任意接続情報の設定（DECLARE AUDIT INFO SET）に該当します。

【発生する例外】

次の場合，SQLException を投入します。

- 該当する Connection オブジェクトがクローズされている場合
- ユーザ付加情報のコード変換に失敗した場合
- pos に 1～3 以外の値が指定された場合
- ユーザ付加情報のコード変換後のデータ長が 100 バイトを超えている場合

(ae)　isValid(int timeout)

【機能】

現在の接続状態を確認します。

【形式】

```
public synchronized boolean isValid (int timeout) throws SQLException
```

【引数】

int timeout：

待ち時間（秒）。0 の場合，無制限となります。

【戻り値】

- true：接続中であることを確認できました。
- false：接続中であることを確認できませんでした。次のどれかの場合，false となります。
  ・該当する Connection オブジェクトがクローズされている場合
  ・引数で指定された時間でのタイムアウト期間が過ぎた場合
  ・接続先が XDM/RD E2 の場合

【発生する例外】

引数 timeout が-1 以下の場合，例外 SQLException を投入します。

(af)　setSchema(String schema)

【機能】

SQL 実行時，認可識別子を省略した場合に仮定する認可識別子を設定します。

【形式】

```
public void setSchema (String schema) throws SQLException
```

【引数】

String schema：

認可識別子名。

【戻り値】

なし。

【発生する例外】

次の場合，SQLException を投入します。

- データベースアクセスエラーが発生した場合
- 該当する Connection オブジェクトがクローズされている場合
- 引数 schema に指定された文字列の長さが 0 又は 30 バイトを超える場合
- 接続先が XDM/RD E2 の場合

(ag)　getSchema()

【機能】

この Connection オブジェクトで現在有効となっている SQL 実行時，認可識別子を省略した場合に仮定する認可識別子を返却します。

【機能】

```
public String getSchema () throws SQLException
```

【機能】

なし。

【戻り値】

現在の認可識別子。

【発生する例外】

次の場合，SQLException を投入します。

- 該当する Connection オブジェクトがクローズされている場合

(ah)　beginRequest()

【機能】

コネクションの使用開始を JDBC ドライバに通知します。

【形式】

public void beginRequest() throws SQLException

【引数】

なし。

【戻り値】

なし。

【発生する例外】

次の場合，SQLException を投入します。

- 該当する Connection オブジェクトがクローズされている場合

(ai)　endRequest()

【機能】

コネクションの使用終了を JDBC ドライバに通知します。

【形式】

public void endRequest() throws SQLException

【引数】

なし。

【戻り値】

なし。

【発生する例外】

次の場合，SQLException を投入します。

- 該当する Connection オブジェクトがクローズされている場合

(3)　パッケージ名称及びクラス名称

このインタフェースを実装するパッケージ名称とクラス名称を次に示します。

パッケージ名称：**JP.co.Hitachi.soft.HiRDB.JDBC**

クラス名称：**PrdbConnection**

(4)　注意事項

(a)　ホールダビリティの指定について

ResultSet のホールダビリティに ResultSet.HOLD_CURSORS_OVER_COMMIT を指定した場合，JDBC ドライバは HiRDB のホールダブルカーソル機能を使用します。

次に示す方法でホールダビリティを指定すると，URL の構文 HIRDB_CURSOR，又はプロパティ HIRDB_CURSOR の指定値を，ステートメントオブジェクト（Statement，PreparedStatement，CallableStatement オブジェクト）及び Connection オブジェクトごとに上書きできます。

- createStatement，preparedStatement，prepareCall メソッドの引数 resultSetHoldability
- setHoldability メソッドの引数 holdability
- 実行する SQL 文（SELECT 文）の UNTIL DISCONNECT の有無

該当するオブジェクトから生成した ResultSet オブジェクト又は DatabaseMetaData オブジェクト（setHoldability メソッド使用時）は，これらの指定と HIRDB_CURSOR の指定の組み合わせによって有効になるホールダビリティの指定が変わります。

次のメソッドで生成したステートメントオブジェクトの場合に有効になるホールダビリティの指定を，次の表に示します。

- createStatement(int resultSetType,int resultSetConcurrency, int resultSetHoldability)
- prepareStatement(int resultSetType,int resultSetConcurrency, int resultSetHoldability)
- prepareCall(int resultSetType,int resultSetConcurrency, int resultSetHoldability)

表 17–17　有効になるホールダビリティの指定（1/2）

HIRDB_CURSOR，又は setHiRDBCursorMode の指定		引数 resultSetHoldability の指定値		
		ResultSet.HOLD_CURSORS_OVER_COMMIT	ResultSet.CLOSE_CURSORS_AT_COMMIT	
			UNTIL DISCONNECT 付きの SELECT 文を実行	左記以外の SQL 文を実行
プロパティ HIRDB_CURSOR に TRUE を指定	URL の構文 HIRDB_CURSOR に TRUE を指定	T	T	F
	URL の構文 HIRDB_CURSOR に FALSE を指定	T	T	F

HIRDB_CURSOR，又は setHiRDBCursorMode の指定		引数 resultSetHoldability の指定値		
		ResultSet.HOLD_CURSORS_OVER_COMMIT	ResultSet.CLOSE_CURSORS_AT_COMMIT	
			UNTIL DISCONNECT 付きの SELECT 文を実行	左記以外の SQL 文を実行
プロパティ HIRDB_CURSOR に FALSE を指定	URL の構文 HIRDB_CURSOR に TRUE を指定	T	T	F
	URL の構文 HIRDB_CURSOR に FALSE を指定	T	T	F
setHiRDBCursorMode に true を指定		T	T	F
setHiRDBCursorMode に false を指定		T	T	F

（凡例）

T：HIRDB_CURSOR に TRUE を指定したものとして動作します。

F：HIRDB_CURSOR に FALSE を指定したものとして動作します。

表「有効になるホールダビリティの指定（1/2）」以外のメソッドで生成したステートメントオブジェクト，又は DatabaseMetaData オブジェクトの場合に有効になるホールダビリティの指定を，次の表に示します。

表 17–18　有効になるホールダビリティの指定（2/2）

HIRDB_CURSOR，又は setHiRDBCursorMode の指定		setHoldability メソッドの指定値			setHoldability メソッドの実行なし	
		ResultSet.HOLD_CURSORS_OVER_COMMIT	ResultSet.CLOSE_CURSORS_AT_COMMIT			
			UNTIL DISCONNECT 付きの SELECT 文を実行	左記以外の SQL 文を実行	UNTIL DISCONNECT 付きの SELECT 文を実行	左記以外の SQL 文を実行
プロパティ HIRDB_CURSOR に TRUE を指定	URL の構文 HIRDB_CURSOR に TRUE を指定	T	T	F	T	T
	URL の構文 HIRDB_CURSOR に FALSE を指定	T	T	F	T	F
プロパティ HIRDB_CURSOR に FALSE を指定	URL の構文 HIRDB_CURSOR に TRUE を指定	T	T	F	T	T

HIRDB_CURSOR，又は setHiRDBCursorMode の指定		setHoldability メソッドの指定値			setHoldability メソッドの実行なし	
		ResultSet.HOLD_CURSORS_OVER_COMMIT	ResultSet.CLOSE_CURSORS_AT_COMMIT			
			UNTIL DISCONNECT 付きの SELECT 文を実行	左記以外の SQL 文を実行	UNTIL DISCONNECT 付きの SELECT 文を実行	左記以外の SQL 文を実行
	URL の構文 HIRDB_CURSOR に FALSE を指定	T	T	F	T	F
setHiRDBCursorMode に true を指定		T	T	F	T	T
setHiRDBCursorMode に false を指定		T	T	F	T	F

（凡例）

T：HIRDB_CURSOR に TRUE を指定したものとして動作します。

F：HIRDB_CURSOR に FALSE を指定したものとして動作します。

HIRDB_CURSOR については，「getConnection メソッドによる HiRDB への接続」を参照してください。

17.4.3　Statement インタフェース

(1)　概要

Statement インタフェースでは，主に次の機能が提供されます。

- SQL の実行
- 検索結果としての結果セット（ResultSet オブジェクト）の生成
- 更新結果としての更新行数の返却
- 最大検索行数の設定
- 検索制限時間の設定

(2)　メソッド

Statement インタフェースのメソッド一覧を次の表に示します。なお，表に記載されていないメソッドはサポートしていません。サポートしていないメソッドを指定すると，SQLException を投入します。

表 17–19　Statement インタフェースのメソッド一覧

メソッド	提供ドライバ			機能
	JDBC2.0	JDBC4.0	JDBC4.3	
addBatch(String sql)	○	○	○	この Statement オブジェクトのバッチに，指定された SQL を追加します。
cancel()	○	○	○	該当するオブジェクト，及び該当するオブジェクトと同一接続のオブジェクトが実行中の SQL を取り消します。
clearBatch()	○	○	○	この Statement オブジェクトのバッチに登録された SQL を，すべてクリアします。
clearWarnings()	○	○	○	この Statement オブジェクトに関して報告された，すべての警告をクリアします。
close()	○	○	○	Statement オブジェクト，及びこの Statement オブジェクトから生成した ResultSet オブジェクトのクローズを行います。
execute(String sql)	○	○	○	指定された SQL 文を実行します。
executeBatch()	○	○	○	バッチに登録された SQL を実行し，更新行数の配列を返します。
executeQuery(String sql)	○	○	○	指定された検索系 SQL を実行し，その結果の ResultSet オブジェクトを返します。
executeUpdate(String sql)	○	○	○	指定された検索系以外の SQL を実行し，更新行数を返します。
getConnection()	○	○	○	この Statement オブジェクトを生成した Connection オブジェクトを返します。
getFetchDirection()	○	○	○	この Statement オブジェクトから生成される結果集

メソッド	提供ドライバ			機能
	JDBC2.0	JDBC4.0	JDBC4.3	
				合の，デフォルトのフェッチ方向を取得します。
getFetchSize()	○	○	○	この Statement オブジェクトから生成される ResultSet オブジェクトの，デフォルトのフェッチサイズを取得します。
getMaxFieldSize()	○	○	○	この Statement オブジェクトによって生成される，ResultSet オブジェクトの文字及びバイナリの各列値に対し，返される最大バイト数を取得します。
getMaxRows()	○	○	○	この Statement オブジェクトによって生成される ResultSet オブジェクトが，含むことのできる最大格納行数を取得します。
getMoreResults()	○	○	○	次の結果集合に移動します。
getQueryTimeout()	○	○	○	SQL 実行がタイムアウトになる秒数を返します。
getResultSet()	○	○	○	ResultSet オブジェクトとして，現在の結果を取得します。
getResultSetConcurrency()	○	○	○	この Statement オブジェクトから生成される ResultSet オブジェクトの並行処理モードを取得します。
getResultSetHoldability()	○	○	○	この Statement オブジェクトから生成される ResultSet オブジェクトの保持機能を取得します。
getResultSetType()	○	○	○	この Statement オブジェクトから生成される ResultSet オブジェクトの，結果集合の型を取得します。
getUpdateCount()	○	○	○	更新行数を返します。

メソッド	提供ドライバ			機能
	JDBC2.0	JDBC4.0	JDBC4.3	
getWarnings()	○	○	○	この Statement オブジェクトに関する呼び出しによって報告される，最初の警告を取得します。
setCursorName(String name)	○	○	○	このドライバでは，位置決めされた更新及び削除をサポートしていないため，このメソッドは指定値を無視します。
setEscapeProcessing(boolean enable)	○	○	○	この Statement オブジェクトによるエスケープ構文の解析を，有効又は無効にします。
setFetchDirection(int direction)	○	○	○	この Statement オブジェクトから生成される結果集合の，デフォルトのフェッチ方向を指定します。
setFetchSize(int rows)	○	○	○	この Statement オブジェクトから生成される ResultSet オブジェクトの，デフォルトのフェッチサイズを設定します。
setMaxFieldSize(int max)	○	○	○	この Statement オブジェクトによって生成される ResultSet オブジェクトの，文字及びバイナリの各列に対する最大バイト数を，指定されたバイト数に設定します。
setMaxRows(int max)	○	○	○	この Statement オブジェクトによって生成される ResultSet オブジェクトが，含むことのできる最大格納行数を設定します。
setQueryTimeout(int seconds)	○	○	○	SQL 実行がタイムアウトになる秒数を設定します。
isClosed()	×	○	○	Statement オブジェクトがクローズされているかどうかの情報を取得します。
isPoolable()	×	○	○	Statement オブジェクトがプールできるかどうかを示す値を取得します。

メソッド	提供ドライバ			機能
	JDBC2.0	JDBC4.0	JDBC4.3	
closeOnCompletion()	×	×	○	ResultSet オブジェクトがクローズされたときにこの Statement オブジェクトをクローズするかどうかを設定します。
isCloseOnCompletion()	×	×	○	ResultSet オブジェクトがクローズされたときにこの Statement オブジェクトをクローズするかどうかを返却します。
getLargeUpdateCount()	×	×	○	更新行数を返します。
setLargeMaxRows(long max)	×	×	○	この Statement オブジェクトによって生成される ResultSet オブジェクトが含むことのできる最大格納行数を設定します。
getLargeMaxRows()	×	×	○	この Statement オブジェクトによって生成される ResultSet オブジェクトが，含むことのできる最大格納行数を取得します。
executeLargeBatch()	×	×	○	バッチに登録された SQL を実行し，更新行数の配列を返します。
executeLargeUpdate(String sql)	×	×	○	指定された検索系以外の SQL を実行し，更新行数を返します。
enquoteLiteral(String val)	×	×	○	指定された文字列に対し，HiRDB の SQL 記述での文字列定数表記を返却します。
enquoteIdentifier(String identifier, boolean alwaysQuote)	×	×	○	指定された文字列に対し，HiRDB の SQL 記述での名前の指定に使用可能な文字列か判定し，元の値の文字列又は二重引用符で囲まれた文字列を返します。
isSimpleIdentifier(String identifier)	×	×	○	HiRDB の SQL 記述での名前の指定に使用可能な文字列か判定します。
enquoteNCharLiteral(String val)	×	×	○	指定された文字列に対し，HiRDB の SQL 記述での各

メソッド	提供ドライバ			機能
	JDBC2.0	JDBC4.0	JDBC4.3	
				国文字列定数表記を返却します。

（凡例）

○：提供されます。

×：提供されません。

(a) addBatch(String sql)

【機能】

この Statement オブジェクトのバッチに，指定された SQL を追加します。

最大 2,147,483,647 個実行する SQL 文を登録できます。

【形式】

```
public synchronized void addBatch(String sql) throws SQLException
```

【引数】

String sql：

実行する SQL 文

【戻り値】

なし。

【発生する例外】

次の場合，SQLException を投入します。

- Statement オブジェクトに対して close()が既に発行されている場合
- この Statement オブジェクトを生成した Connection オブジェクトに対して，close()が既に発行されている場合
- 上限値 2,147,483,647 個を超える SQL 文を登録しようとした場合
- SQL に null 又は 0 長文字列を指定している場合

(b) cancel()

【機能】

該当するオブジェクト，及び該当するオブジェクトと同一接続のオブジェクトが実行中の SQL を取り消します。

【形式】

```
public void cancel() throws SQLException
```

【引数】

なし。

【戻り値】

なし。

【機能詳細】

cancel メソッドを使用して，実行中の SQL※に非同期キャンセルを実行できます。

このメソッドは，該当する Statement オブジェクトが SQL 実行中でなくても，同一接続オブジェクトに対して，ほかのオブジェクトが SQL を実行している場合は，非同期キャンセルを行います。

キャンセルが HiRDB サーバで実行された場合，次の方法による設定値に関係なく，キャンセル前に作成した ResultSet オブジェクト，PreparedStatement オブジェクト，及び CallableStatement オブジェクトが無効となります。

- DriverManager.getConnection の Properties 引数中のプロパティ HiRDB_for_Java_STATEMENT_COMMIT_BEHAVIOR
- URL 中の STATEMENT_COMMIT_BEHAVIOR
- DataSource 系インタフェースの setStatementCommitBehavior
- DriverManager.getConnection の Properties 引数中のプロパティ HIRDB_CURSOR
- URL 中の HIRDB_CURSOR
- DataSource 系インタフェースの setHiRDBCursorMode
- Connection インタフェースの setHoldability メソッド
- Connection インタフェースの createStatement，parepareStatement メソッドの引数 resultSetHoldability
- SQL 文（until disconnect の指定）

該当する Statement オブジェクトが SQL 実行中ではなく，かつ同一接続オブジェクトに対してほかのオブジェクトが SQL を実行していない場合，このメソッドは HiRDB に対してキャンセルを実行しません。

XADataSource を使用した接続の場合，非同期キャンセルの要求は無効になります。

注※

「HiRDB サーバ側に制御があり（この JDBC ドライバは応答待ち），サーバで処理中の SQL」のことを指します。

【発生する例外】

次の場合，SQLException を投入します。

- Statement オブジェクトに対して close()が既に発行されている場合
- この Statement オブジェクトを生成した Connection オブジェクトに対して，close()が既に発行されている場合

(c) clearBatch()

【機能】

この Statement オブジェクトのバッチに登録された SQL を，すべてクリアします。

【形式】

```
public synchronized void clearBatch() throws SQLException
```

【引数】

なし。

【戻り値】

なし。

【発生する例外】

次の場合，SQLException を投入します。

- Statement オブジェクトに対して close()が既に発行されている場合
- この Statement オブジェクトを生成した Connection オブジェクトに対して，close()が既に発行されている場合

(d) clearWarnings()

【機能】

この Statement オブジェクトに関して報告された，すべての警告をクリアします。

【形式】

```
public synchronized void clearWarnings() throws SQLException
```

【引数】

なし。

【戻り値】

なし。

【発生する例外】

なし。

(e) close()

【機能】

Statement オブジェクト，及びこの Statement オブジェクトから生成した ResultSet オブジェクトのクローズを行います。

【形式】

```
public void close() throws SQLException
```

【引数】

なし。

【戻り値】

なし。

【機能詳細】

Statement オブジェクト，及びこの Statement オブジェクトから生成した ResultSet オブジェクトのクローズを行います。

プーリング接続時に，Statement の close メソッドの発行でエラーが発生した場合，SQLException を投入しません。

また，プーリング環境下及び XA 環境下で，Statement の close メソッドでデータベースとの物理的な切断でエラーが発生し，コネクションプーリングが使用できなくなった場合，ConnectionEventListener.connectionErrorOccurred()は発生しません。

【発生する例外】

データベースアクセスエラーが発生した場合，SQLException を投入します。

(f)　execute(String sql)

【機能】

指定された SQL 文を実行します。ResultSet オブジェクトや更新行数を，Statement.getResultSet，Statement.getUpdateCount で取得できます。

【形式】

```
public synchronized boolean execute(String sql) throws SQLException
```

【引数】

String sql：

実行する SQL 文

【戻り値】

実行した SQL が検索系 SQL の場合は true，そうでない場合は false を返します。

【機能詳細】

指定された SQL 文を実行します。ResultSet オブジェクトや更新行数を，Statement.getResultSet，Statement.getUpdateCount で取得できます。

このメソッド実行後の Statement.getResultSet，及び Statement.getUpdateCount の戻り値を次の表に示します。

表 17–20　実行した SQL と Statement.getResultSet 及び Statement.getUpdateCount の戻り値の関係

実行した SQL の種類	Statement.getResultSet の戻り値		Statement.getUpdateCount の戻り値	
	システムプロパティ HiRDB_for_Java_DAB_EXECUTESQL_NOCHK 指定		システムプロパティ HiRDB_for_Java_UPDATECOUNT_BEHAVIOR 指定	
	TRUE 以外	TRUE	TRUE	FALSE
検索系 SQL	実行結果の ResultSet オブジェクト		-1	
検索系以外の SQL	null	0 列の ResultSet オブジェクト	1 以上の値又は-1（更新行なし）	0 以上の値
SQL の実行がエラーとなった場合	null	null	-1	

【発生する例外】

次の場合，SQLException を投入します。

- Statement オブジェクトに対して close()が既に発行されている場合
- この Statement オブジェクトを生成した Connection オブジェクトに対して，close()が既に発行されている場合
- 引数 sql に null 又は 0 長文字列を指定した場合
- データベースアクセスエラーが発生した場合
- HiRDB SQL Tuning Advisor 用アクセスパス情報ファイルへの出力でエラーが発生した場合（JDBC4.0 以降のときだけ）

(g)　executeBatch()

【機能】

バッチに登録された SQL を実行し，更新行数の配列を返します。

バッチに登録されたすべての SQL を実行後，又は途中でエラーが起きた場合，Statement.clearBatch() を呼び，バッチ登録情報をクリアします。

【形式】

```
public synchronized int[] executeBatch() throws SQLException
```

【引数】

なし。

【戻り値】

実行した SQL ごとの更新行数を，配列にして返します。配列は，バッチに登録された順序になります。バッチに一つも登録されていない場合，又はバッチの一つ目がエラーだった場合，要素数 0 の配列を返します。

【発生する例外】

次の場合，SQLException を投入します。

- Statement オブジェクトに対して close()が既に発行されている場合
- この Statement オブジェクトを生成した Connection オブジェクトに対して，close()が既に発行されている場合

次の場合，例外 BatchUpdateException（SQLException のサブクラス）を投入します。

- 検索系 SQL がバッチで実行された場合
- データベースアクセスエラーが発生した場合
- HiRDB SQL Tuning Advisor 用アクセスパス情報ファイルへの出力でエラーが発生した場合（JDBC4.0 以降のときだけ）

(h) executeQuery(String sql)

【機能】

指定された検索系 SQL を実行し，その結果の ResultSet オブジェクトを返します。

【形式】

```
public synchronized ResultSet executeQuery(String sql) throws SQLException
```

【引数】

String sql：

実行する SQL 文

【戻り値】

実行結果の ResultSet オブジェクト

【機能詳細】

- システムプロパティ HiRDB_for_Java_DAB_EXECUTESQL_NOCHK に TRUE を設定していない場合
  指定された検索系 SQL を実行し，その結果の ResultSet オブジェクトを返却します。検索結果のない SQL 文（INSERT 文など）の場合は，SQLException を投入します。
- システムプロパティ HiRDB_for_Java_DAB_EXECUTESQL_NOCHK に TRUE を設定している場合
  指定された SQL を実行し，その結果の ResultSet オブジェクトを返却します。
  検索結果のない SQL 文（INSERT 文など）の場合，0 列の ResultSet オブジェクトを返却します。また，Statement.getUpdateCount メソッドを使用して，更新行数を取得できます。executeQuery メソッドの戻り値と，メソッド実行後に実行する Statement.getUpdateCount メソッドの戻り値を次の表に示します。

表 17–21　executeQuery メソッドの戻り値と，メソッド実行後に実行する Statement.getUpdateCount メソッドの戻り値

実行した SQL の種類		executeQuery メソッドの戻り値	Statement.getUpdateCount メソッドの戻り値	
			HiRDB_for_Java_UPDATECOUNT_BEHAVIOR 指定	
			TRUE	FALSE
検索系 SQL		実行結果の ResultSet オブジェクト	-1	
検索系以外の SQL	INSERT，UPDATE，DELETE	0 列の ResultSet オブジェクト	1 以上の更新行数又は-1（更新行数なし）	0 以上の更新行数
	その他	0 列の ResultSet オブジェクト	-1	0
SQL の実行がエラーとなった場合		−	-1	

【発生する例外】

次の場合，SQLException を投入します。

- Statement オブジェクトに対して close()が既に発行されている場合
- この Statement オブジェクトを生成した Connection オブジェクトに対して，close()が既に発行されている場合
- 検索系以外の SQL を指定した場合（システムプロパティ HiRDB_for_Java_DAB_EXECUTESQL_NOCHK に TRUE を設定している場合を除く）
- 引数 sql に null 又は 0 長文字列を指定した場合
- データベースアクセスエラーが発生した場合
- HiRDB SQL Tuning Advisor 用アクセスパス情報ファイルへの出力でエラーが発生した場合（JDBC4.0 以降のときだけ）

(i)　executeUpdate(String sql)

【機能】

指定された検索系以外の SQL を実行し，更新行数を返します。

【形式】

```
public synchronized int executeUpdate(String sql) throws SQLException
```

【引数】

String sql：

実行する SQL 文

【戻り値】

次の表に示します。

表 17–22　executeUpdate メソッドの戻り値

実行した SQL の種類		システムプロパティ HiRDB_for_Java_DAB_EXECUTESQL_NOCHK 指定	
		TRUE 以外	TRUE
検索系 SQL		–	-1
検索系以外の SQL	INSERT，UPDATE，DELETE	更新行数	更新行数
	その他	0	0

【機能詳細】

- システムプロパティ HiRDB_for_Java_DAB_EXECUTESQL_NOCHK に TRUE を設定していない場合
  指定された検索系以外の SQL を実行し，更新行数を返却します。
  検索結果を返却する SQL 文（SELECT 文）の場合は，SQLException を投入します。
- システムプロパティ HiRDB_for_Java_DAB_EXECUTESQL_NOCHK に TRUE を設定している場合
  指定された SQL を実行します。
  戻り値が-1 の場合，Statement.getResultSet メソッドを使用して ResultSet オブジェクトを取得できます。executeUpdate メソッド実行後に実行する Statement.getResultSet メソッドの戻り値を次の表に示します。

  表 17–23　executeUpdate メソッド実行後に実行する Statement.getResultSet メソッドの戻り値

実行した SQL の種類		Statement.getResultSet メソッドの戻り値
検索系 SQL		実行結果の ResultSet オブジェクト
検索系以外の SQL	INSERT，UPDATE，DELETE	0 列の ResultSet オブジェクト
	その他	0 列の ResultSet オブジェクト
SQL の実行がエラーとなった場合		null

【発生する例外】

次の場合，SQLException を投入します。

- Statement オブジェクトに対して close()が既に発行されている場合
- この Statement オブジェクトを生成した Connection オブジェクトに対して，close()が既に発行されている場合
- 検索系 SQL を指定した場合（システムプロパティ HiRDB_for_Java_DAB_EXECUTESQL_NOCHK に TRUE を設定している場合を除く）
- 引数 sql に null 又は 0 長文字列を指定した場合
- データベースアクセスエラーが発生した場合

- HiRDB SQL Tuning Advisor 用アクセスパス情報ファイルへの出力でエラーが発生した場合（JDBC4.0 以降のときだけ）

(j) getConnection()

【機能】

この Statement オブジェクトを生成した Connection オブジェクトを返します。

【形式】

```
public synchronized Connection getConnection() throws SQLException
```

【引数】

なし。

【戻り値】

Connection オブジェクト

【発生する例外】

次の場合，SQLException を投入します。

- Statement オブジェクトに対して close()が既に発行されている場合
- この Statement オブジェクトを生成した Connection オブジェクトに対して，close()が既に発行されている場合

(k) getFetchDirection()

【機能】

この Statement オブジェクトから生成される結果集合の，デフォルトのフェッチ方向を取得します。

HiRDB では，フェッチ方向は順方向しかサポートしないため，このメソッドの戻り値は，常に ResultSet.FETCH_FORWARD となります。

【形式】

```
public synchronized int getFetchDirection() throws SQLException
```

【引数】

なし。

【戻り値】

常に ResultSet.FETCH_FORWARD を返します。

【発生する例外】

次の場合，SQLException を投入します。

- Statement オブジェクトに対して close()が既に発行されている場合
- この Statement オブジェクトを生成した Connection オブジェクトに対して，close()が既に発行されている場合

(l)　getFetchSize()

【機能】

この Statement オブジェクトから生成される ResultSet オブジェクトの，デフォルトのフェッチサイズを取得します。

【形式】

```
public synchronized int getFetchSize() throws SQLException
```

【引数】

なし。

【戻り値】

この Statement オブジェクトから生成される ResultSet オブジェクトの，デフォルトのフェッチサイズ

【機能詳細】

この Statement オブジェクトから生成される ResultSet オブジェクトの，デフォルトのフェッチサイズを取得します。

setFetchSize で 0 が指定されている場合，実際のフェッチサイズはクライアント環境定義に依存しますが，このメソッドは 0 を返します。フェッチサイズと戻り値の関係を次の表に示します。

表 17–24　フェッチサイズと戻り値の関係

setFetchSize の設定値（m）	戻り値
0	0
1≦m≦4096	m

【発生する例外】

次の場合，SQLException を投入します。

- Statement オブジェクトに対して close()が既に発行されている場合
- この Statement オブジェクトを生成した Connection オブジェクトに対して，close()が既に発行されている場合

(m)　getMaxFieldSize()

【機能】

この Statement オブジェクトによって生成される，ResultSet オブジェクトの文字及びバイナリの各列値に対し，返される最大バイト数を取得します。この最大バイト数は，[M ｜ N][VAR]CHAR，BINARY，及び BLOB の各列にだけ適用されます。最大バイト数を超えた分は切り捨てます。

setMaxFieldSize で設定した値を返します。

【形式】

```
public synchronized int getMaxFieldSize() throws SQLException
```

【引数】

なし。

【戻り値】

[M ｜ N][VAR]CHAR，BINARY，及び BLOB 列に対する現在の最大バイト数。0 は無制限を意味します。

【発生する例外】

次の場合，SQLException を投入します。

- Statement オブジェクトに対して close()が既に発行されている場合
- この Statement オブジェクトを生成した Connection オブジェクトに対して，close()が既に発行されている場合

(n) getMaxRows()

【機能】

この Statement オブジェクトによって生成される ResultSet オブジェクトが，含むことのできる最大格納行数を取得します。最大格納行数を超えた行は，通知なしに除外されます。

setMaxRows で設定した値を返します。

【形式】

```
public synchronized int getMaxRows() throws SQLException
```

【引数】

なし。

【戻り値】

この Statement オブジェクトによって生成される，ResultSet オブジェクトの最大格納行数（0 は無制限を意味します）

【発生する例外】

次の場合，SQLException を投入します。

- Statement オブジェクトに対して close()が既に発行されている場合
- この Statement オブジェクトを生成した Connection オブジェクトに対して，close()が既に発行されている場合

(o) getMoreResults()

【機能】

次の結果集合に移動します。

【形式】

```
public synchronized boolean getMoreResults() throws SQLException
```

【引数】

なし。

【戻り値】

true：

次の結果集合が存在します。

false：

次の結果集合が存在しません。

【発生する例外】

次の場合，SQLException を投入します。

- Statement オブジェクトに対して close()が既に発行されている場合
- この Statement オブジェクトを生成した Connection オブジェクトに対して，close()が既に発行されている場合
- データベースアクセスエラーが発生した場合

(p) getQueryTimeout()

【機能】

SQL 実行がタイムアウトになる秒数を返します。

setQueryTimeout で設定した値を返します。

setQueryTimeout を実行してない場合は，0 を返します。

【形式】

```
public synchronized int getQueryTimeout() throws SQLException
```

【引数】

なし。

【戻り値】

タイムアウトになる秒数

【発生する例外】

次の場合，SQLException を投入します。

- Statement オブジェクトに対して close()が既に発行されている場合
- この Statement オブジェクトを生成した Connection オブジェクトに対して，close()が既に発行されている場合

(q) getResultSet()

【機能】

ResultSet オブジェクトとして，現在の結果を取得します。

【形式】

```
public synchronized ResultSet getResultSet() throws SQLException
```

【引数】

なし。

【戻り値】

Statement オブジェクトが保持している ResultSet オブジェクト（結果がない場合は null を返します）

【発生する例外】

次の場合，SQLException を投入します。

- Statement オブジェクトに対して close()が既に発行されている場合
- この Statement オブジェクトを生成した Connection オブジェクトに対して，close()が既に発行されている場合

(r)　getResultSetConcurrency()

【機能】

この Statement オブジェクトから生成される ResultSet オブジェクトの並行処理モードを取得します。

更新カーソルは未サポートのため，常に ResultSet.CONCUR_READ_ONLY を返します。

【形式】

```
public synchronized int getResultSetConcurrency() throws SQLException
```

【引数】

なし。

【戻り値】

常に ResultSet.CONCUR_READ_ONLY を返します。

【発生する例外】

次の場合，SQLException を投入します。

- Statement オブジェクトに対して close()が既に発行されている場合
- この Statement オブジェクトを生成した Connection オブジェクトに対して，close()が既に発行されている場合

(s)　getResultSetHoldability()

【機能】

この Statement オブジェクトから生成される ResultSet オブジェクトの保持機能を取得します。

保持機能は Statement オブジェクト生成時に決められ，以後変更はできません。

【形式】

```
public synchronized int getResultSetHoldability() throws SQLException
```

【引数】

なし。

【戻り値】

ResultSet.HOLD_CURSORS_OVER_COMMIT：

コミット・ロールバック後も，ResltSet オブジェクトを操作できます。

ResultSet.CLOSE_CURSORS_AT_COMMIT：

コミット・ロールバック後の ResultSet オブジェクトの操作で，close 以外の操作は SQLException が起きます。

【発生する例外】

次の場合，SQLException を投入します。

- Statement オブジェクトに対して close()が既に発行されている場合
- この Statement オブジェクトを生成した Connection オブジェクトに対して，close()が既に発行されている場合

(t)　getResultSetType()

【機能】

この Statement オブジェクトから生成される ResultSet オブジェクトの，結果集合の型を取得します。

更新カーソルは未サポートのため，ResultSet.TYPE_FORWARD_ONLY 又は ResultSet.TYPE_SCROLL_INSENSITIVE を返します。

【形式】

```
public synchronized int getResultSetType() throws SQLException
```

【引数】

なし。

【戻り値】

ResultSet.TYPE_FORWARD_ONLY：

カーソルが順方向だけ移動できます。

ResultSet.TYPE_SCROLL_INSENSITIVE：

カーソルがスクロールできますが，値の変更は反映されません。

【発生する例外】

次の場合，SQLException を投入します。

- Statement オブジェクトに対して close()が既に発行されている場合
- この Statement オブジェクトを生成した Connection オブジェクトに対して，close()が既に発行されている場合

(u)　getUpdateCount()

【機能】

更新行数を返します。

【形式】

```
public synchronized int getUpdateCount() throws SQLException
```

【引数】

なし。

【戻り値】

次の表に示します。

表 17–25　getUpdateCount メソッドの戻り値

<table>
<tr><th colspan="5" rowspan="3">Statement オブジェクトのメソッド実行状態</th><th colspan="2">getUpdateCount の戻り値</th></tr>
<tr><th colspan="2">HiRDB_for_Java_UPDATECOUNT_BEHAVIOR 指定</th></tr>
<tr><th>TRUE</th><th>FALSE</th></tr>
<tr><td colspan="5">executeXXX メソッドを実行していない</td><td colspan="2">-1</td></tr>
<tr><td rowspan="8">executeXXX メソッドを実行している</td><td colspan="4">最後に実行した executeXXX メソッドの後，getMoreResults メソッドを実行した</td><td colspan="2">-1</td></tr>
<tr><td colspan="4">最後に実行した executeXXX メソッドでエラーが発生した</td><td colspan="2">-1</td></tr>
<tr><td colspan="4">最後に executeBatch メソッドを実行した</td><td colspan="2">-1</td></tr>
<tr><td rowspan="7">最後に executeBatch メソッド以外の executeXXX メソッドを実行した</td><td colspan="3">最後に実行した SQL が検索系 SQL</td><td colspan="2">-1</td></tr>
<tr><td rowspan="6">最後に実行した SQL が検索系以外の SQL</td><td rowspan="2">INSERT，UPDATE，DELETE</td><td>更新行あり</td><td colspan="2">更新行数</td></tr>
<tr><td>更新行なし</td><td>-1</td><td>0</td></tr>
<tr><td rowspan="2">ASSIGN LIST</td><td>作成行あり</td><td colspan="2">作成行数</td></tr>
<tr><td>作成行なし</td><td>-1</td><td>0</td></tr>
<tr><td rowspan="2">CALL</td><td>結果集合あり</td><td>-1</td><td></td></tr>
<tr><td>結果集合なし</td><td>-1</td><td>0</td></tr>
<tr><td colspan="2">その他</td><td>-1</td><td>0</td></tr>
</table>

【機能詳細】

更新行数を返します。

【発生する例外】

次の場合，SQLException を投入します。

- Statement オブジェクトに対して close()が既に発行されている場合

- この Statement オブジェクトを生成した Connection オブジェクトに対して，close()が既に発行されている場合

(v) getWarnings()

【機能】

この Statement オブジェクトに関する呼び出しによって報告される，最初の警告を取得します。二つ以上の警告がある場合，後続の警告は，最初の警告にチェーンされ，直前に取得された警告の SQLWarning.getNextWarning メソッドを呼び出すことによって取得されます。

【形式】

```
public synchronized SQLWarning getWarnings() throws SQLException
```

【引数】

なし。

【戻り値】

最初の SQLWarning オブジェクト（SQLWarning オブジェクトがない場合は，null を返します）

【発生する例外】

次の場合，SQLException を投入します。

- Statement オブジェクトに対して close()が既に発行されている場合
- この Statement オブジェクトを生成した Connection オブジェクトに対して，close()が既に発行されている場合

(w) setCursorName(String name)

【機能】

このドライバでは，位置決めされた更新及び削除をサポートしていないため，このメソッドは指定値を無視します。

【形式】

```
public synchronized void setCursorName(String name) throws SQLException
```

【引数】

String name：

カーソル名

【戻り値】

なし。

【発生する例外】

次の場合，SQLException を投入します。

- Statement オブジェクトに対して close()が既に発行されている場合

- この Statement オブジェクトを生成した Connection オブジェクトに対して，close()が既に発行されている場合

(x) setEscapeProcessing(boolean enable)

【機能】

この Statement オブジェクトによるエスケープ構文の解析を，有効又は無効にします。

デフォルトは有効です。

【形式】

```
public synchronized void setEscapeProcessing(boolean enable) throws SQLException
```

【引数】

boolean enable：

エスケープ構文の解析を有効にする場合は true，無効にする場合は false を指定します。

【戻り値】

なし。

【発生する例外】

次の場合，SQLException を投入します。

- Statement オブジェクトに対して close()が既に発行されている場合
- この Statement オブジェクトを生成した Connection オブジェクトに対して，close()が既に発行されている場合

(y) setFetchDirection(int direction)

【機能】

この Statement オブジェクトから生成される結果集合の，デフォルトのフェッチ方向を指定します。

デフォルト値は ResultSet.FETCH_FORWARD です。

【形式】

```
public synchronized void setFetchDirection(int direction) throws SQLException
```

【引数】

int direction：

デフォルトのフェッチ方向 ResultSet.FETCH_FORWARD だけ指定できます。

【戻り値】

なし。

【発生する例外】

次の場合，SQLException を投入します。

- Statement オブジェクトに対して close()が既に発行されている場合

- この Statement オブジェクトを生成した Connection オブジェクトに対して，close()が既に発行されている場合
- direction に ResultSet.FETCH_FORWARD 以外が指定された場合

(z)　setFetchSize(int rows)

【機能】

この Statement オブジェクトから生成される ResultSet オブジェクトの，デフォルトのフェッチサイズを設定します。0 を指定した場合，クライアント環境定義に依存します。

【形式】

```
public synchronized void setFetchSize(int rows) throws SQLException
```

【引数】

int rows：

フェッチする行数。設定できる範囲は 0〜4096 です。

【戻り値】

なし。

【機能詳細】

この Statement オブジェクトから生成される ResultSet オブジェクトの，デフォルトのフェッチサイズを設定します。0 を指定した場合，クライアント環境定義に依存します。

デフォルトは 0 です。0 以外の値を指定した場合，指定された値をクライアント環境定義 PDBLKF の値として，ブロック転送機能を行います。ブロック転送機能については，「ブロック転送機能」を参照してください。

設定値と実際に使用される PDBLKF の値の関係を次の表に示します。

表 17–26　設定値と実際に使用される PDBLKF の関係

setFetchSize の設定値（m）	クライアント環境定義 PDBLKF の値（n）	ブロック転送機能で使用される PDBLKF の値
0	1≦n≦4096	n
	指定なし	指定なし
1≦m≦4096	1≦n≦4096	m
	指定なし	m

【発生する例外】

次の場合，SQLException を投入します。

- Statement オブジェクトに対して close()が既に発行されている場合
- この Statement オブジェクトを生成した Connection オブジェクトに対して，close()が既に発行されている場合

- rows ＜ 0，最大行数（getMaxRows()の値）＜ rows，4096（ブロックフェッチの最大値）＜ rows のどれかに該当する場合

(aa)　setMaxFieldSize(int max)

【機能】

この Statement オブジェクトによって生成される ResultSet オブジェクトの，文字及びバイナリの各列に対する最大バイト数を，指定されたバイト数に設定します。この最大バイト数は，[M ｜ N] [VAR]CHAR，BINARY，及び BLOB の各列にだけ適用されます。最大バイト数を超えた分は切り捨てます。

デフォルトは 0 です。

生成済みの ResultSet オブジェクトには影響しません。

【形式】

```
public synchronized void setMaxFieldSize(int max) throws SQLException
```

【引数】

int max：

新しい最大バイト数。0 は無制限を意味します。最大バイト数が奇数値である場合，NCHAR，NVARCHAR に対しては，最大バイト数-1 を最大バイト数とします。最大バイト数に 1 を指定している場合，NCHAR，NVARCHAR に対する最大バイト数が 0 となりますが，その場合は無制限の 0 として扱います。

【戻り値】

なし。

【発生する例外】

次の場合，SQLException を投入します。

- Statement オブジェクトに対して close()が既に発行されている場合
- この Statement オブジェクトを生成した Connection オブジェクトに対して，close()が既に発行されている場合
- max に 0 未満の値が指定された場合

(ab)　setMaxRows(int max)

【機能】

この Statement オブジェクトによって生成される ResultSet オブジェクトが，含むことのできる最大格納行数を設定します。最大格納行数を超えた行は，通知なしに除外されます。

デフォルトは 0 です。

生成済みの ResultSet オブジェクトには影響しません。

【形式】

```
public synchronized void setMaxRows(int max) throws SQLException
```

【引数】

int max：

新しい最大格納行数。0 は無制限を意味します。ただし，結果集合の型が ResultSet.TYPE_SCROLL_INSENSITIVE の場合は，0 を指定しても Integer.MAX_VALUE が最大格納行数となります。

【戻り値】

なし。

【発生する例外】

次の場合，SQLException を投入します。

- Statement オブジェクトに対して close()が既に発行されている場合
- この Statement オブジェクトを生成した Connection オブジェクトに対して，close()が既に発行されている場合
- max に 0 未満の値が指定された場合

(ac)　setQueryTimeout(int seconds)

【機能】

SQL 実行がタイムアウトになる秒数を設定します。

【形式】

```
public synchronized void setQueryTimeout(int seconds) throws SQLException
```

【引数】

int seconds：

タイムアウトになる秒数

【戻り値】

なし。

【機能詳細】

SQL 実行時の，HiRDB サーバとの通信の最大待ち時間（秒）を指定します。

0 の場合，クライアント環境定義 PDCWAITTIME の設定値に依存します。

デフォルトは 0 です。

65536 以上の値を設定している場合，このメソッドは指定値を無視します。

【発生する例外】

次の場合，SQLException を投入します。

- Statement オブジェクトに対して close()が既に発行されている場合
- この Statement オブジェクトを生成した Connection オブジェクトに対して，close()が既に発行されている場合

- seconds に 0 未満の値が指定された場合

(ad)　isClosed()

【機能】

Statement オブジェクトがクローズされているかどうかの情報を取得します。

【形式】

```
public boolean isClosed() throws SQLException
```

【引数】

なし。

【戻り値】

true：

Statement オブジェクトがクローズされています。

false：

Statement オブジェクトがオープン中です。

【機能詳細】

Statement オブジェクトがクローズされているかどうかの情報を取得します。次の場合は true が返されます。

- Statement オブジェクトに close()が発行されている場合
- この Statement オブジェクトを生成した Connection オブジェクトに対して close()が既に発行されている場合

【発生する例外】

なし。

(ae)　isPoolable()

【機能】

Statement オブジェクトがプールできるかどうかを示す値を取得します。

【形式】

```
public boolean isPoolable() throws SQLException
```

【引数】

なし。

【戻り値】

false：

Statement オブジェクトはプールできません。

【機能詳細】

Statement オブジェクトがプールできるかどうかを示す値を取得します。常に false が返されます。

【発生する例外】

次の場合，SQLException を投入します。

- Statement オブジェクトに対して close()が既に発行されている場合
- この Statement オブジェクトを生成した Connection オブジェクトに対して，close()が既に発行されている場合

(af)　closeOnCompletion()

【機能】

ResultSet オブジェクトがクローズされたときに，この Statement オブジェクトをクローズするかどうかを設定します。

【形式】

```
public void closeOnCompletion() throws SQLException
```

【引数】

なし。

【戻り値】

なし。

【機能詳細】

この Statement オブジェクトから生成された ResultSet オブジェクトがクローズされたときに，この Statement オブジェクトをクローズするかどうかを設定します。

【発生する例外】

次の場合，SQLException を投入します。

- Statement オブジェクトに対して close()が既に発行されている場合
- この Statement オブジェクトを生成した Connection オブジェクトに対して，close()が既に発行されている場合

(ag)　isCloseOnCompletion()

【機能】

ResultSet オブジェクトがクローズされたときに，この Statement オブジェクトをクローズするかどうかを返却します。

【形式】

```
public void isCloseOnCompletion() throws SQLException
```

【引数】

なし。

【戻り値】

true：

ResultSet オブジェクトクローズ時，Statement オブジェクトをクローズします。

false：

ResultSet オブジェクトクローズ時，Statement オブジェクトをクローズしません。

【機能詳細】

ResultSet オブジェクトがクローズされたときに，この Statement オブジェクトがクローズされるかどうかを返却します。

【発生する例外】

次の場合，SQLException を投入します。

- Statement オブジェクトに対して close()が既に発行されている場合
- この Statement オブジェクトを生成した Connection オブジェクトに対して，close()が既に発行されている場合

(ah)　getLargeUpdateCount()

【機能】

更新行数を返します。

【形式】

```
public synchronized long getLargeUpdateCount() throws SQLException
```

【引数】

なし。

【戻り値】

「getUpdateCount()」を参照してください。

【機能詳細】

「getUpdateCount()」を参照してください。

更新行数が 2,147,483,647 件を超える場合は getLargeUpdateCount()メソッドを使用してください。

更新行数が 4,294,967,295 件以上の場合，一度 0 にリセットされ，再度 1 からカウントアップした値が返却されます。

【発生する例外】

「getUpdateCount()」を参照してください。

(ai)　setLargeMaxRows(long max)

【機能】

この Statement オブジェクトによって生成される ResultSet オブジェクトが，含むことのできる最大格納行数を設定します。最大格納行数を超えた行は，通知なしに除外されます。デフォルトは 0 です。

生成済みの ResultSet オブジェクトには影響しません。

【形式】

```
public synchronized void setLargeMaxRows(long max) throws SQLException
```

【引数】

long max：

新しい最大格納行数。0 は無制限を意味します。ただし，結果集合の型が ResultSet.TYPE_SCROLL_INSENSITIVE の場合は，0 を指定しても 2,147,483,647 件が最大格納行数となります。

【戻り値】

なし。

【機能詳細】

「setMaxRows(int max)」を参照してください。

格納行数が 2,147,483,647 件を超える場合は setLargeMaxRows(long max)を使用してください。

【発生する例外】

「setMaxRows(int max)」を参照してください。

(aj)　getLargeMaxRows()

【機能】

この Statement オブジェクトによって生成される ResultSet オブジェクトが，含むことのできる最大格納行数を取得します。最大格納行数を超えた行は，通知なしに除外されます。

「setMaxRows(int max)」，「setLargeMaxRows(long max)」で設定した値を返します。

【形式】

```
public synchronized long getLargeMaxRows() throws SQLException
```

【引数】

なし。

【戻り値】

「getMaxRows()」を参照してください。

格納行数が 2,147,483,647 件を超える場合は getLargeMaxRows()を使用してください。

【機能詳細】

「getMaxRows()」を参照してください。最大格納行数が 2 ギガバイトを超える場合は getLargeMaxRows()を使用してください。

【発生する例外】

「getMaxRows()」を参照してください。

(ak)　executeLargeBatch()

【機能】

バッチに登録された SQL を実行し，更新行数の配列を返します。

バッチに登録されたすべての SQL を実行後，又は途中でエラーが起きた場合，Statement.clearBatch()を呼び，バッチ登録情報をクリアします。

【形式】

```
public synchronized long[] executeLargeBatch() throws SQLException
```

【引数】

なし。

【戻り値】

「executeBatch()」を参照してください。

【機能詳細】

「executeBatch()」を参照してください。

更新行数が 2,147,483,647 件を超える場合は executeLargeBatch()を使用してください。

更新行数が 4,294,967,295 件以上の場合，一度 0 にリセットされ，再度 1 からカウントアップした値が返却されます。

【発生する例外】

「executeBatch()」を参照してください。

(al)　executeLargeUpdate(String sql)

【機能】

指定された検索系以外の SQL を実行し，更新行数を返します。

【形式】

```
public synchronized long executeLargeUpdate(String sql) throws SQLException
```

【引数】

String sql：

実行する SQL 文

【戻り値】

「executeUpdate(String sql)」を参照してください。

【機能詳細】

「executeUpdate(String sql)」を参照してください。

更新行数が 2,147,483,647 件を超える場合は executeLargeUpdate(String sql)を使用してください。

更新行数が 4,294,967,295 件以上の場合，一度 0 にリセットされ，再度 1 からカウントアップした値が返却されます。

【発生する例外】

「executeUpdate(String sql)」を参照してください。

(am)　enquoteLiteral(String val)

【機能】

指定された文字列に対し，HiRDB の SQL 記述での文字列定数表記（単一引用符で囲まれた文字列。文字列内の単一引用符は二つの単一引用符に置き換える）を返却します。

【形式】

```
public String enquoteLiteral(String val) throws SQLException
```

【引数】

String val：

文字列

【戻り値】

単一引用符で囲まれた文字列。

【機能詳細】

指定された文字列に対し，HiRDB の SQL 記述での文字列定数表記（単一引用符で囲まれた文字列。文字列内の単一引用符は二つの単一引用符に置き換える）を返却します。

変換例を次に示します。

引数 val に指定した文字列	戻り値
Hello	'Hello'
G'Day	'G''Day'
'G''Day'	'''G''''Day'''

【発生する例外】

次の場合，SQLException を投入します。

- Statement オブジェクトに対して close()が既に発行されている場合
- この Statement オブジェクトを生成した Connection オブジェクトに対して，close()が既に発行されている場合

次の場合，NullPointerException を投入します。

- 引数 val に null を指定した場合

(an)　enquoteIdentifier(String identifier, boolean alwaysQuote)

【機能】

指定された文字列に対し，HiRDB の SQL 記述での名前の指定に可能な文字列か判定し，元の値の文字列又は二重引用符で囲まれた文字列を返します。

【形式】

```
public String enquoteIdentifier(String identifier, boolean alwaysQuote) throws SQLExcepti
on
```

【引数】

String identifier：

文字列

boolean alwaysQuote：

文字列を二重引用符付きで返す場合は true，二重引用符なしの場合は false。

【戻り値】

文字列又は二重引用符で囲まれた文字列。

【機能詳細】

引数 identifier を以下の条件で判定し（どれか一つでも満たす場合は Exception を返却します），引数 alwaysQuote が true の場合は二重引用符で囲んだ文字列を，false の場合は元の値をそのまま返します。

1. 文字列が二重引用符で囲まれている
2. 文字列に英大文字，数字，英小文字，下線（_），半角空白，ハイフン（-）以外の ASCII 文字を含む
3. 先頭の文字が半角英大文字，半角英小文字以外の ASCII 文字である
4. 文字列の長さが 1～255 バイトの範囲外である

なお，文字列に半角空白又はハイフンが含まれている場合は，引数 alwaysQuote が false でも二重引用符で囲んだ文字列を返します。

文字列が二重引用符で囲まれている場合は，引数 alwaysQuote の値に関係なく元の値をそのまま返します。

変換例を次に示します。

引数 identifier に指定した文字列	引数 alwaysQuote	戻り値
Hello	false	Hello
Hello	true	"Hello"
Table-1	false	"Table-1"
Table-1	true	"Table-1"

引数 identifier に指定した文字列	引数 alwaysQuote	戻り値
"Bruce Wayone"	false	"Bruce Wayone"
"Bruce Wayone"	true	"Bruce Wayone"
Hello"World	false	SQLException
Hello"World	true	SQLException
"Hello"World"	false	SQLException
"Hello"World"	true	SQLException

【発生する例外】

次の場合，SQLException を投入します。

- Statement オブジェクトに対して close()が既に発行されている場合
- この Statement オブジェクトを生成した Connection オブジェクトに対して，close()が既に発行されている場合

次の場合，NullPointerException を投入します。

- 引数 identifier に null を指定した場合

(ao)　isSimpleIdentifier(String identifier)

【機能】

HiRDB の SQL 記述での名前の指定に使用可能な文字列か判定します。

【形式】

```
public boolean isSimpleIdentifier(String identifier) throws SQLException
```

【引数】

String identifier：

文字列

【戻り値】

true：

名前の指定に使用できます。

false：

名前の指定に使用できません。

【機能詳細】

引数 identifier を以下の条件で判定し（どれか一つでも満たす場合は Exception を返却します），名前の指定に使用可能な文字列かどうかを返却します

1. 文字列が二重引用符で囲まれている
2. 文字列に英大文字，数字，英小文字，下線（_），半角空白，ハイフン（-）以外の ASCII 文字を含む

3. 先頭の文字が半角英大文字，半角英小文字以外の ASCII 文字である

4. 文字列の長さが 1〜255 バイトの範囲外である

変換例を次に示します。

引数 identifier に指定した文字列	戻り値
Hello	true
Table-1	false
"Table-1"	false
"Hello World"	SQLException

【発生する例外】

次の場合，SQLException を投入します。

- Statement オブジェクトに対して close()が既に発行されている場合
- この Statement オブジェクトを生成した Connection オブジェクトに対して，close()が既に発行されている場合

次の場合，NullPointerException を投入します。

- 引数 identifier に null を指定した場合

(ap)　enquoteNCharLiteral(String val)

【機能】

指定された文字列に対し，HiRDB の SQL 記述での各国文字列定数表記（単一引用符で囲まれ大文字の N で始まる文字列。文字列内の単一引用符は二つの単一引用符に置き換える）を返却します。

【形式】

```
public String enquoteNCharLiteral(String val) throws SQLException
```

【引数】

String val：

文字列

【戻り値】

N というプレフィクスが付加された単一引用符で囲まれた文字列。

【機能詳細】

指定された文字列に対し，HiRDB の SQL 記述での各国文字列定数表記（単一引用符で囲まれ大文字 N で始まる文字列。文字列内の単一引用符は二つの単一引用符に置き換える）を返却します。

変換例を次に示します。

引数 val に指定した文字列	戻り値
Hello	N'Hello'

引数 val に指定した文字列	戻り値
G'Day	N'G''Day'
'G''Day'	N'''G''''Day'''
I'''M	N'I''''''M'

【発生する例外】

次の場合，SQLException を投入します。

- Statement オブジェクトに対して close()が既に発行されている場合
- この Statement オブジェクトを生成した Connection オブジェクトに対して，close()が既に発行されている場合

次の場合，NullPointerException を投入します。

- 引数 val に null を指定した場合

(3)　パッケージ名称及びクラス名称

このインタフェースを実装するパッケージ名称とクラス名称を次に示します。

パッケージ名称：**JP.co.Hitachi.soft.HiRDB.JDBC**

クラス名称：**PrdbStatement**

(4)　注意事項

(a)　setFetchSize メソッドの指定によるブロック転送機能の利用

setFetchSize メソッドに 1 以上を指定すると，JDBC ドライバはブロック転送機能を利用して，引数で指定した行数の検索結果を HiRDB サーバに対して一度に要求します。ブロック転送機能の詳細は，「ブロック転送機能」を参照してください。

setFetchSize メソッドの指定値に上限はありませんが，ブロック転送機能で一度に転送できる行数の最大は 4,096 行のため，4,097 以上を指定しても実際の転送行数は 4,096 行以下になります。

JDBC ドライバが HiRDB サーバに対して一度の通信で要求する行数は，次の表に示す優先順位によって決定します。

表 17–27　JDBC ドライバが HiRDB サーバに対して一度の通信で要求する行数の優先順位

優先順位	指定値
1	ResultSet クラスの setFetchSize メソッドの引数で指定した値。
2	Statement クラスの setFetchSize メソッドの引数で指定した値。
3	クライアント環境定義 PDBLKF で設定した値。

表「JDBC ドライバが HiRDB サーバに対して一度の通信で要求する行数の優先順位」に示す転送要求行数に対して，実際に JDBC ドライバが HiRDB サーバから一度の通信で受け取る行数は，「1 回の通信で転送する行数」を参照してください。ただし，「PDBLKF」は「表「JDBC ドライバが HiRDB サーバに対して一度の通信で要求する行数の優先順位」に示す優先順位で決定した転送要求行数」に，「FETCH 文」は「ResultSet クラスの next メソッド」に読み替えてください。

なお，検索結果が表「JDBC ドライバが HiRDB サーバに対して一度の通信で要求する行数の優先順位」に示す転送行数より多い場合，JDBC ドライバは検索が終了するまで（又は，UAP からの検索要求がなくなるまで），HiRDB サーバに対して転送を要求します。

また，次のどれかの条件に一致する場合，HiRDB サーバから JDBC ドライバが一度に受け取る行数は 1 になります。

- 結果集合の射影列に HiRDB の BLOB 型を含む。
- 結果集合の射影列に HiRDB の定義長 32,001 以上の BINARY 型を含み，クライアント環境定義 PDBINARYBLKF の指定が NO である。
- 次の条件すべてに一致する。
  - 接続時のプロパティ LONGVARBINARY_ACCESS，又は DataSource クラスの setLONGVARBINARY_Access の引数に，LOCATOR を指定している。
  - 次のどれかを指定している。
    ・SELECT 文に UNTIL DISCONNECT を指定している。
    ・Connection クラスの createStatement，prepareStatement メソッドの引数 resultSetHoldability に，ResultSet.HOLD_CURSORS_OVER_COMMIT を指定している。
    ・接続時のプロパティ，又は URL の設定項目 HIRDB_CURSOR に TRUE を指定する。
    ・DataSource クラスの setHiRDBCursorMode の引数に，true を指定している。

(b)　cancel メソッドによる非同期キャンセル

cancel メソッドを使用すると，HiRDB サーバで処理中の SQL に対する非同期キャンセルを実行できます。該当する Statement オブジェクトが SQL を実行中でなくても，同一の接続オブジェクトに対してほかのオブジェクトが SQL を実行していると，非同期キャンセルを実行します。

HiRDB サーバで非同期キャンセルが実行されると，Statement オブジェクト及び ResultSet オブジェクトをコミット実行後も有効とするかどうかの指定に関係なく，非同期キャンセル前に作成した PreparedStatement オブジェクト及び ResultSet オブジェクトが無効になります。

オブジェクトをコミット実行後も有効とするかどうかの指定方法を次に示します。

- DriverManager クラスの getConnection メソッドの，引数 Properties 中のプロパティ HiRDB_for_Java_STATEMENT_COMMIT_BEHAVIOR
- URL 中の STATEMENT_COMMIT_BEHAVIOR
- DataSource 系インタフェースの setStatementCommitBehavior

- DriverManager クラスの getConnection メソッドの，引数 Properties 中のプロパティ HIRDB_CURSOR
- URL 中の HIRDB_CURSOR
- DataSource 系インタフェースの setHiRDBCursorMode
- Connection インタフェースの setHoldability メソッド
- Connection インタフェースの createStatement，prepareStatement メソッドの引数 resultSetHoldability
- SQL 文（UNTIL DISCONNECT の指定）

該当する Statement オブジェクトが SQL を実行中でなく，かつ同一の接続オブジェクトに対してほかのオブジェクトが SQL を実行していないと，HiRDB サーバに対する非同期キャンセルを実行しません。

また，XADataSource を使用した接続の場合は，非同期キャンセルの要求は有効となりません。

(c) executeXXX メソッド実行時の Resultset オブジェクトのクローズ

該当する Statement オブジェクトが生成した ResultSet オブジェクトがクローズされていない状態で executeXXX メソッドを実行すると，以前生成した ResultSet オブジェクトをクローズします。このため，executeXXX メソッド実行後，以前に生成した ResultSet オブジェクトを使用して検索結果を取得しようとすると，SQLException を投入します。SQLException が発生する例を次に示します。

```
Statement st  = con.createStatement();
ResultSet rs1 = st.executeQuery("select * from tb1");
ResultSet rs2 = st.executeQuery("select * from tb2");
rs1.next();  // SQLExceptionを投入する。
rs2.next();
```

(d) 1 コネクション当たりのステートメントオブジェクト数の上限

1 つの Connection オブジェクトに対し，使用中のステートメントオブジェクトの数の合計が 4096 以上となった場合，SQLException を投入します。対象となるオブジェクトを次に示します。

- Statement オブジェクト
- PreparedStatement オブジェクト※1
- CallableStatement オブジェクト※1
- DatabaseMetaData オブジェクトのメソッドから返却された ResultSet オブジェクト※2

注※1

ステートメントプーリング／ステートメントキャッシュ機能を使用している場合，プール／キャッシュされているオブジェクトを含んだ数になります。該当する機能の詳細は，機能を提供する製品のマニュアルを参照してください。

注※2

ResultSet オブジェクトを返却する DatabaseMetaData オブジェクトのメソッドでは，内部的にステートメントオブジェクトを 1 つ使用しています（この場合，ResultSet オブジェクトの close 時に，内部的なステートメントオブジェクトを close します）。

ステートメントオブジェクトを使用するメソッドを次に示します。

- getProcedures
- getProcedureColumns
- getTables
- getSchemas
- getCatalogs
- getTableTypes
- getColumns
- getColumnPrivileges
- getTablePrivileges
- getBestRowIdentifier
- getVersionColumns
- getPrimaryKeys
- getImportedKeys
- getExportedKeys
- getCrossReference
- getTypeInfo
- getIndexInfo
- getUDTs
- getSuperTypes
- getSuperTables
- getAttributes
- getClientInfoProperties
- getFunctionColumns
- getFunctions

17.4.4　PreparedStatement インタフェース

(1)　概要

PreparedStatement インタフェースでは，主に次の機能が提供されます。

- ？パラメタ指定の SQL の実行※
- ？パラメタの設定
- 検索結果としての ResultSet オブジェクトの生成，返却
- 更新結果としての更新行数の返却

注※

出力パラメタは指定できません（指定時の動作は不定です）。

また，PreparedStatement インタフェースは Statement インタフェースのサブインタフェースであるため，Statement インタフェースの機能をすべて継承します。

(2)　メソッド

PreparedStatement インタフェースのメソッド一覧を次の表に示します。なお，表に記載されていないメソッドはサポートしていません。サポートしていないメソッドを指定すると，SQLException を投入します。

表 17–28　PreparedStatement インタフェースのメソッド一覧

メソッド	提供ドライバ			機能
	JDBC2.0	JDBC4.0	JDBC4.3	
addBatch()	○	○	○	この PreparedStatement オブジェクトのバッチに，現在のパラメタセットを追加します。
clearParameters()	○	○	○	現在設定されているパラメタセットの値をすべてクリアします。
execute()	○	○	○	前処理済みの SQL を実行します。
execute(String sql)	○	○	○	指定された SQL 文を実行します。
executeQuery()	○	○	○	前処理済みの検索系 SQL を実行し，その結果の ResultSet オブジェクトを返します。
executeQuery(String sql)	○	○	○	指定された検索系 SQL を実行し，その結果の ResultSet オブジェクトを返します。

メソッド	提供ドライバ			機能
	JDBC2.0	JDBC4.0	JDBC4.3	
executeUpdate()	○	○	○	前処理済みの検索系以外の SQL を実行し，更新行数を返します。
executeUpdate(String sql)	○	○	○	指定された検索系以外の SQL を実行し，更新行数を返します。
getMetaData()	×	○	○	この PreparedStatement オブジェクトの実行時に返される ResultSet のメタ情報を表す ResultSetMetaData を返却します。
getParameterMetaData()	×	○	○	この PreparedStatement オブジェクト内のパラメタのメタ情報を表す ParameterMetaData を返却します。
setArray(int i,Array x)	○	○	○	Array オブジェクトを指定されたパラメタに設定します。
setAsciiStream(int parameterIndex, java.io.InputStream x, int length)	○	○	○	指定された InputStream オブジェクトの持つ値を？パラメタ値に設定します。
setBigDecimal(int parameterIndex, BigDecimal x)	○	○	○	指定された BigDecimal オブジェクトを？パラメタ値に設定します。
setBinaryStream(int parameterIndex, java.io.InputStream x, int length)	○	○	○	指定された InputStream オブジェクトの持つ値を？パラメタ値に設定します。
setBlob(int parameterIndex, Blob x)	○	○	○	指定された Blob オブジェクトの持つ値を？パラメタ値に設定します。
setBoolean(int parameterIndex, boolean x)	○	○	○	指定された boolean 値を？パラメタ値に設定します。
setByte(int parameterIndex, byte x)	○	○	○	指定された byte 値を？パラメタ値に設定します。
setBytes(int parameterIndex, byte x[])	○	○	○	指定された byte 配列を？パラメタ値に設定します。
setCharacterStream(int parameterIndex, Reader reader, int length)	○	○	○	指定された Reader オブジェクトの持つ値を？パラメタ値に設定します。

メソッド	提供ドライバ			機能
	JDBC2.0	JDBC4.0	JDBC4.3	
setDate(int parameterIndex, java.sql.Date x)	○	○	○	指定された java.sql.Date オブジェクトを？パラメタ値に設定します。
setDate(int parameterIndex, java.sql.Date x,Calendar cal)	○	○	○	ローカルタイムで指定された java.sql.Date オブジェクトを，指定されたカレンダーのタイムゾーンの時間に変換し，？パラメタ値に設定します。
setDouble(int parameterIndex, double x)	○	○	○	指定された double 値を？パラメタ値に設定します。
setFloat(int parameterIndex, float x)	○	○	○	指定された float 値を？パラメタ値に設定します。
setInt(int parameterIndex, int x)	○	○	○	指定された int 値を？パラメタ値に設定します。
setLong(int parameterIndex, long x)	○	○	○	指定された long 値を？パラメタ値に設定します。
setNull(int parameterIndex,int sqlType)	○	○	○	指定された？パラメタに NULL 値を設定します。
setObject(int parameterIndex, Object x)	○	○	○	指定されたオブジェクトの持つ値を？パラメタ値に設定します。
setObject(int parameterIndex, Object x, int targetSqlType)	○	○	○	指定されたオブジェクトの持つ値を？パラメタ値に設定します。
setObject(int parameterIndex, Object x, int targetSqlType, int scale)	○	○	○	指定されたオブジェクトの持つ値を？パラメタ値に設定します。
setShort(int parameterIndex, short x)	○	○	○	指定された short 値を？パラメタ値に設定します。
setString(int parameterIndex, String x)	○	○	○	指定された String オブジェクトを？パラメタ値に設定します。
setTime(int parameterIndex, java.sql.Time x)	○	○	○	指定された java.sql.Time オブジェクトを？パラメタ値に設定します。
setTime(int parameterIndex, java.sql.Time x,Calendar cal)	○	○	○	ローカルタイムで指定された java.sql.Time オブジェクトを，指定されたカレンダーのタイムゾーンの時間に変換し，？パラメタ値に設定します。

メソッド	提供ドライバ			機能
	JDBC2.0	JDBC4.0	JDBC4.3	
setTimestamp(int parameterIndex, java.sql.Timestamp x)	○	○	○	指定された java.sql.Timestamp オブジェクトを？パラメタ値に設定します。
setTimestamp(int parameterIndex, java.sql.Timestamp x,Calendar cal)	○	○	○	ローカルタイムで指定された java.sql.Timestamp オブジェクトを，指定されたカレンダーのタイムゾーンの時間に変換し，？パラメタ値に設定します。
executeLargeUpdate()	×	×	○	前処理済みの検索系以外の SQL を実行し，更新行数を返します。

（凡例）

○：提供されます。

×：提供されません。

(a)　addBatch()

【機能】

この PreparedStatement オブジェクトのバッチに，現在のパラメタセットを追加します。

最大 2,147,483,647 個パラメタセットを登録できます。

【形式】

```
public synchronized void addBatch() throws SQLException
```

【引数】

なし。

【戻り値】

なし。

【発生する例外】

次の場合，SQLException を投入します。

- PreparedStatement オブジェクトに対して close()が既に発行されている場合
- この PreparedStatement オブジェクトを生成した Connection オブジェクトに対して，close()が既に発行されている場合
- すべての？パラメタに値がセットされていない場合
- バッチの登録数が 2,147,483,647 個を超える場合

(b) clearParameters()

【機能】

現在設定されているパラメタセットの値をすべてクリアします。

【形式】

```
public synchronized void clearParameters() throws SQLException
```

【引数】

なし。

【戻り値】

なし。

【発生する例外】

次の場合，SQLException を投入します。

- PreparedStatement オブジェクトに対して close()が既に発行されている場合
- この PreparedStatement オブジェクトを生成した Connection オブジェクトに対して，close()が既に発行されている場合

(c) execute()

【機能】

前処理済みの SQL を実行します。

実行結果の ResultSet オブジェクトや更新行数を，PreparedStatement.getResultSet，PreparedStatement.getUpdateCount で取得できます。

【形式】

```
public synchronized boolean execute() throws SQLException
```

【引数】

なし。

【戻り値】

実行した SQL が検索系 SQL の場合は true，そうでない場合は false を返します。

【機能詳細】

前処理済みの SQL を実行します。

ResultSet オブジェクトや更新行数を，PreparedStatement.getResultSet，PreparedStatement.getUpdateCount で取得できます。メソッド実行後の Statement.getResultSet，Statement.getUpdateCount の戻り値については，「execute(String sql)」を参照してください。

【発生する例外】

次の場合，SQLException を投入します。

- PreparedStatement オブジェクトに対して close()が既に発行されている場合

- この PreparedStatement オブジェクトを生成した Connection オブジェクトに対して，close()が既に発行されている場合
- 値を設定していない？パラメタがある場合
- データベースアクセスエラーが発生した場合

(d)　execute(String sql)

【機能】

指定された SQL 文を実行します。実行結果の ResultSet オブジェクトや更新行数を，PreparedStatement.getResultSet，PreparedStatement.getUpdateCount で取得できます。

【形式】

```
public synchronized boolean execute(String sql) throws SQLException
```

【引数】

String sql：

実行する SQL 文

【戻り値】

実行した SQL が検索系 SQL の場合は true，そうでない場合は false を返します。

【機能詳細】

指定された SQL 文を実行します。ResultSet オブジェクトや更新行数を，PreparedStatement.getResultSet，PreparedStatement.getUpdateCount で取得できます。

このメソッド実行後の PreparedStatement.getResultSet，PreparedStatement.getUpdateCount の戻り値については，「実行した SQL と Statement.getResultSet 及び Statement.getUpdateCount の戻り値の関係」を参照してください。

【発生する例外】

次の場合，SQLException を投入します。

- PreparedStatement オブジェクトに対して close()が既に発行されている場合
- この PreparedStatement オブジェクトを生成した Connection オブジェクトに対して，close()が既に発行されている場合
- 引数 sql に null 又は 0 長文字列を指定した場合
- データベースアクセスエラーが発生した場合
- HiRDB SQL Tuning Advisor 用アクセスパス情報ファイルへの出力でエラーが発生した場合（JDBC4.0 以降のときだけ）

(e)　executeQuery()

【機能】

前処理済みの検索系 SQL を実行し，その結果の ResultSet オブジェクトを返します。

【形式】

```
public synchronized ResultSet executeQuery() throws SQLException
```

【引数】

なし。

【戻り値】

実行結果の ResultSet オブジェクト

【機能詳細】

- システムプロパティ HiRDB_for_Java_DAB_EXECUTESQL_NOCHK に TRUE を設定していない場合
  前処理済みの検索系 SQL を実行し，その結果の ResultSet オブジェクトを返します。検索系以外の SQL を実行した場合，SQLException を投入します。
- システムプロパティ HiRDB_for_Java_DAB_EXECUTESQL_NOCHK に TRUE を設定している場合
  前処理済みの SQL を実行し，その結果の ResultSet オブジェクトを返します。検索系以外の SQL を実行した場合，0 列の ResultSet オブジェクトを返します。
  また，Statement.getUpdateCount メソッドを使用して更新行数を取得できます。
  executeQuery メソッドの戻り値と，メソッド実行後に実行する Statement.getUpdateCount メソッドの戻り値については，「executeQuery(String sql)」を参照してください。

【発生する例外】

次の場合，SQLException を投入します。

- PreparedStatement オブジェクトに対して close()が既に発行されている場合
- この PreparedStatement オブジェクトを生成した Connection オブジェクトに対して，close()が既に発行されている場合
- 検索系以外の SQL を指定した場合（システムプロパティ HiRDB_for_Java_DAB_EXECUTESQL_NOCHK に TRUE を設定している場合を除く）
- 値を設定していない？パラメタがある場合
- データベースアクセスエラーが発生した場合

(f)　executeQuery(String sql)

【機能】

指定された検索系 SQL を実行し，その結果の ResultSet オブジェクトを返します。

【形式】

```
public synchronized ResultSet executeQuery(String sql) throws SQLException
```

【引数】

String sql：

実行する SQL 文

【戻り値】

実行結果の ResultSet オブジェクト

【機能詳細】

- システムプロパティ HiRDB_for_Java_DAB_EXECUTESQL_NOCHK に TRUE を設定していない場合

  指定された検索系 SQL を実行し，その結果の ResultSet オブジェクトを返却します。検索結果のない SQL 文（INSERT 文など）の場合は，SQLException を投入します。

- システムプロパティ HiRDB_for_Java_DAB_EXECUTESQL_NOCHK に TRUE を設定している場合

  指定された SQL を実行し，その結果の ResultSet オブジェクトを返却します。検索系以外の SQL を実行した場合，0 列の ResultSet オブジェクトを返却します。また，Statement.getUpdateCount メソッドを使用して更新行数を取得できます。

  executeQuery メソッドの戻り値と，メソッド実行後に実行する Statement.getUpdateCount メソッドの戻り値については，「executeQuery(String sql)」を参照してください。

【発生する例外】

次の場合，SQLException を投入します。

- PreparedStatement オブジェクトに対して close()が既に発行されている場合
- この PreparedStatement オブジェクトを生成した Connection オブジェクトに対して，close()が既に発行されている場合
- 検索系以外の SQL を指定した場合（システムプロパティ HiRDB_for_Java_DAB_EXECUTESQL_NOCHK に TRUE を設定している場合を除く）
- 引数 sql に null 又は 0 長文字列を指定した場合
- データベースアクセスエラーが発生した場合
- HiRDB SQL Tuning Advisor 用アクセスパス情報ファイルへの出力でエラーが発生した場合（JDBC4.0 以降のときだけ）

(g)　executeUpdate()

【機能】

前処理済みの検索系以外の SQL を実行し，更新行数を返します。

【形式】

```
public synchronized int executeUpdate() throws SQLException
```

【引数】

なし。

【戻り値】

次の表に示します。

表 17–29　executeUpdate メソッドの戻り値

実行した SQL の種類		システムプロパティ HiRDB_for_Java_DAB_EXECUTESQL_NOCHK 指定	
		TRUE 以外	TRUE
検索系 SQL		–	-1
検索系以外の SQL	INSERT，UPDATE，DELETE	更新行数	更新行数
	その他	0	0

【機能詳細】

- システムプロパティ HiRDB_for_Java_DAB_EXECUTESQL_NOCHK に TRUE を設定していない場合
  前処理済みの検索系以外の SQL を実行し，更新行数を返します。
  検索系 SQL を実行した場合，SQLException を投入します。
- システムプロパティ HiRDB_for_Java_DAB_EXECUTESQL_NOCHK に TRUE を設定している場合
  前処理済みの SQL を実行します。
  戻り値が-1 の場合，Statement.getResultSet メソッドを使用して ResultSet オブジェクトを取得できます。executeUpdate メソッド実行後に実行する Statement.getResultSet メソッドの戻り値については，「executeUpdate(String sql)」を参照してください。

【発生する例外】

次の場合，SQLException を投入します。

- オブジェクトに対して close()が既に発行されている場合
- このオブジェクトを生成した Connection オブジェクトに対して，close()が既に発行されている場合
- 検索系 SQL を実行した場合（システムプロパティ HiRDB_for_Java_DAB_EXECUTESQL_NOCHK に TRUE を設定している場合を除く）
- 値を設定していない？パラメタがある場合
- データベースアクセスエラーが発生した場合

(h)　executeUpdate(String sql)

【機能】

指定された検索系以外の SQL を実行し，更新行数を返します。

【形式】

```
public synchronized int executeUpdate(String sql) throws SQLException
```

【引数】

String sql：

実行する SQL 文

【戻り値】

次の表に示します。

表 17–30　executeUpdate メソッドの戻り値

実行した SQL の種類		システムプロパティ HiRDB_for_Java_DAB_EXECUTESQL_NOCHK 指定	
		TRUE 以外	TRUE
検索系 SQL		–	-1
検索系以外の SQL	INSERT，UPDATE，DELETE	更新行数	更新行数
	その他	0	0

【機能詳細】

- システムプロパティ HiRDB_for_Java_DAB_EXECUTESQL_NOCHK に TRUE を設定していない場合
  指定された検索系以外の SQL を実行し，更新行数を返却します。
  検索結果を返却する SQL 文（SELECT 文）の場合は，SQLException を投入します。
- システムプロパティ HiRDB_for_Java_DAB_EXECUTESQL_NOCHK に TRUE を設定している場合
  指定された SQL を実行します。
  戻り値が-1 の場合，Statement.getResultSet メソッドを使用して ResultSet オブジェクトを取得できます。executeUpdate メソッド実行後に実行する Statement.getResultSet メソッドの戻り値については，「executeUpdate(String sql)」を参照してください。

【発生する例外】

次の場合，SQLException を投入します。

- PreparedStatement オブジェクトに対して close()が既に発行されている場合
- この PreparedStatement オブジェクトを生成した Connection オブジェクトに対して，close()が既に発行されている場合
- 検索系 SQL を指定した場合（システムプロパティ HiRDB_for_Java_DAB_EXECUTESQL_NOCHK に TRUE を設定している場合を除く）
- 引数 sql に null 又は 0 長文字列を指定した場合
- データベースアクセスエラーが発生した場合
- HiRDB SQL Tuning Advisor 用アクセスパス情報ファイルへの出力でエラーが発生した場合（JDBC4.0 以降のときだけ）

(i)　getMetaData()

【機能】

この PreparedStatement オブジェクトの実行時に返される ResultSet のメタ情報を表す ResultSetMetaData を返却します。

【形式】

```
public synchronized ResultSetMetaData getMetaData() throws SQLException
```

【引数】

なし。

【戻り値】

次の表に示します。

表 17–31　getMetaData メソッドの戻り値

前処理済み SQL の種類	getMetaData メソッドの戻り値
検索系 SQL	実行時に返される ResultSet のメタ情報を表す ResultSetMetaData
上記以外	null

注

引数に SQL 文を持つ executeXXX メソッドを実行した場合，引数を持たない executeXXX メソッドを実行するまで，引数に指定された SQL 文に対応するメタ情報を返却します。

(例)

```
pstmt = Connection.prepareStatement(<SQL1>)
pstmt.getMetaData() ..........SQL1の結果セットのメタ情報を返却
pstmt.execute()
pstmt.getMetaData() ..........SQL1の結果セットのメタ情報を返却
pstmt.executeQuery(<SQL2>)
pstmt.getMetaData() ..........SQL2の結果セットのメタ情報を返却
pstmt.execute()
pstmt.getMetaData() ..........SQL1の結果セットのメタ情報を返却
```

【機能詳細】

この PreparedStatement オブジェクトの実行時に返される ResultSet のメタ情報を表す ResultSetMetaData を返却します。

【発生する例外】

次の場合，SQLException を投入します。

- PreparedStatement オブジェクトに対して close()が既に発行されている場合
- この PreparedStatement オブジェクトを生成した Connection オブジェクトに対して close()が既に発行されている場合

(j)　getParameterMetaData()

【機能】

この PreparedStatement オブジェクト内のパラメタのメタ情報を表す ParameterMetaData を返却します。

【形式】

```
public ParameterMetaData getParameterMetaData() throws SQLException
```

【引数】

なし。

【戻り値】

この PreparedStatement オブジェクト内のパラメタのメタ情報を表す ParameterMetaData

【機能詳細】

この PreparedStatement オブジェクト内のパラメタのメタ情報を表す ParameterMetaData を返却します。返却する ParameterMetaData は，Connection.prepareStatement()又は Connection.prepareCall()実行時点でサーバから取得したパラメタのメタ情報となります。

【発生する例外】

次の場合，SQLException を投入します。

- PreparedStatement オブジェクトに対して close()が既に発行されている場合
- この PreparedStatement オブジェクトを生成した Connection オブジェクトに対して close()が既に発行されている場合

(k)　setArray(int i,Array x)

【機能】

Array オブジェクトを指定されたパラメタに設定します。

【形式】

```
public void setArray(int i,Array x) throws SQLException
```

【引数】

int i：

？パラメタの番号。最初のパラメタは 1，2 番目のパラメタは 2，...と指定します。

Array x：

?パラメタに設定する Array オブジェクト

【戻り値】

なし。

【機能詳細】

Array オブジェクトを，指定されたパラメタ番号のパラメタに設定します。

Array オブジェクトの getArray()で取得した Object 配列のデータ型と，Array オブジェクトの getBaseType()で取得したデータ型が対応していない場合は，SQLException となります。

Array オブジェクトの getArray()で取得した Object 配列のデータ型と，Array オブジェクトの getBaseType()で取得したデータ型の対応を次に示します。

getBaseType()で取得したデータ型	getArray()で取得した Object 配列のデータ型
java.sql.Types.SMALLINT	short[],java.lang.Short[]
java.sql.Types.INTEGER	int[],java.lang.Integer[]
java.sql.Types.REAL	float[],java.lang.Float[]
java.sql.Types.FLOAT	double[],java.lang.Double[]
java.sql.Types.Decimal	java.math.BigDecimal[]
java.sql.Types.CHAR	java.lang.String[]
java.sql.Types.VARCHAR	java.lang.String[]
java.sql.Types.DATE	java.sql.Date
java.sql.Types.TIME	java.sql.Time
java.sql.Types.TIMESTAMP	java.sql.Timestamp

引数 i 及び引数 x と設定される値の関係を次に示します。

引数 i	引数 x	getArray で取り出した Object 配列の要素数	Object 配列の各要素	HiRDB に設定される繰返し列の要素
存在する?パラメタ番号	!=null	0<要素数<=30000	すべての要素が null	すべての要素が null である繰返し列
			上記以外	上記以外の繰返し列
		要素数>30000	−	SQLException
		0	−	要素数 0 の繰返し列
	null	−	−	列全体が null
存在するが，繰返し列ではない?パラメタ番号	−	−	−	SQLException
存在しない?パラメタ番号	SQLException			

（凡例） −：該当しません。

【発生する例外】

次の場合，SQLException を投入します。

- PreparedStatement オブジェクトに対して close()が既に発行されている場合
- この PreparedStatement オブジェクトを生成した Connection オブジェクトに対して close()が既に発行されている場合
- 引数 i に存在しない？パラメタの番号を指定した場合
- 引数 i で指定したパラメタ番号の列が，HiRDB の繰返し列ではない場合
- 引数 x に指定した Array オブジェクトから取り出した Object 配列のデータ型が，？パラメタの列のデータ型に変換できない場合

- 引数 x に指定した Array オブジェクトから取り出した Object 配列の長さが 30000 を超えている（配列要素数が 30000 を超えている）場合

(l)　setAsciiStream(int parameterIndex, java.io.InputStream x, int length)

【機能】

指定された InputStream オブジェクトの持つ値を？パラメタ値に設定します。

【形式】

```
public synchronized void setAsciiStream(int parameterIndex, java.io.InputStream x, int le
ngth) throws SQLException
```

【引数】

int parameterIndex：

？パラメタの番号

java.io.InputStream x：

？パラメタに設定する値を持つ InputStream オブジェクト

int length：

設定するバイト数

【戻り値】

なし。

【機能詳細】

指定された InputStream オブジェクトの持つ値を？パラメタ値に設定します。

このメソッドは，x からの入力が終わった後でも，x に対して close()メソッドを実行しません。

？パラメタの HiRDB のデータ型が[M ｜ N][VAR]CHAR，BINARY，BLOB 以外の場合は，SQLException となります。

【発生する例外】

次の場合，SQLException を投入します。

- PreparedStatement オブジェクトに対して close()が既に発行されている場合
- この PreparedStatement オブジェクトを生成した Connection オブジェクトに対して close()が既に発行されている場合
- length に 0 未満の値が指定された場合
- 存在しない？パラメタの番号を指定した場合
- ？パラメタの HiRDB データ型がこのメソッドでは設定できないデータ型の場合
- 指定された値が列のデータ型の範囲外，又は変換できない形式の場合
- 指定した？パラメタが OUT パラメタである場合

(m)　setBigDecimal(int parameterIndex, BigDecimal x)

【機能】

指定された BigDecimal オブジェクトを？パラメタ値に設定します。

【形式】

```
public synchronized void setBigDecimal(int parameterIndex, BigDecimal x) throws SQLExcept
ion
```

【引数】

int parameterIndex：

？パラメタの番号

BigDecimal x：

？パラメタに設定する BigDecimal オブジェクト

【戻り値】

なし。

【発生する例外】

次の場合，SQLException を投入します。

- PreparedStatement オブジェクトに対して close()が既に発行されている場合
- この PreparedStatement オブジェクトを生成した Connection オブジェクトに対して close()が既に発行されている場合
- 存在しない？パラメタの番号を指定した場合
- ？パラメタの HiRDB データ型がこのメソッドでは設定できないデータ型の場合
- 指定された値が列のデータ型の範囲外，又は変換できない形式の場合
- 指定した？パラメタが OUT パラメタである場合

(n)　setBinaryStream(int parameterIndex, java.io.InputStream x, int length)

【機能】

指定された InputStream オブジェクトの持つ値を？パラメタ値に設定します。

【形式】

```
public synchronized void setBinaryStream(int parameterIndex, java.io.InputStream x, int l
ength) throws SQLException
```

【引数】

int parameterIndex：

？パラメタの番号

java.io.InputStream x：

？パラメタに設定する値を持つ InputStream オブジェクト

int length：

設定するバイト数

【戻り値】

なし。

【機能詳細】

指定された InputStream オブジェクトの持つ値を？パラメタ値に設定します。

このメソッドは，x からの入力が終わった後でも，x に対して close()メソッドを実行しません。

？パラメタの HiRDB のデータ型が BINARY，BLOB 以外の場合は，SQLException となります。

【発生する例外】

次の場合，SQLException を投入します。

- PreparedStatement オブジェクトに対して close()が既に発行されている場合
- この PreparedStatement オブジェクトを生成した Connection オブジェクトに対して close()が既に発行されている場合
- length に 0 未満の値が指定された場合
- 存在しない？パラメタの番号を指定した場合
- ？パラメタの HiRDB データ型がこのメソッドでは設定できないデータ型の場合
- 指定された値が列のデータ型の範囲外，又は変換できない形式の場合
- 指定した？パラメタが OUT パラメタである場合

(o)　setBlob(int parameterIndex, Blob x)

【機能】

指定された Blob オブジェクトの持つ値を？パラメタ値に設定します。

【形式】

```
public synchronized void setBlob(int parameterIndex, Blob x) throws SQLException
```

【引数】

int parameterIndex：

？パラメタの番号

Blob x：

？パラメタに設定する値を持つ Blob オブジェクト

【戻り値】

なし。

【機能詳細】

指定された InputStream オブジェクトの持つ値を？パラメタ値に設定します。

？パラメタの HiRDB のデータ型が BINARY，BLOB 以外の場合は，SQLException となります。

【発生する例外】

次の場合，SQLException を投入します。

- PreparedStatement オブジェクトに対して close()が既に発行されている場合
- この PreparedStatement オブジェクトを生成した Connection オブジェクトに対して close()が既に発行されている場合
- 存在しない？パラメタの番号を指定した場合
- ？パラメタの HiRDB データ型がこのメソッドでは設定できないデータ型の場合
- 指定された値が列のデータ型の範囲外，又は変換できない形式の場合
- 指定した？パラメタが OUT パラメタである場合

(p)　setBoolean(int parameterIndex, boolean x)

【機能】

指定された boolean 値を？パラメタ値に設定します。

【形式】

```
public synchronized void setBoolean(int parameterIndex, boolean x) throws SQLException
```

【引数】

int parameterIndex：

？パラメタの番号

boolean x：

？パラメタに設定する値

【戻り値】

なし。

【機能詳細】

指定された boolean 値を？パラメタ値に設定します。

parameterIndex で指定した?パラメタの HiRDB のデータ型が CHAR，MCHAR，NCHAR，VARCHAR，MVARCHAR，及び NVARHAR の場合，?パラメタへの設定値は x が true のときは"true"，false のときは"false△"（△は半角スペース）となります。

【発生する例外】

次の場合，SQLException を投入します。

- PreparedStatement オブジェクトに対して close()が既に発行されている場合

- この PreparedStatement オブジェクトを生成した Connection オブジェクトに対して close()が既に発行されている場合
- 存在しない？パラメタの番号を指定した場合
- ？パラメタの HiRDB データ型がこのメソッドでは設定できないデータ型の場合
- 指定した？パラメタが OUT パラメタである場合

(q)　setByte(int parameterIndex, byte x)

【機能】

指定された byte 値を？パラメタ値に設定します。

【形式】

```
public synchronized void setByte(int parameterIndex, byte x) throws SQLException
```

【引数】

int parameterIndex：

？パラメタの番号

byte x：

？パラメタに設定する値

【戻り値】

なし。

【発生する例外】

次の場合，SQLException を投入します。

- PreparedStatement オブジェクトに対して close()が既に発行されている場合
- この PreparedStatement オブジェクトを生成した Connection オブジェクトに対して close()が既に発行されている場合
- 存在しない？パラメタの番号を指定した場合
- ？パラメタの HiRDB データ型がこのメソッドでは設定できないデータ型の場合
- 指定した？パラメタが OUT パラメタである場合

(r)　setBytes(int parameterIndex, byte x[])

【機能】

指定された byte 配列を？パラメタ値に設定します。

【形式】

```
public synchronized void setBytes(int parameterIndex, byte x[]) throws SQLException
```

【引数】

int parameterIndex：

？パラメタの番号

byte x[]：

？パラメタに設定する値を持つ byte 配列

【戻り値】

なし。

【機能詳細】

このメソッドでは，byte 配列のコピーをしないで，参照だけを保持します。そのため，executeXXX メソッドの実行前に byte 配列の値を変更した場合は，変更後の値が設定値となります。

【発生する例外】

次の場合，SQLException を投入します。

- PreparedStatement オブジェクトに対して close()が既に発行されている場合
- この PreparedStatement オブジェクトを生成した Connection オブジェクトに対して close()が既に発行されている場合
- 存在しない？パラメタの番号を指定した場合
- ？パラメタの HiRDB データ型がこのメソッドでは設定できないデータ型（BINARY 型及び BLOB 型以外のデータ型）の場合
- 指定された値が列のデータ型の範囲外，又は変換できない形式の場合
- 指定した？パラメタが OUT パラメタである場合

(s)　setCharacterStream(int parameterIndex, Reader reader, int length)

【機能】

指定された Reader オブジェクトの持つ値を？パラメタ値に設定します。

【形式】

```
public synchronized void setCharacterStream(int parameterIndex,
Reader x, int length) throws SQLException
```

【引数】

int parameterIndex：

？パラメタの番号

Reader reader：

？パラメタに設定する値を持つ Reader オブジェクト

int length：

文字数

【戻り値】

なし。

【機能詳細】

指定された Reader オブジェクトの持つ値を？パラメタ値に設定します。

？パラメタの HiRDB のデータ型が[M ｜ N][VAR]CHAR，BINARY，BLOB 以外の場合は，SQLException となります。

【発生する例外】

次の場合，SQLException を投入します。

- PreparedStatement オブジェクトに対して close()が既に発行されている場合
- この PreparedStatement オブジェクトを生成した Connection オブジェクトに対して close()が既に発行されている場合
- length に 0 未満の値が指定された場合
- 存在しない？パラメタの番号を指定した場合
- ？パラメタの HiRDB データ型がこのメソッドでは設定できないデータ型の場合
- エンコードに失敗した場合
- 指定した？パラメタが OUT パラメタである場合

(t)　setDate(int parameterIndex, java.sql.Date x)

【機能】

指定された java.sql.Date オブジェクトを？パラメタ値に設定します。

【形式】

```
public synchronized void setDate(int parameterIndex, java.sql.Date x) throws SQLException
```

【引数】

int parameterIndex：

？パラメタの番号

java.sql.Date x：

？パラメタに設定する値を持つ java.sql.Date オブジェクト

【戻り値】

なし。

【発生する例外】

次の場合，SQLException を投入します。

- PreparedStatement オブジェクトに対して close()が既に発行されている場合
- この PreparedStatement オブジェクトを生成した Connection オブジェクトに対して close()が既に発行されている場合

- 存在しない？パラメタの番号を指定した場合
- ？パラメタの HiRDB データ型がこのメソッドでは設定できないデータ型の場合
- 指定された値が列のデータ型の範囲外，又は変換できない形式の場合
- 指定した？パラメタが OUT パラメタである場合

(u) setDate(int parameterIndex, java.sql.Date x,Calendar cal)

【機能】

ローカルタイムで指定された java.sql.Date オブジェクトを，指定されたカレンダーのタイムゾーンの時間に変換し，？パラメタ値に設定します。

【形式】

```
public synchronized void setDate(int parameterIndex, java.sql.Date x,Calendar cal) throws
 SQLException
```

【引数】

int parameterIndex：

？パラメタの番号

java.sql.Date x：

？パラメタに設定する値を持つ java.sql.Date オブジェクト

Calendar cal：

データベースに格納する値のタイムゾーンが設定されたカレンダー。null が指定された場合，JavaVM のデフォルトのタイムゾーンのカレンダーが指定されたとみなされます。

【戻り値】

なし。

【発生する例外】

次の場合，SQLException を投入します。

- PreparedStatement オブジェクトに対して close()が既に発行されている場合
- この PreparedStatement オブジェクトを生成した Connection オブジェクトに対して close()が既に発行されている場合
- 存在しない？パラメタの番号を指定した場合
- ？パラメタの HiRDB データ型がこのメソッドでは設定できないデータ型の場合
- 指定された値が列のデータ型の範囲外，又は変換できない形式の場合
- 指定した？パラメタが OUT パラメタである場合

(v)　setDouble(int parameterIndex, double x)

【機能】

指定された double 値を？パラメタ値に設定します。

【形式】

```
public synchronized void setDouble(int parameterIndex, double x) throws SQLException
```

【引数】

int parameterIndex：

？パラメタの番号

double x：

？パラメタに設定する値

【戻り値】

なし。

【発生する例外】

次の場合，SQLException を投入します。

- PreparedStatement オブジェクトに対して close()が既に発行されている場合
- この PreparedStatement オブジェクトを生成した Connection オブジェクトに対して close()が既に発行されている場合
- 存在しない？パラメタの番号を指定した場合
- ？パラメタの HiRDB データ型がこのメソッドでは設定できないデータ型の場合
- 指定された値が列のデータ型の範囲外，又は変換できない形式の場合
- 指定した？パラメタが OUT パラメタである場合

(w)　setFloat(int parameterIndex, float x)

【機能】

指定された float 値を？パラメタ値に設定します。

【形式】

```
public synchronized void setFloat(int parameterIndex, float x) throws SQLException
```

【引数】

int parameterIndex：

？パラメタの番号

float x：

？パラメタに設定する値

【戻り値】

なし。

【発生する例外】

次の場合，SQLException を投入します。

- PreparedStatement オブジェクトに対して close()が既に発行されている場合
- この PreparedStatement オブジェクトを生成した Connection オブジェクトに対して close()が既に発行されている場合
- 存在しない？パラメタの番号を指定した場合
- ？パラメタの HiRDB データ型がこのメソッドでは設定できないデータ型の場合
- 指定された値が列のデータ型の範囲外，又は変換できない形式の場合
- 指定した？パラメタが OUT パラメタである場合

(x)　setInt(int parameterIndex, int x)

【機能】

指定された int 値を？パラメタ値に設定します。

【形式】

```
public synchronized void setInt(int parameterIndex, int x) throws SQLException
```

【引数】

int parameterIndex：

？パラメタの番号

int x：

？パラメタに設定する値

【戻り値】

なし。

【発生する例外】

次の場合，SQLException を投入します。

- PreparedStatement オブジェクトに対して close()が既に発行されている場合
- この PreparedStatement オブジェクトを生成した Connection オブジェクトに対して close()が既に発行されている場合
- 存在しない？パラメタの番号を指定した場合
- ？パラメタの HiRDB データ型がこのメソッドでは設定できないデータ型の場合
- 指定された値が列のデータ型の範囲外，又は変換できない形式の場合
- 指定した？パラメタが OUT パラメタである場合

(y)　setLong(int parameterIndex, long x)

【機能】

指定された long 値を？パラメタ値に設定します。

【形式】

```
public synchronized void setLong(int parameterIndex, long x) throws SQLException
```

【引数】

int parameterIndex：

？パラメタの番号

long x：

？パラメタに設定する値

【戻り値】

なし。

【発生する例外】

次の場合，SQLException を投入します。

- PreparedStatement オブジェクトに対して close()が既に発行されている場合
- この PreparedStatement オブジェクトを生成した Connection オブジェクトに対して close()が既に発行されている場合
- 存在しない？パラメタの番号を指定した場合
- ？パラメタの HiRDB データ型がこのメソッドでは設定できないデータ型の場合
- 指定された値が列のデータ型の範囲外，又は変換できない形式の場合
- 指定した？パラメタが OUT パラメタである場合

(z)　setNull(int parameterIndex,int sqlType)

【機能】

指定された？パラメタに NULL 値を設定します。

引数 sqlType はこのドライバでは無視します。

【形式】

```
public synchronized void setNull(int parameterIndex,int sqlType) throws SQLException
```

【引数】

int parameterIndex：

？パラメタの番号

int sqlType：

JDBC の SQL データ型

【戻り値】

なし。

【発生する例外】

次の場合，SQLException を投入します。

- PreparedStatement オブジェクトに対して close()が既に発行されている場合
- この PreparedStatement オブジェクトを生成した Connection オブジェクトに対して close()が既に発行されている場合
- 存在しない？パラメタの番号を指定した場合
- 指定した？パラメタが OUT パラメタである場合

(aa)　setObject(int parameterIndex, Object x)

【機能】

指定されたオブジェクトの持つ値を？パラメタ値に設定します。

【形式】

```
public synchronized void setObject(int parameterIndex, Object x) throws SQLException
```

【引数】

int parameterIndex：

？パラメタの番号

Object x：

？パラメタに設定する値を持つオブジェクト

【戻り値】

なし。

【機能詳細】

指定されたオブジェクトの持つ値を？パラメタ値に設定します。

parameterIndex で指定した?パラメタの型が HiRDB の CHAR，MCHAR，NCHAR，VARCHAR，MVARCHAR，又は NVARHAR で，x が Boolean オブジェクトの場合，?パラメタへの設定値は x が true のときは"true"，false のときは"false△"（△は半角スペース）となります。

x が byte 配列の場合，byte 配列のコピーをしないで，参照だけを保持します。そのため，executeXXX メソッドの実行前に byte 配列の値を変更した場合は，変更後の値が設定値となります。

【発生する例外】

次の場合，SQLException を投入します。

- PreparedStatement オブジェクトに対して close()が既に発行されている場合
- この PreparedStatement オブジェクトを生成した Connection オブジェクトに対して close()が既に発行されている場合

- 存在しない？パラメタの番号を指定した場合
- ？パラメタの HiRDB データ型がこのメソッドでは設定できないデータ型の場合
- 指定された値が列のデータ型の範囲外，又は変換できない形式の場合
- 指定した？パラメタが OUT パラメタである場合

(ab)　setObject(int parameterIndex, Object x, int targetSqlType)

【機能】

指定されたオブジェクトの持つ値を？パラメタ値に設定します。

【形式】

```
public synchronized void setObject(int parameterIndex, Object x, int targetSqlType) throw
s SQLException
```

【引数】

int parameterIndex：

？パラメタの番号

Object x：

？パラメタに設定する値を持つオブジェクト

int targetSqlType：

JDBC の SQL データ型

【戻り値】

なし。

【機能詳細】

指定されたオブジェクトの持つ値を？パラメタ値に設定します。

targetSqlType が java.sql.Types.CHAR，java.sql.Types.VARCHAR，又は java.sql.Types.LONGVARCHAR で，x が Boolean オブジェクトの場合，?パラメタへの設定値は x が true のときは”1”,”0”になります。

なお，HiRDB データ型である NCHAR，NVARCHAR 型の?パラメタへの設定値が”1”,”0”になる場合，SQLException を投入します。

x が byte 配列の場合，byte 配列のコピーをしないで，参照だけを保持します。そのため，executeXXX メソッドの実行前に byte 配列の値を変更した場合は，変更後の値が設定値となります。

【発生する例外】

次の場合，SQLException を投入します。

- PreparedStatement オブジェクトに対して close()が既に発行されている場合
- この PreparedStatement オブジェクトを生成した Connection オブジェクトに対して close()が既に発行されている場合
- 存在しない？パラメタの番号を指定した場合

- ？パラメタの HiRDB データ型がこのメソッドでは設定できないデータ型の場合
- 指定された値が列のデータ型の範囲外，又は変換できない形式の場合
- targetSqlType が次のどれかの場合
  Types.ARRAY，Types.CLOB，Types.REF，Types.STRUCT
- 指定した？パラメタが OUT パラメタである場合

(ac)　setObject(int parameterIndex, Object x, int targetSqlType, int scale)

【機能】

指定されたオブジェクトの持つ値を？パラメタ値に設定します。

【形式】

```
public synchronized void setObject(int parameterIndex, Object x, int targetSqlType, int s
cale) throws SQLException
```

【引数】

int parameterIndex：

？パラメタの番号

Object x：

？パラメタに設定する値を持つオブジェクト

int targetSqlType：

JDBC の SQL データ型

int scale：

位取り（指定値は無視します）

【戻り値】

なし。

【機能詳細】

指定されたオブジェクトの持つ値を？パラメタ値に設定します。

targetSqlType が java.sql.Types.CHAR，java.sql.Types.VARCHAR，又は java.sql.Types.LONGVARCHAR で，x が Boolean オブジェクトの場合，?パラメタへの設定値は x が true のときは”1”,”0”になります。

なお，HiRDB データ型である NCHAR，NVARCHAR 型の?パラメタへの設定値が”1”,”0”になる場合，SQLException を投入します。

x が byte 配列の場合，byte 配列のコピーをしないで，参照だけを保持します。そのため，executeXXX メソッドの実行前に byte 配列の値を変更した場合は，変更後の値が設定値となります。

【発生する例外】

次の場合，SQLException を投入します。

- PreparedStatement オブジェクトに対して close()が既に発行されている場合

- この PreparedStatement オブジェクトを生成した Connection オブジェクトに対して close()が既に発行されている場合
- 存在しない？パラメタの番号を指定した場合
- ？パラメタの HiRDB データ型がこのメソッドでは設定できないデータ型の場合
- 指定された値が列のデータ型の範囲外，又は変換できない形式の場合
- targetSqlType が次のどれかの場合
  Types.ARRAY，Types.CLOB，Types.REF，Types.STRUCT
- 指定した？パラメタが OUT パラメタである場合

(ad)　setShort(int parameterIndex, short x)

【機能】

指定された short 値を？パラメタ値に設定します。

【形式】

```
public synchronized void setShort(int parameterIndex, short x) throws SQLException
```

【引数】

int parameterIndex：

？パラメタの番号

short x：

？パラメタに設定する値

【戻り値】

なし。

【発生する例外】

次の場合，SQLException を投入します。

- PreparedStatement オブジェクトに対して close()が既に発行されている場合
- この PreparedStatement オブジェクトを生成した Connection オブジェクトに対して close()が既に発行されている場合
- 存在しない？パラメタの番号を指定した場合
- ？パラメタの HiRDB データ型がこのメソッドでは設定できないデータ型の場合
- 指定した？パラメタが OUT パラメタである場合

(ae)　setString(int parameterIndex, String x)

【機能】

指定された String オブジェクトを？パラメタ値に設定します。

【形式】

```
public synchronized void setString(int parameterIndex, String x) throws SQLException
```

【引数】

int parameterIndex：

？パラメタの番号

String x：

？パラメタに設定する値を持つ String オブジェクト

【戻り値】

なし。

【機能詳細】

引数 x で指定された String オブジェクトを？パラメタに設定します。

該当するインスタンスが CallableStatement で，かつ引数 x の指定値が 0 長文字列の場合の，？パラメタ設定値を次の表に示します。

表 17–32　？パラメタの設定値

？パラメタのデータ型	？パラメタの設定値
[M ｜ N][VAR]CHAR	• システムプロパティ HiRDB_for_Java_DAB_CONVERT_NULL に TRUE を設定している場合 null • 上記以外 0 長文字列
BINARY，又は BLOB	0 長文字列
その他	null

【発生する例外】

次の場合，SQLException を投入します。

- PreparedStatement オブジェクトに対して close()が既に発行されている場合
- この PreparedStatement オブジェクトを生成した Connection オブジェクトに対して close()が既に発行されている場合
- 存在しない？パラメタの番号を指定した場合
- 指定された値が列のデータ型の範囲外，又は変換できない形式の場合
- エンコードに失敗した場合
- 指定した？パラメタが OUT パラメタである場合

(af)　setTime(int parameterIndex, java.sql.Time x)

【機能】

指定された java.sql.Time オブジェクトを？パラメタ値に設定します。

【形式】

```
public synchronized void setTime(int parameterIndex, java.sql.Time x) throws SQLException
```

【引数】

int parameterIndex：

？パラメタの番号

java.sql.Time x：

？パラメタに設定する値を持つ java.sql.Time オブジェクト

【戻り値】

なし。

【発生する例外】

次の場合，SQLException を投入します。

- PreparedStatement オブジェクトに対して close()が既に発行されている場合
- この PreparedStatement オブジェクトを生成した Connection オブジェクトに対して close()が既に発行されている場合
- 存在しない？パラメタの番号を指定した場合
- ？パラメタの HiRDB データ型がこのメソッドでは設定できないデータ型の場合
- 指定した？パラメタが OUT パラメタである場合

(ag)　setTime(int parameterIndex, java.sql.Time x,Calendar cal)

【機能】

ローカルタイムで指定された java.sql.Time オブジェクトを，指定されたカレンダーのタイムゾーンの時間に変換し，？パラメタ値に設定します。

【形式】

```
public synchronized void setTime(int parameterIndex, java.sql.Time x,Calendar cal) throws
 SQLException
```

【引数】

int parameterIndex：

？パラメタの番号

java.sql.Time x：

？パラメタに設定する値を持つ java.sql.Time オブジェクト

Calendar cal：

データベースに格納する値のタイムゾーンが設定されたカレンダー。null が指定された場合，JavaVM のデフォルトのタイムゾーンのカレンダーが指定されたとします。

【戻り値】

なし。

【発生する例外】

次の場合，SQLException を投入します。

- PreparedStatement オブジェクトに対して close()が既に発行されている場合
- この PreparedStatement オブジェクトを生成した Connection オブジェクトに対して close()が既に発行されている場合
- 存在しない？パラメタの番号を指定した場合
- ？パラメタの HiRDB データ型がこのメソッドでは設定できないデータ型の場合
- 指定した？パラメタが OUT パラメタである場合

(ah)　setTimestamp(int parameterIndex, java.sql.Timestamp x)

【機能】

指定された java.sql.Timestamp オブジェクトを？パラメタ値に設定します。

【形式】

```
public synchronized void setTimestamp(int parameterIndex, java.sql.Timestamp x) throws SQ
LException
```

【引数】

int parameterIndex：

？パラメタの番号

java.sql.Timestamp x：

？パラメタに設定する値を持つ java.sql.Timestamp オブジェクト

【戻り値】

なし。

【発生する例外】

次の場合，SQLException を投入します。

- PreparedStatement オブジェクトに対して close()が既に発行されている場合
- この PreparedStatement オブジェクトを生成した Connection オブジェクトに対して close()が既に発行されている場合
- 存在しない？パラメタの番号を指定した場合
- ？パラメタの HiRDB データ型がこのメソッドでは設定できないデータ型の場合

- 指定された値が列のデータ型の範囲外，又は変換できない形式の場合
- 指定した？パラメタが OUT パラメタである場合

(ai)　setTimestamp(int parameterIndex, java.sql.Timestamp x,Calendar cal)

【機能】

ローカルタイムで指定された java.sql.Timestamp オブジェクトを，指定されたカレンダーのタイムゾーンの時間に変換し，？パラメタ値に設定します。

【形式】

```
public synchronized void setTimestamp(int parameterIndex, java.sql.Timestamp x,Calendar cal) throws SQLException
```

【引数】

int parameterIndex：

？パラメタの番号

java.sql.Timestamp x：

？パラメタに設定する値を持つ java.sql.Timestamp オブジェクト

Calendar cal：

データベースに格納する値のタイムゾーンが設定されたカレンダー。null が指定された場合，JavaVM のデフォルトのタイムゾーンのカレンダーが指定されたとします。

【戻り値】

なし。

【発生する例外】

次の場合，SQLException を投入します。

- PreparedStatement オブジェクトに対して close()が既に発行されている場合
- この PreparedStatement オブジェクトを生成した Connection オブジェクトに対して close()が既に発行されている場合
- 存在しない？パラメタの番号を指定した場合
- ？パラメタの HiRDB データ型がこのメソッドでは設定できないデータ型の場合
- 指定された値が列のデータ型の範囲外，又は変換できない形式の場合
- 指定した？パラメタが OUT パラメタである場合

(aj)　executeLargeUpdate()

【機能】

前処理済みの検索系以外の SQL を実行し，更新行数を返します。更新行数が 2 ギガバイトを超える場合は executeLargeUpdate()を使用してください。

【形式】

```
public synchronized long executeLargeUpdate() throws SQLException
```

【引数】

なし。

【戻り値】

「executeUpdate()」を参照してください。

【機能詳細】

「executeUpdate()」を参照してください。

【発生する例外】

「executeUpdate()」を参照してください。

(3)　パッケージ名称及びクラス名称

このインタフェースを実装するパッケージ名称とクラス名称を次に示します。

パッケージ名称：**JP.co.Hitachi.soft.HiRDB.JDBC**

クラス名称：**PrdbPreparedStatement**

(4)　注意事項

PreparedStatement インタフェースは Statement インタフェースのサブインタフェースであるため，Statement インタフェースの注意事項はすべて該当します。

ここでは，それ以外の PreparedStatement インタフェースの注意事項を次に示します。

(a)　？パラメタの設定

- setXXX メソッドによってマッピングできるかどうかについては，「？パラメタ設定時のマッピング」を参照してください。
- setXXX メソッドに指定した列番号又は列名称が存在しない場合は，SQLException を投入します。
- setXXX メソッドに指定した値が，対応する？パラメタのデータ型が表現できる値の範囲を超える場合，オーバフローによって SQLException を投入します。オーバフローが発生する可能性のある setXXX メソッドと HiRDB のデータ型の組み合わせについては，「オーバフローの扱い」を参照してください。
- setXXX メソッドで指定した値は，次の操作のどれかを実行するまで有効です。
  - 該当する PreparedStatement オブジェクトに対して，clearParameters メソッドを実行する。
  - 該当する PreparedStatement オブジェクトに対して，setXXX メソッドを実行し，かつ設定対象の？パラメタは同じである。
  - 該当する PreparedStatement オブジェクトに対して，close メソッドを実行する。

(b)　コミット，ロールバックをわたる SQL の前処理結果保持

コミット，ロールバックをわたる SQL の前処理結果保持については，「HIRDB_CURSOR 及び STATEMENT_COMMIT_BEHAVIOR 指定時の注意事項」を参照してください。

(c)　HiRDB の DECIMAL 型の？パラメタに対する値指定

HiRDB の DECIMAL 型の？パラメタに対して setXXX メソッドで値を指定する場合，？パラメタの精度及び位取りと値の精度及び位取りが一致していないときの動作を次に示します。

実際の精度よりも大きい場合：SQLException を投入する

実際の精度よりも小さい場合：拡張する

実際の位取りよりも大きい場合：実際の位取りで切り捨てる

実際の位取りよりも小さい場合：0 で補完し拡張する

(d)　HiRDB の TIMESTAMP 型の？パラメタに対する値指定

HiRDB の TIMESTAMP 型の？パラメタに対して setXXX メソッドで値を指定する場合，？パラメタの小数秒精度よりも値の小数秒精度が大きいときは，？パラメタの小数秒精度に合わせて，小数秒精度を切り捨てます。

(e)　HiRDB の CHAR，VARCHAR，NCHAR，NVARCHAR，MCHAR，又は MVARCHAR 型の？パラメタに対する値指定

HiRDB の CHAR，VARCHAR，NCHAR，NVARCHAR，MCHAR，又は MVARCHAR 型の？パラメタに対して setXXX メソッドで値を指定する場合，？パラメタの定義長よりも値を文字列表現にしたときの長さが大きいと，SQLException を投入します。

(f)　setObject で指定できるオブジェクト

setObject の引数 x に指定できるオブジェクトは，次に示す型のオブジェクトです。

- byte[]
- java.lang.Byte
- java.lang.Double
- java.lang.Float
- java.lang.Integer
- java.lang.Long
- java.lang.Short
- java.lang.String
- java.math.BigDecimal

- java.sql.Blob
- java.sql.Boolean
- java.sql.Date
- java.sql.Time
- java.sql.Timestamp
- java.sql.Array

(g)　1 コネクション当たりのステートメントオブジェクト数の上限

1 つの Connection オブジェクトに対し，使用中のステートメントオブジェクト（Statement，PreparedStatement，及び CallableStatement)の数の合計が 4096 以上となった場合，SQLException を投入します。詳細は「Statement インタフェース」の「注意事項」を参照してください。

17.4.5　CallableStatement インタフェース

(1)　概要

CallableStatement インタフェースでは，主に次の機能が提供されます。

- ストアドプロシジャの実行
- IN パラメタ及び INOUT パラメタの設定（PreparedStatement クラスの setXXX メソッドを使用）
- INOUT パラメタ及び OUT パラメタの登録
- INOUT パラメタ及び OUT パラメタ値の取得

また，CallableStatement クラスは PreparedStatement クラスのサブクラスであるため，PreparedStatement クラス及び Statement クラスの機能をすべて継承します。

(2)　メソッド

CallableStatement インタフェースのメソッド一覧を次の表に示します。なお，表に記載されていないメソッドはサポートしていません。サポートしていないメソッドを指定すると，SQLException を投入します。

表 17–33　CallableStatement インタフェースのメソッド一覧

メソッド	提供ドライバ			機能
	JDBC2.0	JDBC4.0	JDBC4.3	
getBigDecimal(int parameterIndex)	○	○	○	指定された？パラメタの値を，Java プログラミング言語の java.math.BigDecimal オブジェクトとして取得します。

メソッド	提供ドライバ			機能
	JDBC2.0	JDBC4.0	JDBC4.3	
getBigDecimal(int parameterIndex, int scale)	○	○	○	指定された？パラメタの値を，scale で指定された小数点以下のけた数を持つ Java プログラミング言語の java.math.BigDecimal オブジェクトとして取得します。
getBlob(int parameterIndex)	○	○	○	指定された？パラメタの値を，Java プログラミング言語の java.sql.Blob オブジェクトとして取得します。
getBlob(String parameterName)	○	○	○	指定された？パラメタの値を，Java プログラミング言語の java.sql.Blob オブジェクトとして取得します。
getBoolean(int parameterIndex)	○	○	○	指定された？パラメタの値を，Java プログラミング言語の boolean として取得します。
getBoolean(String parameterName)	○	○	○	指定された？パラメタの値を，Java プログラミング言語の boolean として取得します。
getByte(int parameterIndex)	○	○	○	指定された？パラメタの値を，Java プログラミング言語の byte として取得します。
getByte(String parameterName)	○	○	○	指定された？パラメタの値を，Java プログラミング言語の byte として取得します。
getBytes(int parameterIndex)	○	○	○	指定された？パラメタの値を，Java プログラミング言語の byte の配列として取得します。
getBytes(String parameterName)	○	○	○	指定された？パラメタの値を，Java プログラミング言語の byte の配列として取得します。
getDate(int parameterIndex)	○	○	○	指定された？パラメタの値を，Java プログラミング言語の java.sql.Date オブジェクトとして取得します。
getDate(int parameterIndex, java.util.Calendar cal)	○	○	○	指定された？パラメタの値を，Java プログラミング言語の java.sql.Date オブジェクトとして取得します。

メソッド	提供ドライバ			機能
	JDBC2.0	JDBC4.0	JDBC4.3	
getDate(String parameterName)	○	○	○	指定された？パラメタの値を，Java プログラミング言語の java.sql.Date オブジェクトとして取得します。
getDate(String parameterName, java.util.Calendar cal)	○	○	○	指定された？パラメタの値を，Java プログラミング言語の java.sql.Date オブジェクトとして取得します。
getDouble(int parameterIndex)	○	○	○	指定された？パラメタの値を，Java プログラミング言語の double として取得します。
getDouble(String parameterName)	○	○	○	指定された？パラメタの値を，Java プログラミング言語の double として取得します。
getFloat(int parameterIndex)	○	○	○	指定された？パラメタの値を，Java プログラミング言語の float として取得します。
getFloat(String parameterName)	○	○	○	指定された？パラメタの値を，Java プログラミング言語の float として取得します。
getInt(int parameterIndex)	○	○	○	指定された？パラメタの値を，Java プログラミング言語の int として取得します。
getInt(String parameterName)	○	○	○	指定された？パラメタの値を，Java プログラミング言語の int として取得します。
getLong(int parameterIndex)	○	○	○	指定された？パラメタの値を，Java プログラミング言語の long として取得します。
getLong(String parameterName)	○	○	○	指定された？パラメタの値を，Java プログラミング言語の long として取得します。
getObject(int parameterIndex)	○	○	○	？パラメタの値を，Java プログラミング言語の java.lang.Object として取得します。
getObject(String parameterName)	○	○	○	？パラメタの値を，Java プログラミング言語の java.lang.Object として取得します。

メソッド	提供ドライバ			機能
	JDBC2.0	JDBC4.0	JDBC4.3	
getShort(int parameterIndex)	○	○	○	指定された？パラメタの値を，Java プログラミング言語の short として取得します。
getShort(String parameterName)	○	○	○	指定された？パラメタの値を，Java プログラミング言語の short として取得します。
getString(int parameterIndex)	○	○	○	指定された？パラメタの値を，Java プログラミング言語の java.lang.String オブジェクトとして取得します。
getString(String parameterName)	○	○	○	指定された？パラメタの値を，Java プログラミング言語の java.lang.String オブジェクトとして取得します。
getTime(int parameterIndex)	○	○	○	指定された？パラメタの値を，Java プログラミング言語の java.sql.Time オブジェクトとして取得します。
getTime(int parameterIndex, java.util.Calendar cal)	○	○	○	指定された？パラメタの値を，Java プログラミング言語の java.sql.Time オブジェクトとして取得します。
getTime(String parameterName)	○	○	○	指定された？パラメタの値を，Java プログラミング言語の java.sql.Time オブジェクトとして取得します。
getTime(String parameterName, java.util.Calendar cal)	○	○	○	指定された？パラメタの値を，Java プログラミング言語の java.sql.Time オブジェクトとして取得します。
getTimestamp(int parameterIndex)	○	○	○	指定された？パラメタの値を，Java プログラミング言語の java.sql.Timestamp オブジェクトとして取得します。
getTimestamp(int parameterIndex, java.util.Calendar cal)	○	○	○	指定された？パラメタの値を，Java プログラミング言語の java.sql.Timestamp オブジェクトとして取得します。
getTimestamp(String parameterName)	○	○	○	指定された？パラメタの値を，Java プログラミング言語の java.sql.Timestamp オブジェクトとして取得します。

メソッド	提供ドライバ			機能
	JDBC2.0	JDBC4.0	JDBC4.3	
getTimestamp(String parameterName, java.util.Calendar cal)	○	○	○	指定された？パラメタの値を，Java プログラミング言語の java.sql.Timestamp オブジェクトとして取得します。
registerOutParameter(int parameterIndex, int sqlType)	○	○	○	指定された OUT パラメタのデータ型を，指定された JDBC の型として登録します。
registerOutParameter(int parameterIndex, int sqlType, int scale)	○	○	○	指定された OUT パラメタのデータ型を，指定された JDBC の型として登録します。
registerOutParameter(String parameterName, int sqlType)	○	○	○	指定された OUT パラメタのデータ型を，指定された JDBC の型として登録します。
registerOutParameter(String parameterName, int sqlType, int scale)	○	○	○	指定された OUT パラメタのデータ型を，指定された JDBC の型として登録します。
setAsciiStream(String parameterName, java.io.InputStream x, int length)	○	○	○	指定された java.io.InputStream オブジェクトの持つ値を指定された？パラメタ値に指定された長さだけ設定します。
setBigDecimal(String parameterName, java.math.BigDecimal x)	○	○	○	指定された java.math.BigDecimal 値を指定された？パラメタ値に設定します。
setBinaryStream(String parameterName, java.io.InputStream x, int length)	○	○	○	指定された？パラメタの値を，Java プログラミング言語の java.math.BigDecimal オブジェクトとして取得します。
setBoolean(String parameterName, boolean x)	○	○	○	指定された boolean 値を？パラメタ値に設定します。
setByte(String parameterName, byte x)	○	○	○	指定された byte 値を？パラメタ値に設定します。
setBytes(String parameterName,byte[] x)	○	○	○	指定された byte 配列を？パラメタ値に設定します。
setCharacterStream(String parameterName, Reader x, int length)	○	○	○	指定された Reader オブジェクトの持つ値を？パラメタ値に設定します。

メソッド	提供ドライバ			機能
	JDBC2.0	JDBC4.0	JDBC4.3	
setDate(String parameterName, java.sql.Date x)	○	○	○	指定された java.sql.Date オブジェクトの持つ値を？パラメタ値に設定します。
setDate(String parameterName, java.sql.Date x,Calendar cal)	○	○	○	ローカルタイムで指定された java.sql.Date オブジェクトを指定されたカレンダーのタイムゾーンの時間に変換し，？パラメタ値に設定します。
setDouble(String parameterName, double x)	○	○	○	指定された double 値を？パラメタ値に設定します。
setFloat(String parameterName, float x)	○	○	○	指定された float 値を？パラメタ値に設定します。
setInt(String parameterName, int x)	○	○	○	指定された int 値を？パラメタ値に設定します。
setLong(String parameterName, long x)	○	○	○	指定された long 値を？パラメタ値に設定します。
setNull(String parameterName,int sqlType)	○	○	○	指定された？パラメタに NULL 値を設定します。
setObject(String parameterName, Object x)	○	○	○	指定されたオブジェクトの持つ値を？パラメタ値に設定します。
setObject(String parameterName, Object x, int targetSqlType)	○	○	○	指定されたオブジェクトの持つ値を？パラメタ値に設定します。
setObject(String parameterName, Object x, int targetSqlType, int scale)	○	○	○	指定されたオブジェクトの持つ値を？パラメタ値に設定します。
setShort(String parameterName, short x)	○	○	○	指定された short 値を？パラメタ値に設定します。
setString(String parameterName, String x)	○	○	○	指定された String オブジェクトを？パラメタ値に設定します。
setTime(String parameterName, Time x)	○	○	○	指定された java.sql.Time オブジェクトを？パラメタ値に設定します。
setTime(String parameterName, java.sql.Time x,Calendar cal)	○	○	○	ローカルタイムで指定された java.sql.Time オブジェクトを指定されたカレンダーのタイムゾーンの時間に変換し，？パラメタ値に設定します。
setTimestamp(String parameterName, java.sql.Timestamp x)	○	○	○	指定された java.sql.Timestamp オブジェクトを？パラメタ値に設定します。

メソッド	提供ドライバ			機能
	JDBC2.0	JDBC4.0	JDBC4.3	
setTimestamp(String parameterName, java.sql.Timestamp x,Calendar cal)	○	○	○	ローカルタイムで指定された java.sql.Timestamp オブジェクトを指定されたカレンダーのタイムゾーンの時間に変換し，？パラメタ値に設定します。
wasNull()	○	○	○	最後に読み込まれた OUT パラメタ又は INOUT パラメタの値が，NULL かどうかを取得します。

（凡例）

○：提供されます。

(a)　getBigDecimal(int parameterIndex)

【機能】

指定された？パラメタの値を，Java プログラミング言語の java.math.BigDecimal オブジェクトとして取得します。

【形式】

```
public synchronized java.math.BigDecimal getBigDecimal(int parameterIndex) throws SQLException
```

【引数】

int parameterIndex：

？パラメタの番号。最初のパラメタは 1，2 番目のパラメタは 2，...と指定します。

【戻り値】

指定された？パラメタの値を持つ java.math.BigDecimal オブジェクト（値が NULL の場合，null を返します）

【機能詳細】

指定された？パラメタの値を，Java プログラミング言語の java.math.BigDecimal オブジェクトとして取得します。

HiRDB のデータ型による？パラメタ値と戻り値の関係を次に示します。

HiRDB のデータ型	？パラメタの値	戻り値
[M | N][VAR]CHAR	NULL	null
	[半角空白]整数の文字列表現，10 進数の文字列表現又は浮動小数点数の文字列表現[半角空白]	？パラメタの値を持つ BigDecimal オブジェクト。文字列の前後の半角空白を取り除いた値を BigDecimal オブジェクトにします。
	上記以外	SQLException を投入

HiRDB のデータ型	？パラメタの値	戻り値
SMALLINT	NULL	null
	NULL 以外	？パラメタの値を持つ BigDecimal オブジェクト
INTEGER	NULL	null
	NULL 以外	？パラメタの値を持つ BigDecimal オブジェクト
REAL	NULL	null
	Infinity	SQLException を投入
	-Infinity	SQLException を投入
	NaN	SQLException を投入
	上記以外	？パラメタの値を持つ BigDecimal オブジェクト
FLOAT	NULL	null
	Infinity	SQLException を投入
	-Infinity	SQLException を投入
	NaN	SQLException を投入
	上記以外	？パラメタの値を持つ BigDecimal オブジェクト
DECIMAL	NULL	null
	NULL 以外	？パラメタの値を持つ BigDecimal オブジェクト
BOOLEAN	NULL	null
	true	BigDecimal(1)で BigDecimal オブジェクトにしたもの
	false	BigDecimal(0)で BigDecimal オブジェクトにしたもの
その他	−	SQLException を投入

（凡例）

−：該当しません。

【発生する例外】

次の場合，SQLException を投入します。

- CallableStatement オブジェクトに対して close()が既に発行されている場合
- この CallableStatement オブジェクトを生成した Connection オブジェクトに対して close()が既に発行されている場合

- 存在しない？パラメタの番号を指定した場合
- 指定した？パラメタが IN パラメタの場合
- 指定した？パラメタはこのメソッドでは取得できないデータ型である場合

(b)　getBigDecimal(int parameterIndex, int scale)

【機能】

指定された？パラメタの値を，scale で指定された小数点以下のけた数を持つ Java プログラミング言語の java.math.BigDecimal オブジェクトとして取得します。

【形式】

```
public synchronized java.math.BigDecimal getBigDecimal(int parameterIndex, int scale) thr
ows SQLException
```

【引数】

int parameterIndex：

？パラメタの番号。最初のパラメタは 1，2 番目のパラメタは 2，...と指定します。

int scale：

位取り。指定できる範囲は 0≦scale≦2147483647 です。

【戻り値】

指定された？パラメタの値で小数点以下のけた数を引数 scale にした値を持つ java.math.BigDecimal オブジェクト（値が NULL の場合，null を返します）

【機能詳細】

指定された？パラメタの値を，scale で指定された小数点以下のけた数を持つ Java プログラミング言語の java.math.BigDecimal オブジェクトとして取得します。

？パラメタの値が持つ小数点以下のけた数より引数 scale の値が小さい場合，？パラメタの小数点以下のけた数を scale で切り捨てます。？パラメタの値が持つ小数点以下のけた数より引数 scale の値が大きい場合，？パラメタの小数点以下のけた数を scale まで 0 で補完します。

HiRDB のデータ型による？パラメタ値と戻り値の関係については，「getBigDecimal(int parameterIndex)」の【機能詳細】を参照してください。

【発生する例外】

次の場合，SQLException を投入します。

- CallableStatement オブジェクトに対して close()が既に発行されている場合
- この CallableStatement オブジェクトを生成した Connection オブジェクトに対して close()が既に発行されている場合
- 存在しない？パラメタの番号を指定した場合
- 指定した？パラメタが IN パラメタである場合
- scale に 0 未満を指定した場合

- 指定した？パラメタはこのメソッドでは取得できないデータ型である場合

(c) getBlob(int parameterIndex)

【機能】

指定された？パラメタの値を，Java プログラミング言語の java.sql.Blob オブジェクトとして取得します。

【形式】

```
public synchronized java.sql.Blob getBlob(int parameterIndex) throws SQLException
```

【引数】

int parameterIndex：

？パラメタの番号。最初のパラメタは 1，2 番目のパラメタは 2...と指定します。

【戻り値】

指定された？パラメタの値を持つ java.sql.Blob オブジェクト（値が NULL の場合，null を返します）

【機能詳細】

指定された？パラメタの値を，Java プログラミング言語の java.sql.Blob オブジェクトとして取得します。

HiRDB のデータ型による？パラメタ値と戻り値の関係を次に示します。

HiRDB のデータ型	？パラメタの値	戻り値
BINARY BLOB	NULL	null
	NULL 以外	？パラメタの値を持つ java.sql.Blob オブジェクト
その他	−	SQLException を投入

（凡例）

−：該当しません。

【発生する例外】

次の場合，SQLException を投入します。

- CallableStatement オブジェクトに対して close()が既に発行されている場合
- この CallableStatement オブジェクトを生成した Connection オブジェクトに対して close()が既に発行されている場合
- 存在しない？パラメタの番号を指定した場合
- 指定した？パラメタが IN パラメタである場合
- 指定した？パラメタはこのメソッドでは取得できないデータ型である場合

(d)　getBlob(String parameterName)

【機能】

指定された？パラメタの値を，Java プログラミング言語の java.sql.Blob オブジェクトとして取得します。

HiRDB のデータ型による？パラメタ値と戻り値の関係は「getBlob(int parameterIndex)」の【機能詳細】を参照してください。

【形式】

```
public synchronized java.sql.Blob getBlob (String parameterName) throws SQLException
```

【引数】

String parameterName：

パラメタ名。パラメタ名の大文字と小文字は区別しません。parameterName の文字列すべてを？パラメタ名として扱うため，文字列に引用符「"」が含まれている場合，引用符「"」も？パラメタ名に含まれます。

【戻り値】

指定された？パラメタの値を持つ java.sql.Blob オブジェクト（値が NULL の場合，null を返します）

【発生する例外】

次の場合，SQLException を投入します。

- CallableStatement オブジェクトに対して close()が既に発行されている場合
- この CallableStatement オブジェクトを生成した Connection オブジェクトに対して close()が既に発行されている場合
- 存在しないパラメタ名を指定した場合（parameterName の指定値が null 又は 0 長文字列の場合も含む）
- 指定した？パラメタが IN パラメタである場合
- 指定した？パラメタはこのメソッドでは取得できないデータ型である場合

(e)　getBoolean(int parameterIndex)

【機能】

指定された？パラメタの値を，Java プログラミング言語の boolean として取得します。

【形式】

```
public synchronized boolean getBoolean(int parameterIndex) throws SQLException
```

【引数】

int parameterIndex：

？パラメタの番号。最初のパラメタは 1，2 番目のパラメタは 2，...と指定します。

【戻り値】

指定された？パラメタの値（値が NULL の場合，false を返します）

【機能詳細】

指定された？パラメタの値を，Java プログラミング言語の boolean として取得します。

HiRDB のデータ型による？パラメタ値と戻り値の関係を次に示します。

HiRDB のデータ型	？パラメタの値	戻り値
[M][VAR]CHAR NVARCHAR	NULL	false
	[半角空白]半角 true（大文字と小文字は区別しない）[半角空白]	true
	[半角空白]半角 1[半角空白]	true
	上記以外	false
NCHAR	NULL	false
	先頭が[半角空白]半角 true（大文字と小文字は区別しない）	true
	上記以外	false
SMALLINT	NULL	false
	0	false
	0 以外	true
INTEGER	NULL	false
	0	false
	0 以外	true
REAL	NULL	false
	Infinity	true
	-Inifnity	true
	NaN	true
	0.0 又は-0.0	false
	上記以外	true
FLOAT	NULL	false
	Infinity	true
	-Inifnity	true
	NaN	true
	0.0 又は-0.0	false
	上記以外	true

HiRDB のデータ型	？パラメタの値	戻り値
DECIMAL	NULL	false
	0[.00…0]	false
	上記以外	true
BOOLEAN	NULL	false
	NULL 以外	？パラメタ値
その他	−	SQLException を投入

（凡例）

−：該当しません。

【発生する例外】

次の場合，SQLException を投入します。

- CallableStatement オブジェクトに対して close()が既に発行されている場合
- この CallableStatement オブジェクトを生成した Connection オブジェクトに対して close()が既に発行されている場合
- 存在しない？パラメタの番号を指定した場合
- 指定した？パラメタが IN パラメタである場合
- 指定した？パラメタはこのメソッドでは取得できないデータ型である場合

(f)　getBoolean(String parameterName)

【機能】

指定された？パラメタの値を，Java プログラミング言語の boolean として取得します。

HiRDB のデータ型による？パラメタ値と戻り値の関係については，「getBoolean(int parameterIndex)」の【機能詳細】を参照してください。

【形式】

```
public synchronized boolean getBoolean (String parameterName) throws SQLException
```

【引数】

String parameterName：

パラメタ名。パラメタ名の大文字と小文字は区別しません。parameterName の文字列すべてを？パラメタ名として扱うため，文字列に引用符「"」が含まれている場合，引用符「"」も？パラメタ名に含まれます。

【戻り値】

指定された？パラメタの値（値が NULL の場合，false を返します）

【発生する例外】

次の場合，SQLException を投入します。

- CallableStatement オブジェクトに対して close()が既に発行されている場合
- この CallableStatement オブジェクトを生成した Connection オブジェクトに対して close()が既に発行されている場合
- 存在しないパラメタ名を指定した場合（parameterName の指定値が null 又は 0 長文字列の場合も含む）
- 指定した？パラメタが IN パラメタである場合
- 指定した？パラメタはこのメソッドでは取得できないデータ型である場合

(g)　getByte(int parameterIndex)

【機能】

指定された？パラメタの値を，Java プログラミング言語の byte として取得します。

【形式】

```
public synchronized byte getByte(int parameterIndex) throws SQLException
```

【引数】

int parameterIndex：

？パラメタの番号。最初のパラメタは 1，2 番目のパラメタは 2，...と指定します。

【戻り値】

指定された？パラメタの値（値が NULL の場合，0 を返します）

【機能詳細】

指定された？パラメタの値を，Java プログラミング言語の byte として取得します。

HiRDB のデータ型による？パラメタ値と戻り値の関係を次に示します。

HiRDB のデータ型	？パラメタの値	戻り値
[M ｜ N][VAR]CHAR	NULL	0
	[半角空白]整数の文字列表現，10 進数の文字列表現又は浮動小数点数の文字列表現[半角空白]であり，かつ Byte.MIN_VALUE 以上，Byte.MAX_VALUE 以下	？パラメタ値を byte 値にしたもの
	[半角空白]整数の文字列表現，10 進数の文字列表現又は浮動小数点数の文字列表現[半角空白]であり，かつ Byte.MAX_VALUE より大きいか Byte.MIN_VALUE より小さい	SQLException を投入
	[半角空白][+]Infinity[半角空白]	SQLException を投入
	[半角空白]-Infinity[半角空白]	SQLException を投入
	[半角空白][+｜-]NaN[半角空白]	SQLException を投入
	上記以外（double 値にできない）	SQLException を投入

HiRDB のデータ型	？パラメタの値	戻り値
SMALLINT	NULL	0
	Byte.MIN_VALUE 以上かつ Byte.MAX_VALUE 以下	？パラメタ値を byte 値にしたもの
	上記以外	SQLException を投入
INTEGER	NULL	0
	Byte.MIN_VALUE 以上かつ Byte.MAX_VALUE 以下	？パラメタ値を byte 値にしたもの
	上記以外	SQLException を投入
REAL	NULL	0
	Infinity	SQLException を投入
	-Infinity	SQLException を投入
	NaN	SQLException を投入
	Byte.MIN_VALUE 以上かつ Byte.MAX_VALUE 以下	？パラメタ値の整数部分の値を byte 値にしたもの
	上記以外	SQLException を投入
FLOAT	NULL	0
	Infinity	SQLException を投入
	-Infinity	SQLException を投入
	NaN	SQLException を投入
	Byte.MIN_VALUE 以上かつ Byte.MAX_VALUE 以下	？パラメタ値の整数部分の値を byte 値にしたもの
	上記以外	SQLException を投入
DECIMAL	NULL	0
	Byte.MIN_VALUE 以上かつ Byte.MAX_VALUE 以下	？パラメタ値の整数部分の値を byte 値にしたもの
	上記以外	SQLException を投入
BOOLEAN	NULL	0
	true	1
	false	0
その他	−	SQLException を投入

（凡例）

−：該当しません。

【発生する例外】

次の場合，SQLException を投入します。

- CallableStatement オブジェクトに対して close()が既に発行されている場合
- この CallableStatement オブジェクトを生成した Connection オブジェクトに対して close()が既に発行されている場合
- 存在しない？パラメタの番号を指定した場合
- 指定した？パラメタが IN パラメタである場合
- 指定した？パラメタはこのメソッドでは取得できないデータ型である場合

(h)　getByte(String parameterName)

【機能】

指定された？パラメタの値を，Java プログラミング言語の byte として取得します。

HiRDB のデータ型による？パラメタ値と戻り値の関係は，「getByte(int parameterIndex)」の【機能詳細】を参照してください。

【形式】

```
public synchronized byte getByte (String parameterName) throws SQLException
```

【引数】

String parameterName：

パラメタ名。パラメタ名の大文字と小文字は区別しません。parameterName の文字列すべてを？パラメタ名として扱うため，文字列に引用符「"」が含まれている場合，引用符「"」も？パラメタ名に含まれます。

【戻り値】

指定された？パラメタの値（値が NULL の場合，0 を返します）

【発生する例外】

次の場合，SQLException を投入します。

- CallableStatement オブジェクトに対して close()が既に発行されている場合
- この CallableStatement オブジェクトを生成した Connection オブジェクトに対して close()が既に発行されている場合
- 存在しないパラメタ名を指定した場合（parameterName の指定値が null 又は 0 長文字列の場合も含む）
- 指定した？パラメタが IN パラメタである場合
- 指定した？パラメタはこのメソッドでは取得できないデータ型である場合

(i)　getBytes(int parameterIndex)

【機能】

指定された？パラメタの値を，Java プログラミング言語の byte の配列として取得します。

【形式】

```
public synchronized byte[] getBytes(int parameterIndex) throws SQLException
```

【引数】

int parameterIndex：

？パラメタの番号。最初のパラメタは 1，2 番目のパラメタは 2，...と指定します。

【戻り値】

指定された？パラメタの値を持つ byte 配列（値が NULL の場合，null を返します）

【機能詳細】

指定された？パラメタの値を，Java プログラミング言語の byte の配列として取得します。

HiRDB のデータ型による？パラメタ値と戻り値の関係を次に示します。

HiRDB のデータ型	？パラメタの値	戻り値
[M | N][VAR]CHAR BINARY BLOB	NULL	null
	NULL 以外	？パラメタ値を byte 配列にしたもの
その他	−	SQLException を投入

（凡例）

−：該当しません。

【発生する例外】

次の場合，SQLException を投入します。

- CallableStatement オブジェクトに対して close()が既に発行されている場合
- この CallableStatement オブジェクトを生成した Connection オブジェクトに対して close()が既に発行されている場合
- 存在しない？パラメタの番号を指定した場合
- 指定した？パラメタが IN パラメタである場合
- 指定した？パラメタはこのメソッドでは取得できないデータ型である場合
- データベースアクセスエラーが発生した場合

(j)　getBytes(String parameterName)

【機能】

指定された？パラメタの値を，Java プログラミング言語の byte の配列として取得します。

HiRDB のデータ型による？パラメタ値と戻り値の関係については，「getBytes(int parameterIndex)」の【機能詳細】を参照してください。

【形式】

```
public synchronized byte[] getBytes(String parameterName) throws SQLException
```

【引数】

String parameterName：

パラメタ名。パラメタ名の大文字と小文字は区別しません。parameterName の文字列すべてを？パラメタ名として扱うため，文字列に引用符「"」が含まれている場合，引用符「"」も？パラメタ名に含まれます。

【戻り値】

指定された？パラメタの値を持つ byte 配列（値が NULL の場合，null を返します）

【発生する例外】

次の場合，SQLException を投入します。

- CallableStatement オブジェクトに対して close()が既に発行されている場合
- この CallableStatement オブジェクトを生成した Connection オブジェクトに対して close()が既に発行されている場合
- 存在しないパラメタ名を指定した場合（parameterName の指定値が null 又は 0 長文字列の場合も含む）
- 指定した？パラメタが IN パラメタである場合
- 指定した？パラメタはこのメソッドでは取得できないデータ型である場合
- データベースアクセスエラーが発生した場合

(k)　getDate(int parameterIndex)

【機能】

指定された？パラメタの値を，Java プログラミング言語の java.sql.Date オブジェクトとして取得します。

【形式】

```
public synchronized java.sql.Date getDate(int parameterIndex) throws SQLException
```

【引数】

int parameterIndex：

？パラメタの番号。最初のパラメタは 1，2 番目のパラメタは 2，...と指定します。

【戻り値】

指定された？パラメタの値を持つ java.sql.Date オブジェクト（値が NULL の場合，null を返します）

【機能詳細】

指定された？パラメタの値を，Java プログラミング言語の java.sql.Date オブジェクトとして取得します。

HiRDB のデータ型による？パラメタ値と戻り値の関係を次に示します。

HiRDB のデータ型	？パラメタの値	戻り値
[M ｜ N][VAR]CHAR	NULL	null
	[半角空白]日付形式[半角空白]	？パラメタ値の前後の半角空白を取り除き java.sql.Date オブジェクトにしたもの
	上記以外	SQLException を投入
DATE	NULL	null
	NULL 以外	？パラメタ値を java.sql.Date オブジェクトにしたもの
TIMESTAMP	NULL	null
	NULL 以外	？パラメタ値を java.sql.Date オブジェクトにしたもの
その他	－	SQLException を投入

（凡例）

－：該当しません。

【発生する例外】

次の場合，SQLException を投入します。

- CallableStatement オブジェクトに対して close()が既に発行されている場合
- この CallableStatement オブジェクトを生成した Connection オブジェクトに対して close()が既に発行されている場合
- 存在しない？パラメタの番号を指定した場合
- 指定した？パラメタが IN パラメタである場合
- 指定した？パラメタはこのメソッドでは取得できないデータ型である場合
- ？パラメタの値が java.sql.Date として取得できない値の場合

(l)　getDate(int parameterIndex, java.util.Calendar cal)

【機能】

指定された？パラメタの値を，Java プログラミング言語の java.sql.Date オブジェクトとして取得します。このメソッドは，指定されたカレンダーを使って取得した適切なミリ秒値から日付を作成します。

HiRDB のデータ型による？パラメタ値と戻り値の関係については，「getDate(int parameterIndex)」の【機能詳細】を参照してください。

【形式】

```
public synchronized java.sql.Date getDate (int parameterIndex, java.util.Calendar cal) th
rows SQLException
```

【引数】

int parameterIndex：

？パラメタの番号。最初のパラメタは 1，2 番目のパラメタは 2，...と指定します。

java.util.Calendar cal：

データベースに格納されている値のタイムゾーンが設定されたカレンダー

【戻り値】

指定された？パラメタの値を持つ java.sql.Date オブジェクト（値が NULL の場合，null を返します）

【発生する例外】

次の場合，SQLException を投入します。

- CallableStatement オブジェクトに対して close()が既に発行されている場合
- この CallableStatement オブジェクトを生成した Connection オブジェクトに対して close()が既に発行されている場合
- 存在しない？パラメタの番号を指定した場合
- 指定した？パラメタが IN パラメタである場合
- 指定した？パラメタはこのメソッドでは取得できないデータ型である場合
- ？パラメタの値が java.sql.Date として取得できない値の場合

(m)　getDate(String parameterName)

【機能】

指定された？パラメタの値を，Java プログラミング言語の java.sql.Date オブジェクトとして取得します。

HiRDB のデータ型による？パラメタ値と戻り値の関係については，「getDate(int parameterIndex)」の【機能詳細】を参照してください。

【形式】

```
public synchronized java.sql.Date getDate (String parameterName) throws SQLException
```

【引数】

String parameterName：

パラメタ名。パラメタ名の大文字と小文字は区別しません。parameterName の文字列すべてを？パラメタ名として扱うため，文字列に引用符「"」が含まれている場合，引用符「"」も？パラメタ名に含まれます。

【戻り値】

指定された？パラメタの値を持つ java.sql.Date オブジェクト（値が NULL の場合，null を返します）

【発生する例外】

次の場合，SQLException を投入します。

- CallableStatement オブジェクトに対して close()が既に発行されている場合
- この CallableStatement オブジェクトを生成した Connection オブジェクトに対して close()が既に発行されている場合
- 存在しないパラメタ名を指定した場合（parameterName の指定値が null 又は 0 長文字列の場合も含む）
- 指定した？パラメタが IN パラメタである場合
- 指定した？パラメタはこのメソッドでは取得できないデータ型である場合
- ？パラメタの値が java.sql.Date として取得できない値の場合

(n)　getDate(String parameterName, java.util.Calendar cal)

【機能】

指定された？パラメタの値を，Java プログラミング言語の java.sql.Date オブジェクトとして取得します。このメソッドは，指定されたカレンダーを使って取得した適切なミリ秒値から日付を作成します。

HiRDB のデータ型による？パラメタ値と戻り値の関係については，「getDate(int parameterIndex)」の【機能詳細】を参照してください。

【形式】

```
public synchronized java.sql.Date getDate (String parameterName, java.util.Calendar cal)
throws SQLException
```

【引数】

String parameterName：

パラメタ名。パラメタ名の大文字と小文字は区別しません。parameterName の文字列すべてを？パラメタ名として扱うため，文字列に引用符「"」が含まれている場合，引用符「"」も？パラメタ名に含まれます。

java.util.Calendar cal：

データベースに格納されている値のタイムゾーンが設定されたカレンダー

【戻り値】

指定された？パラメタの値を持つ java.sql.Date オブジェクト（値が NULL の場合，null を返します）

【発生する例外】

次の場合，SQLException を投入します。

- CallableStatement オブジェクトに対して close()が既に発行されている場合

- この CallableStatement オブジェクトを生成した Connection オブジェクトに対して close()が既に発行されている場合
- 存在しないパラメタ名を指定した場合（parameterName の指定値が null 又は 0 長文字列の場合も含む）
- 指定した？パラメタが IN パラメタである場合
- 指定した？パラメタはこのメソッドでは取得できないデータ型である場合
- ？パラメタの値が java.sql.Date として取得できない値の場合

(o)　getDouble(int parameterIndex)

【機能】

指定された？パラメタの値を，Java プログラミング言語の double として取得します。

【形式】

```
public synchronized double getDouble(int parameterIndex) throws SQLException
```

【引数】

int parameterIndex：

？パラメタの番号。最初のパラメタは 1，2 番目のパラメタは 2，...と指定します。

【戻り値】

指定された？パラメタの値（値が NULL の場合，0 を返します）

【機能詳細】

指定された？パラメタの値を，Java プログラミング言語の double として取得します。

HiRDB のデータ型による？パラメタ値と戻り値の関係を次に示します。

HiRDB のデータ型	？パラメタの値	戻り値
[M | N][VAR]CHAR	NULL	0.0
	[半角空白]整数の文字列表現，10 進数の文字列表現又は浮動小数点数の文字列表現[半角空白]であり，かつ-Double.MAX_VALUE 以上，かつ Double.MIN_VALUE 以下，かつ Double.MIN_VALUE 以上かつ Double.MAX_VALUE 以下	？パラメタ値を double 値にしたもの
	[半角空白]整数の文字列表現，10 進数の文字列表現又は浮動小数点数の文字列表現[半角空白]かつ Double.MAX_VALUE より大きい	Infinity
	[半角空白]整数の文字列表現，10 進数の文字列表現又は浮動小数点数の文字列表現[半角空白]かつ-Double.MAX_VALUE より小さい	-Infinity

HiRDB のデータ型	？パラメタの値	戻り値
	[半角空白]整数の文字列表現，10 進数の文字列表現又は浮動小数点数の文字列表現[半角空白]かつ Double.MIN_VALUE より小さく 0 より大きい	0.0
	[半角空白]整数の文字列表現，10 進数の文字列表現又は浮動小数点数の文字列表現[半角空白]かつ-Double.MIN_VALUE より大きく 0 より小さい	-0.0
	[半角空白]-Infinity[半角空白]	-Infinity
	[半角空白][+]Infinity[半角空白]	Infinity
	[半角空白][+｜-]NaN[半角空白]	NaN
	上記以外（double 値にできない）	SQLException を投入
SMALLINT	NULL	0.0
	NULL 以外	？パラメタ値を double 値にしたもの
INTEGER	NULL	0
	NULL 以外	？パラメタ値を double 値にしたもの
REAL	NULL	0.0
	Infinity	Infinity
	-Infinity	-Infinity
	NaN	NaN
	上記以外	？パラメタ値を double 値にしたもの
FLOAT	NULL	0.0
	Infinity	Infinity
	-Infinity	-Infinity
	NaN	NaN
	上記以外	？パラメタ値
DECIMAL	NULL	0
	NULL 以外	？パラメタ値を double 値にしたもの
BOOLEAN	NULL	0.0
	true	1.0
	false	0.0
その他	−	SQLException を投入

（凡例）

－：該当しません。

【発生する例外】

次の場合，SQLException を投入します。

- CallableStatement オブジェクトに対して close()が既に発行されている場合
- この CallableStatement オブジェクトを生成した Connection オブジェクトに対して close()が既に発行されている場合
- 存在しない？パラメタの番号を指定した場合
- 指定した？パラメタが IN パラメタである場合
- 指定した？パラメタはこのメソッドでは取得できないデータ型である場合

(p)　getDouble(String parameterName)

【機能】

指定された？パラメタの値を，Java プログラミング言語の double として取得します。

HiRDB のデータ型による？パラメタ値と戻り値の関係については，「getDouble(int parameterIndex)」の【機能詳細】を参照してください。

【形式】

```
public synchronized double getDouble (String parameterName) throws SQLException
```

【引数】

String parameterName：

パラメタ名。パラメタ名の大文字と小文字は区別しません。parameterName の文字列すべてを？パラメタ名として扱うため，文字列に引用符「"」が含まれている場合，引用符「"」も？パラメタ名に含まれます。

【戻り値】

指定された？パラメタの値（値が NULL の場合，0 を返します）

【発生する例外】

次の場合，SQLException を投入します。

- CallableStatement オブジェクトに対して close()が既に発行されている場合
- この CallableStatement オブジェクトを生成した Connection オブジェクトに対して close()が既に発行されている場合
- 存在しないパラメタ名を指定した場合（parameterName の指定値が null 又は 0 長文字列の場合も含む）
- 指定した？パラメタが IN パラメタである場合
- 指定した？パラメタはこのメソッドでは取得できないデータ型である場合

(q)　getFloat(int parameterIndex)

【機能】

指定された？パラメタの値を，Java プログラミング言語の float として取得します。

【形式】

```
public synchronized float getFloat(int parameterIndex) throws SQLException
```

【引数】

int parameterIndex：

？パラメタの番号。最初のパラメタは 1，2 番目のパラメタは 2，...と指定します。

【戻り値】

指定された？パラメタの値（値が NULL の場合，0 を返します）

【機能詳細】

指定された？パラメタの値を，Java プログラミング言語の float として取得します。

HiRDB のデータ型による？パラメタ値と戻り値の関係を次に示します。

HiRDB のデータ型	？パラメタの値	戻り値
[M ｜ N][VAR]CHAR	NULL	0.0
	[半角空白]整数の文字列表現，10 進数の文字列表現又は浮動小数点数の文字列表現[半角空白]であり，かつ次のどちらか • -Float.MAX_VALUE 以上-Float.MIN_VALUE 以下 • Float.MIN_VALUE 以上Float.MAX_VALUE 以下	？パラメタ値を float 値にしたもの
	[半角空白]整数の文字列表現，10 進数の文字列表現又は浮動小数点数の文字列表現[半角空白]かつ Float.MAX_VALUE より大きい	Infinity
	[半角空白]整数の文字列表現，10 進数の文字列表現又は浮動小数点数の文字列表現[半角空白]かつ-Float.MAX_VALUE より小さい	-Infinity
	[半角空白]整数の文字列表現，10 進数の文字列表現又は浮動小数点数の文字列表現[半角空白]かつ Float.MIN_VALUE より小さく 0 より大きい	0.0
	[半角空白]整数の文字列表現，10 進数の文字列表現又は浮動小数点数の文字列表現[半角空白]かつ-Float.MIN_VALUE より大きく 0 より小さい	-0.0
	[半角空白]-Infinity[半角空白]	-Infinity
	[半角空白][+]Infinity[半角空白]	Infinity

HiRDB のデータ型	？パラメタの値	戻り値
	[半角空白][+｜-]NaN[半角空白]	NaN
	上記以外（float 値にできない）	SQLException を投入
SMALLINT	NULL	0.0
	NULL 以外	？パラメタ値を float 値にしたもの
INTEGER	NULL	0.0
	NULL 以外	？パラメタ値を float 値にしたもの
REAL	NULL	0.0
	Infinity	Infinity
	-Infinity	-Infinity
	NaN	NaN
	上記以外	？パラメタ値
FLOAT	NULL	0.0
	Infinity	Infinity
	-Infinity	-Infinity
	NaN	NaN
	-Float.MAX_VALUE 以上-Float.MIN_VALUE 以下，又はFloat.MIN_VALUE 以上 Float.MAX_VALUE 以下	？パラメタ値を float 値にしたもの
	Float.MAX_VALUE より大きい	Infinity
	-Float.MAX_VALUE より小さい	-Infinity
	Float.MIN_VALUE より小さく 0 より大きい	0.0
	-Float.MIN_VALUE より大きく 0 より小さい	-0.0
DECIMAL	NULL	0.0
	NULL 以外	？パラメタ値を float 値にしたもの
BOOLEAN	NULL	0.0
	true	1.0
	false	0.0
その他	－	SQLException を投入

（凡例）

－：該当しません。

【発生する例外】

次の場合，SQLException を投入します。

- CallableStatement オブジェクトに対して close()が既に発行されている場合
- この CallableStatement オブジェクトを生成した Connection オブジェクトに対して close()が既に発行されている場合
- 存在しない？パラメタの番号を指定した場合
- 指定した？パラメタが IN パラメタである場合
- 指定した？パラメタはこのメソッドでは取得できないデータ型である場合

(r)　getFloat(String parameterName)

【機能】

指定された？パラメタの値を，Java プログラミング言語の float として取得します。

HiRDB のデータ型による？パラメタ値と戻り値の関係については，「getFloat(int parameterIndex)」の【機能詳細】を参照してください。

【形式】

```
public synchronized float getFloat (String parameterName) throws SQLException
```

【引数】

String parameterName：

パラメタ名。パラメタ名の大文字と小文字は区別しません。parameterName の文字列すべてを？パラメタ名として扱うため，文字列に引用符「"」が含まれている場合，引用符「"」も？パラメタ名に含まれます。

【戻り値】

指定された？パラメタの値（値が NULL の場合，0 を返します）

【発生する例外】

次の場合，SQLException を投入します。

- CallableStatement オブジェクトに対して close()が既に発行されている場合
- この CallableStatement オブジェクトを生成した Connection オブジェクトに対して close()が既に発行されている場合
- 存在しないパラメタ名を指定した場合（parameterName の指定値が null 又は 0 長文字列の場合も含む）
- 指定した？パラメタが IN パラメタである場合
- 指定した？パラメタはこのメソッドでは取得できないデータ型である場合

(s)　getInt(int parameterIndex)

【機能】

指定された？パラメタの値を，Java プログラミング言語の int として取得します。

【形式】

```
public synchronized int getInt(int parameterIndex) throws SQLException
```

【引数】

int parameterIndex：

？パラメタの番号。最初のパラメタは 1，2 番目のパラメタは 2，...と指定します。

【戻り値】

指定された？パラメタの値（値が NULL の場合，0 を返します）

【機能詳細】

指定された？パラメタの値を，Java プログラミング言語の int として取得します。

HiRDB のデータ型による？パラメタ値と戻り値の関係を次に示します。

HiRDB のデータ型	？パラメタの値	戻り値
[M ｜ N][VAR]CHAR	NULL	0
	[半角空白]整数の文字列表現，10 進数の文字列表現又は浮動小数点数の文字列表現[半角空白]であり，かつ Integer.MIN_VALUE 以上，Integer.MAX_VALUE 以下	？パラメタ値の整数部分の値を int 値にしたもの
	[半角空白]整数の文字列表現，10 進数の文字列表現又は浮動小数点数の文字列表現[半角空白]であり，かつ Integer.MAX_VALUE より大きいか Integer.MIN_VALUE より小さい	SQLException を投入
	[半角空白]-Infinity[半角空白]	SQLException を投入
	[半角空白][+]Infinity[半角空白]	SQLException を投入
	[半角空白][+｜-]NaN[半角空白]	SQLException を投入
	上記以外（double 値にできない）	SQLException を投入
SMALLINT	NULL	0
	NULL 以外	？パラメタ値を int 値にしたもの
INTEGER	NULL	0
	NULL 以外	？パラメタ値
REAL	NULL	0
	Infinity	SQLException を投入

HiRDB のデータ型	？パラメタの値	戻り値
	-Infinity	SQLException を投入
	NaN	SQLException を投入
	Integer.MIN_VALUE 以上かつ， Integer.MAX_VALUE 以下	？パラメタ値の整数部分の値を int 値にしたもの
	上記以外	SQLException を投入
FLOAT	NULL	0
	Infinity	SQLException を投入
	-Infinity	SQLException を投入
	NaN	SQLException を投入
	Integer.MIN_VALUE 以上かつ， Integer.MAX_VALUE 以下	？パラメタ値の整数部分の値を int 値にしたもの
	上記以外	SQLException を投入
DECIMAL	NULL	0
	Integer.MIN_VALUE 以上かつ Integer.MAX_VALUE 以下	？パラメタ値の整数部分の値を int 値にしたもの
	上記以外	SQLException を投入
BOOLEAN	NULL	0
	true	1
	false	0
その他	－	SQLException を投入

（凡例）

－：該当しません。

【発生する例外】

次の場合，SQLException を投入します。

- CallableStatement オブジェクトに対して close()が既に発行されている場合
- この CallableStatement オブジェクトを生成した Connection オブジェクトに対して close()が既に発行されている場合
- 存在しない？パラメタの番号を指定した場合
- 指定した？パラメタが IN パラメタである場合
- 指定した？パラメタはこのメソッドでは取得できないデータ型である場合

(t)　getInt(String parameterName)

【機能】

指定された？パラメタの値を，Java プログラミング言語の int として取得します。

HiRDB のデータ型による？パラメタ値と戻り値の関係については，「getInt(int parameterIndex)」の【機能詳細】を参照してください。

【形式】

```
public synchronized int getInt (String parameterName) throws SQLException
```

【引数】

String parameterName：

パラメタ名。パラメタ名の大文字と小文字は区別しません。parameterName の文字列すべてを？パラメタ名として扱うため，文字列に引用符「"」が含まれている場合，引用符「"」も？パラメタ名に含まれます。

【戻り値】

指定された？パラメタの値（値が NULL の場合，0 を返します）

【発生する例外】

次の場合，SQLException を投入します。

- CallableStatement オブジェクトに対して close()が既に発行されている場合
- この CallableStatement オブジェクトを生成した Connection オブジェクトに対して close()が既に発行されている場合
- 存在しないパラメタ名を指定した場合（parameterName の指定値が null 又は 0 長文字列の場合も含む）
- 指定した？パラメタが IN パラメタである場合
- 指定した？パラメタはこのメソッドでは取得できないデータ型である場合

(u)　getLong(int parameterIndex)

【機能】

指定された？パラメタの値を，Java プログラミング言語の long として取得します。

【形式】

```
public synchronized long getlong(int parameterIndex) throws SQLException
```

【引数】

int parameterIndex：

？パラメタの番号。最初のパラメタは 1，2 番目のパラメタは 2，...と指定します。

【戻り値】

指定された？パラメタの値（値が NULL の場合，0 を返します）

【機能詳細】

指定された？パラメタの値を，Java プログラミング言語の long として取得します。

HiRDB のデータ型による？パラメタ値と戻り値の関係を次に示します。

HiRDB のデータ型	？パラメタの値	戻り値
[M ｜ N][VAR]CHAR	NULL	0
	[半角空白]整数の文字列表現又は 10 進数の文字列表現かつ 15 文字以内[半角空白]	？パラメタ値の整数部分の値を long 値にしたもの
	[半角空白]整数の文字列表現又は 10 進数の文字列表現であり，文字以上又は浮動小数点の文字列表現[半角空白]であり，かつ Long.MIN_VALUE 以上，Long.MAX_VALUE 以下	？パラメタ値の整数部分の値を long 値にしたもの
	[半角空白]整数の文字列表現又は 10 進数の文字列表現であり，16 文字以上又は浮動小数点の文字列表現[半角空白]であり，かつ Long.MAX_VALUE より大きいか Long.MIN_VALUE より小さい	SQLException を投入
	[半角空白]-Infinity[半角空白]	SQLException を投入
	[半角空白][+]Infinity[半角空白]	SQLException を投入
	[半角空白][+｜-]NaN[半角空白]	SQLException を投入
	上記以外（double 値又は BigDecimal オブジェクトにできない）	SQLException を投入
SMALLINT	NULL	0
	NULL 以外	？パラメタ値を long 値にしたもの
INTEGER	NULL	0
	NULL 以外	？パラメタ値を long 値にしたもの
REAL	NULL	0
	Infinity	SQLException を投入
	-Infinity	SQLException を投入
	NaN	SQLException を投入
	Long.MIN_VALUE 以上かつ，Long.MAX_VALUE 以下	？パラメタ値の整数部分の値を long 値にしたもの
	上記以外	SQLException を投入
FLOAT	NULL	0
	Infinity	SQLException を投入
	-Infinity	SQLException を投入

HiRDB のデータ型	？パラメタの値	戻り値
	NaN	SQLException を投入
	Long.MIN_VALUE 以上かつ， Long.MAX_VALUE 以下	？パラメタ値の整数部分の値を long 値にしたもの
	上記以外	SQLException を投入
DECIMAL	NULL	0
	Long.MIN_VALUE 以上かつ Long.MAX_VALUE 以下	？パラメタ値の整数部分の値を long 値にしたもの
	上記以外	SQLException を投入
BINARY BLOB	NULL	0
	0 長データ	0
	1 バイト以上	最大 8 バイトをリトルエンディアン形式で long 値にしたもの
BOOLEAN	NULL	0
	true	1
	false	0
その他	－	SQLException を投入

（凡例）

－：該当しません。

【発生する例外】

次の場合，SQLException を投入します。

- CallableStatement オブジェクトに対して close()が既に発行されている場合
- この CallableStatement オブジェクトを生成した Connection オブジェクトに対して close()が既に発行されている場合
- 存在しない？パラメタの番号を指定した場合
- 指定した？パラメタが IN パラメタである場合
- 指定した？パラメタはこのメソッドでは取得できないデータ型である場合

(v)　getLong(String parameterName)

【機能】

指定された？パラメタの値を，Java プログラミング言語の long として取得します。

HiRDB のデータ型による？パラメタ値と戻り値の関係については，「getLong(int parameterIndex)」の【機能詳細】を参照してください。

【形式】

```
public synchronized long getLong (String parameterName) throws SQLException
```

【引数】

String parameterName：

パラメタ名。パラメタ名の大文字と小文字は区別しません。parameterName の文字列すべてを？パラメタ名として扱うため，文字列に引用符「"」が含まれている場合，引用符「"」も？パラメタ名に含まれます。

【戻り値】

指定された？パラメタの値（値が NULL の場合，0 を返します）

【発生する例外】

次の場合，SQLException を投入します。

- CallableStatement オブジェクトに対して close()が既に発行されている場合
- この CallableStatement オブジェクトを生成した Connection オブジェクトに対して close()が既に発行されている場合
- 存在しないパラメタ名を指定した場合（parameterName の指定値が null 又は 0 長文字列の場合も含む）
- 指定した？パラメタが IN パラメタである場合
- 指定した？パラメタはこのメソッドでは取得できないデータ型である場合

(w)　getObject(int parameterIndex)

【機能】

？パラメタの値を，Java プログラミング言語の java.lang.Object として取得します。

【形式】

```
public synchronized Object getObject (int parameterIndex) throws SQLException
```

【引数】

int parameterIndex：

？パラメタの番号。最初のパラメタは 1，2 番目のパラメタは 2，...と指定します。

【戻り値】

指定された？パラメタの値を持つ java.lang.Object（値が NULL の場合，null を返します）

【機能詳細】

？パラメタの値を，Java プログラミング言語の java.lang.Object として取得します。

HiRDB のデータ型による？パラメタ値と戻り値の関係を次に示します。

HiRDB のデータ型	？パラメタの値	戻り値
[M | N][VAR]CHAR	NULL	null
	NULL 以外	？パラメタ値
SMALLINT	NULL	null
	NULL 以外	？パラメタ値で生成した Integer オブジェクト
INTEGER	NULL	null
	NULL 以外	？パラメタ値で生成した Integer オブジェクト
REAL	NULL	null
	NULL 以外	？パラメタ値で生成した Float オブジェクト
FLOAT	NULL	null
	NULL 以外	？パラメタ値で生成した Double オブジェクト
DECIMAL	NULL	null
	NULL 以外	？パラメタ値
DATE	NULL	null
	NULL 以外	？パラメタ値で生成した java.sql.Date オブジェクト
TIME	NULL	null
	NULL 以外	？パラメタ値で生成した java.sql.Time オブジェクト
TIMESTAMP	NULL	null
	NULL 以外	？パラメタ値で生成した java.sql.Timestamp オブジェクト
BINARY BLOB	NULL	null
	NULL 以外	？パラメタ値
BOOLEAN	NULL	null
	NULL 以外	？パラメタ値で生成した Boolean オブジェクト

【発生する例外】

次の場合，SQLException を投入します。

- CallableStatement オブジェクトに対して close()が既に発行されている場合

- この CallableStatement オブジェクトを生成した Connection オブジェクトに対して close()が既に発行されている場合
- 存在しない？パラメタの番号を指定した場合
- 指定した？パラメタが IN パラメタである場合
- データベースアクセスエラーが発生した場合

(x)　getObject(String parameterName)

【機能】

？パラメタの値を，Java プログラミング言語の java.lang.Object として取得します。

HiRDB のデータ型による？パラメタ値と戻り値の関係については，「getObject(int parameterIndex)」の【機能詳細】を参照してください。

【形式】

```
public synchronized Object getObject (String parameterName) throws SQLException
```

【引数】

String parameterName：

パラメタ名。パラメタ名の大文字と小文字は区別しません。parameterName の文字列すべてを？パラメタ名として扱うため，文字列に引用符「"」が含まれている場合，引用符「"」も？パラメタ名に含まれます。

【戻り値】

指定された？パラメタの値を持つ java.lang.Object（値が NULL の場合，null を返します）

【発生する例外】

次の場合，SQLException を投入します。

- CallableStatement オブジェクトに対して close()が既に発行されている場合
- この CallableStatement オブジェクトを生成した Connection オブジェクトに対して close()が既に発行されている場合
- 存在しないパラメタ名を指定した場合（parameterName の指定値が null 又は 0 長文字列の場合も含む）
- 指定した？パラメタが IN パラメタである場合
- 指定した？パラメタはこのメソッドでは取得できないデータ型である場合

(y)　getShort(int parameterIndex)

【機能】

指定された？パラメタの値を，Java プログラミング言語の short として取得します。

【形式】

```
public synchronized short getShort(int parameterIndex) throws SQLException
```

【引数】

int parameterIndex：

？パラメタの番号。最初のパラメタは 1，2 番目のパラメタは 2，...と指定します。

【戻り値】

指定された？パラメタの値（値が NULL の場合，0 を返します）

【機能詳細】

指定された？パラメタの値を，Java プログラミング言語の short として取得します。

HiRDB のデータ型による？パラメタ値と戻り値の関係を次に示します。

HiRDB のデータ型	？パラメタの値	戻り値
[M | N][VAR]CHAR	NULL	0
	[半角空白]整数の文字列表現，10 進数の文字列表現又は浮動小数点数の文字列表現[半角空白]であり，かつ Short.MIN_VALUE 以上，Short.MAX_VALUE 以下	？パラメタ値の整数部分の値を short 値にしたもの
	[半角空白]整数の文字列表現，10 進数の文字列表現又は浮動小数点数の文字列表現[半角空白]であり，かつ Short.MAX_VALUE より大きいか Short.MIN_VALUE より小さい	SQLException を投入
	[半角空白]-Infinity[半角空白]	SQLException を投入
	[半角空白][+]Infinity[半角空白]	SQLException を投入
	[半角空白][+|-]NaN[半角空白]	SQLException を投入
	上記以外（double 値にできない）	SQLException を投入
SMALLINT	NULL	0
	NULL 以外	？パラメタ値
INTEGER	NULL	0
	Short.MIN_VALUE 以上かつ，Short.MAX_VALUE 以下	？パラメタ値を short 値にしたもの
	上記以外	SQLException を投入
REAL	NULL	0
	Infinity	SQLException を投入
	-Infinity	SQLException を投入
	NaN	SQLException を投入

HiRDB のデータ型	？パラメタの値	戻り値
	Short.MIN_VALUE 以上かつ， Short.MAX_VALUE 以下	？パラメタ値の整数部分の値を short 値にしたもの
	上記以外	SQLException を投入
FLOAT	NULL	0
	Infinity	SQLException を投入
	-Infinity	SQLException を投入
	NaN	SQLException を投入
	Short.MIN_VALUE 以上かつ， Short.MAX_VALUE 以下	？パラメタ値の整数部分の値を short 値にしたもの
	上記以外	SQLException を投入
DECIMAL	NULL	0
	Short.MIN_VALUE 以上かつ Short.MAX_VALUE 以下	？パラメタ値の整数部分の値を short 値にしたもの
	上記以外	SQLException を投入
BOOLEAN	NULL	0
	true	1
	false	0
その他	－	SQLException を投入

（凡例）

－：該当しません。

【発生する例外】

次の場合，SQLException を投入します。

- CallableStatement オブジェクトに対して close()が既に発行されている場合
- この CallableStatement オブジェクトを生成した Connection オブジェクトに対して close()が既に発行されている場合
- 存在しない？パラメタの番号を指定した場合
- 指定した？パラメタが IN パラメタである場合
- 指定した？パラメタはこのメソッドでは取得できないデータ型である場合

(z)　getShort(String parameterName)

【機能】

指定された？パラメタの値を，Java プログラミング言語の short として取得します。

HiRDB のデータ型による？パラメタ値と戻り値の関係は，「getShort(int parameterIndex)」の【機能詳細】を参照してください。

【形式】

```
public synchronized short getShort (String parameterName) throws SQLException
```

【引数】

String parameterName：

パラメタ名。パラメタ名の大文字と小文字は区別しません。parameterName の文字列すべてを？パラメタ名として扱うため，文字列に引用符「"」が含まれている場合，引用符「"」も？パラメタ名に含まれます。

【戻り値】

指定された？パラメタの値（値が NULL の場合，0 を返します）

【発生する例外】

次の場合，SQLException を投入します。

- CallableStatement オブジェクトに対して close()が既に発行されている場合
- この CallableStatement オブジェクトを生成した Connection オブジェクトに対して close()が既に発行されている場合
- 存在しないパラメタ名を指定した場合（parameterName の指定値が null 又は 0 長文字列の場合も含む）
- 指定した？パラメタが IN パラメタである場合
- 指定した？パラメタはこのメソッドでは取得できないデータ型である場合

(aa)　getString(int parameterIndex)

【機能】

指定された？パラメタの値を，Java プログラミング言語の java.lang.String オブジェクトとして取得します。

【形式】

```
public synchronized String getString(int parameterIndex) throws SQLException
```

【引数】

int parameterIndex：

？パラメタの番号。最初のパラメタは 1，2 番目のパラメタは 2，...と指定します。

【戻り値】

指定された？パラメタの値を持つ java.lang.String オブジェクト（値が NULL の場合，null を返します）

【機能詳細】

指定された？パラメタの値を，Java プログラミング言語の java.lang.String オブジェクトとして取得します。

HiRDB のデータ型による？パラメタ値と戻り値の関係を次に示します。

HiRDB のデータ型	？パラメタの値	戻り値
[M ｜ N][VAR]CHAR	NULL	null
	NULL 以外	？パラメタ値 ただし，システムプロパティ HiRDB_for_Java_DAB_CONVERT_NULL に TRUE を設定し，かつ？パラメタの値が 0 長文字列の場合は null
SMALLINT	NULL	null
	NULL 以外	？パラメタ値を文字列表現にした String オブジェクト
INTEGER	NULL	null
	NULL 以外	？パラメタ値を文字列表現にした String オブジェクト
REAL	NULL	null
	Infinity	文字列"Infinity"の String オブジェクト
	-Infinity	文字列"-Infinity"の String オブジェクト
	NaN	文字列"NaN"の String オブジェクト
	上記以外	？パラメタ値を文字列表現にした String オブジェクト
FLOAT	NULL	null
	Infinity	文字列"Infinity"の String オブジェクト
	-Infinity	文字列"-Infinity"の String オブジェクト
	NaN	文字列"NaN"の String オブジェクト
	上記以外	？パラメタ値を文字列表現にした String オブジェクト
DECIMAL	NULL	null
	NULL 以外	？パラメタ値を文字列表現にした String オブジェクト
DATE	NULL	null
	NULL 以外	JdbConvert.convertDate()で取得した yyyy-MM-DD 形式の文字列の String オブジェクト
TIME	NULL	null
	NULL 以外	hh:mm:ss 形式の文字列の String オブジェクト
TIMESTAMP	NULL	null
	NULL 以外	yyyy-MM-DD△hh:mm:ss[.ffffff]形式の文字列の String オブジェクト

HiRDB のデータ型	？パラメタの値	戻り値
BINARY BLOB	NULL	null
	NULL 以外	？パラメタ値の String オブジェクト
BOOLEAN	NULL	null
	true	文字列"true"の String オブジェクト
	false	文字列"false"の String オブジェクト

【発生する例外】

次の場合，SQLException を投入します。

- CallableStatement オブジェクトに対して close()が既に発行されている場合
- この CallableStatement オブジェクトを生成した Connection オブジェクトに対して close()が既に発行されている場合
- 存在しない？パラメタの番号を指定した場合
- 指定した？パラメタが IN パラメタである場合
- 指定した？パラメタはこのメソッドでは取得できないデータ型である場合

(ab)　getString(String parameterName)

【機能】

指定された？パラメタの値を，Java プログラミング言語の java.lang.String オブジェクトとして取得します。

HiRDB のデータ型による？パラメタ値と戻り値の関係は，「getString(int parameterIndex)」の【機能詳細】を参照してください。

【形式】

```
public synchronized String getString(String parameterName) throws SQLException
```

【引数】

String parameterName：

パラメタ名。パラメタ名の大文字と小文字は区別しません。parameterName の文字列すべてを？パラメタ名として扱うため，文字列に引用符「"」が含まれている場合，引用符「"」も？パラメタ名に含まれます。

【戻り値】

指定された？パラメタの値を持つ java.lang.String オブジェクト（値が NULL の場合，null を返します）。

【発生する例外】

次の場合，SQLException を投入します。

- CallableStatement オブジェクトに対して close()が既に発行されている場合

- この CallableStatement オブジェクトを生成した Connection オブジェクトに対して close()が既に発行されている場合
- 存在しないパラメタ名を指定した場合（parameterName の指定値が null 又は 0 長文字列の場合も含む）
- 指定した？パラメタが IN パラメタである場合
- 指定した？パラメタはこのメソッドでは取得できないデータ型である場合

(ac)　getTime(int parameterIndex)

【機能】

指定された？パラメタの値を，Java プログラミング言語の java.sql.Time オブジェクトとして取得します。

【形式】

```
public synchronized java.sql.Time getTime(int parameterIndex) throws SQLException
```

【引数】

int parameterIndex：

？パラメタの番号。最初のパラメタは 1，2 番目のパラメタは 2，...と指定します。

【戻り値】

指定された？パラメタの値を持つ java.sql.Time オブジェクト（値が NULL の場合，null を返します）

【機能詳細】

指定された？パラメタの値を，Java プログラミング言語の java.sql.Time オブジェクトとして取得します。

HiRDB のデータ型による？パラメタ値と戻り値の関係を次に示します。

HiRDB のデータ型	？パラメタの値	戻り値
[M | N][VAR]CHAR	NULL	null
	[半角空白]時刻形式[半角空白]	？パラメタ値の前後の半角空白を取り除き java.sql.Time オブジェクトにしたもの
	上記以外	SQLException を返します
TIME	NULL	null
	NULL 以外	？パラメタ値を java.sql.Time オブジェクトにしたもの
TIMESTAMP	NULL	null
	NULL 以外	？パラメタ値を java.sql.Time オブジェクトにしたもの
その他	−	SQLException を返します

（凡例）

－：該当しません。

【発生する例外】

次の場合，SQLException を投入します。

- CallableStatement オブジェクトに対して close()が既に発行されている場合
- この CallableStatement オブジェクトを生成した Connection オブジェクトに対して close()が既に発行されている場合
- 存在しない？パラメタの番号を指定した場合
- 指定した？パラメタが IN パラメタである場合
- 指定した？パラメタはこのメソッドでは取得できないデータ型である場合
- ？パラメタの値が java.sql.Time として取得できない値の場合

(ad)　getTime(int parameterIndex, java.util.Calendar cal)

【機能】

指定された？パラメタの値を，Java プログラミング言語の java.sql.Time オブジェクトとして取得します。このメソッドは指定されたカレンダーを使って取得した適切なミリ秒値から時刻を作成します。

HiRDB のデータ型による？パラメタ値と戻り値の関係については，「getTime(int parameterIndex)」の【機能詳細】を参照してください。

【形式】

```
public synchronized java.sql.Time getTime (int parameterIndex, java.util.Calendar cal) th
rows SQLException
```

【引数】

int parameterIndex：

？パラメタの番号。最初のパラメタは 1，2 番目のパラメタは 2，...と指定します。

java.util.Calendar cal：

データベースに格納されている値のタイムゾーンが設定されたカレンダー

【戻り値】

指定された？パラメタの値を持つ java.sql.Time オブジェクト（値が NULL の場合，null を返します）

【発生する例外】

次の場合，SQLException を投入します。

- CallableStatement オブジェクトに対して close()が既に発行されている場合
- この CallableStatement オブジェクトを生成した Connection オブジェクトに対して close()が既に発行されている場合
- 存在しない？パラメタの番号を指定した場合
- 指定した？パラメタが IN パラメタである場合

- 指定した？パラメタはこのメソッドでは取得できないデータ型である場合
- ？パラメタの値が java.sql.Time として取得できない値の場合

(ae)　getTime(String parameterName)

【機能】

指定された？パラメタの値を，Java プログラミング言語の java.sql.Time オブジェクトとして取得します。

HiRDB のデータ型による？パラメタ値と戻り値の関係については，「getTime(int parameterIndex)」の【機能詳細】を参照してください。

【形式】

```
public synchronized java.sql.Time getTime (String parameterName) throws SQLException
```

【引数】

String parameterName：

パラメタ名。パラメタ名の大文字と小文字は区別しません。parameterName の文字列すべてを？パラメタ名として扱うため，文字列に引用符「"」が含まれている場合，引用符「"」も？パラメタ名に含まれます。

【戻り値】

指定された？パラメタの値を持つ java.sql.Time オブジェクト（値が NULL の場合，null を返します）

【発生する例外】

次の場合，SQLException を投入します。

- CallableStatement オブジェクトに対して close()が既に発行されている場合
- この CallableStatement オブジェクトを生成した Connection オブジェクトに対して close()が既に発行されている場合
- 存在しないパラメタ名を指定した場合（parameterName の指定値が null 又は 0 長文字列の場合も含む）
- 指定した？パラメタが IN パラメタである場合
- 指定した？パラメタはこのメソッドでは取得できないデータ型である場合
- ？パラメタの値が java.sql.Time として取得できない値の場合

(af)　getTime(String parameterName, java.util.Calendar cal)

【機能】

指定された？パラメタの値を，Java プログラミング言語の java.sql.Time オブジェクトとして取得します。このメソッドは指定されたカレンダーを使って取得した適切なミリ秒値から時刻を作成します。

HiRDB のデータ型による？パラメタ値と戻り値の関係については，「getTime(int parameterIndex)」の【機能詳細】を参照してください。

【形式】

```
public synchronized java.sql.Time getTime (String parameterName, java.util.Calendar cal)
throws SQLException
```

【引数】

String parameterName：

パラメタ名。パラメタ名の大文字と小文字は区別しません。parameterName の文字列すべてを？パラメタ名として扱うため，文字列に引用符「"」が含まれている場合，引用符「"」も？パラメタ名に含まれます。

java.util.Calendar cal：

データベースに格納されている値のタイムゾーンが設定されたカレンダー

【戻り値】

指定された？パラメタの値を持つ java.sql.Time オブジェクト（値が NULL の場合，null を返します）

【発生する例外】

次の場合，SQLException を投入します。

- CallableStatement オブジェクトに対して close()が既に発行されている場合
- この CallableStatement オブジェクトを生成した Connection オブジェクトに対して close()が既に発行されている場合
- 存在しないパラメタ名を指定した場合（parameterName の指定値が null 又は 0 長文字列の場合も含む）
- 指定した？パラメタが IN パラメタである場合
- 指定した？パラメタはこのメソッドでは取得できないデータ型である場合
- ？パラメタの値が java.sql.Time として取得できない値の場合

(ag)　getTimestamp(int parameterIndex)

【機能】

指定された？パラメタの値を，Java プログラミング言語の java.sql.Timestamp オブジェクトとして取得します。

【形式】

```
public synchronized java.sql.Timestamp getTimestamp (int parameterIndex) throws SQLExcept
ion
```

【引数】

int parameterIndex：

？パラメタの番号。最初のパラメタは 1，2 番目のパラメタは 2，...と指定します。

【戻り値】

指定された？パラメタの値を持つ java.sql.Timestamp オブジェクト（値が NULL の場合，null を返します）

【機能詳細】

指定された？パラメタの値を，Java プログラミング言語の java.sql.Timestamp オブジェクトとして取得します。

HiRDB のデータ型による？パラメタ値と戻り値の関係については「TIME 型，DATE 型，及び TIMESTAMP 型列のデータ変換処理」を参照してください。TIME 型，DATE 型，TIMESTAMP 型，文字列型以外のデータ型を取得した場合，SQLException を投入します。

【発生する例外】

次の場合，SQLException を投入します。

- CallableStatement オブジェクトに対して close()が既に発行されている場合
- この CallableStatement オブジェクトを生成した Connection オブジェクトに対して close()が既に発行されている場合
- 存在しない？パラメタの番号を指定した場合
- 指定した？パラメタが IN パラメタである場合
- 指定した？パラメタはこのメソッドでは取得できないデータ型である場合
- ？パラメタの値が java.sql.Timestamp として取得できない値の場合

(ah)　getTimestamp(int parameterIndex, java.util.Calendar cal)

【機能】

指定された？パラメタの値を，Java プログラミング言語の java.sql.Timestamp オブジェクトとして取得します。このメソッドは指定されたカレンダーを使って取得した適切なミリ秒値からタイムスタンプを作成します。

各 HiRDB のデータ型での？パラメタ値と戻り値の関係については，「TIME 型，DATE 型，及び TIMESTAMP 型列のデータ変換処理」を参照してください。TIME 型，DATE 型，TIMESTAMP 型，文字列型以外のデータ型を取得した場合，SQLException を投入します。

【形式】

```
public synchronized java.sql.Timestamp getTimestamp (int parameterIndex, java.util.Calendar cal) throws SQLException
```

【引数】

int parameterIndex：

？パラメタの番号。最初のパラメタは 1，2 番目のパラメタは 2，...と指定します。

java.util.Calendar cal：

データベースに格納されている値のタイムゾーンが設定されたカレンダー

【戻り値】

指定された？パラメタの値を持つ java.sql.Timestamp オブジェクト（値が NULL の場合，null を返します）

【発生する例外】

次の場合，SQLException を投入します。

- CallableStatement オブジェクトに対して close()が既に発行されている場合
- この CallableStatement オブジェクトを生成した Connection オブジェクトに対して close()が既に発行されている場合
- 存在しない？パラメタの番号を指定した場合
- 指定した？パラメタが IN パラメタである場合
- 指定した？パラメタはこのメソッドでは取得できないデータ型である場合
- ？パラメタの値が java.sql.Timestamp として取得できない値の場合

(ai)　getTimestamp(String parameterName)

【機能】

指定された？パラメタの値を，Java プログラミング言語の java.sql.Timestamp オブジェクトとして取得します。

各 HiRDB のデータ型での？パラメタ値と戻り値の関係については，「TIME 型，DATE 型，及び TIMESTAMP 型列のデータ変換処理」を参照してください。TIME 型，DATE 型，TIMESTAMP 型，文字列型以外のデータ型を取得した場合，SQLException を投入します。

【形式】

```
public synchronized java.sql.Timestamp getTimestamp (String parameterName) throws SQLExce
ption
```

【引数】

String parameterName：

パラメタ名。パラメタ名の大文字と小文字は区別しません。parameterName の文字列すべてを？パラメタ名として扱うため，文字列に引用符「"」が含まれている場合，引用符「"」も？パラメタ名に含まれます。

【戻り値】

指定された？パラメタの値を持つ java.sql.Timestamp オブジェクト（値が NULL の場合，null を返します）

【発生する例外】

次の場合，SQLException を投入します。

- CallableStatement オブジェクトに対して close()が既に発行されている場合
- この CallableStatement オブジェクトを生成した Connection オブジェクトに対して close()が既に発行されている場合

- 存在しないパラメタ名を指定した場合（parameterName の指定値が null 又は 0 長文字列の場合も含む）
- 指定した？パラメタが IN パラメタである場合
- 指定した？パラメタはこのメソッドでは取得できないデータ型である場合
- ？パラメタの値が java.sql.Timestamp として取得できない値の場合

(aj)　getTimestamp(String parameterName, java.util.Calendar cal)

【機能】

指定された？パラメタの値を，Java プログラミング言語の java.sql.Timestamp オブジェクトとして取得します。このメソッドは指定されたカレンダーを使って取得した適切なミリ秒値からタイムスタンプを作成します。

各 HiRDB のデータ型での？パラメタ値と戻り値の関係については，「TIME 型，DATE 型，及び TIMESTAMP 型列のデータ変換処理」を参照してください。TIME 型，DATE 型，TIMESTAMP 型，文字列型以外のデータ型を取得した場合，SQLException を投入します。

【形式】

```
public synchronized java.sql.Timestamp getTimestamp (String parameterName, java.util.Calendar cal) throws SQLException
```

【引数】

String parameterName：

パラメタ名。パラメタ名の大文字と小文字は区別しません。parameterName の文字列すべてを？パラメタ名として扱うため，文字列に引用符「"」が含まれている場合，引用符「"」も？パラメタ名に含まれます。

java.util.Calendar cal：

データベースに格納されている値のタイムゾーンが設定されたカレンダー

【戻り値】

指定された？パラメタの値を持つ java.sql.Timestamp オブジェクト（値が NULL の場合，null を返します）

【発生する例外】

次の場合，SQLException を投入します。

- CallableStatement オブジェクトに対して close()が既に発行されている場合
- この CallableStatement オブジェクトを生成した Connection オブジェクトに対して close()が既に発行されている場合
- 存在しないパラメタ名を指定した場合（parameterName の指定値が null 又は 0 長文字列の場合も含む）
- 指定した？パラメタが IN パラメタである場合
- 指定した？パラメタはこのメソッドでは取得できないデータ型である場合

- ？パラメタの値が java.sql.Timestamp として取得できない値の場合

(ak)　registerOutParameter(int parameterIndex, int sqlType)

【機能】

指定された OUT パラメタのデータ型を，指定された JDBC の型として登録します。

このメソッドで，DECIMAL 型の OUT パラメタ及び INOUT パラメタを設定する場合，スケール値は 0 とみなします。

【形式】

```
public synchronized void registerOutParameter(int parameterIndex, int sqlType) throws SQL
Exception
```

【引数】

int parameterIndex：

？パラメタの番号。最初のパラメタは 1，2 番目のパラメタは 2，...と指定します。

int sqlType：

登録する JDBC の型

【戻り値】

なし。

【発生する例外】

次の場合，SQLException を投入します。

- CallableStatement オブジェクトに対して close()が既に発行されている場合
- この CallableStatement オブジェクトを生成した Connection オブジェクトに対して，close()が既に発行されている場合
- 存在しない？パラメタの番号を指定した場合
- 指定した？パラメタが IN パラメタである場合
- HiRDB のデータ型とマッピングできない JDBC の型を指定した場合

(al)　registerOutParameter(int parameterIndex, int sqlType, int scale)

【機能】

指定された OUT パラメタのデータ型を，指定された JDBC の型として登録します。

【形式】

```
public synchronized void registerOutParameter(int parameterIndex, int sqlType, int scale)
 throws SQLException
```

【引数】

int parameterIndex：

？パラメタの番号。最初のパラメタは 1，2 番目のパラメタは 2，...と指定します。

int sqlType：

登録する JDBC の型

int scale：

位取り。指定できる範囲は 0≦scale≦38 です。

【戻り値】

なし。

【発生する例外】

次の場合，SQLException を投入します。

- CallableStatement オブジェクトに対して close()が既に発行されている場合
- この CallableStatement オブジェクトを生成した Connection オブジェクトに対して，close()が既に発行されている場合
- 存在しない？パラメタの番号を指定した場合
- 指定した？パラメタが IN パラメタである場合
- HiRDB のデータ型とマッピングできない JDBC の型を指定した場合
- scale に指定した値が 0 未満，又は 38 より大きい場合

(am)　registerOutParameter(String parameterName, int sqlType)

【機能】

指定された OUT パラメタのデータ型を，指定された JDBC の型として登録します。

このメソッドで，DECIMAL 型の OUT パラメタ及び INOUT パラメタを設定する場合，スケール値は 0 とみなします。

【形式】

```
public synchronized void registerOutParameter(String parameterName, int sqlType) throws S
QLException
```

【引数】

String parameterName：

パラメタ名。大文字と小文字は区別しません。parameterName の文字列すべてを？パラメタ名として扱うため，文字列に引用符（''）が含まれている場合，引用符（''）も？パラメタ名に含まれます。

int sqlType：

登録する JDBC の型

【戻り値】

なし。

【発生する例外】

次の場合，SQLException を投入します。

- CallableStatement オブジェクトに対して close()が既に発行されている場合
- この CallableStatement オブジェクトを生成した Connection オブジェクトに対して，close()が既に発行されている場合
- 存在しないパラメタ名を指定した場合

  parameterName の指定値が，null 又は 0 長文字列の場合も含みます。
- 指定した？パラメタが IN パラメタである場合
- HiRDB のデータ型とマッピングできない JDBC の型を指定した場合

(an)　registerOutParameter(String parameterName, int sqlType, int scale)

【機能】

指定された OUT パラメタのデータ型を，指定された JDBC の型として登録します。

【形式】

```
public synchronized void registerOutParameter(String parameterName, int sqlType, int scal
e) throws SQLException
```

【引数】

String parameterName：

パラメタ名。大文字と小文字は区別しません。parameterName の文字列すべてを？パラメタ名として扱うため，文字列に引用符（''）が含まれている場合，引用符（''）も？パラメタ名に含まれます。

int sqlType：

登録する JDBC の型

int scale：

位取り。指定できる範囲は 0≦scale≦38 です。

【戻り値】

なし。

【発生する例外】

次の場合，SQLException を投入します。

- CallableStatement オブジェクトに対して close()が既に発行されている場合
- この CallableStatement オブジェクトを生成した Connection オブジェクトに対して，close()が既に発行されている場合
- 存在しないパラメタ名を指定した場合

parameterName の指定値が，null 又は 0 長文字列の場合も含みます。

- 指定した？パラメタが IN パラメタである場合
- HiRDB のデータ型とマッピングできない JDBC の型を指定した場合
- scale に指定した値が 0 未満，又は 38 より大きい場合

(ao)　setAsciiStream(String parameterName, java.io.InputStream x, int length)

【機能】

指定された java.io.InputStream オブジェクトの持つ値を指定された？パラメタ値に指定された長さだけ設定します。

【形式】

```
public synchronized void setAsciiStream(String parameterName, java.io.InputStream x, int
length) throws SQLException
```

【引数】

String parameterName：

パラメタ名。パラメタ名の大文字と小文字は区別しません。parameterName の文字列すべてを？パラメタ名として扱うため，文字列に引用符「"」が含まれている場合，引用符「"」も？パラメタ名に含まれます。

java.io.InputStream x：

？パラメタに設定する値を持つ InputStream オブジェクト

int length：

設定するストリームのバイト数

【戻り値】

なし。

【機能詳細】

指定された InputStream オブジェクトの持つ値を？パラメタ値に設定します。

このメソッドは，x から入力を終えた後，x に対して close()メソッドを実行しません。

？パラメタの HiRDB のデータ型が[M ｜ N][VAR]CHAR，BINARY，BLOB 以外は SQLException となります。

【発生する例外】

次の場合，SQLException を投入します。

- CallableStatement オブジェクトに対して close()が既に発行されている場合
- この CallableStatement オブジェクトを生成した Connection オブジェクトに対して close()が既に発行されている場合
- 存在しないパラメタ名を指定した場合（parameterName の指定値が null 又は 0 長文字列の場合も含む）

- 指定した？パラメタが OUT パラメタである場合
- length に 0 未満の値が指定された場合
- このメソッドでは設定できない？パラメタのデータ型である場合
- 指定された値が？パラメタのデータ型の範囲外，又は変換できない形式である場合
- ストリームからの読み込みに失敗した場合

(ap)　setBigDecimal(String parameterName, java.math.BigDecimal x)

【機能】

指定された java.math.BigDecimal 値を指定された？パラメタ値に設定します。

【形式】

```
public synchronized void setBigDecimal(String parameterName, java.math.BigDecimal x) thro
ws SQLException
```

【引数】

String parameterName：

パラメタ名。パラメタ名の大文字と小文字は区別しません。parameterName の文字列すべてを？パラメタ名として扱うため，文字列に引用符「"」が含まれている場合，引用符「"」も？パラメタ名に含まれます。

java.math.BigDecimal x：

？パラメタに設定する値を持つ BigDecimal オブジェクト

【戻り値】

なし。

【発生する例外】

次の場合，SQLException を投入します。

- CallableStatement オブジェクトに対して close()が既に発行されている場合
- この CallableStatement オブジェクトを生成した Connection オブジェクトに対して close()が既に発行されている場合
- 存在しないパラメタ名を指定した場合（parameterName の指定値が null 又は 0 長文字列の場合も含む）
- 指定した？パラメタが OUT パラメタである場合
- 指定したデータの整数部のけた数が，表定義の整数部のけた数を超えるため，表定義のデータ型に変換できない場合
- このメソッドでは設定できない？パラメタのデータ型である場合
- 指定された値が？パラメタのデータ型の範囲外，又は変換できない形式である場合

(aq)　setBinaryStream(String parameterName, java.io.InputStream x, int length)

【機能】

指定された？パラメタの値を，Java プログラミング言語の java.math.BigDecimal オブジェクトとして取得します。

【形式】

```
public synchronized void setBinaryStream(String parameterName, java.io.InputStream x, int
 length) throws SQLException
```

【引数】

String parameterName：

パラメタ名。パラメタ名の大文字と小文字は区別しません。parameterName の文字列すべてを？パラメタ名として扱うため，文字列に引用符「"」が含まれている場合，引用符「"」も？パラメタ名に含まれます。

java.io.InputStream x：

？パラメタに設定する値を持つ InputStream オブジェクト

int length：

設定するストリームのバイト数

【戻り値】

なし。

【発生する例外】

次の場合，SQLException を投入します。

- CallableStatement オブジェクトに対して close()が既に発行されている場合
- この CallableStatement オブジェクトを生成した Connection オブジェクトに対して close()が既に発行されている場合
- 存在しないパラメタ名を指定した場合（parameterName の指定値が null 又は 0 長文字列の場合も含む）
- 指定した？パラメタが OUT パラメタである場合
- length に 0 未満の値が指定された場合
- このメソッドでは設定できない？パラメタのデータ型である場合
- 指定された値が？パラメタのデータ型の範囲外，又は変換できない形式である場合
- ストリームからの読み込みに失敗した場合

(ar)　setBoolean(String parameterName, boolean x)

【機能】

指定された boolean 値を？パラメタ値に設定します。

指定した?パラメタの HiRDB のデータ型が CHAR，MCHAR，NCHAR，VARCHAR，MVARCHAR，NVARHAR の場合，?パラメタへの設定値は x が true のときは"true"，false のときは"false△"（△は半角スペース）となります。

【形式】

```
public synchronized void setBoolean(String parameterName, boolean x) throws SQLException
```

【引数】

String parameterName：

パラメタ名。パラメタ名の大文字と小文字は区別しません。parameterName の文字列すべてを？パラメタ名として扱うため，文字列に引用符「"」が含まれている場合，引用符「"」も？パラメタ名に含まれます。

boolean x：

？パラメタに設定する値

【戻り値】

なし。

【発生する例外】

次の場合，SQLException を投入します。

- CallableStatement オブジェクトに対して close()が既に発行されている場合
- この CallableStatement オブジェクトを生成した Connection オブジェクトに対して close()が既に発行されている場合
- 存在しないパラメタ名を指定した場合（parameterName の指定値が null 又は 0 長文字列の場合も含む）
- ？パラメタの HiRDB データ型がこのメソッドでは設定できないデータ型の場合
- 指定した？パラメタが OUT パラメタである場合

(as)　setByte(String parameterName, byte x)

【機能】

指定された byte 値を？パラメタ値に設定します。

【形式】

```
public synchronized void setByte(String parameterName, byte x) throws SQLException
```

【引数】

String parameterName：

パラメタ名。パラメタ名の大文字と小文字は区別しません。parameterName の文字列すべてを？パラメタ名として扱うため，文字列に引用符「"」が含まれている場合，引用符「"」も？パラメタ名に含まれます。

byte x：

？パラメタに設定する値

【戻り値】

なし。

【発生する例外】

次の場合，SQLException を投入します。

- CallableStatement オブジェクトに対して close()が既に発行されている場合
- この CallableStatement オブジェクトを生成した Connection オブジェクトに対して close()が既に発行されている場合
- 存在しないパラメタ名を指定した場合（parameterName の指定値が null 又は 0 長文字列の場合も含む）
- ？パラメタの HiRDB データ型がこのメソッドでは設定できないデータ型の場合
- 指定した？パラメタが OUT パラメタである場合

(at)　setBytes(String parameterName,byte[] x)

【機能】

指定された byte 配列を？パラメタ値に設定します。

【形式】

```
public synchronized void setBytes(String parameterName, byte[] x) throws SQLException
```

【引数】

String parameterName：

パラメタ名。パラメタ名の大文字と小文字は区別しません。parameterName の文字列すべてを？パラメタ名として扱うため，文字列に引用符「"」が含まれている場合，引用符「"」も？パラメタ名に含まれます。

byte[] x：

？パラメタに設定する値を持つ byte 配列

【戻り値】

なし。

【機能詳細】

？パラメタの HiRDB のデータ型が BINARY，BLOB 以外は SQLException となります。

このメソッドでは，byte 配列中の設定値を参照しません。executeXXX()メソッド実行時に byte 配列中の設定値を参照します。そのため，このメソッド実行から executeXXX()メソッド実行するまでの間に byte 配列中の設定値を変更した場合，変更後の設定値が？パラメタの値となります。

【発生する例外】

次の場合，SQLException を投入します。

- CallableStatement オブジェクトに対して close()が既に発行されている場合
- この CallableStatement オブジェクトを生成した Connection オブジェクトに対して close()が既に発行されている場合
- 存在しないパラメタ名を指定した場合（parameterName の指定値が null 又は 0 長文字列の場合も含む）
- ？パラメタの HiRDB データ型がこのメソッドでは設定できないデータ型の場合
- 指定された値が列のデータ型の範囲外，又は変換できない形式の場合
- 指定した？パラメタが OUT パラメタである場合

(au)　setCharacterStream(String parameterName, Reader x, int length)

【機能】

指定された Reader オブジェクトの持つ値を？パラメタ値に設定します。

？パラメタの HiRDB のデータ型が[M ｜ N][VAR]CHAR，BINARY，BLOB 以外は SQLException となります。

【形式】

```
public synchronized void setCharacterStream(String parameterName, Reader x, int length) t
hrows SQLException
```

【引数】

String parameterName：

パラメタ名。パラメタ名の大文字と小文字は区別しません。parameterName の文字列すべてを？パラメタ名として扱うため，文字列に引用符「"」が含まれている場合，引用符「"」も？パラメタ名に含まれます。

Reader x：

？パラメタに設定する値を持つ Reader オブジェクト

int length：

文字数

【戻り値】

なし。

【発生する例外】

次の場合，SQLException を投入します。

- CallableStatement オブジェクトに対して close()が既に発行されている場合
- この CallableStatement オブジェクトを生成した Connection オブジェクトに対して close()が既に発行されている場合
- length に 0 未満の値が指定された場合

- 存在しないパラメタ名を指定した場合（parameterName の指定値が null 又は 0 長文字列の場合も含む）
- ？パラメタの HiRDB データ型がこのメソッドでは設定できないデータ型の場合
- エンコード失敗
- 指定した？パラメタが OUT パラメタである場合

(av)　setDate(String parameterName, java.sql.Date x)

【機能】

指定された java.sql.Date オブジェクトの持つ値を？パラメタ値に設定します。

【形式】

```
public synchronized void setDate(String parameterName, java.sql.Date x) throws SQLExcepti
on
```

【引数】

String parameterName：

パラメタ名。パラメタ名の大文字と小文字は区別しません。parameterName の文字列すべてを？パラメタ名として扱うため，文字列に引用符「"」が含まれている場合，引用符「"」も？パラメタ名に含まれます。

java.sql.Date x：

？パラメタに設定する値を持つ java.sql.Date オブジェクト

【戻り値】

なし。

【機能詳細】

ローカルタイムで指定された java.sql.Date オブジェクトを指定されたカレンダーのタイムゾーンの時間に変換し，？パラメタ値に設定します。

【発生する例外】

次の場合，SQLException を投入します。

- CallableStatement オブジェクトに対して close()が既に発行されている場合
- この CallableStatement オブジェクトを生成した Connection オブジェクトに対して close()が既に発行されている場合
- 存在しないパラメタ名を指定した場合（parameterName の指定値が null 又は 0 長文字列の場合も含む）
- ？パラメタの HiRDB データ型がこのメソッドでは設定できないデータ型の場合
- 指定された値が列のデータ型の範囲外，又は変換できない形式の場合
- 指定した？パラメタが OUT パラメタである場合

(aw)　setDate(String parameterName, java.sql.Date x,Calendar cal)

【機能】

ローカルタイムで指定された java.sql.Date オブジェクトを指定されたカレンダーのタイムゾーンの時間に変換し，？パラメタ値に設定します。

【形式】

```
public synchronized void setDate(String parameterName, java.sql.Date x,Calendar cal) thro
ws SQLException
```

【引数】

String parameterName：

パラメタ名。パラメタ名の大文字と小文字は区別しません。parameterName の文字列すべてを？パラメタ名として扱うため，文字列に引用符「"」が含まれている場合，引用符「"」も？パラメタ名に含まれます。

java.sql.Date x：

？パラメタに設定する値を持つ java.sql.Date オブジェクト

Calendar cal：

データベースに格納する値のタイムゾーンが設定されたカレンダー。null が指定された場合，JavaVM のデフォルトのタイムゾーンのカレンダーが指定されたとします。

【戻り値】

なし。

【発生する例外】

次の場合，SQLException を投入します。

- CallableStatement オブジェクトに対して close()が既に発行されている場合
- この CallableStatement オブジェクトを生成した Connection オブジェクトに対して close()が既に発行されている場合
- 存在しないパラメタ名を指定した場合（parameterName の指定値が null 又は 0 長文字列の場合も含む）
- ？パラメタの HiRDB データ型がこのメソッドでは設定できないデータ型の場合
- 指定された値が列のデータ型の範囲外，又は変換できない形式の場合
- 指定した？パラメタが OUT パラメタである場合

(ax)　setDouble(String parameterName, double x)

【機能】

指定された double 値を？パラメタ値に設定します。

【形式】

```
public synchronized void setDouble(String parameterName, double x) throws SQLException
```

【引数】

String parameterName：

パラメタ名。パラメタ名の大文字と小文字は区別しません。parameterName の文字列すべてを？パラメタ名として扱うため，文字列に引用符「"」が含まれている場合，引用符「"」も？パラメタ名に含まれます。

double x：

？パラメタに設定する値

【戻り値】

なし。

【発生する例外】

次の場合，SQLException を投入します。

- CallableStatement オブジェクトに対して close()が既に発行されている場合
- この CallableStatement オブジェクトを生成した Connection オブジェクトに対して close()が既に発行されている場合
- 存在しないパラメタ名を指定した場合（parameterName の指定値が null 又は 0 長文字列の場合も含む）
- ？パラメタの HiRDB データ型がこのメソッドでは設定できないデータ型の場合
- 指定された値が列のデータ型の範囲外，又は変換できない形式の場合
- 指定した？パラメタが OUT パラメタである場合

(ay)　setFloat(String parameterName, float x)

【機能】

指定された float 値を？パラメタ値に設定します。

【形式】

```
public synchronized void setFloat(String parameterName, float x) throws SQLException
```

【引数】

String parameterName：

パラメタ名。パラメタ名の大文字と小文字は区別しません。parameterName の文字列すべてを？パラメタ名として扱うため，文字列に引用符「"」が含まれている場合，引用符「"」も？パラメタ名に含まれます。

float x：

？パラメタに設定する値

【戻り値】

なし。

【発生する例外】

次の場合，SQLException を投入します。

- CallableStatement オブジェクトに対して close()が既に発行されている場合
- この CallableStatement オブジェクトを生成した Connection オブジェクトに対して close()が既に発行されている場合
- 存在しないパラメタ名を指定した場合（parameterName の指定値が null 又は 0 長文字列の場合も含む）
- ？パラメタの HiRDB データ型がこのメソッドでは設定できないデータ型の場合
- 指定された値が列のデータ型の範囲外，又は変換できない形式の場合
- 指定した？パラメタが OUT パラメタである場合

(az)　setInt(String parameterName, int x)

【機能】

指定された int 値を？パラメタ値に設定します。

【形式】

```
public synchronized void setInt(String parameterName, int x) throws SQLException
```

【引数】

String parameterName：

パラメタ名。パラメタ名の大文字と小文字は区別しません。parameterName の文字列すべてを？パラメタ名として扱うため，文字列に引用符「"」が含まれている場合，引用符「"」も？パラメタ名に含まれます。

int x：

？パラメタに設定する値

【戻り値】

なし。

【発生する例外】

次の場合，SQLException を投入します。

- CallableStatement オブジェクトに対して close()が既に発行されている場合
- この CallableStatement オブジェクトを生成した Connection オブジェクトに対して close()が既に発行されている場合
- 存在しないパラメタ名を指定した場合（parameterName の指定値が null 又は 0 長文字列の場合も含む）

- ？パラメタの HiRDB データ型がこのメソッドでは設定できないデータ型の場合
- 指定された値が列のデータ型の範囲外，又は変換できない形式の場合
- 指定した？パラメタが OUT パラメタである場合

(ba)　setLong(String parameterName, long x)

【機能】

指定された long 値を？パラメタ値に設定します。

【形式】

```
public synchronized void setLong(String parameterName, long x) throws SQLException
```

【引数】

String parameterName：

パラメタ名。パラメタ名の大文字と小文字は区別しません。parameterName の文字列すべてを？パラメタ名として扱うため，文字列に引用符「"」が含まれている場合，引用符「"」も？パラメタ名に含まれます。

long x：

？パラメタに設定する値

【戻り値】

なし。

【発生する例外】

次の場合，SQLException を投入します。

- CallableStatement オブジェクトに対して close()が既に発行されている場合
- この CallableStatement オブジェクトを生成した Connection オブジェクトに対して close()が既に発行されている場合
- 存在しないパラメタ名を指定した場合（parameterName の指定値が null 又は 0 長文字列の場合も含む）
- ？パラメタの HiRDB データ型がこのメソッドでは設定できないデータ型の場合
- 指定された値が列のデータ型の範囲外，又は変換できない形式の場合
- 指定した？パラメタが OUT パラメタである場合

(bb)　setNull(String parameterName,int sqlType)

【機能】

指定された？パラメタに NULL 値を設定します。

引数 sqlType はこのドライバでは無視します。

【形式】

```
public synchronized void setNull(String parameterName,int sqlType) throws SQLException
```

【引数】

String parameterName：

パラメタ名。パラメタ名の大文字と小文字は区別しません。parameterName の文字列すべてを？パラメタ名として扱うため，文字列に引用符「"」が含まれている場合，引用符「"」も？パラメタ名に含まれます。

int sqlType：

JDBC の SQL データ型

【戻り値】

なし。

【発生する例外】

次の場合，SQLException を投入します。

- CallableStatement オブジェクトに対して close()が既に発行されている場合
- この CallableStatement オブジェクトを生成した Connection オブジェクトに対して close()が既に発行されている場合
- 存在しないパラメタ名を指定した場合（parameterName の指定値が null 又は 0 長文字列の場合も含む）
- 指定した？パラメタが OUT パラメタである場合

(bc)　setObject(String parameterName, Object x)

【機能】

指定されたオブジェクトの持つ値を？パラメタ値に設定します。

【形式】

```
public synchronized void setObject(String parameterName, Object x) throws SQLException
```

【引数】

String parameterName：

パラメタ名。パラメタ名の大文字と小文字は区別しません。parameterName の文字列すべてを？パラメタ名として扱うため，文字列に引用符「"」が含まれている場合，引用符「"」も？パラメタ名に含まれます。

Object x：

？パラメタに設定する値を持つオブジェクト

【戻り値】

なし。

【機能詳細】

指定されたオブジェクトの持つ値を？パラメタ値に設定します。

parameterIndex で指定した?パラメタの型が HiRDB の CHAR，MCHAR，NCHAR，VARCHAR，MVARCHAR，又は NVARHAR で，x が Boolean オブジェクトの場合，?パラメタへの設定値は x が true のときは"true"，false のときは"false△"(△は半角スペース)となります。

x が byte 配列の場合，byte 配列中の設定値を参照しません。executeXXX()メソッド実行時に byte 配列中の設定値を参照します。そのため，このメソッド実行から executeXXX()メソッド実行するまでの間に byte 配列中の設定値を変更した場合，変更後の設定値が？パラメタの値となります。

【発生する例外】

次の場合，SQLException を投入します。

- CallableStatement オブジェクトに対して close()が既に発行されている場合
- この CallableStatement オブジェクトを生成した Connection オブジェクトに対して close()が既に発行されている場合
- 存在しないパラメタ名を指定した場合（parameterName の指定値が null 又は 0 長文字列の場合も含む）
- ？パラメタの HiRDB データ型がこのメソッドでは設定できないデータ型の場合
- 指定された値が列のデータ型の範囲外，又は変換できない形式の場合
- 指定した？パラメタが OUT パラメタである場合

(bd)　setObject(String parameterName, Object x, int targetSqlType)

【機能】

指定されたオブジェクトの持つ値を？パラメタ値に設定します。

【形式】

```
public synchronized void setObject(String parameterName, Object x, int targetSqlType) thr
ows SQLException
```

【引数】

String parameterName：

パラメタ名。パラメタ名の大文字と小文字は区別しません。parameterName の文字列すべてを？パラメタ名として扱うため，文字列に引用符「"」が含まれている場合，引用符「"」も？パラメタ名に含まれます。

Object x：

？パラメタに設定する値を持つオブジェクト

int targetSqlType：

JDBC の SQL データ型

【戻り値】

なし。

【機能詳細】

指定されたオブジェクトの持つ値を？パラメタ値に設定します。

targetSqlType が java.sql.Types.CHAR，java.sql.Types.VARCHAR，又は java.sql.Types.LONGVARCHAR で，x が Boolean オブジェクトの場合，?パラメタへの設定値は x が true のときは"1"，"0"になります。

なお，HiRDB データ型である NCHAR，NVARCHAR 型の?パラメタへの設定値が"1"，"0"になる場合，SQLException を投入します。

x が byte 配列の場合，byte 配列中の設定値を参照しません。executeXXX()メソッド実行時に byte 配列中の設定値を参照します。そのため，このメソッド実行から executeXXX()メソッド実行するまでの間に byte 配列中の設定値を変更した場合，変更後の設定値が？パラメタの値となります。

【発生する例外】

次の場合，SQLException を投入します。

- CallableStatement オブジェクトに対して close()が既に発行されている場合
- この CallableStatement オブジェクトを生成した Connection オブジェクトに対して close()が既に発行されている場合
- 存在しないパラメタ名を指定した場合（parameterName の指定値が null 又は 0 長文字列の場合も含む）
- ？パラメタの HiRDB データ型がこのメソッドでは設定できないデータ型の場合
- 指定された値が列のデータ型の範囲外，又は変換できない形式の場合
- targetSqlType が次の場合

  Types.ARRAY，Types.CLOB，Types.REF，Types.STRUCT
- 指定した？パラメタが OUT パラメタである場合

(be)　setObject(String parameterName, Object x, int targetSqlType, int scale)

【機能】

指定されたオブジェクトの持つ値を？パラメタ値に設定します。

【形式】

```
public synchronized void setObject(String parameterName, Object x, int targetSqlType, int
 scale) throws SQLException
```

【引数】

String parameterName：

パラメタ名。パラメタ名の大文字と小文字は区別しません。parameterName の文字列すべてを？パラメタ名として扱うため，文字列に引用符「"」が含まれている場合，引用符「"」も？パラメタ名に含まれます。

Object x：

？パラメタに設定する値を持つオブジェクト

int targetSqlType：

JDBC の SQL データ型

int scale：

位取り（指定値は無視する）

【戻り値】

なし。

【機能詳細】

指定されたオブジェクトの持つ値を？パラメタ値に設定します。

targetSqlType が java.sql.Types.CHAR，java.sql.Types.VARCHAR，又は java.sql.Types.LONGVARCHAR で，x が Boolean オブジェクトの場合，?パラメタへの設定値は x が true のときは"1"，"0"になります。

なお，HiRDB データ型である NCHAR，NVARCHAR 型の?パラメタへの設定値が"1"，"0"になる場合，SQLException を投入します。

x が byte 配列の場合，byte 配列中の設定値を参照しません。executeXXX()メソッド実行時に byte 配列中の設定値を参照します。そのため，このメソッド実行から executeXXX()メソッド実行するまでの間に byte 配列中の設定値を変更した場合，変更後の設定値が？パラメタの値となります。

【発生する例外】

次の場合，SQLException を投入します。

- CallableStatement オブジェクトに対して close()が既に発行されている場合
- この CallableStatement オブジェクトを生成した Connection オブジェクトに対して close()が既に発行されている場合
- 存在しないパラメタ名を指定した場合（parameterName の指定値が null 又は 0 長文字列の場合も含む）
- ？パラメタの HiRDB データ型がこのメソッドでは設定できないデータ型の場合
- 指定された値が列のデータ型の範囲外，又は変換できない形式の場合
- targetSqlType が次の場合

  Types.ARRAY，Types.CLOB，Types.REF，Types.STRUCT
- 指定した？パラメタが OUT パラメタである場合

(bf)　setShort(String parameterName, short x)

【機能】

指定された short 値を？パラメタ値に設定します。

【形式】

```
public synchronized void setShort(String parameterName, short x) throws SQLException
```

【引数】

String parameterName：

パラメタ名。パラメタ名の大文字と小文字は区別しません。parameterName の文字列すべてを？パラメタ名として扱うため，文字列に引用符「"」が含まれている場合，引用符「"」も？パラメタ名に含まれます。

short x：

？パラメタに設定する値

【戻り値】

なし。

【発生する例外】

次の場合，SQLException を投入します。

- CallableStatement オブジェクトに対して close()が既に発行されている場合
- この CallableStatement オブジェクトを生成した Connection オブジェクトに対して close()が既に発行されている場合
- 存在しないパラメタ名を指定した場合（parameterName の指定値が null 又は 0 長文字列の場合も含む）
- ？パラメタの HiRDB データ型がこのメソッドでは設定できないデータ型の場合
- 指定した？パラメタが OUT パラメタである場合

(bg)　setString(String parameterName, String x)

【機能】

指定された String オブジェクトを？パラメタ値に設定します。

【形式】

```
public synchronized void setString(String parameterName, String x) throws SQLException
```

【引数】

String parameterName：

パラメタ名。パラメタ名の大文字と小文字は区別しません。parameterName の文字列すべてを？パラメタ名として扱うため，文字列に引用符「"」が含まれている場合，引用符「"」も？パラメタ名に含まれます。

String x：

？パラメタに設定する値を持つ String オブジェクト

【戻り値】

なし。

【機能詳細】

引数 x で指定された String オブジェクトを？パラメタに設定します。

引数 x の指定値が 0 長文字列の場合の？パラメタ設定値を次の表に示します。

表 17–34　？パラメタの設定値

？パラメタのデータ型	？パラメタの設定値
[M ｜ N][VAR]CHAR	• システムプロパティ HiRDB_for_Java_DAB_CONVERT_NULL に TRUE を設定している場合 null • 上記以外 0 長文字列
BINARY，又は BLOB	0 長文字列
その他	null

【発生する例外】

次の場合，SQLException を投入します。

- CallableStatement オブジェクトに対して close()が既に発行されている場合
- この CallableStatement オブジェクトを生成した Connection オブジェクトに対して close()が既に発行されている場合
- 存在しないパラメタ名を指定した場合（parameterName の指定値が null 又は 0 長文字列の場合も含む）
- 指定された値が列のデータ型の範囲外，又は変換できない形式の場合
- エンコード失敗
- 指定した？パラメタが OUT パラメタである場合

(bh)　setTime(String parameterName, Time x)

【機能】

指定された java.sql.Time オブジェクトを？パラメタ値に設定します。

【形式】

```
public synchronized void setTime(String parameterName, Time x) throws SQLException
```

【引数】

String parameterName：

パラメタ名。パラメタ名の大文字と小文字は区別しません。parameterName の文字列すべてを？パラメタ名として扱うため，文字列に引用符「"」が含まれている場合，引用符「"」も？パラメタ名に含まれます。

java.sql.Time x：

？パラメタに設定する値を持つ java.sql.Time オブジェクト

【戻り値】

なし。

【発生する例外】

次の場合，SQLException を投入します。

- CallableStatement オブジェクトに対して close()が既に発行されている場合
- この CallableStatement オブジェクトを生成した Connection オブジェクトに対して close()が既に発行されている場合
- 存在しないパラメタ名を指定した場合（parameterName の指定値が null 又は 0 長文字列の場合も含む）
- ？パラメタの HiRDB データ型がこのメソッドでは設定できないデータ型の場合
- 指定した？パラメタが OUT パラメタである場合

(bi)　setTime(String parameterName, java.sql.Time x,Calendar cal)

【機能】

ローカルタイムで指定された java.sql.Time オブジェクトを指定されたカレンダーのタイムゾーンの時間に変換し，？パラメタ値に設定します。

【形式】

```
public synchronized void setTime(String parameterName, java.sql.Time x,Calendar cal) thro
ws SQLException
```

【引数】

String parameterName：

パラメタ名。パラメタ名の大文字と小文字は区別しません。parameterName の文字列すべてを？パラメタ名として扱うため，文字列に引用符「"」が含まれている場合，引用符「"」も？パラメタ名に含まれます。

java.sql.Time x：

？パラメタに設定する値を持つ java.sql.Time オブジェクト

Calendar cal：

データベースに格納する値のタイムゾーンが設定されたカレンダー。null が指定された場合，JavaVM のデフォルトのタイムゾーンのカレンダーが指定されたとします。

【戻り値】

なし。

【発生する例外】

次の場合，SQLException を投入します。

- CallableStatement オブジェクトに対して close()が既に発行されている場合

- この CallableStatement オブジェクトを生成した Connection オブジェクトに対して close()が既に発行されている場合
- 存在しないパラメタ名を指定した場合（parameterName の指定値が null 又は 0 長文字列の場合も含む）
- ？パラメタの HiRDB データ型がこのメソッドでは設定できないデータ型の場合
- 指定した？パラメタが OUT パラメタである場合

(bj)　setTimestamp(String parameterName, java.sql.Timestamp x)

【機能】

指定された java.sql.Timestamp オブジェクトを？パラメタ値に設定します。

【形式】

```
public synchronized void setTimestamp(String parameterName, java.sql.Timestamp x) throws
SQLException
```

【引数】

String parameterName：

パラメタ名。パラメタ名の大文字と小文字は区別しません。parameterName の文字列すべてを？パラメタ名として扱うため，文字列に引用符「"」が含まれている場合，引用符「"」も？パラメタ名に含まれます。

java.sql.Timestamp x：

？パラメタに設定する値を持つ java.sql.Timestamp オブジェクト

【戻り値】

なし。

【発生する例外】

次の場合，SQLException を投入します。

- CallableStatement オブジェクトに対して close()が既に発行されている場合
- この CallableStatement オブジェクトを生成した Connection オブジェクトに対して close()が既に発行されている場合
- 存在しないパラメタ名を指定した場合（parameterName の指定値が null 又は 0 長文字列の場合も含む）
- ？パラメタの HiRDB データ型がこのメソッドでは設定できないデータ型の場合
- 指定された値が列のデータ型の範囲外，又は変換できない形式の場合
- 指定した？パラメタが OUT パラメタである場合

(bk)　setTimestamp(String parameterName, java.sql.Timestamp x,Calendar cal)

【機能】

ローカルタイムで指定された java.sql.Timestamp オブジェクトを指定されたカレンダーのタイムゾーンの時間に変換し，？パラメタ値に設定します。

【形式】

```
public synchronized void setTimestamp(String parameterName, java.sql.Timestamp x,Calendar
 cal) throws SQLException
```

【引数】

String parameterName：

パラメタ名。パラメタ名の大文字と小文字は区別しません。parameterName の文字列すべてを？パラメタ名として扱うため，文字列に引用符「"」が含まれている場合，引用符「"」も？パラメタ名に含まれます。

java.sql.Timestamp x：

？パラメタに設定する値を持つ java.sql.Timestamp オブジェクト

Calendar cal：

データベースに格納する値のタイムゾーンが設定されたカレンダー。null が指定された場合，JavaVM のデフォルトのタイムゾーンのカレンダーが指定されたとします。

【戻り値】

なし。

【発生する例外】

次の場合，SQLException を投入します。

- CallableStatement オブジェクトに対して close()が既に発行されている場合
- この CallableStatement オブジェクトを生成した Connection オブジェクトに対して close()が既に発行されている場合
- 存在しないパラメタ名を指定した場合（parameterName の指定値が null 又は 0 長文字列の場合も含む）
- ？パラメタの HiRDB データ型がこのメソッドでは設定できないデータ型の場合
- 指定された値が列のデータ型の範囲外，又は変換できない形式の場合
- 指定した？パラメタが OUT パラメタである場合

(bl)　wasNull()

【機能】

最後に読み込まれた OUT パラメタ又は INOUT パラメタの値が，NULL かどうかを取得します。

【形式】

```
public synchronized boolean wasNull() throws SQLException
```

【引数】

なし。

【戻り値】

最後に読み込まれたパラメタが NULL の場合は true，そうでない場合は false を返します。

各条件での戻り値を次の表に示します。

表 17–35　各条件での戻り値

条件	戻り値
最後に読み込まれたパラメタが NULL の場合	true
最後に読み込まれたパラメタが NULL 以外の場合	false
execute 系メソッド実行前（OUT パラメタ用 ResultSet が null）の場合	false
一度も値の取得（getXXX）を行っていない場合	false
clearParameters 実行後の場合	clearParameters 実行前と同じ値
execute 系メソッドの再実行後（OUT パラメタ用 ResultSet の close をまたぐ）の場合	false

【発生する例外】

次の場合，SQLException を投入します。

- CallableStatement オブジェクトに対して close()が既に発行されている場合
- この CallableStatement オブジェクトを生成した Connection オブジェクトに対して，close()が既に発行されている場合

(3)　パッケージ名称及びクラス名称

このインタフェースを実装するパッケージ名称とクラス名称を次に示します。

パッケージ名称：**JP.co.Hitachi.soft.HiRDB.JDBC**

クラス名称：**PrdbCallableStatement**

(4)　注意事項

(a)　ストアドプロシジャの OUT パラメタの属性

- JDBC 規格では，executeXXX メソッドを実行する前に registerOutParameter メソッドを実行して，OUT パラメタ及び INOUT パラメタの属性を登録する必要があります。しかし，Type4 JDBC ドライバでは，CALL 文の前処理で取得した OUT パラメタ及び INOUT パラメタの属性を使用するため，registerOutParameter メソッドの実行は任意です。

- registerOutParameter メソッドで設定した情報は，executeXXX メソッド実行時に有効となります。そのため，executeXXX メソッド実行後に registerOutParameter メソッドで設定しても実行結果には影響しません。また，executeXXX メソッド実行後に clearParameters で設定情報をクリアしても実行結果には影響しません。

(b)　DECIMAL 型使用時の注意事項

DECIMAL 型の OUT パラメタ及び INOUT パラメタを設定する場合，小数点以下のけた数を受け入れない形式の registerOutParameter(int parameterIndex, int sqlType)メソッドを使用したときは，けた数を 0 とみなします。

(c)　DECIMAL 型の？パラメタに対する値指定

HiRDB の DECIMAL 型の？パラメタに対して setXXX メソッドで値を指定する場合，CALL 文の前処理で取得した精度及び位取りと，値の精度及び位取りが一致していないときの動作を次に示します。

CALL 文の前処理で取得した精度よりも大きい場合：SQLException を投入します。

CALL 文の前処理で取得した精度よりも小さい場合：拡張します。

CALL 文の前処理で取得した位取りよりも大きい場合：CALL 文の前処理で取得した位取りで切り捨てます。

CALL 文の前処理で取得した位取りよりも小さい場合：0 で補完し拡張します。

(d)　1 コネクション当たりのステートメントオブジェクト数の上限

1 つの Connection オブジェクトに対し，使用中のステートメントオブジェクト（Statement，PreparedStatement，及び CallableStatement)の数の合計が 4096 以上となった場合，SQLException を投入します。詳細は「Statement インタフェース」の「注意事項」を参照してください。

17.4.6　ResultSet インタフェース

(1)　概要

ResultSet インタフェースでは，主に次の機能が提供されます。

- 行単位の結果セット内の移動
- 結果データの返却
- 検索結果データがナル値かどうかの通知

(2)　メソッド

ResultSet インタフェースのメソッド一覧を次の表に示します。なお，表に記載されていないメソッドはサポートしていません。サポートしていないメソッドを指定すると，SQLException を投入します。

表 17–36　ResultSet インタフェースのメソッド一覧

メソッド	提供ドライバ			機能
	JDBC2.0	JDBC4.0	JDBC4.3	
absolute(int row)	○	○	○	カーソルを，この ResultSet オブジェクト内の指定された行に移動します。
afterLast()	○	○	○	カーソルを，この ResultSet オブジェクトの最終行の後ろに移動します。
beforeFirst()	○	○	○	カーソルを，この ResultSet オブジェクトの先頭行の前に移動します。
clearWarnings()	○	○	○	この ResultSet オブジェクトに関して報告された，すべての警告をクリアします。
close()	○	○	○	この ResultSet オブジェクトでオープンしていたデータベースのカーソルをクローズし，JDBC リソースを解放します。
findColumn(String columnName)	○	○	○	指定された列名を列番号にマッピングします。
first()	○	○	○	カーソルを，この ResultSet オブジェクト内の先頭行に移動します。
getArray(int i)	○	○	○	この ResultSet オブジェクトの現在行にある，指定された列番号の繰返し列の要素を，Array オブジェクトとして取得します。
getArray(String colName)	○	○	○	ResultSet オブジェクトの現在行にある，指定された列名の繰返し列の要素を，Array オブジェクトとして取得します。
getAsciiStream(int columnIndex)	○	○	○	この ResultSet オブジェクトの現在行にある指定された列の値を，PrdbDataStream オブジェクトとして取得します。
getAsciiStream(String columnName)	○	○	○	この ResultSet オブジェクトの現在行にある指定された列の値を，PrdbDataStream オブジェクトとして取得します。

メソッド	提供ドライバ			機能
	JDBC2.0	JDBC4.0	JDBC4.3	
getBigDecimal(int columnIndex)	○	○	○	この ResultSet オブジェクトの現在行にある指定された列の値を，java.math.BigDecimal オブジェクトとして取得します。
getBigDecimal(int columnIndex, int scale)	○	○	○	この ResultSet オブジェクトの現在行にある指定された列の値を，scale で指定された小数点以下のけた数を持つ java.math.BigDecimal オブジェクトとして取得します。
getBigDecimal(String columnName)	○	○	○	この ResultSet オブジェクトの現在行にある指定された列の値を，java.math.BigDecimal オブジェクトとして取得します。
getBigDecimal(String columnName, int scale)	○	○	○	この ResultSet オブジェクトの現在行にある指定された列の値を，scale で指定された小数点以下のけた数を持つ java.math.BigDecimal オブジェクトとして取得します。
getBinaryStream(int columnIndex)	○	○	○	この ResultSet オブジェクトの現在行にある，列番号で指定された列の値を，バイナリストリームとして取得します。
getBinaryStream(String columnName)	○	○	○	この ResultSet オブジェクトの現在行にある，指定された列の値を，バイナリストリームとして取得します。
getBlob(int i)	○	○	○	この ResultSet オブジェクトの現在行にある，列番号で指定された列の値を，java.sql.Blob オブジェクトとして取得します。
getBlob(String colName)	○	○	○	この ResultSet オブジェクトの現在行にある，指定された列の値を，java.sql.Blob オブジェクトとして取得します。
getBoolean(int columnIndex)	○	○	○	この ResultSet オブジェクトの現在行にある指定された列の値を，Java プログラミング言語の boolean として取得します。

メソッド	提供ドライバ			機能
	JDBC2.0	JDBC4.0	JDBC4.3	
getBoolean(String columnName)	○	○	○	この ResultSet オブジェクトの現在行にある指定された列の値を，Java プログラミング言語の boolean として取得します。
getByte(int columnIndex)	○	○	○	この ResultSet オブジェクトの現在行にある指定された列の値を，Java プログラミング言語の byte として取得します。
getByte(String columnName)	○	○	○	この ResultSet オブジェクトの現在行にある指定された列の値を，Java プログラミング言語の byte として取得します。
getBytes(int columnIndex)	○	○	○	ResultSet オブジェクトの現在行にある指定された列の値を，Java プログラミング言語の byte 配列として取得します。
getBytes(String columnName)	○	○	○	この ResultSet オブジェクトの現在行にある指定された列の値を，Java プログラミング言語の byte 配列として取得します。
getCharacterStream(int columnIndex)	○	○	○	この ResultSet オブジェクトの現在行にある指定された列の値を，java.io.Reader オブジェクトとして取得します。
getCharacterStream(String columnName)	○	○	○	この ResultSet オブジェクトの現在行にある指定された列の値を，java.io.Reader オブジェクトとして取得します。
getConcurrency()	○	○	○	この ResultSet オブジェクトの並行処理モードを返します。
getCursorName()	○	○	○	この ResultSet オブジェクトが使用する SQL カーソルの名前を取得します。
getDate(int columnIndex)	○	○	○	この ResultSet オブジェクトの現在行にある指定された列の値を，java.sql.Date オブジェクトとして取得します。
getDate(int columnIndex, Calendar cal)	○	○	○	この ResultSet オブジェクトの現在行にある指定された列の値を，java.sql.Date オブジェクトとして取得します。

メソッド	提供ドライバ			機能
	JDBC2.0	JDBC4.0	JDBC4.3	
getDate(String columnName)	○	○	○	この ResultSet オブジェクトの現在行にある指定された列の値を，java.sql.Date オブジェクトとして取得します。
getDate(String columnName, Calendar cal)	○	○	○	この ResultSet オブジェクトの現在行にある指定された列の値を，java.sql.Date オブジェクトとして取得します。
getDouble(int columnIndex)	○	○	○	この ResultSet オブジェクトの現在行にある指定された列の値を，Java プログラミング言語の double として取得します。
getDouble(String columnName)	○	○	○	この ResultSet オブジェクトの現在行にある指定された列の値を，Java プログラミング言語の double として取得します。
getFetchDirection()	○	○	○	この ResultSet オブジェクトのフェッチ方向を取得します。
getFetchSize()	○	○	○	この ResultSet オブジェクトのフェッチサイズを取得します。
getFloat(int columnIndex)	○	○	○	この ResultSet オブジェクトの現在行にある指定された列の値を，Java プログラミング言語の float として取得します。
getFloat(String columnName)	○	○	○	この ResultSet オブジェクトの現在行にある指定された列の値を，Java プログラミング言語の float として取得します。
getInt(int columnIndex)	○	○	○	この ResultSet オブジェクトの現在行にある指定された列の値を，Java プログラミング言語の int として取得します。
getInt(String columnName)	○	○	○	この ResultSet オブジェクトの現在行にある指定された列の値を，Java プログラミング言語の int として取得します。
getLong(int columnIndex)	○	○	○	この ResultSet オブジェクトの現在行にある指定された列の値を，Java プログラミング言語の long として取得します。

メソッド	提供ドライバ			機能
	JDBC2.0	JDBC4.0	JDBC4.3	
getLong(String columnName)	○	○	○	この ResultSet オブジェクトの現在行にある指定された列の値を，Java プログラミング言語の long として取得します。
getMetaData()	○	○	○	この ResultSet オブジェクトのメタ情報を表す ResultSetMetaData を返します。
getObject(int columnIndex)	○	○	○	この ResultSet オブジェクトの現在行にある指定された列の値を，Java プログラミング言語の Object として取得します。
getObject(String columnName)	○	○	○	この ResultSet オブジェクトの現在行にある指定された列の値を，Java プログラミング言語の Object として取得します。
getRow()	○	○	○	現在の行の番号を取得します。
getShort(int columnIndex)	○	○	○	この ResultSet オブジェクトの現在行にある指定された列の値を，Java プログラミング言語の short として取得します。
getShort(String columnName)	○	○	○	この ResultSet オブジェクトの現在行にある指定された列の値を，Java プログラミング言語の short として取得します。
getStatement()	○	○	○	この ResultSet オブジェクトを生成した Statement オブジェクトを取得します。
getString(int columnIndex)	○	○	○	この ResultSet オブジェクトの現在行にある指定された列の値を，Java プログラミング言語の String として取得します。
getString(String columnName)	○	○	○	この ResultSet オブジェクトの現在行にある指定された列の値を，Java プログラミング言語の String として取得します。
getTime(int columnIndex)	○	○	○	この ResultSet オブジェクトの現在行にある指定された列の値を，java.sql.Time として取得します。

メソッド	提供ドライバ			機能
	JDBC2.0	JDBC4.0	JDBC4.3	
getTime(int columnIndex, Calendar cal)	○	○	○	この ResultSet オブジェクトの現在行にある指定された列の値を，java.sql.Time オブジェクトとして取得します。
getTime(String columnName)	○	○	○	この ResultSet オブジェクトの現在行にある指定された列の値を，java.sql.Time オブジェクトとして取得します。
getTime(String columnName, Calendar cal)	○	○	○	この ResultSet オブジェクトの現在行にある指定された列の値を，java.sql.Time オブジェクトとして取得します。
getTimestamp(int columnIndex)	○	○	○	この ResultSet オブジェクトの現在行にある指定された列の値を，java.sql.Timestamp オブジェクトとして取得します。
getTimestamp(int columnIndex, Calendar cal)	○	○	○	この ResultSet オブジェクトの現在行にある指定された列の値を，java.sql.Timestamp オブジェクトとして取得します。
getTimestamp(String columnName)	○	○	○	この ResultSet オブジェクトの現在行にある指定された列の値を，java.sql.Timestamp オブジェクトとして取得します。
getTimestamp(String columnName, Calendar cal)	○	○	○	この ResultSet オブジェクトの現在行にある指定された列の値を，java.sql.Timestamp オブジェクトとして取得します。
getType()	○	○	○	この ResultSet オブジェクトの型を返します。
getWarnings()	○	○	○	この ResultSet オブジェクトに関する呼び出しによって報告される，最初の警告を取得します。
isAfterLast()	○	○	○	カーソルが，この ResultSet オブジェクト内の最終行の後ろにあるかどうかを取得します。
isBeforeFirst()	○	○	○	カーソルが，この ResultSet オブジェクト内の先頭行の前にあるかどうかを取得します。

メソッド	提供ドライバ			機能
	JDBC2.0	JDBC4.0	JDBC4.3	
isFirst()	○	○	○	カーソルが，この ResultSet オブジェクト内の先頭行にあるかどうかを取得します。
isLast()	○	○	○	カーソルが，この ResultSet オブジェクトの最終行にあるかどうかを取得します。
last()	○	○	○	カーソルを，この ResultSet オブジェクトの最終行に移動します。
next()	○	○	○	カーソルを，現在の位置から次の行に移動します。
previous()	○	○	○	カーソルを，この ResultSet オブジェクト内の一つ前の行に移動します。
relative(int rows)	○	○	○	カーソルを，正又は負の相対行数だけ移動します。
setFetchDirection(int direction)	○	○	○	この Statement オブジェクトから生成される結果集合の，デフォルトのフェッチ方向を指定します。
setFetchSize(int rows)	○	○	○	この ResultSet オブジェクトのフェッチサイズを設定します。
wasNull()	○	○	○	最後に取得した列の値が，NULL 値であるかどうかを通知します。
getHoldability()	×	○	○	ResultSet オブジェクトの保持機能についての状態を示す値を取得します。
isClosed()	×	○	○	ResultSet オブジェクトがクローズされているかどうかの情報を取得します。

（凡例）

○：提供されます。

×：提供されません。

(a)　absolute(int row)

【機能】

カーソルを，この ResultSet オブジェクト内の指定された行に移動します。

【形式】

```
public synchronized boolean absolute(int row) throws SQLException
```

【引数】

int row：

カーソルの移動先の行番号。正の番号は行番号が結果セットの先頭からカウントされることを示し，負の番号は終端からカウントされることを示します。

【戻り値】

このメソッド呼び出し後のカーソル位置が，先頭行の前又は最終行の後ろの場合は false，そうでない場合は true を返します。

【機能詳細】

カーソルを，この ResultSet オブジェクト内の指定された行に移動します。

このメソッド呼び出し後のカーソル移動位置と戻り値を次の表に示します。

表 17–37　absolute の移動先と戻り値

結果集合の行数※	row の指定値	移動先	戻り値
0	0 以外	先頭行の前のまま	false
n	n ＜ row	最終行の後ろ	false
	1≦row≦n	row	true
	-n≦row≦-1	(n+1)＋ row	true
	row ＜-n	先頭行の前	false

注※

setMaxRows の値より実際の行数の方が多い場合は，setMaxRows の値です。

【発生する例外】

次の場合，SQLException を投入します。

- この ResultSet オブジェクトが close されている場合

  この ResultSet オブジェクトを生成した Statement を close したことによって，このドライバが ResultSet を close した場合を含みます。
- この ResultSet オブジェクトを生成した Statement を作成した Connection が close されている場合
- この ResultSet オブジェクトの型が TYPE_FORWARD_ONLY の場合
- row=0 が指定された場合
- トランザクションの決着によって ResultSet が無効になった場合
- データベースアクセスでエラーが発生した場合

(b)　afterLast()

【機能】

カーソルを，この ResultSet オブジェクトの最終行の後ろに移動します。

【形式】

```
public synchronized void afterLast() throws SQLException
```

【引数】

なし。

【戻り値】

なし。

【機能詳細】

カーソルを，この ResultSet オブジェクトの最終行の後ろに移動します。

このメソッド呼び出し時のカーソル位置移動先を次の表に示します。

表 17–38　afterLast の移動先

結果集合の行数※	現在の行	afterLast()呼び出し後の行
0	先頭行の前	先頭行の前のまま
n	先頭行の前	最終行の後ろ
	1≦現在の行≦n	最終行の後ろ
	最終行の後ろ	最終行の後ろのまま

注※

setMaxRows の値より実際の行数の方が多い場合は，setMaxRows の値です。

【発生する例外】

次の場合，SQLException を投入します。

- この ResultSet オブジェクトが close されている場合

  この ResultSet オブジェクトを生成した Statement を close したことによって，このドライバが ResultSet を close した場合を含みます。
- この ResultSet オブジェクトを生成した Statement を作成した Connection が close されている場合
- この ResultSet オブジェクトの型が TYPE_FORWARD_ONLY の場合
- トランザクションの決着によって ResultSet が無効になった場合
- データベースアクセスでエラーが発生した場合

(c)　beforeFirst()

【機能】

カーソルを，この ResultSet オブジェクトの先頭行の前に移動します。

【形式】

```
public synchronized void beforeFirst() throws SQLException
```

【引数】

なし。

【戻り値】

なし。

【機能詳細】

カーソルを，この ResultSet オブジェクトの先頭行の前に移動します。

このメソッド呼び出し時のカーソル位置移動先を次の表に示します。

表 17–39　beforeFirst の移動先

結果集合の行数※	現在の行	beforeFirst()呼び出し後の行
0	先頭行の前	先頭行の前のまま
n	先頭行の前	先頭行の前のまま
	1≦現在の行≦n	先頭行の前
	最終行の後ろ	先頭行の前

注※

setMaxRows の値より実際の行数の方が多い場合は，setMaxRows の値です。

【発生する例外】

次の場合，SQLException を投入します。

- この ResultSet オブジェクトが close されている場合

  この ResultSet オブジェクトを生成した Statement を close したことによって，このドライバが ResultSet を close した場合を含みます。
- この ResultSet オブジェクトを生成した Statement を作成した Connection が close されている場合
- この ResultSet オブジェクトの型が TYPE_FORWARD_ONLY の場合
- トランザクションの決着によって ResultSet が無効になった場合
- データベースアクセスでエラーが発生した場合

(d)　clearWarnings()

【機能】

この ResultSet オブジェクトに関して報告された，すべての警告をクリアします。

【形式】

```
public synchronized void clearWarnings() throws SQLException
```

【引数】

なし。

【戻り値】

なし。

【発生する例外】

トランザクションの決着によって ResultSet が無効になった場合，SQLException を投入します。

(e) close()

【機能】

この ResultSet オブジェクトでオープンしていたデータベースのカーソルをクローズし，JDBC リソースを解放します。

【形式】

```
public synchronized void close() throws SQLException
```

【引数】

なし。

【戻り値】

なし。

【発生する例外】

データベースアクセスでエラーが発生した場合，SQLException を投入します。

(f) findColumn(String columnName)

【機能】

指定された列名を列番号にマッピングします。

【形式】

```
public synchronized int findColumn(String columnName) throws SQLException
```

【引数】

String columnName：

列名。大文字と小文字は区別しません。columnName の文字列すべてを列名として扱うため，文字列に引用符「"」が含まれている場合，引用符「"」も列名に含まれます。columnName の指定値が null，又は 0 長文字列の場合，必ず列が存在しないエラーとなります。

columnName の値と一致する列が複数ある場合，列番号が小さい方が優先されます。また，列名が長いため HiRDB サーバの仕様によって後ろが切り捨てられる場合，切り捨てられた列名が指定されていれば，一致しているとみなします。

【戻り値】

指定された列名に対応する列番号

【機能詳細】

引数 columnName を列名とし，対応する列番号を取得して返します。

【発生する例外】

次の場合，SQLException を投入します。

- この ResultSet オブジェクトが close されている場合

  この ResultSet オブジェクトを生成した Statement を close したことによって，このドライバが ResultSet を close した場合を含みます。
- この ResultSet オブジェクトを生成した Statement を作成した Connection が close されている場合
- トランザクションの決着によって ResultSet が無効になった場合
- 指定された列がない場合
- データベースアクセスでエラーが発生した場合

(g) first()

【機能】

カーソルを，この ResultSet オブジェクト内の先頭行に移動します。

【形式】

```
public synchronized boolean first() throws SQLException
```

【引数】

なし。

【戻り値】

結果集合の行数が 0 の場合は false，そうでない場合は true を返します。

【発生する例外】

次の場合，SQLException を投入します。

- この ResultSet オブジェクトが close されている場合

  この ResultSet オブジェクトを生成した Statement を close したことによって，このドライバが ResultSet を close した場合を含みます。
- この ResultSet オブジェクトを生成した Statement を作成した Connection が close されている場合
- この ResultSet オブジェクトの型が TYPE_FORWARD_ONLY の場合
- トランザクションの決着によって ResultSet が無効になった場合
- データベースアクセスでエラーが発生した場合

(h)　getArray(int i)

【機能】

この ResultSet オブジェクトの現在行にある，指定された列番号の繰返し列の要素を，Array オブジェクトとして取得します。

【形式】

```
public Array getArray(int i) throws SQLException
```

【引数】

int i：

列番号

【戻り値】

指定された列に対応する Array オブジェクトを返します。繰返し列の列全体が null 値の場合，null を返します。

【発生する例外】

次の場合，SQLException を投入します。

- この ResultSet オブジェクトが close されている場合

  この ResultSet オブジェクトを生成した Statement オブジェクトを close したことによって，このドライバが ResultSet オブジェクトを close した場合を含みます。
- この ResultSet オブジェクトを生成した Statement オブジェクトを作成した Connection オブジェクトが close されている場合
- トランザクションの決着によって ResultSet オブジェクトが無効になった場合
- 引数 i に存在しない列番号が指定されている場合
- 引数 i に指定された列番号が示す列が繰返し列ではない場合
- JDBC ドライバ内でエラーが発生した場合

(i)　getArray(String colName)

【機能】

ResultSet オブジェクトの現在行にある，指定された列名の繰返し列の要素を，Array オブジェクトとして取得します。

【形式】

```
public Array getArray(String colName) throws SQLException
```

【引数】

String colName：

列名

【戻り値】

指定された列名に対応する Array オブジェクト（繰返し列の列全体が null 値の場合，null を返します）

【発生する例外】

次の場合，SQLException を投入します。

- この ResultSet オブジェクトが close されている場合

  この ResultSet オブジェクトを生成した Statement オブジェクトを close したことによって，このドライバが ResultSet オブジェクトを close した場合を含みます。
- この ResultSet オブジェクトを生成した Statement オブジェクトを作成した Connection オブジェクトが close されている場合
- トランザクションの決着によって ResultSet オブジェクトが無効になった場合
- 引数 colName に指定した列が存在しない，又は null である場合
- JDBC ドライバ内でエラー発生した場合

(j)　getAsciiStream(int columnIndex)

【機能】

この ResultSet オブジェクトの現在行にある指定された列の値を，PrdbDataStream オブジェクトとして取得します。

【形式】

```
public synchronized InputStream getAsciiStream (int columnIndex) throws SQLException
```

【引数】

int columnIndex：

列番号

【戻り値】

列値を設定した PrdbDataStream オブジェクト（値が NULL 値の場合は null を返します）

【機能詳細】

この ResultSet オブジェクトの現在行にある指定された列の値を，PrdbDataStream オブジェクトとして取得します。ASCII 文字への変換は行いません。

HiRDB のデータ型による検索結果と戻り値の関係を次に示します。

HiRDB のデータ型	検索結果	戻り値
[M ｜ N][VAR]CHAR BINARY BLOB	NULL	null
	上記以外	検索結果を格納した InputStream オブジェクト
その他	−	SQLException を投入

（凡例）　−：該当しません。

【発生する例外】

次の場合，SQLException を投入します。

- この ResultSet オブジェクトが close されている場合

  この ResultSet オブジェクトを生成した Statement を close したことによって，このドライバが ResultSet を close した場合を含みます。
- この ResultSet オブジェクトを生成した Statement を作成した Connection が close されている場合
- トランザクションの決着によって ResultSet が無効になった場合
- 存在しない列番号が指定されている場合
- このメソッドでは取得できないデータ型の場合
- JDBC ドライバ内でエラーが発生した場合

(k)　getAsciiStream(String columnName)

【機能】

この ResultSet オブジェクトの現在行にある指定された列の値を，PrdbDataStream オブジェクトとして取得します。ASCII 文字への変換は行いません。

HiRDB のデータ型による検索結果と戻り値の関係については，「getAsciiStream(int columnIndex)」の【機能詳細】を参照してください。

【形式】

```
public synchronized InputStream getAsciiStream (String columnName) throws SQLException
```

【引数】

String columnName：

列名

【戻り値】

列値を設定した PrdbDataStream オブジェクト（値が NULL 値の場合は null を返します）

【発生する例外】

次の場合，SQLException を投入します。

- この ResultSet オブジェクトが close されている場合

  この ResultSet オブジェクトを生成した Statement を close したことによって，このドライバが ResultSet を close した場合を含みます。
- この ResultSet オブジェクトを生成した Statement を作成した Connection が close されている場合
- トランザクションの決着によって ResultSet が無効になった場合
- 存在しない列名が指定されている場合
- このメソッドでは取得できないデータ型の場合
- JDBC ドライバ内でエラーが発生した場合

(l)　getBigDecimal(int columnIndex)

【機能】

この ResultSet オブジェクトの現在行にある指定された列の値を，java.math.BigDecimal オブジェクトとして取得します。

【形式】

```
public synchronized BigDecimal getBigDecimal (int columnIndex) throws SQLException
```

【引数】

int columnIndex：

列番号

【戻り値】

指定された列番号に対応する列値（値が NULL 値の場合は null を返します）

【機能詳細】

この ResultSet オブジェクトの現在行にある指定された列の値を，java.math.BigDecimal オブジェクトとして取得します。

HiRDB のデータ型による検索結果と戻り値の関係を次に示します。

HiRDB のデータ型	検索結果	戻り値
[M | N][VAR]CHAR	NULL	null
	[半角空白]整数の文字列表現，10進数の文字列表現，又は浮動小数点数の文字列表現[半角空白]	検索結果を持つ BigDecimal オブジェクト。 文字列の前後の半角空白を取り除いた値を BigDecimal オブジェクトにします。
	上記以外	SQLException を投入
SMALLINT	NULL	null
	上記以外	検索結果を持つ BigDecimal オブジェクト。
INTEGER	NULL	null
	上記以外	検索結果を持つ BigDecimal オブジェクト。
REAL	NULL	null
	Infinity	SQLException を投入
	-Infinity	SQLException を投入
	NaN	SQLException を投入
	上記以外	検索結果を持つ BigDecimal オブジェクト。
FLOAT	NULL	null
	Infinity	SQLException を投入
	-Infinity	SQLException を投入

HiRDB のデータ型	検索結果	戻り値
	NaN	SQLException を投入
	上記以外	検索結果を持つ BigDecimal オブジェクト。
DECIMAL	NULL	null
	上記以外	検索結果を持つ BigDecimal オブジェクト。
BOOLEAN※	NULL	null
	true	BigDecimal(1)で BigDecimal オブジェクトにしたもの。
	false	BigDecimal(0)で BigDecimal オブジェクトにしたもの。
その他	−	SQLException を投入

（凡例） −：該当しません。

注※　DatabaseMetadata から生成した Resultset の場合，BOOLEAN 型データが存在します。

【発生する例外】

次の場合，SQLException を投入します。

- この ResultSet オブジェクトが close されている場合

  この ResultSet オブジェクトを生成した Statement を close したことによって，このドライバが ResultSet を close した場合を含みます。
- この ResultSet オブジェクトを生成した Statement を作成した Connection が close されている場合
- トランザクションの決着によって ResultSet が無効になった場合
- 存在しない列番号が指定されている場合
- このメソッドでは取得できないデータ型の場合
- 列の値が BigDecimal として取得できない場合
- JDBC ドライバ内でエラーが発生した場合

(m)　getBigDecimal(int columnIndex, int scale)

【機能】

この ResultSet オブジェクトの現在行にある指定された列の値を，scale で指定された小数点以下のけた数を持つ java.math.BigDecimal オブジェクトとして取得します。

【形式】

```
public synchronized BigDecimal getBigDecimal(int columnIndex,int scale) throws SQLExcepti
on
```

【引数】

int columnIndex：

列番号

int scale：

位取り

【戻り値】

指定された列番号に対応する列値の小数点以下のけた数を引数 scale にした値（値が NULL 値の場合は null を返します）

【機能詳細】

この ResultSet オブジェクトの現在行にある指定された列の値を，scale で指定された小数点以下のけた数を持つ java.math.BigDecimal オブジェクトとして取得します。

列値の小数点以下のけた数より引数 scale が小さい場合，scale で切り捨てます。scale の方が大きい場合 0 で補完します。

検索結果の小数点以下のけた数より引数 scale の値が小さい場合，検索結果の小数点以下のけた数を scale で切り捨てます。検索結果の小数点以下のけた数より引数 scale の値が大きい場合，検索結果の小数点以下のけた数を scale まで 0 で補完します。

HiRDB のデータ型による検索結果と戻り値の関係については，「getBigDecimal(int columnIndex)」の【機能詳細】を参照してください。

【発生する例外】

次の場合，SQLException を投入します。

- この ResultSet オブジェクトが close されている場合

  この ResultSet オブジェクトを生成した Statement を close したことによって，このドライバが ResultSet を close した場合を含みます。
- この ResultSet オブジェクトを生成した Statement を作成した Connection が close されている場合
- トランザクションの決着によって ResultSet が無効になった場合
- 存在しない列番号が指定されている場合
- このメソッドでは取得できないデータ型の場合
- 列の値が BigDecimal として取得できない場合
- scale に 0 未満を指定した場合
- JDBC ドライバ内でエラーが発生した場合

(n)　getBigDecimal(String columnName)

【機能】

この ResultSet オブジェクトの現在行にある指定された列の値を，java.math.BigDecimal オブジェクトとして取得します。

HiRDB のデータ型による検索結果と戻り値の関係については，「getBigDecimal(int columnIndex)」の【機能詳細】を参照してください。

【形式】

```
public synchronized BigDecimal getBigDecimal (String columnName) throws SQLException
```

【引数】

String columnName：

列名

【戻り値】

指定された列名に対応する列値（値が NULL 値の場合は null を返します）

【発生する例外】

次の場合，SQLException を投入します。

- この ResultSet オブジェクトが close されている場合

  この ResultSet オブジェクトを生成した Statement を close したことによって，このドライバが ResultSet を close した場合を含みます。
- この ResultSet オブジェクトを生成した Statement を作成した Connection が close されている場合
- トランザクションの決着によって ResultSet が無効になった場合
- 存在しない列名が指定されている場合
- このメソッドでは取得できないデータ型の場合
- 列の値が BigDecimal として取得できない場合
- JDBC ドライバ内でエラーが発生した場合

(o) getBigDecimal(String columnName, int scale)

【機能】

この ResultSet オブジェクトの現在行にある指定された列の値を，scale で指定された小数点以下のけた数を持つ java.math.BigDecimal オブジェクトとして取得します。

【形式】

```
public synchronized BigDecimal getBigDecimal(String columnName,int scale) throws SQLExcep
tion
```

【引数】

String columnName：

列名

int scale：

位取り

【戻り値】

指定された列名に対応する列値の小数点以下のけた数を引数 scale にした値（値が NULL 値の場合は null を返します）

【機能詳細】

この ResultSet オブジェクトの現在行にある指定された列の値を，scale で指定された小数点以下のけた数を持つ java.math.BigDecimal オブジェクトとして取得します。

検索結果の小数点以下のけた数より引数 scale の値が小さい場合，検索結果の小数点以下のけた数を scale で切り捨てます。検索結果の小数点以下のけた数より引数 scale の値が大きい場合，検索結果の小数点以下のけた数を scale まで 0 で補完します。

HiRDB のデータ型による検索結果と戻り値の関係については，「getBigDecimal(int columnIndex)」の【機能詳細】を参照してください。

【発生する例外】

次の場合，SQLException を投入します。

- この ResultSet オブジェクトが close されている場合

  この ResultSet オブジェクトを生成した Statement を close したことによって，このドライバが ResultSet を close した場合を含みます。
- この ResultSet オブジェクトを生成した Statement を作成した Connection が close されている場合
- トランザクションの決着によって ResultSet が無効になった場合
- 存在しない列名が指定されている場合
- このメソッドでは取得できないデータ型の場合
- 列の値が BigDecimal として取得できない場合
- scale に 0 未満を指定した場合
- JDBC ドライバ内でエラーが発生した場合

(p)　getBinaryStream(int columnIndex)

【機能】

この ResultSet オブジェクトの現在行にある，列番号で指定された列の値を，バイナリストリームとして取得します。

【形式】

```
public synchronized InputStream getBinaryStream(int columnIndex) throws SQLException
```

【引数】

int columnIndex：

列番号

【戻り値】

列値を未解釈のバイトストリームとして送る Java 入力ストリーム（値が NULL 値の場合は null を返します）

【機能詳細】

この ResultSet オブジェクトの現在行にある，列番号で指定された列の値を，バイナリストリームとして取得します。

HiRDB のデータ型による検索結果と戻り値の関係を次に示します。

HiRDB のデータ型	検索結果	戻り値
BINARYBLOB	NULL	null
	上記以外	検索結果を格納した InputStream オブジェクト
その他	−	SQLException を投入

（凡例） −：該当しません。

【発生する例外】

次の場合，SQLException を投入します。

- この ResultSet オブジェクトが close されている場合

  この ResultSet オブジェクトを生成した Statement を close したことによって，このドライバが ResultSet を close した場合を含みます。
- この ResultSet オブジェクトを生成した Statement を作成した Connection が close されている場合
- トランザクションの決着によって ResultSet が無効になった場合
- 存在しない列番号が指定されている場合
- このメソッドでは取得できないデータ型の場合
- JDBC ドライバ内でエラーが発生した場合

(q)　getBinaryStream(String columnName)

【機能】

この ResultSet オブジェクトの現在行にある，指定された列の値を，バイナリストリームとして取得します。

HiRDB のデータ型による検索結果と戻り値の関係については，「getBinaryStream(int columnIndex)」の【機能詳細】を参照してください。

【形式】

```
public synchronized InputStream getBinaryStream (String columnName) throws SQLException
```

【引数】

String columnName：

列名

【戻り値】

列値を未解釈のバイトストリームとして送る Java 入力ストリーム（値が NULL 値の場合は null を返します）

【発生する例外】

次の場合，SQLException を投入します。

- この ResultSet オブジェクトが close されている場合

  この ResultSet オブジェクトを生成した Statement を close したことによって，このドライバが ResultSet を close した場合を含みます。
- この ResultSet オブジェクトを生成した Statement を作成した Connection が close されている場合
- トランザクションの決着によって ResultSet が無効になった場合
- 存在しない列名が指定されている場合
- このメソッドでは取得できないデータ型の場合
- JDBC ドライバ内でエラーが発生した場合

(r)　getBlob(int i)

【機能】

この ResultSet オブジェクトの現在行にある，列番号で指定された列の値を，java.sql.Blob オブジェクトとして取得します。

【形式】

```
public synchronized Blob getBlob(int i) throws SQLException
```

【引数】

int i：

列番号

【戻り値】

指定された列の値を表す Blob オブジェクト（値が NULL 値の場合は null を返します）

【機能詳細】

この ResultSet オブジェクトの現在行にある，列番号で指定された列の値を，java.sql.Blob オブジェクトとして取得します。

HiRDB のデータ型による検索結果と戻り値の関係を次に示します。

HiRDB のデータ型	検索結果	戻り値
BINARYBLOB	NULL	null
	上記以外	検索結果を持つ java.sql.Blob オブジェクト
その他	−	SQLException を投入

（凡例）　−：該当しません。

【発生する例外】

次の場合，SQLException を投入します。

- この ResultSet オブジェクトが close されている場合

この ResultSet オブジェクトを生成した Statement を close したことによって，このドライバが ResultSet を close した場合を含みます。

- この ResultSet オブジェクトを生成した Statement を作成した Connection が close されている場合
- トランザクションの決着によって ResultSet が無効になった場合
- 存在しない列番号が指定されている場合
- このメソッドでは取得できないデータ型の場合
- JDBC ドライバ内でエラーが発生した場合

(s)　getBlob(String colName)

【機能】

この ResultSet オブジェクトの現在行にある，指定された列の値を，java.sql.Blob オブジェクトとして取得します。

HiRDB のデータ型による検索結果と戻り値の関係については，「getBlob(int i)」の【機能詳細】を参照してください。

【形式】

```
public synchronized Blob getBlob(String colName) throws SQLException
```

【引数】

String colName：

列名

【戻り値】

指定された列の値を表す Blob オブジェクト（値が NULL 値の場合は null を返します）

【発生する例外】

次の場合，SQLException を投入します。

- この ResultSet オブジェクトが close されている場合

  この ResultSet オブジェクトを生成した Statement を close したことによって，このドライバが ResultSet を close した場合を含みます。
- この ResultSet オブジェクトを生成した Statement を作成した Connection が close されている場合
- トランザクションの決着によって ResultSet が無効になった場合
- 存在しない列名が指定されている場合
- このメソッドでは取得できないデータ型の場合
- JDBC ドライバ内でエラーが発生した場合

(t)　getBoolean(int columnIndex)

【機能】

この ResultSet オブジェクトの現在行にある指定された列の値を，Java プログラミング言語の boolean として取得します。

【形式】

```
public synchronized boolean getBoolean(int columnIndex) throws SQLException
```

【引数】

int columnIndex：

列番号

【戻り値】

true 又は false（値が NULL 値の場合は false が返されます）

【機能詳細】

この ResultSet オブジェクトの現在行にある指定された列の値を，Java プログラミング言語の boolean として取得します。

HiRDB のデータ型による検索結果と戻り値の関係を次に示します。

HiRDB のデータ型	検索結果	戻り値
[M][VAR]CHAR NVARCHAR	NULL	false
	[半角空白]半角 true（大文字と小文字は区別しない）[半角空白]	true
	[半角空白]半角 1[半角空白]	true
	上記以外	false
NCHAR	NULL	false
	先頭が[半角空白]半角 true（大文字と小文字は区別しない）	true
	上記以外	false
SMALLINT	NULL	false
	0	false
	0 以外	true
INTEGER	NULL	false
	0	false
	0 以外	true
REAL	NULL	false
	Infinity	true
	-Inifnity	true

HiRDB のデータ型	検索結果	戻り値
	NaN	true
	0.0 又は-0.0	false
	上記以外	true
FLOAT	NULL	false
	Infinity	true
	-Inifnity	true
	NaN	true
	0.0 又は-0.0	false
	上記以外	true
DECIMAL	NULL	false
	0[.00…0]	false
	上記以外	true
BOOLEAN※	NULL	false
	NULL 以外	検索結果
その他	−	SQLException を投入

（凡例） −：該当しません。

注※ DatabaseMetadata から生成した Resultset の場合，BOOLEAN 型データが存在します。

【発生する例外】

次の場合，SQLException を投入します。

- この ResultSet オブジェクトが close されている場合

  この ResultSet オブジェクトを生成した Statement を close したことによって，このドライバが ResultSet を close した場合を含みます。
- この ResultSet オブジェクトを生成した Statement を作成した Connection が close されている場合
- トランザクションの決着によって ResultSet が無効になった場合
- 存在しない列番号が指定されている場合
- このメソッドでは取得できないデータ型の場合
- JDBC ドライバ内でエラーが発生した場合

(u) getBoolean(String columnName)

【機能】

この ResultSet オブジェクトの現在行にある指定された列の値を，Java プログラミング言語の boolean として取得します。

HiRDB のデータ型による検索結果と戻り値の関係については，「getBoolean(int columnIndex)」の【機能詳細】を参照してください。

【形式】

```
public synchronized boolean getBoolean(String columnName) throws SQLException
```

【引数】

String columnName：

列名

【戻り値】

true 又は false（値が NULL 値の場合は false が返されます）

【発生する例外】

次の場合，SQLException を投入します。

- この ResultSet オブジェクトが close されている場合

  この ResultSet オブジェクトを生成した Statement を close したことによって，このドライバが ResultSet を close した場合を含みます。
- この ResultSet オブジェクトを生成した Statement を作成した Connection が close されている場合
- トランザクションの決着によって ResultSet が無効になった場合
- 存在しない列名が指定されている場合
- このメソッドでは取得できないデータ型の場合
- JDBC ドライバ内でエラーが発生した場合

(v)　getByte(int columnIndex)

【機能】

この ResultSet オブジェクトの現在行にある指定された列の値を，Java プログラミング言語の byte として取得します。

【形式】

```
public synchronized byte getByte(int columnIndex) throws SQLException
```

【引数】

int columnIndex：

列番号

【戻り値】

列値（値が NULL 値の場合は 0 を返します）

【機能詳細】

この ResultSet オブジェクトの現在行にある指定された列の値を，Java プログラミング言語の byte として取得します。

HiRDB のデータ型による検索結果と戻り値の関係を次に示します。

HiRDB のデータ型	検索結果	戻り値
[M ｜ N][VAR]CHAR	NULL	0
	[半角空白]整数の文字列表現，10 進数の文字列表現，又は浮動小数点数の文字列表現[半角空白]であり，かつ Byte.MIN_VALUE 以上，Byte.MAX_VALUE 以下	検索結果を byte 値にしたもの
	[半角空白]整数の文字列表現，10 進数の文字列表現，又は浮動小数点数の文字列表現[半角空白]であり，かつ Byte.MAX_VALUE より大きいか Byte.MIN_VALUE より小さい	SQLException を投入
	[半角空白][+]Infinity[半角空白]	SQLException を投入
	[半角空白]-Infinity[半角空白]	SQLException を投入
	[半角空白][+｜-]NaN[半角空白]	SQLException を投入
	上記以外（double 値にできない）	SQLException を投入
SMALLINT	NULL	0
	Byte.MIN_VALUE 以上かつ Byte.MAX_VALUE 以下	検索結果を byte 値にしたもの
	上記以外	SQLException を投入
INTEGER	NULL	0
	Byte.MIN_VALUE 以上かつ Byte.MAX_VALUE 以下	検索結果を byte 値にしたもの
	上記以外	SQLException を投入
REAL	NULL	0
	Infinity	SQLException を投入
	-Infinity	SQLException を投入
	NaN	SQLException を投入
	Byte.MIN_VALUE 以上かつ Byte.MAX_VALUE 以下	検索結果の整数部分の値を byte 値にしたもの
	上記以外	SQLException を投入
FLOAT	NULL	0
	Infinity	SQLException を投入
	-Infinity	SQLException を投入
	NaN	SQLException を投入
	Byte.MIN_VALUE 以上かつ Byte.MAX_VALUE 以下	検索結果の整数部分の値を byte 値にしたもの
	上記以外	SQLException を投入

HiRDB のデータ型	検索結果	戻り値
DECIMAL	NULL	0
	Byte.MIN_VALUE 以上かつ Byte.MAX_VALUE 以下	検索結果の整数部分の値を byte 値にしたもの
	上記以外	SQLException を投入
BOOLEAN※	NULL	0
	true	1
	false	0
その他	−	SQLException を投入

（凡例） −：該当しません。

注※　DatabaseMetadata から生成した Resultset の場合，BOOLEAN 型データが存在します。

【発生する例外】

次の場合，SQLException を投入します。

- この ResultSet オブジェクトが close されている場合

  この ResultSet オブジェクトを生成した Statement を close したことによって，このドライバが ResultSet を close した場合を含みます。
- この ResultSet オブジェクトを生成した Statement を作成した Connection が close されている場合
- トランザクションの決着によって ResultSet が無効になった場合
- 存在しない列番号が指定されている場合
- このメソッドでは取得できないデータ型の場合
- 列の値が byte として取得できない場合
- JDBC ドライバ内でエラーが発生した場合

(w)　getByte(String columnName)

【機能】

この ResultSet オブジェクトの現在行にある指定された列の値を，Java プログラミング言語の byte として取得します。

HiRDB のデータ型による検索結果と戻り値の関係については，「getByte(int columnIndex)」の【機能詳細】を参照してください。

【形式】

```
public synchronized byte getByte(String columnName) throws SQLException
```

【引数】

String columnName：

列名

【戻り値】

列値（値が NULL 値の場合は 0 を返します）

【発生する例外】

次の場合，SQLException を投入します。

- この ResultSet オブジェクトが close されている場合

  この ResultSet オブジェクトを生成した Statement を close したことによって，このドライバが ResultSet を close した場合を含みます。
- この ResultSet オブジェクトを生成した Statement を作成した Connection が close されている場合
- トランザクションの決着によって ResultSet が無効になった場合
- 存在しない列名が指定されている場合
- このメソッドでは取得できないデータ型の場合
- 列の値が byte として取得できない場合
- JDBC ドライバ内でエラーが発生した場合

(x)　getBytes(int columnIndex)

【機能】

ResultSet オブジェクトの現在行にある指定された列の値を，Java プログラミング言語の byte 配列として取得します。

【形式】

```
public synchronized byte[] getBytes(int columnIndex) throws SQLException
```

【引数】

int columnIndex：

列番号

【戻り値】

列値（値が NULL 値の場合は null を返します）

【機能詳細】

ResultSet オブジェクトの現在行にある指定された列の値を，Java プログラミング言語の byte 配列として取得します。

HiRDB のデータ型による検索結果と戻り値の関係を次に示します。

HiRDB のデータ型	検索結果	戻り値
[M ｜ N][VAR]CHAR BINARY BLOB	NULL	null
	上記以外	検索結果を byte 配列にしたもの
その他	−	SQLException を投入

（凡例） －：該当しません。

【発生する例外】

次の場合，SQLException を投入します。

- この ResultSet オブジェクトが close されている場合

  この ResultSet オブジェクトを生成した Statement を close したことによって，このドライバが ResultSet を close した場合を含みます。
- この ResultSet オブジェクトを生成した Statement を作成した Connection が close されている場合
- トランザクションの決着によって ResultSet が無効になった場合
- 存在しない列番号が指定されている場合
- このメソッドでは取得できないデータ型の場合
- JDBC ドライバ内でエラーが発生した場合

(y) getBytes(String columnName)

【機能】

この ResultSet オブジェクトの現在行にある指定された列の値を，Java プログラミング言語の byte 配列として取得します。バイトはドライバによって返された行の値を表します。

HiRDB のデータ型による検索結果と戻り値の関係については，「getBytes(int columnIndex)」の【機能詳細】を参照してください。

【形式】

```
public synchronized byte[] getBytes (String columnName) throws SQLException
```

【引数】

String columnName：

列名

【戻り値】

列値（値が NULL 値の場合は null を返します）

【発生する例外】

次の場合，SQLException を投入します。

- この ResultSet オブジェクトが close されている場合

  この ResultSet オブジェクトを生成した Statement を close したことによって，このドライバが ResultSet を close した場合を含みます。
- この ResultSet オブジェクトを生成した Statement を作成した Connection が close されている場合
- トランザクションの決着によって ResultSet が無効になった場合
- 存在しない列名が指定されている場合
- このメソッドでは取得できないデータ型の場合

- JDBC ドライバ内でエラーが発生した場合

(z) getCharacterStream(int columnIndex)

【機能】

この ResultSet オブジェクトの現在行にある指定された列の値を，java.io.Reader オブジェクトとして取得します。

【形式】

```
public synchronized Reader getCharacterStream (int columnIndex) throws SQLException
```

【引数】

int columnIndex：

列番号

【戻り値】

列値を格納する java.io.Reader オブジェクト（値が NULL 値の場合は null を返します）

【機能詳細】

この ResultSet オブジェクトの現在行にある指定された列の値を，java.io.Reader オブジェクトとして取得します。

HiRDB のデータ型による検索結果と戻り値の関係を次に示します。

HiRDB のデータ型	検索結果	戻り値
[M | N][VAR]CHAR BINARY BLOB	NULL	null
	上記以外	検索結果を格納した Reader オブジェクト
その他	−	SQLException を投入

（凡例） −：該当しません。

【発生する例外】

次の場合，SQLException を投入します。

- この ResultSet オブジェクトが close されている場合

  この ResultSet オブジェクトを生成した Statement を close したことによって，このドライバが ResultSet を close した場合を含みます。
- この ResultSet オブジェクトを生成した Statement を作成した Connection が close されている場合
- トランザクションの決着によって ResultSet が無効になった場合
- 存在しない列番号が指定されている場合
- このメソッドでは取得できないデータ型の場合
- エンコードに失敗した場合
- JDBC ドライバ内でエラーが発生した場合

(aa)　getCharacterStream(String columnName)

【機能】

この ResultSet オブジェクトの現在行にある指定された列の値を，java.io.Reader オブジェクトとして取得します。

HiRDB のデータ型による検索結果と戻り値の関係については，「getCharacterStream(int columnIndex)」の【機能詳細】を参照してください。

【形式】

```
public synchronized Reader getCharacterStream (String columnName) throws SQLException
```

【引数】

String columnName：

列名

【戻り値】

列値を格納する java.io.Reader オブジェクト（値が NULL 値の場合は null を返します）

【発生する例外】

次の場合，SQLException を投入します。

- この ResultSet オブジェクトが close されている場合

  この ResultSet オブジェクトを生成した Statement を close したことによって，このドライバが ResultSet を close した場合を含みます。
- この ResultSet オブジェクトを生成した Statement を作成した Connection が close されている場合
- トランザクションの決着によって ResultSet が無効になった場合
- 存在しない列名が指定されている場合
- このメソッドでは取得できないデータ型の場合
- エンコードに失敗した場合
- JDBC ドライバ内でエラーが発生した場合

(ab)　getConcurrency()

【機能】

この ResultSet オブジェクトの並行処理モードを返します。更新カーソルは未サポートのため，必ず ResultSet.CONCUR_READ_ONLY を返します。

【形式】

```
public synchronized int getConcurrency() throws SQLException
```

【引数】

なし。

【戻り値】

ResultSet.CONCUR_READ_ONLY：

列の値を更新できません。

ResultSet.CONCUR_UPDATABLE：

列の値を更新できます。

【発生する例外】

次の場合，SQLException を投入します。

- この ResultSet オブジェクトが close されている場合

  この ResultSet オブジェクトを生成した Statement を close したことによって，このドライバが ResultSet を close した場合を含みます。
- この ResultSet オブジェクトを生成した Statement を作成した Connection が close されている場合
- トランザクションの決着によって ResultSet が無効になった場合

(ac)　getCursorName()

【機能】

この ResultSet オブジェクトが使用する SQL カーソルの名前を取得します。

HiRDB では常に null を返します。

【形式】

```
public synchronized String getCursorName() throws SQLException
```

【引数】

なし。

【戻り値】

null

【発生する例外】

次の場合，SQLException を投入します。

- この ResultSet オブジェクトが close されている場合

  この ResultSet オブジェクトを生成した Statement を close したことによって，このドライバが ResultSet を close した場合を含みます。
- この ResultSet オブジェクトを生成した Statement を作成した Connection が close されている場合
- トランザクションの決着によって ResultSet が無効になった場合

(ad)　getDate(int columnIndex)

【機能】

この ResultSet オブジェクトの現在行にある指定された列の値を，java.sql.Date オブジェクトとして取得します。

【形式】

```
public synchronized java.sql.Date getDate(int columnIndex) throws SQLException
```

【引数】

int columnIndex：

列番号

【戻り値】

列値（値が NULL 値の場合は null を返します）

【機能詳細】

この ResultSet オブジェクトの現在行にある指定された列の値を，java.sql.Date オブジェクトとして取得します。

HiRDB のデータ型による検索結果と戻り値の関係を次に示します。

HiRDB のデータ型	検索結果	戻り値
[M ｜ N][VAR]CHAR	NULL	null
	[半角空白]日付形式[半角空白]	検索結果の前後の半角空白を取り除き java.sql.Date オブジェクトにしたもの
	上記以外	SQLException を投入
DATE	NULL	null
	上記以外	検索結果を java.sql.Date オブジェクトにしたもの
TIMESTAMP	NULL	null
	上記以外	検索結果を java.sql.Date オブジェクトにしたもの
その他	－	SQLException を投入

（凡例）　－：該当しません。

【発生する例外】

次の場合，SQLException を投入します。

- この ResultSet オブジェクトが close されている場合

  この ResultSet オブジェクトを生成した Statement を close したことによって，このドライバが ResultSet を close した場合を含みます。
- この ResultSet オブジェクトを生成した Statement を作成した Connection が close されている場合
- トランザクションの決着によって ResultSet が無効になった場合

- 存在しない列番号が指定されている場合
- このメソッドでは取得できないデータ型の場合
- 列の値が java.sql.Date として取得できない場合
- JDBC ドライバ内でエラーが発生した場合

(ae)　getDate(int columnIndex, Calendar cal)

【機能】

この ResultSet オブジェクトの現在行にある指定された列の値を，java.sql.Date オブジェクトとして取得します。このメソッドは指定されたカレンダーを使って日付に適切なミリ秒値を作成します。

HiRDB のデータ型による検索結果と戻り値の関係については，「getDate(int columnIndex)」の【機能詳細】を参照してください。

【形式】

```
public synchronized java.sql.Date getDate(int columnIndex, Calendar cal) throws SQLExcept
ion
```

【引数】

int columnIndex：

列番号

Calendar cal：

データベースに格納されている値のタイムゾーンが設定されたカレンダー

【戻り値】

列値（値が NULL 値の場合は null を返します）

【発生する例外】

次の場合，SQLException を投入します。

- この ResultSet オブジェクトが close されている場合

  この ResultSet オブジェクトを生成した Statement を close したことによって，このドライバが ResultSet を close した場合を含みます。
- この ResultSet オブジェクトを生成した Statement を作成した Connection が close されている場合
- トランザクションの決着によって ResultSet が無効になった場合
- 存在しない列番号が指定されている場合
- このメソッドでは取得できないデータ型の場合
- 列の値が java.sql.Date として取得できない場合
- JDBC ドライバ内でエラーが発生した場合

(af)　getDate(String columnName)

【機能】

この ResultSet オブジェクトの現在行にある指定された列の値を，java.sql.Date オブジェクトとして取得します。

HiRDB のデータ型による検索結果と戻り値の関係については，「getDate(int columnIndex)」の【機能詳細】を参照してください。

【形式】

```
public synchronized java.sql.Date getDate (String columnName) throws SQLException
```

【引数】

String columnName：

列名

【戻り値】

列値（値が NULL 値の場合は null を返します）

【発生する例外】

次の場合，SQLException を投入します。

- この ResultSet オブジェクトが close されている場合

  この ResultSet オブジェクトを生成した Statement を close したことによって，このドライバが ResultSet を close した場合を含みます。
- この ResultSet オブジェクトを生成した Statement を作成した Connection が close されている場合
- トランザクションの決着によって ResultSet が無効になった場合
- 存在しない列名が指定されている場合
- このメソッドでは取得できないデータ型の場合
- 列の値が java.sql.Date として取得できない場合
- JDBC ドライバ内でエラーが発生した場合

(ag)　getDate(String columnName, Calendar cal)

【機能】

この ResultSet オブジェクトの現在行にある指定された列の値を，java.sql.Date オブジェクトとして取得します。このメソッドは指定されたカレンダーを使って日付に適切なミリ秒値を作成します。

HiRDB のデータ型による検索結果と戻り値の関係については，「getDate(int columnIndex)」の【機能詳細】を参照してください。

【形式】

```
public synchronized java.sql.Date getDate(String columnName, Calendar cal) throws SQLExce
ption
```

【引数】

String columnName：

列名

Calendar cal：

データベースに格納されている値のタイムゾーンが設定されたカレンダー

【戻り値】

列値（値が NULL 値の場合は null を返します）

【発生する例外】

次の場合，SQLException を投入します。

- この ResultSet オブジェクトが close されている場合

  この ResultSet オブジェクトを生成した Statement を close したことによって，このドライバが ResultSet を close した場合を含みます。
- この ResultSet オブジェクトを生成した Statement を作成した Connection が close されている場合
- トランザクションの決着によって ResultSet が無効になった場合
- 存在しない列名が指定されている場合
- このメソッドでは取得できないデータ型の場合
- 列の値が java.sql.Date として取得できない場合
- JDBC ドライバ内でエラーが発生した場合

(ah)　getDouble(int columnIndex)

【機能】

この ResultSet オブジェクトの現在行にある指定された列の値を，Java プログラミング言語の double として取得します。

【形式】

```
public synchronized double getDouble (int columnIndex) throws SQLException
```

【引数】

int columnIndex：

列番号

【戻り値】

列値（値が NULL 値の場合は null を返します）

【機能詳細】

この ResultSet オブジェクトの現在行にある指定された列の値を，Java プログラミング言語の double として取得します。

HiRDB のデータ型による検索結果と戻り値の関係を次に示します。

HiRDB のデータ型	検索結果	戻り値
[M ｜ N][VAR]CHAR	NULL	0.0
	[半角空白]整数の文字列表現，10 進数の文字列表現，又は浮動小数点数の文字列表現[半角空白]であり，かつ-Double.MAX_VALUE 以上，かつ Double.MIN_VALUE 以下，かつ Double.MIN_VALUE 以上かつ Double.MAX_VALUE 以下	検索結果を double 値にしたもの
	[半角空白]整数の文字列表現，10 進数の文字列表現，又は浮動小数点数の文字列表現[半角空白]かつ Double.MAX_VALUE より大きい	Infinity
	[半角空白]整数の文字列表現，10 進数の文字列表現，又は浮動小数点数の文字列表現[半角空白]かつ-Double.MAX_VALUE より小さい	-Infinity
	[半角空白]整数の文字列表現，10 進数の文字列表現，又は浮動小数点数の文字列表現[半角空白]かつ Double.MIN_VALUE より小さく 0 より大きい	0.0
	[半角空白]整数の文字列表現，10 進数の文字列表現，又は浮動小数点数の文字列表現[半角空白]かつ-Double.MIN_VALUE より大きく 0 より小さい	-0.0
	[半角空白]-Infinity[半角空白]	-Infinity
	[半角空白][+]Infinity[半角空白]	Infinity
	[半角空白][+｜-]NaN[半角空白]	NaN
	上記以外（double 値にできない）	SQLException を投入
SMALLINT	NULL	0.0
	上記以外	検索結果を double 値にしたもの
INTEGER	NULL	0
	上記以外	検索結果を double 値にしたもの
REAL	NULL	0.0
	Infinity	Infinity
	-Infinity	-Infinity
	NaN	NaN
	上記以外	検索結果を double 値にしたもの
FLOAT	NULL	0.0
	Infinity	Infinity
	-Infinity	-Infinity

HiRDB のデータ型	検索結果	戻り値
	NaN	NaN
	上記以外	検索結果を double 値にしたもの
DECIMAL	NULL	0
	上記以外	検索結果を double 値にしたもの
BOOLEAN※	NULL	0.0
	true	1.0
	false	0.0
その他	−	SQLException を投入

（凡例） −：該当しません。

注※　DatabaseMetadata から生成した Resultset の場合，BOOLEAN 型データが存在します。

【発生する例外】

次の場合，SQLException を投入します。

- この ResultSet オブジェクトが close されている場合

  この ResultSet オブジェクトを生成した Statement を close したことによって，このドライバが ResultSet を close した場合を含みます。
- この ResultSet オブジェクトを生成した Statement を作成した Connection が close されている場合
- トランザクションの決着によって ResultSet が無効になった場合
- 存在しない列番号が指定されている場合
- このメソッドでは取得できないデータ型の場合
- 列の値が double として取得できない場合
- JDBC ドライバ内でエラーが発生した場合

(ai)　getDouble(String columnName)

【機能】

この ResultSet オブジェクトの現在行にある指定された列の値を，Java プログラミング言語の double として取得します。

HiRDB のデータ型による検索結果と戻り値の関係については，「getDouble(int columnIndex)」の【機能詳細】を参照してください。

【形式】

```
public synchronized double getDouble (String columnName) throws SQLException
```

【引数】

String columnName：

列名

【戻り値】

列値（値が NULL 値の場合は null を返します）

【発生する例外】

次の場合，SQLException を投入します。

- この ResultSet オブジェクトが close されている場合

  この ResultSet オブジェクトを生成した Statement を close したことによって，このドライバが ResultSet を close した場合を含みます。
- この ResultSet オブジェクトを生成した Statement を作成した Connection が close されている場合
- トランザクションの決着によって ResultSet が無効になった場合
- 存在しない列名が指定されている場合
- このメソッドでは取得できないデータ型の場合
- 列の値が double として取得できない場合
- JDBC ドライバ内でエラーが発生した場合

(aj)　getFetchDirection()

【機能】

この ResultSet オブジェクトのフェッチ方向を取得します。HiRDB では常に ResultSet.FETCH_FORWARD を返します。

【形式】

```
public synchronized int getFetchDirection() throws SQLException
```

【引数】

なし。

【戻り値】

ResultSet.FETCH_FORWARD：

結果集合が順方向に処理されます。

ResultSet.FETCH_REVERSE：

結果集合が逆方向に処理されます。

ResultSet.FETCH_UNKNOWN：

結果集合が処理される方向が不明です。

【発生する例外】

次の場合，SQLException を投入します。

- この ResultSet オブジェクトが close されている場合
  この ResultSet オブジェクトを生成した Statement を close したことによって，このドライバが ResultSet を close した場合を含みます。
- この ResultSet オブジェクトを生成した Statement を作成した Connection が close されている場合
- トランザクションの決着によって ResultSet が無効になった場合

(ak)　getFetchSize()

【機能】

この ResultSet オブジェクトのフェッチサイズを取得します。

setFetchSize で設定した値を返します。

setFetchSize で設定していない場合は 0 を返します。

【形式】

```
public synchronized int getFetchSize() throws SQLException
```

【引数】

なし。

【戻り値】

この ResultSet オブジェクトの現在のフェッチサイズ

【発生する例外】

次の場合，SQLException を投入します。

- この ResultSet オブジェクトが close されている場合
  この ResultSet オブジェクトを生成した Statement を close したことによって，このドライバが ResultSet を close した場合を含みます。
- この ResultSet オブジェクトを生成した Statement を作成した Connection が close されている場合
- トランザクションの決着によって ResultSet が無効になった場合

(al)　getFloat(int columnIndex)

【機能】

この ResultSet オブジェクトの現在行にある指定された列の値を，Java プログラミング言語の float として取得します。

【形式】

```
public synchronized float getFloat (int columnIndex) throws SQLException
```

【引数】

int columnIndex：

列番号

【戻り値】

列値（値が NULL 値の場合は null を返します）

【機能詳細】

この ResultSet オブジェクトの現在行にある指定された列の値を，Java プログラミング言語の float として取得します。

HiRDB のデータ型による検索結果と戻り値の関係を次に示します。

HiRDB のデータ型	検索結果	戻り値
[M ｜ N][VAR]CHAR	NULL	0.0
	[半角空白]整数の文字列表現，10 進数の文字列表現，又は浮動小数点数の文字列表現[半角空白]であり，かつ次のどちらか • -Float.MAX_VALUE 以上-Float.MIN_VALUE 以下 • Float.MIN_VALUE 以上 Float.MAX_VALUE 以下	検索結果を float 値にしたもの
	[半角空白]整数の文字列表現，10 進数の文字列表現，又は浮動小数点数の文字列表現[半角空白]かつ Float.MAX_VALUE より大きい	Infinity
	[半角空白]整数の文字列表現，10 進数の文字列表現，又は浮動小数点数の文字列表現[半角空白]かつ-Float.MAX_VALUE より小さい	-Infinity
	[半角空白]整数の文字列表現，10 進数の文字列表現，又は浮動小数点数の文字列表現[半角空白]かつ Float.MIN_VALUE より小さく 0 より大きい	0.0
	[半角空白]整数の文字列表現，10 進数の文字列表現，又は浮動小数点数の文字列表現[半角空白]かつ-Float.MIN_VALUE より大きく 0 より小さい	-0.0
	[半角空白]-Infinity[半角空白]	-Infinity
	[半角空白][+]Infinity[半角空白]	Infinity
	[半角空白][+｜-]NaN[半角空白]	NaN
	上記以外（float 値にできない）	SQLException を投入
SMALLINT	NULL	0.0
	上記以外	検索結果を float 値にしたもの
INTEGER	NULL	0.0
	上記以外	検索結果を float 値にしたもの
REAL	NULL	0.0
	Infinity	Infinity
	-Infinity	-Infinity
	NaN	NaN

HiRDB のデータ型	検索結果	戻り値
	上記以外	検索結果
FLOAT	NULL	0.0
	Infinity	Infinity
	-Infinity	-Infinity
	NaN	NaN
	-Float.MAX_VALUE 以上-Float.MIN_VALUE 以下，又は Float.MIN_VALUE 以上 Float.MAX_VALUE 以下	検索結果を float 値にしたもの
	Float.MAX_VALUE より大きい	Infinity
	-Float.MAX_VALUE より小さい	-Infinity
	Float.MIN_VALUE より小さく 0 より大きい	0.0
	-Float.MIN_VALUE より大きく 0 より小さい	-0.0
DECIMAL	NULL	0.0
	上記以外	検索結果を float 値にしたもの
BOOLEAN※	NULL	0.0
	true	1.0
	false	0.0
その他	−	SQLException を投入

（凡例） −：該当しません。

注※　DatabaseMetadata から生成した Resultset の場合，BOOLEAN 型データが存在します。

【発生する例外】

次の場合，SQLException を投入します。

- この ResultSet オブジェクトが close されている場合

  この ResultSet オブジェクトを生成した Statement を close したことによって，このドライバが ResultSet を close した場合を含みます。
- この ResultSet オブジェクトを生成した Statement を作成した Connection が close されている場合
- トランザクションの決着によって ResultSet が無効になった場合
- 存在しない列番号が指定されている場合
- このメソッドでは取得できないデータ型の場合
- 列の値が float として取得できない場合
- JDBC ドライバ内でエラーが発生した場合

(am)　getFloat(String columnName)

【機能】

この ResultSet オブジェクトの現在行にある指定された列の値を，Java プログラミング言語の float として取得します。

HiRDB のデータ型による検索結果と戻り値の関係については，「getFloat(int columnIndex)」の【機能詳細】を参照してください。

【形式】

```
public synchronized float getFloat (String columnName) throws SQLException
```

【引数】

String columnName：

列名

【戻り値】

列値（値が NULL 値の場合は null を返します）

【発生する例外】

次の場合，SQLException を投入します。

- この ResultSet オブジェクトが close されている場合

  この ResultSet オブジェクトを生成した Statement を close したことによって，このドライバが ResultSet を close した場合を含みます。
- この ResultSet オブジェクトを生成した Statement を作成した Connection が close されている場合
- トランザクションの決着によって ResultSet が無効になった場合
- 存在しない列名が指定されている場合
- このメソッドでは取得できないデータ型の場合
- 列の値が float として取得できない場合
- JDBC ドライバ内でエラーが発生した場合

(an)　getInt(int columnIndex)

【機能】

この ResultSet オブジェクトの現在行にある指定された列の値を，Java プログラミング言語の int として取得します。

【形式】

```
public synchronized int getInt(int columnIndex) throws SQLException
```

【引数】

int columnIndex：

列番号

【戻り値】

列値（値が NULL 値の場合は null を返します）

【機能詳細】

この ResultSet オブジェクトの現在行にある指定された列の値を，Java プログラミング言語の int として取得します。

HiRDB のデータ型による検索結果と戻り値の関係を次に示します。

HiRDB のデータ型	検索結果	戻り値
[M ｜ N][VAR]CHAR	NULL	0
	[半角空白]整数の文字列表現，10 進数の文字列表現，又は浮動小数点数の文字列表現[半角空白]であり，かつ Integer.MIN_VALUE 以上，Integer.MAX_VALUE 以下	検索結果の整数部分の値を int 値にしたもの
	[半角空白]整数の文字列表現，10 進数の文字列表現，又は浮動小数点数の文字列表現[半角空白]であり，かつ Integer.MAX_VALUE より大きいか Integer.MIN_VALUE より小さい	SQLException を投入
	[半角空白]-Infinity[半角空白]	SQLException を投入
	[半角空白][+]Infinity[半角空白]	SQLException を投入
	[半角空白][+｜-]NaN[半角空白]	SQLException を投入
	上記以外（double 値にできない）	SQLException を投入
SMALLINT	NULL	0
	上記以外	検索結果を int 値にしたもの
INTEGER	NULL	0
	上記以外	検索結果を int 値にしたもの
REAL	NULL	0
	Infinity	SQLException を投入
	-Infinity	SQLException を投入
	NaN	SQLException を投入
	Integer.MIN_VALUE 以上かつ，Integer.MAX_VALUE 以下	検索結果の整数部分の値を int 値にしたもの
	上記以外	SQLException を投入
FLOAT	NULL	0
	Infinity	SQLException を投入

HiRDB のデータ型	検索結果	戻り値
	-Infinity	SQLException を投入
	NaN	SQLException を投入
	Integer.MIN_VALUE 以上かつ，Integer.MAX_VALUE 以下	検索結果の整数部分の値を int 値にしたもの
	上記以外	SQLException を投入
DECIMAL	NULL	0
	Integer.MIN_VALUE 以上かつ Integer.MAX_VALUE 以下	検索結果の整数部分の値を int 値にしたもの
	上記以外	SQLException を投入
BOOLEAN※	NULL	0
	true	1
	false	0
その他	−	SQLException を投入

（凡例） −：該当しません。

注※　DatabaseMetadata から生成した Resultset の場合，BOOLEAN 型データが存在します。

【発生する例外】

次の場合，SQLException を投入します。

- この ResultSet オブジェクトが close されている場合

  この ResultSet オブジェクトを生成した Statement を close したことによって，このドライバが ResultSet を close した場合を含みます。
- この ResultSet オブジェクトを生成した Statement を作成した Connection が close されている場合
- トランザクションの決着によって ResultSet が無効になった場合
- 存在しない列番号が指定されている場合
- このメソッドでは取得できないデータ型の場合
- 列の値が int として取得できない場合
- JDBC ドライバ内でエラーが発生した場合

(ao)　getInt(String columnName)

【機能】

この ResultSet オブジェクトの現在行にある指定された列の値を，Java プログラミング言語の int として取得します。

HiRDB のデータ型による検索結果と戻り値の関係については，「getInt(int columnIndex)」の【機能詳細】を参照してください。

【形式】

```
public synchronized int getInt(String columnName) throws SQLException
```

【引数】

String columnName：

列名

【戻り値】

列値（値が NULL 値の場合は null を返します）

【発生する例外】

次の場合，SQLException を投入します。

- この ResultSet オブジェクトが close されている場合

  この ResultSet オブジェクトを生成した Statement を close したことによって，このドライバが ResultSet を close した場合を含みます。
- この ResultSet オブジェクトを生成した Statement を作成した Connection が close されている場合
- トランザクションの決着によって ResultSet が無効になった場合
- 存在しない列名が指定されている場合
- このメソッドでは取得できないデータ型の場合
- 列の値が int として取得できない場合
- JDBC ドライバ内でエラーが発生した場合

(ap)　getLong(int columnIndex)

【機能】

この ResultSet オブジェクトの現在行にある指定された列の値を，Java プログラミング言語の long として取得します。

【形式】

```
public synchronized long getLong (int columnIndex) throws SQLException
```

【引数】

int columnIndex：

列番号

【戻り値】

列値（値が NULL 値の場合は null を返します）

【機能詳細】

この ResultSet オブジェクトの現在行にある指定された列の値を，Java プログラミング言語の long として取得します。

HiRDB のデータ型による検索結果と戻り値の関係を次に示します。

HiRDB のデータ型	検索結果	戻り値
[M ｜ N][VAR]CHAR	NULL	0
	[半角空白]整数の文字列表現，又は 10 進数の文字列表現かつ 15 文字以内[半角空白]	検索結果の整数部分の値を long 値にしたもの
	[半角空白]整数の文字列表現，又は 10 進数の文字列表現であり，16 文字以上，又は浮動小数点の文字列表現[半角空白]であり，かつ Long.MIN_VALUE 以上，Long.MAX_VALUE 以下	検索結果の整数部分の値を long 値にしたもの
	[半角空白]整数の文字列表現，又は 10 進数の文字列表現であり 16 文字以上，又は浮動小数点の文字列表現[半角空白]であり，かつ Long.MAX_VALUE より大きいか Long.MIN_VALUE より小さい	SQLException を投入
	[半角空白]-Infinity[半角空白]	SQLException を投入
	[半角空白][+]Infinity[半角空白]	SQLException を投入
	[半角空白][+｜-]NaN[半角空白]	SQLException を投入
	上記以外（double 値，又は BigDecimal オブジェクトにできない）	SQLException を投入
SMALLINT	NULL	0
	上記以外	検索結果を long 値にしたもの
INTEGER	NULL	0
	上記以外	検索結果を long 値にしたもの
REAL	NULL	0
	Infinity	SQLException を投入
	-Infinity	SQLException を投入
	NaN	SQLException を投入
	Long.MIN_VALUE 以上かつ，Long.MAX_VALUE 以下	検索結果の整数部分の値を long 値にしたもの
	上記以外	SQLException を投入
FLOAT	NULL	0
	Infinity	SQLException を投入
	-Infinity	SQLException を投入
	NaN	SQLException を投入
	Long.MIN_VALUE 以上かつ，Long.MAX_VALUE 以下	検索結果の整数部分の値を long 値にしたもの
	上記以外	SQLException を投入
DECIMAL	NULL	0

HiRDB のデータ型	検索結果	戻り値
	Long.MIN_VALUE 以上かつ Long.MAX_VALUE 以下	検索結果の整数部分の値を long 値にしたもの
BINARYBLOB	NULL	0
	0 長データ	0
	1 バイト以上	最大 8 バイトまでをリトルエンディアン形式で long 値にしたもの
BOOLEAN※	NULL	0
	true	1
	false	0
その他	−	SQLException を投入

（凡例） −：該当しません。

注※　DatabaseMetadata から生成した Resultset の場合，BOOLEAN 型データが存在します。

【発生する例外】

次の場合，SQLException を投入します。

- この ResultSet オブジェクトが close されている場合

  この ResultSet オブジェクトを生成した Statement を close したことによって，このドライバが ResultSet を close した場合を含みます。
- この ResultSet オブジェクトを生成した Statement を作成した Connection が close されている場合
- トランザクションの決着によって ResultSet が無効になった場合
- 存在しない列番号が指定されている場合
- このメソッドでは取得できないデータ型の場合
- 列の値が long として取得できない場合
- JDBC ドライバ内でエラーが発生した場合

(aq)　getLong(String columnName)

【機能】

この ResultSet オブジェクトの現在行にある指定された列の値を，Java プログラミング言語の long として取得します。

HiRDB のデータ型による検索結果と戻り値の関係については，「getLong(int columnIndex)」の【機能詳細】を参照してください。

【形式】

```
public synchronized long getLong (String columnName) throws SQLException
```

【引数】

String columnName：

列名

【戻り値】

列値（値が NULL 値の場合は null を返します）

【発生する例外】

次の場合，SQLException を投入します。

- この ResultSet オブジェクトが close されている場合

  この ResultSet オブジェクトを生成した Statement を close したことによって，このドライバが ResultSet を close した場合を含みます。
- この ResultSet オブジェクトを生成した Statement を作成した Connection が close されている場合
- トランザクションの決着によって ResultSet が無効になった場合
- 存在しない列名が指定されている場合
- このメソッドでは取得できないデータ型の場合
- 列の値が long として取得できない場合
- JDBC ドライバ内でエラーが発生した場合

(ar)　getMetaData()

【機能】

この ResultSet オブジェクトのメタ情報を表す ResultSetMetaData を返します。

【形式】

```
public synchronized ResultSetMetaData getMetaData() throws SQLException
```

【引数】

なし。

【戻り値】

この ResultSet オブジェクトのメタ情報を，ResultSetMetaData オブジェクトで返します。

【発生する例外】

次の場合，SQLException を投入します。

- この ResultSet オブジェクトが close されている場合

  この ResultSet オブジェクトを生成した Statement を close したことによって，このドライバが ResultSet を close した場合を含みます。
- この ResultSet オブジェクトを生成した Statement を作成した Connection が close されている場合
- トランザクションの決着によって ResultSet が無効になった場合

(as)　getObject(int columnIndex)

【機能】

この ResultSet オブジェクトの現在行にある指定された列の値を，Java プログラミング言語の Object として取得します。

【形式】

```
public synchronized Object getObject (int columnIndex) throws SQLException
```

【引数】

int columnIndex：

列番号

【戻り値】

列の値を Java オブジェクトとして返します。

Java オブジェクトの型は，JDBC 仕様で指定されている組み込み型のマッピングに従って，列の SQL 型に対応するデフォルトの Java オブジェクトの型になります。値が NULL 値の場合，null を返します。

【機能詳細】

この ResultSet オブジェクトの現在行にある指定された列の値を，Java プログラミング言語の Object として取得します。

HiRDB のデータ型による検索結果と戻り値の関係を次に示します。

HiRDB のデータ型	検索結果	戻り値
[M ｜ N][VAR]CHAR	NULL	null
	上記以外	検索結果
SMALLINT	NULL	null
	上記以外	検索結果で生成した Integer オブジェクト
INTEGER	NULL	null
	上記以外	検索結果で生成した Integer オブジェクト
REAL	NULL	null
	上記以外	検索結果で生成した Float オブジェクト
FLOAT	NULL	null
	上記以外	検索結果で生成した Double オブジェクト
DECIMAL	NULL	null
	上記以外	検索結果
DATE	NULL	null
	上記以外	検索結果で生成した java.sql.Date オブジェクト
TIME	NULL	null

HiRDB のデータ型	検索結果	戻り値
	上記以外	検索結果で生成した java.sql.Time オブジェクト
TIMESTAMP	NULL	null
	上記以外	検索結果で生成した java.sql.Timestamp オブジェクト
BINARYBLOB	NULL	null
	上記以外	検索結果
BOOLEAN※	NULL	null
	NULL 以外	検索結果で生成した Boolean オブジェクト

注※　DatabaseMetadata から生成した Resultset の場合，BOOLEAN 型データが存在します。

【発生する例外】

次の場合，SQLException を投入します。

- この ResultSet オブジェクトが close されている場合

  この ResultSet オブジェクトを生成した Statement を close したことによって，このドライバが ResultSet を close した場合を含みます。
- この ResultSet オブジェクトを生成した Statement を作成した Connection が close されている場合
- トランザクションの決着によって ResultSet が無効になった場合
- 存在しない列番号が指定されている場合
- JDBC ドライバ内でエラーが発生した場合

(at)　getObject(String columnName)

【機能】

この ResultSet オブジェクトの現在行にある指定された列の値を，Java プログラミング言語の Object として取得します。

HiRDB のデータ型による検索結果と戻り値の関係については，「getObject(int columnIndex)」の【機能詳細】を参照してください。

【形式】

```
public synchronized Object getObject (String columnName) throws SQLException
```

【引数】

String columnName：

列名

【戻り値】

列の値を Java オブジェクトとして返します。

Java オブジェクトの型は，JDBC 仕様で指定されている組み込み型のマッピングに従って，列の SQL 型に対応するデフォルトの Java オブジェクトの型になります。値が NULL 値の場合，null を返します。

【発生する例外】

次の場合，SQLException を投入します。

- この ResultSet オブジェクトが close されている場合

  この ResultSet オブジェクトを生成した Statement を close したことによって，このドライバが ResultSet を close した場合を含みます。
- この ResultSet オブジェクトを生成した Statement を作成した Connection が close されている場合
- トランザクションの決着によって ResultSet が無効になった場合
- 存在しない列名が指定されている場合
- JDBC ドライバ内でエラーが発生した場合

(au)　getRow()

【機能】

現在の行の番号を取得します。最初の行が 1，2 番目は 2 となります。先頭行の前又は最終行の後ろの場合は 0 となります。

最大検索行数が 2147483647 を超える場合，2147483647 を返します。

【形式】

```
public synchronized int getRow() throws SQLException
```

【引数】

なし。

【戻り値】

現在の行番号（現在の行が Integer.MAX_VALUE より大きい場合は Integer.MAX_VALUE を返します）

【発生する例外】

次の場合，SQLException を投入します。

- この ResultSet オブジェクトが close されている場合

  この ResultSet オブジェクトを生成した Statement を close したことによって，このドライバが ResultSet を close した場合を含みます。
- この ResultSet オブジェクトを生成した Statement を作成した Connection が close されている場合
- トランザクションの決着によって ResultSet が無効になった場合
- データベースアクセスでエラーが発生した場合

(av)　getShort(int columnIndex)

【機能】

この ResultSet オブジェクトの現在行にある指定された列の値を，Java プログラミング言語の short として取得します。

【形式】

```
public synchronized short getShort(int columnIndex) throws SQLException
```

【引数】

int columnIndex：

列番号

【戻り値】

列値（値が NULL 値の場合は 0 を返します）

【機能詳細】

この ResultSet オブジェクトの現在行にある指定された列の値を，Java プログラミング言語の short として取得します。

HiRDB のデータ型による検索結果と戻り値の関係を次に示します。

HiRDB のデータ型	検索結果	戻り値
[M ｜ N][VAR]CHAR	NULL	0
	[半角空白]整数の文字列表現，10 進数の文字列表現，又は浮動小数点数の文字列表現[半角空白]であり，かつ Short.MIN_VALUE 以上，Short.MAX_VALUE 以下	検索結果の整数部分の値を short 値にしたもの
	[半角空白]整数の文字列表現，10 進数の文字列表現，又は浮動小数点数の文字列表現[半角空白]であり，かつ Short.MAX_VALUE より大きいか Short.MIN_VALUE より小さい	SQLException を投入
	[半角空白]-Infinity[半角空白]	SQLException を投入
	[半角空白][+]Infinity[半角空白]	SQLException を投入
	[半角空白][+｜-]NaN[半角空白]	SQLException を投入
	上記以外（double 値にできない）	SQLException を投入
SMALLINT	NULL	0
	上記以外	検索結果
INTEGER	NULL	0
	Short.MIN_VALUE 以上かつ，Short.MAX_VALUE 以下	検索結果を short 値にしたもの
	上記以外	SQLException を投入
REAL	NULL	0
	Infinity	SQLException を投入
	-Infinity	SQLException を投入
	NaN	SQLException を投入

HiRDB のデータ型	検索結果	戻り値
	Short.MIN_VALUE 以上かつ，Short.MAX_VALUE 以下	検索結果の整数部分の値を short 値にしたもの
	上記以外	SQLException を投入
FLOAT	NULL	0
	Infinity	SQLException を投入
	-Infinity	SQLException を投入
	NaN	SQLException を投入
	Short.MIN_VALUE 以上かつ，Short.MAX_VALUE 以下	検索結果の整数部分の値を short 値にしたもの
	上記以外	SQLException を投入
DECIMAL	NULL	0
	Short.MIN_VALUE 以上かつ Short.MAX_VALUE 以下	検索結果の整数部分の値を short 値にしたもの
	上記以外	SQLException を投入
BOOLEAN※	NULL	0
	true	1
	false	0
その他	−	SQLException を投入

（凡例） −：該当しません。

注※　DatabaseMetadata から生成した Resultset の場合，BOOLEAN 型データが存在します。

【発生する例外】

次の場合，SQLException を投入します。

- この ResultSet オブジェクトが close されている場合
  この ResultSet オブジェクトを生成した Statement を close したことによって，このドライバが ResultSet を close した場合を含みます。
- この ResultSet オブジェクトを生成した Statement を作成した Connection が close されている場合
- トランザクションの決着によって ResultSet が無効になった場合
- 存在しない列番号が指定されている場合
- このメソッドでは取得できないデータ型の場合
- 列の値が short として取得できない場合
- JDBC ドライバ内でエラーが発生した場合

(aw)　getShort(String columnName)

【機能】

この ResultSet オブジェクトの現在行にある指定された列の値を，Java プログラミング言語の short として取得します。

HiRDB のデータ型による検索結果と戻り値の関係については，「getShort(int columnIndex)」の【機能詳細】を参照してください。

【形式】

```
public synchronized short getShort (String columnName) throws SQLException
```

【引数】

String columnName：

列名

【戻り値】

列値（値が NULL 値の場合は 0 を返します）

【発生する例外】

次の場合，SQLException を投入します。

- この ResultSet オブジェクトが close されている場合

  この ResultSet オブジェクトを生成した Statement を close したことによって，このドライバが ResultSet を close した場合を含みます。
- この ResultSet オブジェクトを生成した Statement を作成した Connection が close されている場合
- トランザクションの決着によって ResultSet が無効になった場合
- 存在しない列名が指定されている場合
- このメソッドでは取得できないデータ型の場合
- 列の値が short として取得できない場合
- JDBC ドライバ内でエラーが発生した場合

(ax)　getStatement()

【機能】

この ResultSet オブジェクトを生成した Statement オブジェクトを取得します。

結果集合が DatabaseMetaData のメソッドで生成された場合は，null を返します。

【形式】

```
public synchronized Statement getStatement() throws SQLException
```

【引数】

なし。

【戻り値】

この ResultSet オブジェクトを生成した Statment オブジェクト（結果集合が DatabaseMetaData のメソッドで生成された場合は，null を返します）

【発生する例外】

次の場合，SQLException を投入します。

- この ResultSet オブジェクトが close されている場合

  この ResultSet オブジェクトを生成した Statement を close したことによって，このドライバが ResultSet を close した場合を含みます。
- この ResultSet オブジェクトを生成した Statement を作成した Connection が close されている場合
- トランザクションの決着によって ResultSet が無効になる場合

(ay)　getString(int columnIndex)

【機能】

この ResultSet オブジェクトの現在行にある指定された列の値を，Java プログラミング言語の String として取得します。

【形式】

```
public synchronized String getString(int columnIndex) throws SQLException
```

【引数】

int columnIndex：

列番号

【戻り値】

列値（値が NULL 値の場合，null を返します）

【機能詳細】

この ResultSet オブジェクトの現在行にある指定された列の値を，Java プログラミング言語の String として取得します。

HiRDB のデータ型による検索結果と戻り値の関係を次に示します。

HiRDB のデータ型	検索結果	戻り値
[M ｜ N][VAR]CHAR	NULL	null
	上記以外	検索結果
SMALLINT	NULL	null
	上記以外	検索結果を文字列表現にした String オブジェクト
INTEGER	NULL	null
	上記以外	検索結果を文字列表現にした String オブジェクト
REAL	NULL	null

HiRDB のデータ型	検索結果	戻り値
	Infinity	文字列"Infinity"の String オブジェクト
	-Infinity	文字列"-Infinity"の String オブジェクト
	NaN	文字列"NaN"の String オブジェクト
	上記以外	検索結果を文字列表現にした String オブジェクト
FLOAT	NULL	null
	Infinity	文字列"Infinity"の String オブジェクト
	-Infinity	文字列"-Infinity"の String オブジェクト
	NaN	文字列"NaN"の String オブジェクト
	上記以外	検索結果を文字列表現にした String オブジェクト
DECIMAL	NULL	null
	上記以外	検索結果を文字列表現にした String オブジェクト
DATE	NULL	null
	上記以外	JdbConvert.convertDate()で取得した yyyy-MM-DD 形式の文字列の String オブジェクト
TIME	NULL	null
	上記以外	hh:mm:ss 形式の文字列の String オブジェクト
TIMESTAMP	NULL	null
	上記以外	yyyy-MM-DD△hh:mm:ss[.ffffff]形式の文字列の String オブジェクト
BINARYBLOB	NULL	null
	上記以外	検索結果の String オブジェクト
BOOLEAN※	NULL	null
	true	文字列"true"の String オブジェクト
	false	文字列"false"の String オブジェクト

注※　DatabaseMetadata から生成した Resultset の場合，BOOLEAN 型データが存在します。

【発生する例外】

次の場合，SQLException を投入します。

- この ResultSet オブジェクトが close されている場合

  この ResultSet オブジェクトを生成した Statement を close したことによって，このドライバが ResultSet を close した場合を含みます。
- この ResultSet オブジェクトを生成した Statement を作成した Connection が close されている場合
- トランザクションの決着によって ResultSet が無効になった場合
- 存在しない列番号が指定されている場合

- エンコードに失敗した場合
- このメソッドでは取得できないデータ型の場合
- JDBC ドライバ内でエラーが発生した場合

(az)　getString(String columnName)

【機能】

この ResultSet オブジェクトの現在行にある指定された列の値を，Java プログラミング言語の String として取得します。

HiRDB のデータ型による検索結果と戻り値の関係については，「getString(int columnIndex)」の【機能詳細】を参照してください。

【形式】

```
public synchronized String getString(String columnName) throws SQLException
```

【引数】

String columnName：

列名

【戻り値】

列値（値が NULL 値の場合，null を返します）

【発生する例外】

次の場合，SQLException を投入します。

- この ResultSet オブジェクトが close されている場合

  この ResultSet オブジェクトを生成した Statement を close したことによって，このドライバが ResultSet を close した場合を含みます。
- この ResultSet オブジェクトを生成した Statement を作成した Connection が close されている場合
- トランザクションの決着によって ResultSet が無効になった場合
- 存在しない列名が指定されている場合
- エンコードに失敗した場合
- このメソッドでは取得できないデータ型の場合
- JDBC ドライバ内でエラーが発生した場合

(ba)　getTime(int columnIndex)

【機能】

この ResultSet オブジェクトの現在行にある指定された列の値を，java.sql.Time として取得します。

【形式】

```
public synchronized java.sql.Time getTime(int columnIndex) throws SQLException
```

【引数】

int columnIndex：

列番号

【戻り値】

列値（値が NULL 値の場合，null を返します）

【機能詳細】

この ResultSet オブジェクトの現在行にある指定された列の値を，java.sql.Time として取得します。

HiRDB のデータ型による検索結果と戻り値の関係を次に示します。

HiRDB のデータ型	検索結果	戻り値
[M | N][VAR]CHAR	NULL	null
	[半角空白]時刻形式[半角空白]	検索結果の前後の半角空白を取り除き java.sql.Time オブジェクトにしたもの
	上記以外	SQLException を返します
TIME	NULL	null
	上記以外	検索結果を java.sql.Time オブジェクトにしたもの
TIMESTAMP	NULL	null
	上記以外	検索結果を java.sql.Time オブジェクトにしたもの
その他	−	SQLException を返します

（凡例） −：該当しません。

【発生する例外】

次の場合，SQLException を投入します。

- この ResultSet オブジェクトが close されている場合

  この ResultSet オブジェクトを生成した Statement を close したことによって，このドライバが ResultSet を close した場合を含みます。
- この ResultSet オブジェクトを生成した Statement を作成した Connection が close されている場合
- トランザクションの決着によって ResultSet が無効になった場合
- 存在しない列番号が指定されている場合
- このメソッドでは取得できないデータ型の場合
- 列の値が java.sql.Time として取得できない場合
- JDBC ドライバ内でエラーが発生した場合

(bb)　getTime(int columnIndex, Calendar cal)

【機能】

この ResultSet オブジェクトの現在行にある指定された列の値を，java.sql.Time オブジェクトとして取得します。このメソッドは指定されたカレンダーを使って時刻に適切なミリ秒値を作成します。

【形式】

```
public synchronized java.sql.Time getTime (int columnIndex, Calendar cal) throws SQLExcep
tion
```

【引数】

int columnIndex：

列番号

Calendar cal：

データベースに格納されている値のタイムゾーンが設定されたカレンダー

【戻り値】

列値（値が NULL 値の場合，null を返します）

HiRDB のデータ型による検索結果と戻り値の関係については，「getTime(int columnIndex)」の【機能詳細】を参照してください。

【発生する例外】

次の場合，SQLException を投入します。

- この ResultSet オブジェクトが close されている場合

  この ResultSet オブジェクトを生成した Statement を close したことによって，このドライバが ResultSet を close した場合を含みます。
- この ResultSet オブジェクトを生成した Statement を作成した Connection が close されている場合
- トランザクションの決着によって ResultSet が無効になった場合
- 存在しない列番号が指定されている場合
- このメソッドでは取得できないデータ型の場合
- 列の値が java.sql.Time として取得できない場合
- JDBC ドライバ内でエラーが発生した場合

(bc)　getTime(String columnName)

【機能】

この ResultSet オブジェクトの現在行にある指定された列の値を，java.sql.Time オブジェクトとして取得します。

HiRDB のデータ型による検索結果と戻り値の関係については，「getTime(int columnIndex)」の【機能詳細】を参照してください。

【形式】

```
public synchronized java.sql.Time getTime (String columnName) throws SQLException
```

【引数】

String columnName：

列名

【戻り値】

列値（値が NULL 値の場合，null を返します）

【発生する例外】

次の場合，SQLException を投入します。

- この ResultSet オブジェクトが close されている場合

  この ResultSet オブジェクトを生成した Statement を close したことによって，このドライバが ResultSet を close した場合を含みます。
- この ResultSet オブジェクトを生成した Statement を作成した Connection が close されている場合
- トランザクションの決着によって ResultSet が無効になった場合
- 存在しない列名が指定されている場合
- このメソッドでは取得できないデータ型の場合
- 列の値が java.sql.Time として取得できない場合
- JDBC ドライバ内でエラーが発生した場合

(bd)　getTime(String columnName, Calendar cal)

【機能】

この ResultSet オブジェクトの現在行にある指定された列の値を，java.sql.Time オブジェクトとして取得します。このメソッドは指定されたカレンダーを使って時刻に適切なミリ秒値を作成します。

HiRDB のデータ型による検索結果と戻り値の関係については，「getTime(int columnIndex)」の【機能詳細】を参照してください。

【形式】

```
public synchronized java.sql.Time getTime (String columnName, Calendar cal) throws SQLExc
eption
```

【引数】

String columnName：

列名

Calendar cal：

データベースに格納されている値のタイムゾーンが設定されたカレンダー

【戻り値】

列値（値が NULL 値の場合，null を返します）

【発生する例外】

次の場合，SQLException を投入します。

- この ResultSet オブジェクトが close されている場合

  この ResultSet オブジェクトを生成した Statement を close したことによって，このドライバが ResultSet を close した場合を含みます。
- この ResultSet オブジェクトを生成した Statement を作成した Connection が close されている場合
- トランザクションの決着によって ResultSet が無効になった場合
- 存在しない列名が指定されている場合
- このメソッドでは取得できないデータ型の場合
- 列の値が java.sql.Time として取得できない場合
- JDBC ドライバ内でエラーが発生した場合

(be)　getTimestamp(int columnIndex)

【機能】

この ResultSet オブジェクトの現在行にある指定された列の値を，java.sql.Timestamp オブジェクトとして取得します。

【形式】

```
public synchronized java.sql.Timestamp getTimestamp (int columnIndex) throws SQLException
```

【引数】

int columnIndex：

列番号

【戻り値】

列値（値が NULL 値の場合，null を返します）

【機能詳細】

この ResultSet オブジェクトの現在行にある指定された列の値を，java.sql.Timestamp オブジェクトとして取得します。

HiRDB のデータ型による検索結果と戻り値の関係については「TIME 型，DATE 型，及び TIMESTAMP 型列のデータ変換処理」を参照してください。TIME 型，DATE 型，TIMESTAMP 型，文字列型以外のデータ型を取得した場合，SQLException を投入します。

【発生する例外】

次の場合，SQLException を投入します。

- この ResultSet オブジェクトが close されている場合

この ResultSet オブジェクトを生成した Statement を close したことによって，このドライバが ResultSet を close した場合を含みます。

- この ResultSet オブジェクトを生成した Statement を作成した Connection が close されている場合
- トランザクションの決着によって ResultSet が無効になった場合
- 存在しない列番号が指定されている場合
- このメソッドでは取得できないデータ型の場合
- 列の値が java.sql.Timestamp として取得できない場合
- JDBC ドライバ内でエラーが発生した場合

(bf)　getTimestamp(int columnIndex, Calendar cal)

【機能】

この ResultSet オブジェクトの現在行にある指定された列の値を，java.sql.Timestamp オブジェクトとして取得します。このメソッドは指定されたカレンダーを使ってタイムスタンプに適切なミリ秒値を作成します。

HiRDB のデータ型による検索結果と戻り値の関係については，「TIME 型，DATE 型，及び TIMESTAMP 型列のデータ変換処理」を参照してください。TIME 型，DATE 型，TIMESTAMP 型，文字列型以外のデータ型を取得した場合，SQLException を投入します。

【形式】

```
public synchronized java.sql.Timestamp getTimestamp (int columnIndex, Calendar cal) throw
s SQLException
```

【引数】

int columnIndex：

列番号

Calendar cal：

データベースに格納されている値のタイムゾーンが設定されたカレンダー

【戻り値】

列値（値が NULL 値の場合，null を返します）

【発生する例外】

次の場合，SQLException を投入します。

- この ResultSet オブジェクトが close されている場合

  この ResultSet オブジェクトを生成した Statement を close したことによって，このドライバが ResultSet を close した場合を含みます。
- この ResultSet オブジェクトを生成した Statement を作成した Connection が close されている場合
- トランザクションの決着によって ResultSet が無効になった場合
- 存在しない列番号が指定されている場合

- このメソッドでは取得できないデータ型の場合
- 列の値が java.sql.Timestamp として取得できない場合
- JDBC ドライバ内でエラーが発生した場合

(bg)　getTimestamp(String columnName)

【機能】

この ResultSet オブジェクトの現在行にある指定された列の値を，java.sql.Timestamp オブジェクトとして取得します。

HiRDB のデータ型による検索結果と戻り値の関係については，「TIME 型，DATE 型，及び TIMESTAMP 型列のデータ変換処理」を参照してください。TIME 型，DATE 型，TIMESTAMP 型，文字列型以外のデータ型を取得した場合，SQLException を投入します。

【形式】

```
public synchronized java.sql.Timestamp getTimestamp (String columnName) throws SQLExcepti
on
```

【引数】

String columnName：

列名

【戻り値】

列値（値が NULL 値の場合，null を返します）

【発生する例外】

次の場合，SQLException を投入します。

- この ResultSet オブジェクトが close されている場合

  この ResultSet オブジェクトを生成した Statement を close したことによって，このドライバが ResultSet を close した場合を含みます。
- この ResultSet オブジェクトを生成した Statement を作成した Connection が close されている場合
- トランザクションの決着によって ResultSet が無効になった場合
- 存在しない列名が指定されている場合
- このメソッドでは取得できないデータ型の場合
- 列の値が java.sql.Timestamp として取得できない場合
- JDBC ドライバ内でエラーが発生した場合

(bh)　getTimestamp(String columnName, Calendar cal)

【機能】

この ResultSet オブジェクトの現在行にある指定された列の値を，java.sql.Timestamp オブジェクトとして取得します。このメソッドは指定されたカレンダーを使ってタイムスタンプに適切なミリ秒値を作成します。

HiRDB のデータ型による検索結果と戻り値の関係については，「TIME 型，DATE 型，及び TIMESTAMP 型列のデータ変換処理」を参照してください。TIME 型，DATE 型，TIMESTAMP 型，文字列型以外のデータ型を取得した場合，SQLException を投入します。

【形式】

```
public synchronized java.sql.Timestamp getTimestamp (String columnName, Calendar cal) thr
ows SQLException
```

【引数】

String columnName：

列名

Calendar cal：

データベースに格納されている値のタイムゾーンが設定されたカレンダー

【戻り値】

列値（値が NULL 値の場合，null を返します）

【発生する例外】

次の場合，SQLException を投入します。

- この ResultSet オブジェクトが close されている場合

  この ResultSet オブジェクトを生成した Statement を close したことによって，このドライバが ResultSet を close した場合を含みます。
- この ResultSet オブジェクトを生成した Statement を作成した Connection が close されている場合
- トランザクションの決着によって ResultSet が無効になった場合
- 存在しない列名が指定されている場合
- このメソッドでは取得できないデータ型の場合
- 列の値が java.sql.Timestamp として取得できない場合
- JDBC ドライバ内でエラーが発生した場合

(bi)　getType()

【機能】

この ResultSet オブジェクトの型を返します。ResultSet.TYPE_FORWARD_ONLY 又は ResultSet.TYPE_SCROLL_INSENSITIVE を返します。

【形式】

```
public synchronized int getType() throws SQLException
```

【引数】

なし。

【戻り値】

ResultSet.TYPE_FORWARD_ONLY：

カーソルが順方向にだけ移動できます。

ResultSet.TYPE_SCROLL_INSENSITIVE：

カーソルがスクロールできますが，値の変更は反映されません。

ResultSet.TYPE_SCROLL_SENSITIVE：

カーソルはスクロールでき，値の変更が反映されます。

【発生する例外】

次の場合，SQLException を投入します。

- この ResultSet オブジェクトが close されている場合

  この ResultSet オブジェクトを生成した Statement を close したことによって，このドライバが ResultSet を close した場合を含みます。
- この ResultSet オブジェクトを生成した Statement を作成した Connection が close されている場合
- トランザクションの決着によって ResultSet が無効になる場合

(bj)　getWarnings()

【機能】

この ResultSet オブジェクトに関する呼び出しによって報告される，最初の警告を取得します。二つ以上の警告がある場合，後続の警告は最初の警告にチェーンされ，直前に取得された警告の SQLWarning.getNextWarning メソッドを呼び出すことによって取得されます。

【形式】

```
public synchronized SQLWarning getWarnings() throws SQLException
```

【引数】

なし。

【戻り値】

最初の SQLWarning オブジェクト（ない場合は null を返します）

【発生する例外】

次の場合，SQLException を投入します。

- この ResultSet オブジェクトが close されている場合

この ResultSet オブジェクトを生成した Statement を close したことによって，このドライバが ResultSet を close した場合を含みます。

- この ResultSet オブジェクトを生成した Statement を作成した Connection が close されている場合
- トランザクションの決着によって ResultSet が無効になった場合

(bk)　isAfterLast()

【機能】

カーソルが，この ResultSet オブジェクト内の最終行の後ろにあるかどうかを取得します。

【形式】

```
public synchronized boolean isAfterLast() throws SQLException
```

【引数】

なし。

【戻り値】

カーソルが最終行の後ろにある場合は true，最終行の後ろにない場合，又は結果集合の行数が 0 行の場合は false を返します。

【発生する例外】

次の場合，SQLException を投入します。

- この ResultSet オブジェクトが close されている場合
  この ResultSet オブジェクトを生成した Statement を close したことによって，このドライバが ResultSet を close した場合を含みます。
- この ResultSet オブジェクトを生成した Statement を作成した Connection が close されている場合
- トランザクションの決着によって ResultSet が無効になった場合
- データベースアクセスでエラーが発生した場合

(bl)　isBeforeFirst()

【機能】

カーソルが，この ResultSet オブジェクト内の先頭行の前にあるかどうかを取得します。

【形式】

```
public synchronized boolean isBeforeFirst() throws SQLException
```

【引数】

なし。

【戻り値】

カーソルが先頭行の前にある場合は true，先頭行の前にない場合，又は結果集合の行数が 0 行の場合は false を返します。

【発生する例外】

次の場合，SQLException を投入します。

- この ResultSet オブジェクトが close されている場合

  この ResultSet オブジェクトを生成した Statement を close したことによって，このドライバが ResultSet を close した場合を含みます。
- この ResultSet オブジェクトを生成した Statement を作成した Connection が close されている場合
- トランザクションの決着によって ResultSet が無効になった場合
- データベースアクセスでエラーが発生した場合

(bm)　isFirst()

【機能】

カーソルが，この ResultSet オブジェクト内の先頭行にあるかどうかを取得します。

【形式】

```
public synchronized boolean isFirst() throws SQLException
```

【引数】

なし。

【戻り値】

カーソルが先頭行にある場合は true，そうでない場合は false を返します。

【発生する例外】

次の場合，SQLException を投入します。

- この ResultSet オブジェクトが close されている場合

  この ResultSet オブジェクトを生成した Statement を close したことによって，このドライバが ResultSet を close した場合を含みます。
- この ResultSet オブジェクトを生成した Statement を作成した Connection が close されている場合
- トランザクションの決着によって ResultSet が無効になった場合
- データベースアクセスでエラーが発生した場合

(bn)　isLast()

【機能】

カーソルが，この ResultSet オブジェクトの最終行にあるかどうかを取得します。

【形式】

```
public synchronized boolean isLast() throws SQLException
```

【引数】

なし。

【戻り値】

カーソルが最終行にある場合は true，そうでない場合は false を返します。

【発生する例外】

次の場合，SQLException を投入します。

- この ResultSet オブジェクトが close されている場合

  この ResultSet オブジェクトを生成した Statement を close したことによって，このドライバが ResultSet を close した場合を含みます。
- この ResultSet オブジェクトを生成した Statement を作成した Connection が close されている場合
- トランザクションの決着によって ResultSet が無効になった場合
- データベースアクセスでエラーが発生した場合

(bo)　last()

【機能】

カーソルを，この ResultSet オブジェクトの最終行に移動します。

【形式】

```
public synchronized boolean last() throws SQLException
```

【引数】

なし。

【戻り値】

カーソルが最終行に移動した場合は true を，結果集合の行数が 0 の場合は false を返します。

【発生する例外】

次の場合，SQLException を投入します。

- この ResultSet オブジェクトが close されている場合

  この ResultSet オブジェクトを生成した Statement を close したことによって，このドライバが ResultSet を close した場合を含みます。
- この ResultSet オブジェクトを生成した Statement を作成した Connection が close されている場合
- この ResultSet オブジェクトの型が TYPE_FORWARD_ONLY の場合
- トランザクションの決着によって ResultSet が無効になった場合
- データベースアクセスでエラーが発生した場合

(bp)　next()

【機能】

カーソルを，現在の位置から次の行に移動します。先頭行の前の場合は先頭行に，最終行の場合は最終行の後ろに移動します。

next メソッドの最初の呼び出しによって，カーソルがオープンされます。

【形式】

```
public synchronized boolean next() throws SQLException
```

【引数】

なし。

【戻り値】

このメソッド呼び出し後のカーソル位置が先頭行の前，又は最終行の後ろの場合は false，そうでない場合は true を返します。

【発生する例外】

次の場合，SQLException を投入します。

- この ResultSet オブジェクトが close されている場合

  この ResultSet オブジェクトを生成した Statement を close したことによって，このドライバが ResultSet を close した場合を含みます。
- この ResultSet オブジェクトを生成した Statement を作成した Connection が close されている場合
- トランザクションの決着によって ResultSet が無効になった場合
- データベースアクセスでエラーが発生した場合

(bq)　previous()

【機能】

カーソルを，この ResultSet オブジェクト内の一つ前の行に移動します。

【形式】

```
public synchronized boolean previous() throws SQLException
```

【引数】

なし。

【戻り値】

このメソッド呼び出し後のカーソル位置が先頭行の前の場合は false，そうでない場合は true を返します。

【発生する例外】

次の場合，SQLException を投入します。

- この ResultSet オブジェクトが close されている場合

  この ResultSet オブジェクトを生成した Statement を close したことによって，このドライバが ResultSet を close した場合を含みます。
- この ResultSet オブジェクトを生成した Statement を作成した Connection が close されている場合
- この ResultSet オブジェクトの型が TYPE_FORWARD_ONLY の場合

- トランザクションの決着によって ResultSet が無効になった場合
- データベースアクセスでエラーが発生した場合

(br)　relative(int rows)

【機能】

カーソルを，正又は負の相対行数だけ移動します。

正の数はカーソルを順方向に移動し，負の数は逆方向に移動します。

【形式】

```
public synchronized boolean relative(int rows) throws SQLException
```

【引数】

int rows：

現在の行から移動する行数

【戻り値】

このメソッド呼び出し後のカーソル位置が先頭行の前，又は最終行の後ろの場合は false，そうでない場合は true に返します。

【発生する例外】

次の場合，SQLException を投入します。

- この ResultSet オブジェクトが close されている場合

  この ResultSet オブジェクトを生成した Statement を close したことによって，このドライバが ResultSet を close した場合を含みます。
- この ResultSet オブジェクトを生成した Statement を作成した Connection が close されている場合
- この ResultSet オブジェクトの型が TYPE_FORWARD_ONLY の場合
- 現在の位置が取得できない場合
- 現在のカーソル位置が有効な行にない場合
- トランザクションの決着によって ResultSet が無効になった場合
- データベースアクセスでエラーが発生した場合

(bs)　setFetchDirection(int direction)

【機能】

この Statement オブジェクトから生成される結果集合の，デフォルトのフェッチ方向を指定します。

ResultSet.FETCH_FORWARD だけ指定できます。

【形式】

```
public synchronized void setFetchDirection(int direction) throws SQLException
```

【引数】

int direction：

デフォルトのフェッチ方向

【戻り値】

なし。

【発生する例外】

次の場合，SQLException を投入します。

- この Statement オブジェクトが close されている場合
- この Statement オブジェクトを生成した Connection が close されている場合
- direction に ResultSet.FETCH_FORWARD 以外が指定された場合

(bt)　setFetchSize(int rows)

【機能】

この ResultSet オブジェクトのフェッチサイズを設定します。0 を指定した場合は，クライアント環境定義に依存します。

【形式】

```
public synchronized void setFetchSize(int rows) throws SQLException
```

【引数】

int rows：

フェッチする行数。設定できる範囲は 0〜4096 です。

【戻り値】

なし。

【機能詳細】

この ResultSet オブジェクトのフェッチサイズを設定します。0 を指定した場合は，クライアント環境定義に依存します。

このメソッドによる指定がない場合は，Statement オブジェクトに指定した行数値をヒントとして検索します。Statement オブジェクトに行数値を指定していない，又は Statement オブジェクトから生成した ResultSet オブジェクトでない場合は，クライアント環境定義 PDBLKF の値をヒントとして検索します。このメソッドで 0 を指定した場合は，クライアント環境定義 PDBLKF の値をヒントとして検索します。その他の注意事項については，「setFetchSize(int rows)」を参照してください。

【発生する例外】

次の場合，SQLException を投入します。

- この ResultSet オブジェクトが close されている場合

  この ResultSet オブジェクトを生成した Statement を close したことによって，このドライバが ResultSet を close した場合を含みます。

- この ResultSet オブジェクトを生成した Statement を作成した Connection が close されている場合
- この ResultSet オブジェクトの型が TYPE_FORWARD_ONLY の場合
- 現在の位置が取得できない場合
- 現在のカーソル位置が有効な行にない場合
- トランザクションの決着によって ResultSet が無効になった場合
- データベースアクセスでエラーが発生した場合

(bu)　wasNull()

【機能】

最後に取得した列の値が，NULL 値であるかどうかを通知します。

getXXX メソッドによって値を取得する前は，false を返します。

【形式】

```
public synchronized boolean wasNull() throws SQLException
```

【引数】

なし。

【戻り値】

最後に取得した列の値が NULL 値の場合は true，そうでない場合は false を返します。

【発生する例外】

次の場合，SQLException を投入します。

- この ResultSet オブジェクトが close されている場合

  この ResultSet オブジェクトを生成した Statement を close したことによって，このドライバが ResultSet を close した場合を含みます。
- この ResultSet オブジェクトを生成した Statement を作成した Connection が close されている場合
- トランザクションの決着によって ResultSet が無効になった場合

(bv)　getHoldability()

【機能】

ResultSet オブジェクトの保持機能についての状態を示す値を取得します。

【形式】

```
public int getHoldability() throws SQLException
```

【引数】

なし。

【戻り値】

ResultSet.HOLD_CURSORS_OVER_COMMIT：

現在のトランザクションがコミットするとき，オープンしている ResultSet オブジェクトはオープンしたままになります。

ResultSet.CLOSE_CURSORS_AT_COMMIT：

現在のトランザクションがコミットするとき，オープンしている ResultSet オブジェクトはクローズされます。

【発生する例外】

次の場合，SQLException を投入します。

- この ResultSet オブジェクトがクローズされている場合

  この ResultSet オブジェクトを生成した Statement をクローズしたことによって，このドライバが ResultSet をクローズした場合を含みます。

- この ResultSet オブジェクトを生成した Statement を作成した Connection がクローズされている場合

(bw)　isClosed()

【機能】

ResultSet オブジェクトがクローズされているかどうかの情報を取得します。

【形式】

```
public boolean isClosed() throws SQLException
```

【引数】

なし。

【戻り値】

true：

ResultSet オブジェクトはクローズされています。

false：

ResultSet オブジェクトはオープンされています。

【機能詳細】

ResultSet オブジェクトがクローズされているかどうかの情報を取得します。次の場合，true が返されます。

- この ResultSet オブジェクトがクローズされている場合

  この ResultSet オブジェクトを生成した Statement をクローズしたことによって，このドライバが ResultSet をクローズした場合を含みます。

- この ResultSet オブジェクトを生成した Statement を作成した Connection がクローズされている場合

【発生する例外】

なし。

(3)　パッケージ名称及びクラス名称

このインタフェースを実装するパッケージ名称とクラス名称を次に示します。

パッケージ名称：**JP.co.Hitachi.soft.HiRDB.JDBC**

クラス名称：**PrdbResultSet**

(4)　フィールド

ResultSet インタフェースでサポートするフィールドを次の表に示します。

表 17–40　ResultSet インタフェースでサポートするフィールド

フィールド	備考
public static final int FETCH_FORWARD	–
public static final int FETCH_REVERSE	–
public static final int FETCH_UNKNOWN	–
public static final int TYPE_FORWARD_ONLY	–
public static final int TYPE_SCROLL_INSENSITIVE	–
public static final int TYPE_SCROLL_SENSITIVE	JDBC ドライバでは，この値が指定されると，TYPE_SCROLL_INSENSITIVE が指定されたものとみなします。
public static final int CONCUR_READ_ONLY	–
public static final int CONCUR_UPDATABLE	JDBC ドライバでは，この値が指定されると，CONCUR_READ_ONLY が指定されたものとみなします。
public static final int HOLD_CURSORS_OVER_COMMIT	–
public static final int CLOSE_CURSORS_AT_COMMIT	–

（凡例）

–：特にありません。

(5)　注意事項

(a)　getXXX メソッドによる値の取得

- getXXX メソッドによってマッピングできるかどうかについては，「検索データ取得時のマッピング」を参照してください。
- getXXX メソッドに指定した列番号又は列名称が存在しない場合は，SQLException を投入します。

- getXXX メソッドに指定する値が，実際の値を表現できない場合（例：INTEGER 型の 40,000 という値を getShort で取得する場合），オーバフローによって SQLException を投入します。オーバフローが発生する可能性のある getXXX メソッドと HiRDB のデータ型の組み合わせについては，「オーバフローの扱い」を参照してください。

(b)　マッピング（変換）

検索データ取得時に使用する getXXX メソッドによってマッピングできるかどうかについては，「検索データ取得時のマッピング」を参照してください。マッピングできない JDBC SQL タイプに対して getXXX メソッドが呼び出された場合，SQLException を投入します。

(c)　setFetchSize メソッドの指定によるブロック転送機能の利用

「setFetchSize メソッドの指定によるブロック転送機能の利用」を参照してください。

(d)　結果セットタイプが ResultSet.TYPE_SCROLL_INSENSITIVE，又は ResultSet.TYPE_SCROLL_SENSITIVE である場合のメモリ使用量

結果セットタイプが ResultSet.TYPE_SCROLL_INSENSITIVE，又は ResultSet.TYPE_SCROLL_SENSITIVE である場合，ResultSet インタフェースの次のメソッドを実行するとき，検索結果蓄積用のメモリを JDBC ドライバが確保します。

- ResultSet.next メソッド
- ResultSet.last メソッド
- ResultSet.absolute メソッド
- ResultSet.relative メソッド
- ResultSet.afterLast メソッド

JDBC ドライバは，検索結果中のすべての値ごとにメモリオブジェクトを割り当てて蓄積します。値が可変長である場合，メモリオブジェクトは検索データの実サイズに合わせた大きさとなります。

(e)　next，absolute，relative，last，及び afterLast メソッド

next メソッドを実行すると，JDBC ドライバは次の表に示すようにデータベースからデータを取得，蓄積します。

表 17–41　next メソッド実行時の，データベースからのデータの取得，蓄積

状態	結果セットタイプ	
	TYPE_FORWARD_ONLY	TYPE_SCROLL_INSENSITIVE，又は TYPE_SCROLL_SENSITIVE
next メソッドで移動した現在行のデータを JDBC ドライバ内に読み込んでいない。	遷移した現在行を接続先のデータベースから取得します。	遷移した現在行を接続先のデータベースから取得し，JDBC ドライバのメモリに読み込み，蓄積します。

状態	結果セットタイプ	
	TYPE_FORWARD_ONLY	TYPE_SCROLL_INSENSITIVE，又は TYPE_SCROLL_SENSITIVE
next メソッドで移動した現在行のデータを JDBC ドライバ内に読み込んでいる。		接続先のデータベースからデータを取得しません。

absolute，relative，last，及び afterLast メソッドを実行すると，JDBC ドライバは次の表に示すようにデータベースからデータを取得，蓄積します。

表 17–42　absolute，relative，last，及び afterLast メソッド実行時の，データベースからのデータの取得，蓄積

状態	結果セットタイプが TYPE_SCROLL_INSENSITIVE，又は TYPE_SCROLL_SENSITIVE
検索結果の先頭行から指定行※までに，JDBC ドライバが読み込んでいないデータがある。	読み込んでいない行を接続先のデータベースから取得し，JDBC ドライバのメモリに蓄積します。
検索結果の先頭行から指定行※までに，JDBC ドライバが読み込んでいないデータはない。	接続先のデータベースからデータを取得しません。

注

結果セットタイプが TYPE_FORWARD_ONLY の場合，SQLException を投入します。

注※

last メソッド及び afterLast メソッドの場合，先頭行から最終行です。

(f)　getAsciiStream，getBinaryStream，getCharacterStream，及び getUnicodeStream メソッド

getAsciiStream，getBinaryStream，getCharacterStream，及び getUnicodeStream メソッドが返却したオブジェクトは，JDBC ドライバが暗黙的にクローズすることはありません。メソッド呼び出し側で close メソッドを実行してください。

(g)　検索結果行数

ResultSet オブジェクトが HiRDB サーバから取得できる検索結果の行数を次の表に示します。次の表に示す行以上の検索結果は，JDBC ドライバが破棄します。

表 17–43　ResultSet オブジェクトが HiRDB サーバから取得できる検索結果の行数

ResultSet オブジェクト	結果セットタイプ	
	TYPE_SCROLL_INSENSITIVE，又は TYPE_SCROLL_SENSITIVE	左記以外
setMaxRows メソッドを実行した Statement オブジェクトから生成した ResultSet オブジェクト	検索結果行数は，setMaxRows メソッドで指定した行数です。	
上記以外の ResultSet オブジェクト	検索結果行数は，setMaxRows メソッドの上限（2,147,483,647）です。	上限はありません。

(h)　ナル既定値について

ナル値の既定値設定機能を使用している場合（クライアント環境定義 PDDFLNVAL に USE を指定），検索結果がナル既定値のときの，HiRDB データ型での getXXXX メソッドの戻り値を次の表に示します。

ナル値の既定値設定機能については，マニュアル「HiRDB SQL リファレンス」を参照してください。

wasNull メソッドでは，ナル値の既定値設定機能を使用している場合，戻り値に FALSE を設定します。

なお，DatabaseMetaData クラスのメソッドで取得した ResultSet オブジェクトには，ナル値の既定値設定機能は適用されません。ナル値の既定値設定機能を使用している場合でも，getXXXX メソッドの戻り値に NULL を設定します。

表 17–44　検索結果が NULL 既定値の場合の HiRDB のデータ型での getXXX メソッドの戻り値（1/2）

getXXX メソッド	HiRDB データ型				
	[M ｜ N][VAR]CHAR	BINARY BLOB	DATE	TIME	TIMESTAMP
getByte	×	×	×	×	×
getShort	×	×	×	×	×
getInt	×	×	×	×	×
getLong	×	×	×	×	×
getFloat	×	×	×	×	×
getDouble	×	×	×	×	×
getBigDecimal	×	×	×	×	×
getBoolean	FALSE	×	×	×	×
getString	NULL 規定値	NULL 既定値の String オブジェクト	NULL 既定値を変換した yyyy-MM-DD 形	NULL 既定値を変換した hh:mm:ss 形	NULL 既定値を変換した yyyy-MM-DD△hh:mm:ss[.ffffff]形

getXXX メソッド	HiRDB データ型				
	[M | N][VAR]CHAR	BINARY BLOB	DATE	TIME	TIMESTAMP
			式の文字列の String オブジェクト	式の文字列の String オブジェクト	式の文字列の String オブジェクト
getBytes	NULL 規定値を byte 配列にしたもの	同左	×	×	×
getDate	×	×	NULL 規定値を java.sql.Date オブジェクトにしたもの	×	NULL 規定値を java.sql.Date オブジェクトにしたもの
getTime	×	×	×	NULL 既定値を java.sql.Time オブジェクトにしたもの	同左
getTimestamp	×	×	NULL 既定値を java.sql.Timestamp オブジェクトにしたもの	×	NULL 既定値を java.sql.Timestamp オブジェクトにしたもの
getAsciiStream	NULL 既定値を格納した InputStream オブジェクト	同左	×	×	×
getBinaryStream	×	NULL 既定値を格納した InputStream オブジェクト	×	×	×
getObject	NULL 規定値	同左	NULL 既定値で生成した java.sql.Date オブジェクト	NULL 既定値で生成した java.sql.Time オブジェクト	NULL 既定値で生成した java.sql.Timestamp オブジェクト
getCharacterStream	NULL 既定値を格納した Reader オブジェクト	同左	×	×	×
getArray	×	×	×	×	×
getBlob	×	NULL 既定値の値を持つ java.sql.Blob オブジェクト	×	×	×

（凡例） ×：SQLException を投入します。

表 17–45　検索結果が NULL 既定値の場合の HiRDB のデータ型での getXXX メソッドの戻り値（2/2）

getXXX メソッド	JDBC SQL タイプ					
	SMALLINT	INTEGER	REAL	FLOAT	DECIMAL	ARRAY
getByte	NULL 既定値を byte 値にしたもの	同左	同左	同左	同左	×
getShort	NULL 既定値	NULL 規定値を short 値にしたもの	NULL 規定値の整数部分の値を short 値にしたもの	同左	同左	×
getInt	NULL 規定値を int 値にしたもの	同左	NULL 規定値の整数部分の値を int 値にしたもの	同左	同左	×
getLong	NULL 規定値を long 値にしたもの	同左	NULL 規定値の整数部分の値を long 値にしたもの	同左	同左	×
getFloat	NULL 規定値を float 値にしたもの	同左	同左	同左	同左	×
getDouble	NULL 規定値を double 値にしたもの	同左	同左	同左	同左	×
getBigDecimal	NULL 規定値の値を持つ BigDecimal オブジェクト	同左	同左	同左	同左	×
getBoolean	FALSE	同左	同左	同左	同左	×
getString	NULL 既定値を文字列表現にした String オブジェクト	同左	同左	同左	同左	×
getBytes	×	×	×	×	×	×
getDate	×	×	×	×	×	×
getTime	×	×	×	×	×	×
getTimestamp	×	×	×	×	×	×
getAsciiStream	×	×	×	×	×	×
getBinaryStream	×	×	×	×	×	×

getXXX メソッド	JDBC SQL タイプ					
	SMALLINT	INTEGER	REAL	FLOAT	DECIMAL	ARRAY
getObject	NULL 既定値で生成した Integer オブジェクト	同左	NULL 既定値で生成した Float オブジェクト	NULL 既定値で生成した Double オブジェクト	NULL 規定値	NULL 既定値で生成した Array オブジェクト
getCharacterStream	×	×	×	×	×	×
getArray	×	×	×	×	×	要素数 0 の java.sql.Array オブジェクト
getBlob	×	×	×	×	×	×

（凡例） ×：SQLException を投入します。

17.4.7 DatabaseMetaData インタフェース

(1) 概要

DatabaseMetaData インタフェースでは，主に次の機能が提供されます。

- 接続先の DB に関する各種情報の返却
- 表一覧，列一覧などの一覧情報の返却（ResultSet（結果セット）に格納）

DatabaseMetaData クラスのメソッドには，String のパターン文字列を引数とするものがあります。このパターン文字列中に指定できる特殊文字を次に示します。

特殊文字	説明
_（下線）	任意の 1 文字です。
%	0 文字以上の任意の長さの文字列です。
¥	エスケープ文字です。パターン文字列中に記述したエスケープ文字の直後の特殊文字を通常の文字として扱います。

(2) メソッド

DatabaseMetaData インタフェースのメソッド一覧を次の表に示します。なお，表に記載されていないメソッドはサポートしていません。サポートしていないメソッドを指定すると，SQLException を投入します。

表 17–46　DatabaseMetaData インタフェースのメソッド一覧

メソッド	提供ドライバ			機能
	JDBC2.0	JDBC4.0	JDBC4.3	
allProceduresAreCallable()	○	○	○	getProcedures()メソッドによって返されるすべてのプロシジャが，現在のユーザから呼び出せるかどうかを返します。
allTablesAreSelectable()	○	○	○	getTables()メソッドによって返されるすべてのテーブルが，現在のユーザによって使用できるかどうかを返します。
dataDefinitionCausesTransactionCommit()	○	○	○	トランザクションのデータ定義文が，トランザクションを強制的にコミットさせるかどうかを返します。
dataDefinitionIgnoredInTransactions()	○	○	○	トランザクションでデータ定義文が無視されるかどうかを返します。
deletesAreDetected(int type)	○	○	○	ResultSet.rowDeleted()メソッドを呼び出すことによって，可視の行が削除されたことを検出できるかどうかを返します。
doesMaxRowSizeIncludeBlobs()	○	○	○	getMaxRowSize()メソッドに，SQL データ型の LONGVARCHAR 及び LONGVARBINARY を含んでいるかどうか返します。
getAttributes(String catalog,String schemaPattern,String typeNamePattern,String attributeNamePattern)	○	○	○	指定されたスキーマ，及びカタログで使用できるユーザ定義の型のための，属性に関する記述を返します。
getBestRowIdentifier(String catalog, String schema, String table, int scope, boolean nullable)	○	○	○	行を一意に識別するテーブルの，最適な列セットに関する記述を返します。
getCatalogs()	○	○	○	使用できるカタログ名を返します。
getCatalogSeparator()	○	○	○	カタログ名とテーブル名のセパレータを返します。
getCatalogTerm()	○	○	○	"catalog"に対する推奨用語を返します。
getColumnPrivileges(String catalog,String schema,String table,String columnNamePattern)	○	○	○	テーブルの列へのアクセス権に関する記述を返します。
getColumns(String catalog, String schemaPattern, String	○	○	○	指定されたテーブル列の記述を返します。※1

メソッド	提供ドライバ			機能
	JDBC2.0	JDBC4.0	JDBC4.3	
tableNamePattern, String columnNamePattern)				
getConnection()	○	○	○	この DatabaseMetaData インスタンスを生成した Connection インスタンスを返します。
getCrossReference (String primaryCatalog,String primarySchema,String primaryTable,String foreignCatalog,String foreignSchema,String foreignTable)	○	○	○	指定された参照テーブルと指定された被参照テーブルとのクロスリファレンス情報を返します。
getDatabaseMajorVersion()	○	○	○	データベースのメジャーバージョンを返します。
getDatabaseMinorVersion()	○	○	○	データベースのマイナーバージョンを返します。
getDatabaseProductName()	○	○	○	接続データベースの製品名称を返します。
getDatabaseProductVersion()	○	○	○	接続データベースのバージョンを返します。
getDefaultTransactionIsolation()	○	○	○	デフォルトのトランザクション遮断レベルを返します。
getDriverMajorVersion()	○	○	○	この JDBC ドライバのメジャーバージョンを int 型で返します。
getDriverMinorVersion()	○	○	○	この JDBC ドライバのマイナーバージョンを int 型で返します。
getDriverName()	○	○	○	JDBC ドライバの名前を返します。
getDriverVersion()	○	○	○	この JDBC ドライバのバージョンを String として返します。
getExportedKeys (String catalog,String schema,String table)	○	○	○	指定されたテーブルの外部キーに関する情報を返します。
getExtraNameCharacters()	○	○	○	引用符で囲まれていない SQL 識別名に使用できるすべての特殊文字（a-z，A-Z，0-9，及び_以外）を返します。
getIdentifierQuoteString()	○	○	○	SQL 識別子を引用するために使用する文字列を返します。

メソッド	提供ドライバ			機能
	JDBC2.0	JDBC4.0	JDBC4.3	
getImportedKeys (String catalog,String schema,String table)	○	○	○	指定されたテーブルの主キーに関する情報を返します。
getIndexInfo(String catalog, String schema, String table, boolean unique, boolean approximate)	○	○	○	指定されたテーブルのインデクスに関する記述を返します。
getJDBCMajorVersion()	○	○	○	ドライバの JDBC メジャーバージョンを返します。
getJDBCMinorVersion()	○	○	○	ドライバの JDBC マイナーバージョンを返します。
getMaxBinaryLiteralLength()	○	○	○	バイナリリテラル中に入れられる 16 進数の最大文字数を返します。
getMaxCatalogNameLength()	○	○	○	カタログ名の最大文字数を返します。
getMaxCharLiteralLength()	○	○	○	キャラクタリテラルの最大文字数を返します。
getMaxColumnNameLength()	○	○	○	列名の最大文字数を返します。
getMaxColumnsInGroupBy()	○	○	○	GROUP BY 節中の列数の最大値を返します。
getMaxColumnsInIndex()	○	○	○	インデクス中で許される列数の最大値を返します。
getMaxColumnsInOrderBy()	○	○	○	ORDER BY 節中の列数の最大値を返します。
getMaxColumnsInSelect()	○	○	○	SELECT リスト中の列数の最大値を返します。
getMaxColumnsInTable()	○	○	○	テーブル中の列数の最大値を返します。
getMaxConnections()	○	○	○	現在の接続の最大数を返します。
getMaxCursorNameLength()	○	○	○	カーソル名の最大文字数を返します。
getMaxIndexLength()	○	○	○	インデクスの全部分を含む，インデクスの最大長を返します。
getMaxProcedureNameLength()	○	○	○	プロシジャ名の最大文字数を返します。
getMaxRowSize()	○	○	○	1 行の最大バイト数を返します。

メソッド	提供ドライバ			機能
	JDBC2.0	JDBC4.0	JDBC4.3	
getMaxSchemaNameLength()	○	○	○	スキーマ名の最大文字数を返します。
getMaxStatementLength()	○	○	○	SQL 文の最大長を返します。
getMaxStatements()	○	○	○	アクティブにできる SQL 文の最大数を返します。
getMaxTableNameLength()	○	○	○	テーブル名の最大文字数を返します。
getMaxTablesInSelect()	○	○	○	SELECT 文中の最大テーブル数を返します。
getMaxUserNameLength()	○	○	○	ユーザ名の最大文字数を返します。
getNumericFunctions()	○	○	○	数学関数を，コンマで区切ったリストで返します。
getPrimaryKeys(String catalog, String schema, String table)	○	○	○	指定されたテーブルの主キー列の記述を返します。
getProcedureColumns(String catalog,String schemaPattern,String procedureNamePattern, String columnNamePattern)	○	○	○	ストアドプロシジャパラメタに関する記述を返します。※2
getProcedures(String catalog,String schemaPattern,String procedureNamePattern)	○	○	○	ストアドプロシジャに関する記述を返します。※3
getProcedureTerm()	○	○	○	"procedure"に対する推奨用語を返します。
getResultSetHoldability()	○	○	○	ResultSet オブジェクトの保持機能を返します。
getSchemas()	○	○	○	使用できるスキーマ名に関する情報を返します。※4
getSchemaTerm()	○	○	○	"schema"に対する推奨用語を返します。
getSearchStringEscape()	○	○	○	ワイルドカード文字をエスケープするために使用する文字列を返します。
getSQLKeywords()	○	○	○	データベース固有の SQL キーワードであり，かつ SQL92 のキーワードではないすべてのキーワードを，コンマで区切ったリストで返します。

メソッド	提供ドライバ			機能
	JDBC2.0	JDBC4.0	JDBC4.3	
getSQLStateType()	○	○	○	SQLException.getSQLState によって返される SQLSTATE が，X/Open の SQL CLI であるか SQL99 であるかを返します。
getStringFunctions()	○	○	○	文字列関数を，コンマで区切ったリストで返します。
getSuperTables(String catalog,String schemaPattern,String tableNamePattern)	○	○	○	特定のスキーマで定義されているテーブル階層の説明を返します。
getSuperTypes(String catalog,String schemaPattern,String typeNamePattern)	○	○	○	特定のスキーマで定義されているユーザ定義型階層の説明を返します。
getSystemFunctions()	○	○	○	使用できるシステム関数を返します。
getTablePrivileges(String catalog,String schemaPattern,String tableNamePattern)	○	○	○	テーブルのアクセス権に関する記述を返します。
getTables(String catalog, String schemaPattern, String tableNamePattern, String[] types)	○	○	○	テーブルに関する記述を返します。
getTableTypes()	○	○	○	使用できるテーブルの型を返します。
getTimeDateFunctions()	○	○	○	使用できる時間関数と日付関数をコンマで区切ったリストを返します。
getTypeInfo()	○	○	○	標準 SQL の型に関する記述を返します。
getUDTs(String catalog,String schemaPattern,String typeNamePattern,int[] types)	○	○	○	ユーザ定義型に関する情報を返します。
getURL()	○	○	○	HiRDB 又は XDM/RD E2 に接続した URL を返します。
getUserName()	○	○	○	HiRDB 又は XDM/RD E2 に接続する際に使用したユーザ名を返します。
getVersionColumns(String catalog,String schema,String table)	○	○	○	テーブル中の行が更新された場合に，自動的に更新されるテーブルの列に関する記述を返します。

メソッド	提供ドライバ			機能
	JDBC2.0	JDBC4.0	JDBC4.3	
insertsAreDetected(int type)	○	○	○	ResultSet.rowInserted()メソッドを呼び出すことによって，可視の行が挿入されたことを検出できるかどうかを返します。
isCatalogAtStart()	○	○	○	完全修飾されたテーブル名の開始部分（又は終了部分）に，カタログが現れるかどうかを返します。
isReadOnly()	○	○	○	データベースが読み込み専用モードかどうかを返します。
locatorsUpdateCopy()	○	○	○	LOB への変更が，コピーに対して行われたのか，LOB に直接行われたのかを示します。
nullPlusNonNullIsNull()	○	○	○	NULL 値と非 NULL 値の連結を，NULL とするかどうかを返します。
nullsAreSortedAtEnd()	○	○	○	NULL 値が，終了時にソート順に関係なくソートされるかどうかを返します。
nullsAreSortedAtStart()	○	○	○	NULL 値が，開始時にソート順に関係なくソートされるかどうかを返します。
nullsAreSortedHigh()	○	○	○	NULL 値が高位にソートされるかどうかを返します。
nullsAreSortedLow()	○	○	○	NULL 値が下位にソートされるかどうかを返します。
othersDeletesAreVisible(int type)	○	○	○	ほかで行われた削除が可視かどうかを返します。
othersInsertsAreVisible(int type)	○	○	○	ほかで行われた挿入が可視かどうかを返します。
othersUpdatesAreVisible(int type)	○	○	○	ほかで行われた削除が可視かどうかを返します。
ownDeletesAreVisible(int type)	○	○	○	結果セット自身の削除が可視かどうかを返します。
ownInsertsAreVisible(int type)	○	○	○	結果セット自身の挿入が可視かどうかを返します。
ownUpdatesAreVisible(int type)	○	○	○	結果セット自身の更新が可視かどうかを返します。
storesLowerCaseIdentifiers()	○	○	○	大文字と小文字が混在する引用符なしの SQL 識別子を，大文字と小文字を区別しないで処理し，小文字で格納するかどうかを返します。
storesLowerCaseQuotedIdentifiers()	○	○	○	大文字と小文字が混在する引用符付きの SQL 識別子を，大文字と小文字を区別しな

メソッド	提供ドライバ			機能
	JDBC2.0	JDBC4.0	JDBC4.3	
				いで処理し，小文字で格納するかどうかを返します。
storesMixedCaseIdentifiers()	○	○	○	大文字と小文字が混在する引用符なしのSQL 識別子を，大文字と小文字を区別しないで処理し，大文字と小文字を混在して格納するかどうかを返します。
storesMixedCaseQuotedIdentifiers()	○	○	○	大文字と小文字が混在する引用符付きのSQL 識別子を，大文字と小文字を区別しないで処理し，大文字と小文字を混在して格納するかどうかを返します。
storesUpperCaseIdentifiers()	○	○	○	大文字と小文字が混在する引用符なしのSQL 識別子を，大文字と小文字を区別しないで処理し，大文字で格納するかどうかを返します。
storesUpperCaseQuotedIdentifiers()	○	○	○	大文字と小文字が混在する引用符付きのSQL 識別子を，大文字と小文字を区別しないで処理し，大文字で格納するかどうかを返します。
supportsAlterTableWithAddColumn()	○	○	○	追加列のある ALTER TABLE が，サポートされているかどうかを返します。
supportsAlterTableWithDropColumn()	○	○	○	ドロップ列のある ALTER TABLE が，サポートされているかどうかを返します。
supportsANSI92EntryLevelSQL()	○	○	○	ANSI92 エントリレベルの SQL 文法が，サポートされるかどうかを返します。
supportsANSI92FullSQL()	○	○	○	ANSI92 完全レベルの SQL 文法が，サポートされるかどうかを返します。
supportsANSI92IntermediateSQL()	○	○	○	ANSI92 中間レベルの SQL 文法が，サポートされるかどうかを返します。
supportsBatchUpdates()	○	○	○	バッチ更新をサポートしているかどうかを返します。
supportsCatalogsInDataManipulation()	○	○	○	データ操作文でカタログ名を使用できるかどうかを返します。
supportsCatalogsInIndexDefinitions()	○	○	○	インデクス定義文でカタログ名を使用できるかどうかを返します。
supportsCatalogsInPrivilegeDefinitions()	○	○	○	特権定義文でカタログ名を使用できるかどうかを返します。
supportsCatalogsInProcedureCalls()	○	○	○	プロシジャ呼び出し文でカタログ名を使用できるかどうかを返します。

メソッド	提供ドライバ			機能
	JDBC2.0	JDBC4.0	JDBC4.3	
supportsCatalogsInTableDefinitions()	○	○	○	テーブル定義文でカタログ名を使用できるかどうかを返します。
supportsColumnAliasing()	○	○	○	カラムの別名をサポートしているかどうかを返します。
supportsConvert()	○	○	○	SQL タイプ間の CONVERT 関数をサポートしているかどうかを返します。
supportsConvert(int fromType, int toType)	○	○	○	与えられた SQL タイプ間の CONVERT 関数をサポートしているかどうかを返します。
supportsCoreSQLGrammar()	○	○	○	ODBC Core SQL 文法をサポートしているかどうかを返します。
supportsCorrelatedSubqueries()	○	○	○	照合関連サブクエリーをサポートしているかどうかを返します。
supportsDataDefinitionAndDataManipulationTransactions()	○	○	○	トランザクションで，データ定義文とデータ操作文の両方をサポートしているかどうかを返します。
supportsDataManipulationTransactionsOnly()	○	○	○	トランザクションで，データ操作文だけをサポートしているかどうかを返します。
supportsDifferentTableCorrelationNames()	○	○	○	テーブル相互関連名がサポートされている場合，テーブルの名前と異なる名前にするという制限があるかどうかを返します。
supportsExpressionsInOrderBy()	○	○	○	"ORDER BY"リスト中の式をサポートしているかどうかを返します。
supportsExtendedSQLGrammar()	○	○	○	ODBC Extended SQL 文法をサポートしているかどうかを返します。
supportsFullOuterJoins()	○	○	○	完全ネストの外部結合をサポートしているかどうかを返します。
supportsGetGeneratedKeys()	○	○	○	文が実行された後に自動生成キーを取得できるかどうかを返します。
supportsGroupBy()	○	○	○	"GROUP BY"節のフォームをサポートしているかどうかを返します。
supportsGroupByBeyondSelect()	○	○	○	SELECT 中のすべてのカラムを指定するという条件で，"GROUP BY"節が SELECT 中にないカラムを使用できるかどうかを返します。
supportsGroupByUnrelated()	○	○	○	"GROUP BY"節が，SELECT 中にないカラムを使用できるかどうかを返します。
supportsIntegrityEnhancementFacility()	○	○	○	SQL Integrity Enhancement Facility をサポートしているかどうかを返します。

メソッド	提供ドライバ			機能
	JDBC2.0	JDBC4.0	JDBC4.3	
supportsLikeEscapeClause()	○	○	○	"LIKE"節で，エスケープ文字をサポートしているかどうかを返します。
supportsLimitedOuterJoins()	○	○	○	外部結合に関し，制限されたサポートがあるかどうかを返します。
supportsMinimumSQLGrammar()	○	○	○	ODBC Minimum SQL 文法をサポートしているかどうかを返します。
supportsMixedCaseIdentifiers()	○	○	○	大文字と小文字が混在する引用符なしの SQL 識別子を，大文字と小文字を区別して処理し，結果として大文字と小文字を混在して格納するかどうかを返します。
supportsMixedCaseQuotedIdentifiers()	○	○	○	大文字と小文字が混在する引用符付きの SQL 識別子を，大文字と小文字を区別して処理し，結果として大文字と小文字を混在して格納するかどうかを返します。
supportsMultipleOpenResults()	○	○	○	CallableStatement オブジェクトから同時に返された複数の ResultSet オブジェクトを持つことが可能かどうかを返します。
supportsMultipleResultSets()	○	○	○	単一の execute メソッド実行からの複数 ResultSet をサポートしているかどうかを返します。
supportsMultipleTransactions()	○	○	○	一度に複数のトランザクションを（異なった接続に関し）オープンできるかどうかを返します。
supportsNamedParameters()	○	○	○	callable 文への名前付きパラメータがサポートされるかどうかを返します。
supportsNonNullableColumns()	○	○	○	列を非 null として定義できるかどうかを返します。
supportsOpenCursorsAcrossCommit()	○	○	○	カーソルをコミット間でオープンされた状態のままにできるかどうかを返します。
supportsOpenCursorsAcrossRollback()	○	○	○	カーソルをロールバック間でオープンされた状態のままにできるかどうかを返します。
supportsOpenStatementsAcrossCommit()	○	○	○	文をコミット間でオープンされた状態のままにできるかどうかを返します。
supportsOpenStatementsAcrossRollback()	○	○	○	文をロールバック間でオープンされた状態のままにできるかどうかを返します。
supportsOrderByUnrelated()	○	○	○	"ORDER BY"節が，SELECT 中にない列を使用できるかどうかを返します。
supportsOuterJoins()	○	○	○	何らかの外部結合のフォームをサポートしているかどうかを返します。

メソッド	提供ドライバ			機能
	JDBC2.0	JDBC4.0	JDBC4.3	
supportsPositionedDelete()	○	○	○	位置決めされた DELETE をサポートしているかどうかを返します。
supportsPositionedUpdate()	○	○	○	位置決めされた UPDATE をサポートしているかどうかを返します。
supportsResultSetConcurrency(int type, int concurrency)	○	○	○	指定された ResultSet タイプと並行処理タイプの組み合わせをサポートしているかどうかを返します。
supportsResultSetHoldability(int holdability)	○	○	○	指定された ResultSet オブジェクトの保持機能を，サポートしているかどうかを返します。
supportsResultSetType(int type)	○	○	○	指定された ResultSet タイプをサポートしているかどうかを返します。
supportsSavepoints()	○	○	○	セーブポイントをサポートしているかどうかを返します。
supportsSchemasInDataManipulation()	○	○	○	データ操作文でスキーマ名を使用できるかどうかを返します。
supportsSchemasInIndexDefinitions()	○	○	○	インデクス定義文でスキーマ名を使用できるかどうかを返します。
supportsSchemasInPrivilegeDefinitions()	○	○	○	特権定義文でスキーマ名を使用できるかどうかを返します。
supportsSchemasInProcedureCalls()	○	○	○	プロシジャ呼び出し文でスキーマ名を使用できるかどうかを返します。
supportsSchemasInTableDefinitions()	○	○	○	テーブル定義文でスキーマ名を使用できるかどうかを返します。
supportsSelectForUpdate()	○	○	○	UPDATE 用の SELECT をサポートしているかどうかを返します。
supportsStatementPooling()	○	○	○	文のプールをサポートしているかどうかを返します。
supportsStoredProcedures()	○	○	○	ストアドプロシジャコールをサポートしているかどうかを返します。
supportsSubqueriesInComparisons()	○	○	○	比較式中でサブクエリーをサポートしているかどうかを返します。
supportsSubqueriesInExists()	○	○	○	"exists"式中でサブクエリーをサポートしているかどうかを返します。
supportsSubqueriesInIns()	○	○	○	"in"文中でサブクエリーをサポートしているかどうかを返します。

メソッド	提供ドライバ			機能
	JDBC2.0	JDBC4.0	JDBC4.3	
supportsSubqueriesInQuantifieds()	○	○	○	定量化された式中でサブクエリーをサポートしているかどうかを返します。
supportsTableCorrelationNames()	○	○	○	テーブル相互関連名をサポートしているかどうかを返します。
supportsTransactionIsolationLevel(int level)	○	○	○	与えられたトランザクションアイソレーションレベルをサポートしているかどうかを返します。
supportsTransactions()	○	○	○	トランザクションをサポートしているかどうかを返します。
supportsUnion()	○	○	○	SQL UNION をサポートしているかどうかを返します。
supportsUnionAll()	○	○	○	SQL UNION ALL をサポートしているかどうかを返します。
updatesAreDetected(int type)	○	○	○	指定された ResultSet タイプで，ResultSet に行われた更新を ResultSet.rowUpdated メソッドによって検出できるかどうかを返します。
usesLocalFilePerTable()	○	○	○	各テーブルにファイルを使用するかどうかを返します。
usesLocalFiles()	○	○	○	ローカルファイルにテーブルを格納するかどうかを返します。
getRowIdLifetime()	×	○	○	RowId 型をサポートしているかどうかを返します。RowId 型をサポートしている場合は，RowId オブジェクトが有効になっている寿命を返します。
getSchemas(String catalog, String schemaPattern)	×	○	○	使用できるスキーマ名に関する情報を返します。
supportsStoredFunctionsUsingCallSyntax()	×	○	○	プロシジャエスケープ構文を使用した，ユーザ定義関数又はベンダー関数の呼び出しをサポートするかどうかを返します。
autoCommitFailureClosesAllResultSets()	×	○	○	オートコミット有効時に SQLException が発生した場合，オープンされたすべての ResultSet がクローズされるかどうかを返します。
getClientInfoProperties()	×	○	○	JDBC ドライバがサポートするクライアント情報プロパティのリスト情報を取得します。

メソッド	提供ドライバ			機能
	JDBC2.0	JDBC4.0	JDBC4.3	
getFunctions(String catalog, String schemaPattern, String functionNamePattern)	×	○	○	関数に関する情報を返します。
getFunctionColumns(String catalog,String schemaPattern,String functionNamePattern,String columnNamePattern)	×	○	○	関数のパラメタと返される型に関する情報を返します。
getPseudoColumns(String catalog, String schemaPattern, String tableNamePattern, String columnNamePattern)	×	×	○	擬似列，隠し列の説明を返します。
generatedKeyAlwaysReturned()	×	×	○	自動生成キーの列を表す指定された列名又はインデックスが有効であり，SQL 文が成功した場合に生成されたキーが常に返されるかどうかを返します。
getMaxLogicalLobSize()	×	×	○	このデータベースで LOB の論理サイズとして許容される最大バイト数を返します。
supportsRefCursors()	×	×	○	REF CURSOR のサポート有無を返します。
supportsSharding()	×	×	○	シャーディングのサポート有無を返します。

（凡例）

○：提供されます。

×：提供されません。

注※1

JDBC2.0 と JDBC4.0 以降とでは，次のような差異があります。

- 返却される ResultSet の列数が JDBC4.0 以降では JDBC2.0 よりも多い

注※2

JDBC2.0 と JDBC4.0 以降とでは，次のような差異があります。

- 返却される ResultSet の列数が JDBC4.0 以降では JDBC2.0 よりも多い
- 返却される PRECISION 及び LENGTH 列の値が異なる
- SQL_ROUTINES 表，又は SQL_ROUTINE_PARAMS 表が存在しないためディクショナリ検索でエラーになった場合，JDBC2.0 では SQLException が返却されるが，JDBC4.0 以降では検索結果行数 0 の ResultSet が返却される

- JDBC2.0 ではプロシジャ及び関数のパラメタ情報が返却されるが，JDBC4.0 以降ではプロシジャのパラメタ情報だけが返却される

注※3

JDBC2.0 と JDBC4.0 以降では，次のような差異があります。

- 返却される ResultSet の列数が JDBC4.0 以降では JDBC2.0 よりも多い
- SQL_ROUTINES 表，又は SQL_ROUTINE_PARAMS 表が存在しないためディクショナリ検索でエラーになった場合，JDBC2.0 では SQLException が返却されるが，JDBC4.0 以降では検索結果行数 0 の ResultSet が返却される
- JDBC2.0 ではプロシジャ及び関数のパラメタ情報が返却されるが，JDBC4.0 以降ではプロシジャのパラメタ情報だけが返却される

注※4

JDBC2.0 と JDBC4.0 以降では，次のような差異があります。

- JDBC2.0 では表を所有するスキーマが返却されるが，JDBC4.0 以降ではすべてのスキーマが返却される

(a) allProceduresAreCallable()

【機能】

getProcedures()メソッドによって返されるすべてのプロシジャが，現在のユーザから呼び出せるかどうかを返します。

【形式】

```
public boolean allProceduresAreCallable() throws SQLException
```

【引数】

なし。

【戻り値】

boolean 型：

true：呼び出せます。

false：呼び出せません。

【機能詳細】

常に false を返します。

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(b) allTablesAreSelectable()

【機能】

getTables()メソッドによって返されるすべてのテーブルが，現在のユーザによって使用できるかどうかを返します。

【形式】

```
public boolean allTablesAreSelectable() throws SQLException
```

【引数】

なし。

【戻り値】

boolean 型：

true：使用できます。

false：使用できません。

【機能詳細】

常に false を返します。

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(c) dataDefinitionCausesTransactionCommit()

【機能】

トランザクションのデータ定義文が，トランザクションを強制的にコミットさせるかどうかを返します。

【形式】

```
public boolean dataDefinitionCausesTransactionCommit() throws SQLException
```

【引数】

なし。

【戻り値】

boolean 型：

true：コミットさせます。

false：コミットさせません。

【機能詳細】

常に true を返します。

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(d) dataDefinitionIgnoredInTransactions()

【機能】

トランザクションでデータ定義文が無視されるかどうかを返します。

【形式】

```
public boolean dataDefinitionIgnoredInTransactions() throws SQLException
```

【引数】

なし。

【戻り値】

boolean 型：

true：無視されます。

false：無視されません。

【機能詳細】

常に false を返します。

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(e) deletesAreDetected(int type)

【機能】

ResultSet.rowDeleted()メソッドを呼び出すことによって，可視の行が削除されたことを検出できるかどうかを返します。

【形式】

```
public boolean deletesAreDetected(int type) throws SQLException
```

【引数】

int type：

ResultSet の型。次のどれかです。

- ResultSet.TYPE_FORWARD_ONLY
- ResultSet.TYPE_SCROLL_INSENSITIVE
- ResultSet.TYPE_SCROLL_SENSITIVE

【戻り値】

boolean 型：

true：削除が検出されます。

false：削除が検出されません。

【機能詳細】

常に false を返します。

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(f) doesMaxRowSizeIncludeBlobs()

【機能】

getMaxRowSize()メソッドに，SQL データ型の LONGVARCHAR 及び LONGVARBINARY を含んでいるかどうか返します。

【形式】

```
public boolean doesMaxRowSizeIncludeBlobs() throws SQLException
```

【引数】

なし。

【戻り値】

boolean 型：

true：含んでいます。

false：含んでいません。

【機能詳細】

常に false を返します。

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(g) getAttributes(String catalog,String schemaPattern,String typeNamePattern,String attributeNamePattern)

【機能】

指定されたスキーマ，及びカタログで使用できるユーザ定義の型のための，属性に関する記述を返します。

【形式】

```
public ResultSet getAttributes(String catalog, String schemaPattern, String typeNamePatte
rn, String attributeNamePattern) throws SQLException
```

【引数】

String catalog：

カタログ名

String schemaPattern：

スキーマ名パターン

String typeNamePattern：

型名パターン

String attributeNamePattern：

属性名パターン

【戻り値】

ResultSet オブジェクト

【機能詳細】

Type4 JDBC ドライバでは，ユーザ定義の型は未サポートのため，常に検索結果行数 0 の ResultSet を返します。すべての引数について妥当性チェックを行いません。返却する ResultSet の形式を次に示します。

列番号	型	SQL 型（Types）	列名	列の意味
1	String	CHAR	TYPE_CAT	カタログ名
2	String	VARCHAR	TYPE_SCHEM	認可識別子名
3	String	VARCHAR	TYPE_NAME	型名
4	String	VARCHAR	ATTR_NAME	属性名
5	int	INTEGER	DATA_TYPE	属性の型
6	String	VARCHAR	ATTR_TYPE_NAME	型名
7	int	INTEGER	ATTR_SIZE	列サイズ
8	int	INTEGER	DECIMAL_DIGITS	小数点以下のけた数
9	int	INTEGER	NUM_PREC_RADIX	基数
10	int	INTEGER	NULLABLE	NULL 値の可否
11	String	VARCHAR	REMARKS	コメント
12	String	VARCHAR	ATTR_DEF	デフォルト値
13	int	INTEGER	SQL_DATA_TYPE	未使用
14	int	INTEGER	SQL_DATETIME_SUB	未使用
15	int	INTEGER	CHAR_OCTET_LENGTH	char 型列の最大バイト数
16	int	INTEGER	ORDINAL_POSITION	テーブル中の列のインデクス
17	String	VARCHAR	IS_NULLABLE	NULL 値の可否
18	String	VARCHAR	SCOPE_CATALOG	参照属性のスコープであるテーブルのカタログ

列番号	型	SQL 型（Types）	列名	列の意味
19	String	VARCHAR	SCOPE_SCHEMA	参照属性のスコープであるテーブルのスキーマ
20	String	VARCHAR	SCOPE_TABLE	参照属性のスコープであるテーブル名
21	short	SMALLINT	SOURCE_DATA_TYPE	個別の型又はユーザ生成 Ref 型，java.sql.Types の SQL 型のソースの型

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(h) getBestRowIdentifier(String catalog, String schema, String table, int scope, boolean nullable)

【機能】

行を一意に識別するテーブルの，最適な列セットに関する記述を返します。

【形式】

```
public ResultSet getBestRowIdentifier(String catalog, String schema, String table, int sc
ope, boolean nullable) throws SQLException
```

【引数】

String catalog：

カタログ名

String schema：

スキーマ名

String table：

テーブル名

int scope：

対象のスケール

boolean nullable：

NULL 値指定

【戻り値】

ResultSet オブジェクト

【機能詳細】

すべての引数について妥当性チェックを行いません。常に検索結果行数 0 の ResultSet を返します。

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(i) getCatalogs()

【機能】

使用できるカタログ名を返します。

【形式】

```
public ResultSet getCatalogs() throws SQLException
```

【引数】

なし。

【戻り値】

ResultSet オブジェクト

【機能詳細】

常に検索結果行数 0 の ResultSet を返します。

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(j) getCatalogSeparator()

【機能】

カタログ名とテーブル名のセパレータを返します。

【形式】

```
public String getCatalogSeparator() throws SQLException
```

【引数】

なし。

【戻り値】

String オブジェクト

【機能詳細】

null を返します。

システムプロパティ HiRDB_for_Java_METADATA_BEHAVIOR に CAT_SEP_EMPTY が指定されている場合，空文字("")を返します。

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(k) getCatalogTerm()

【機能】

"catalog"に対する推奨用語を返します。

【形式】

```
public String getCatalogTerm() throws SQLException
```

【引数】

なし。

【戻り値】

String オブジェクト

【機能詳細】

常に null を返します。

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(l) getColumnPrivileges (String catalog,String schema,String table,String columnNamePattern)

【機能】

テーブルの列へのアクセス権に関する記述を返します。

【形式】

```
public ResultSet getColumnPrivileges(String catalog, String schema, String table, String
columnNamePattern) throws SQLException
```

【引数】

String catalog：

カタログ名。このドライバでは無視します。

String schema：

スキーマ名パターン。null 又は 0 長文字列を指定している場合，スキーマ名を検索の限定に使用しません。引数の値は大文字と小文字を区別します。

String table：

テーブル名パターン。null 又は 0 長文字列を指定している場合，テーブル名を検索の限定に使用しません。引数の値は大文字と小文字を区別します。

String columnNamePattern：

列名パターン。null 又は 0 長文字列を指定している場合，列名を検索の限定に使用しません。引数の値は大文字と小文字を区別します。

【戻り値】

ResultSet オブジェクト

【機能詳細】

指定されたテーブル列が属する表のアクセス権に関する記述を返します。返却する ResultSet の形式を次に示します。

列番号	型	SQL 型（Types）	列名	ソート※	列の意味
1	String	CHAR	TABLE_CAT	−	カタログ名 常に NULL 値を返します。
2	String	VARCHAR	TABLE_SCHEM	1（昇順）	認可識別子名
3	String	VARCHAR	TABLE_NAME	2（昇順）	テーブル名
4	String	VARCHAR	COLUMN_NAME	3（昇順）	列名
5	String	VARCHAR	GRANTOR	−	アクセス権の付与者
6	String	VARCHAR	GRANTEE	−	アクセス権の被付与者
7	String	VARCHAR	PRIVILEGE	4（昇順）	許可されているアクセス権限名。複数のアクセス権限を許可されている場合は，コンマ（,）で区切って返します。 SELECT：SELECT 権限 INSERT：INSERT 権限 UPDATE：UPDATE 権限 DELETE：DELETE 権限
8	String	VARCHAR	IS_GRANTABLE	−	アクセス権の被付与者が，別のユーザにアクセス権を与えることができるかを表します。 YES：与えることができます。 NO：与えることができません。 null：与えることができるかどうか不明です。 PRIVILEGE に設定されたアクセス権限が複数ある場合，その内の一つ以上のアクセス権限を別のユーザに与えることができれば，YES を設定します。HiRDB，XDM/RD E2 共に，null は返しません。

（凡例）

−：該当しません。

注※

ソートのキー列の優先順位を示します。

【発生する例外】

次のどちらかの場合，例外 SQLException を投入します。

- このメソッド実行前に Connection オブジェクトに対して close()を実行した場合
- データベースアクセスエラーが発生した場合

(m) getColumns(String catalog, String schemaPattern, String tableNamePattern, String columnNamePattern)

【機能】

指定されたテーブル列の記述を返します。

【形式】

```
public ResultSet getColumns(String catalog, String schemaPattern, String tableNamePattern
, String columnNamePattern) throws SQLException
```

【引数】

String catalog：

カタログ名（このドライバでは無視します）

String schemaPattern：

スキーマ名パターン（大文字と小文字を区別します）

String tableNamePattern：

テーブル名パターン（大文字と小文字を区別します）

String columnNamePattern：

列名パターン（大文字と小文字を区別します）

【戻り値】

ResultSet オブジェクト

【機能詳細】

指定されたテーブル列の記述を返します。返却する ResultSet の形式を次の表に示します。

表 17–47　getColumns で返却する ResultSet の形式

列番号	型	SQL 型（Types）	列名	列の意味	ドライバが返却する列	
					JDBC2.0	JDBC4.0 以降
1	String	CHAR	TABLE_CAT	常に NULL 値を返します。	○	○
2	String	VARCHAR	TABLE_SCHEM	認可識別子名	○	○
3	String	VARCHAR	TABLE_NAME	テーブル名	○	○

列番号	型	SQL 型（Types）	列名	列の意味	ドライバが返却する列	
					JDBC2.0	JDBC4.0 以降
4	String	VARCHAR	COLUMN_NAME	列名	○	○
5	int	INTEGER	DATA_TYPE	SQL の型	○	○
6	String	CHAR	TYPE_NAME	型名	○	○
7	int	INTEGER	COLUMN_SIZE	列サイズ	○	○
8	int	INTEGER	BUFFER_LENGTH	常に NULL 値を返します。	○	○
9	int	INTEGER	DECIMAL_DIGITS	小数点以下のけた数	○	○
10	int	INTEGER	NUM_PREC_RADIX	基数 • 概数値：2 • 真数値：10 • 数値以外：0	○	○
11	int	INTEGER	NULLABLE	この型に NULL 値を使用できるかどうかを返します。 • columnNoNulls：NULL 値を使用できない可能性があります。 • columnNullable：NULL 値を使用できます。 • columnNullableUnknown：NULL 値を使用できるか不明です。	○	○
12	String	VARCHAR	REMARKS	コメント記述列	○	○
13	String	VARCHAR	COLUMN_DEF	デフォルト値	○	○
14	int	INTEGER	SQL_DATA_TYPE	常に NULL 値を返します。	○	○
15	int	INTEGER	SQL_DATETIME_SUB	常に NULL 値を返します。	○	○
16	int	INTEGER	CHAR_OCTET_LENGTH	char の型についての列の最大バイト数 COLUMN_SIZE と同じです。	○	○
17	int	SMALLINT	ORDINAL_POSITION	列番号 1 から始まります。	○	○
18	String	CHAR	IS_NULLABLE	この型に NULL 値が使用できるかどうかを返します。 • "NO"：NULL 値を使用できません。	○	○

列番号	型	SQL 型（Types）	列名	列の意味	ドライバが返却する列	
					JDBC2.0	JDBC4.0 以降
				• "YES"：NULL 値を使用できる可能性があります。		
19	String	CHAR	SCOPE_CATLOG	常に NULL 値を返します。	○	○
20	String	CHAR	SCOPE_SCHEMA	常に NULL 値を返します。	○	○
21	String	CHAR	SCOPE_TABLE	常に NULL 値を返します。	○	○
22	short	SMALLINT	SOURCE_DATA_TYPE	常に NULL 値を返します。	○	○
23	String	VARCHAR	IS_AUTOINCREMENT	列が自動インクリメントされるかどうかを示します。 • "YES"：列が自動インクリメントされます。 • "NO"：列が自動インクリメントされません。 • 空白：列が自動インクリメントされるか不明です。 常に NO を返します。	×	○

（凡例）

○：返却されます。

×：返却されません。

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(n) getConnection()

【機能】

この DatabaseMetaData インスタンスを生成した Connection インスタンスを返します。

【形式】

```
public Connection getConnection() throws SQLException
```

【引数】

なし。

【戻り値】

Connenction オブジェクト

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(o) getCrossReference (String primaryCatalog,String primarySchema,String primaryTable,String foreignCatalog,String foreignSchema,String foreignTable)

【機能】

指定された参照テーブルと指定された被参照テーブルとのクロスリファレンス情報を返します。

【形式】

```
public ResultSet getCrossReference(String primaryCatalog, String primarySchema, String pr
imaryTable, String foreignCatalog, String foreignSchema, String foreignTable) throws SQLE
xception
```

【引数】

String primaryCatalog：

被参照テーブルのカタログ名。このドライバでは無視します。

String primarySchema：

被参照テーブルのスキーマ名パターン。null 又は 0 長文字列を指定している場合，スキーマ名を検索の限定に使用しません。引数の値は大文字と小文字を区別します。

String primaryTable：

被参照テーブルのテーブル名パターン。null 又は 0 長文字列を指定している場合，テーブル名を検索の限定に使用しません。引数の値は大文字と小文字を区別します。

String foreignCatalog：

参照テーブルのカタログ名。このドライバでは無視します。

String foreignSchema：

参照テーブルのスキーマ名パターン。null 又は 0 長文字列を指定している場合，スキーマ名を検索の限定に使用しません。引数の値は大文字と小文字を区別します。

String foreignTable：

参照テーブルのテーブル名パターン。null 又は 0 長文字列を指定している場合，テーブル名を検索の限定に使用しません。引数の値は大文字小文字を区別します。

【戻り値】

ResultSet オブジェクト

【機能詳細】

指定された参照テーブルと指定された被参照テーブルとのクロスリファレンス情報を返します。返却する ResultSet の形式を次に示します。

列番号	型	SQL 型（Types）	列名	ソート※	列の意味
1	String	CHAR	PKTABLE_CAT	−	被参照テーブルのカタログ名。 常に NULL 値を返します。
2	String	VARCHAR	PKTABLE_SCHEM	−	被参照テーブルの認可識別子名。
3	String	VARCHAR	PKTABLE_NAME	−	被参照テーブルのテーブル名。
4	String	VARCHAR	PKCOLUMN_NAME	−	主キーの列名。
5	String	CHAR	FKTABLE_CAT	−	参照テーブルのカタログ名。 常に NULL 値を返します。
6	String	VARCHAR	FKTABLE_SCHEM	1（昇順）	参照テーブルの認可識別子名。
7	String	VARCHAR	FKTABLE_NAME	2（昇順）	参照テーブルのテーブル名。
8	String	VARCHAR	FKCOLUMN_NAME	−	外部キーの列名。
9	short	SMALLINT	KEY_SEQ	3（昇順）	外部キーのシーケンス番号。
10	short	SMALLINT	UPDATE_RULE	−	更新規則。 主キーを参照している外部キーが存在する場合に，主キーを更新すると，それを参照している外部キーがどのような影響を受けるのかを表します。 • importedKeyNoAction 主キーは更新できません。 • importedKeyCascade 主キーと同じ値で，外部キーを更新します。 • importedKeySetNull 外部キーの値が NULL になります。 • importedKeySetDefault 外部キーの値がデフォルト値になります。 • importedKeyRestrict 主キーは更新できません。
11	short	SMALLINT	DELETE_RULE	−	削除規則。 主キーを参照している外部キーが存在する場合に，主キーを削除すると，それを参照している外部キーがどのような影響を受けるのかを表します。 • importedKeyNoAction 主キーを削除できません。 • importedKeyCascade 外部キーがある行を，削除します。 • importedKeySetNull 外部キーの値が NULL になります。 • importedKeySetDefault

列番号	型	SQL 型（Types）	列名	ソート※	列の意味
					外部キーの値がデフォルト値になります。 • importedKeyRestrict 主キーを削除できません。
12	String	VARCHAR	FK_NAME	−	参照制約の制約名。
13	String	VARCHAR	PK_NAME	−	主キーのインデクス名。
14	short	SMALLINT	DEFERRABILITY	−	外部キーに対する制約の評価を，トランザクションのコミット時まで延期できるかを表します。 • importedKeyInitiallyDeferred 延期できます。 • importedKeyInitiallyImmediate 現時点では即時に評価されますが，延期できるように変更できます。 • importedKeyNotDeferrable 延期できません。

（凡例）

−：該当しません。

注※

ソートのキー列の優先順位を示します。

【発生する例外】

次のどちらかの場合，例外 SQLException を投入します。

- このメソッド実行前に Connection オブジェクトに対して close()を実行した場合
- データベースアクセスエラーが発生した場合

(p) getDatabaseMajorVersion()

【機能】

データベースのメジャーバージョンを返します。

【形式】

```
public int getDatabaseMajorVersion() throws SQLException
```

【引数】

なし。

【戻り値】

int 型：

データベースのメジャーバージョン

【機能詳細】

サーバが HiRDB の場合，HiRDB サーバのメジャーバージョンを返します。例えば，HiRDB サーバのバージョンが 08-02 の場合，int 型の 8 を返します。

サーバが XDM/RD E2 の場合，XDM/RD E2 のメジャーバージョンに，20 を加算した値を返します。この返却値は，JDBC ドライバが出力する SQL トレースの，ヘッダ部に表示されるサーバのメジャーバージョンと同じです。例えば，XDM/RD E2 のバージョンが 11-03 の場合，int 型の 31 を返します。

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(q) getDatabaseMinorVersion()

【機能】

データベースのマイナーバージョンを返します。

【形式】

```
public int getDatabaseMinorVersion() throws SQLException
```

【引数】

なし。

【戻り値】

int 型：

データベースのマイナーバージョン

【機能詳細】

サーバが HiRDB の場合，HiRDB サーバのマイナーバージョンを返します。例えば，HiRDB サーバのバージョンが 08-02 の場合，int 型の 2 を返します。

サーバが XDM/RD E2 の場合，XDM/RD E2 のマイナーバージョンを返します。例えば，XDM/RD E2 のバージョンが 11-03 の場合，int 型の 3 を返します。

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(r) getDatabaseProductName()

【機能】

接続データベースの製品名称を返します。

サーバが HiRDB の場合，"HiRDB"を返します。サーバが XDM/RD E2 の場合，"XDM/RD E2"を返します。

【形式】

```
public String getDatabaseProductName() throws SQLException
```

【引数】

なし。

【戻り値】

String オブジェクト

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(s) getDatabaseProductVersion()

【機能】

接続データベースのバージョンを返します。

【形式】

```
public String getDatabaseProductVersion() throws SQLException
```

【引数】

なし。

【戻り値】

String オブジェクト

【機能詳細】

サーバが HiRDB の場合，"vv-rr"（例："08-00"）の形式で HiRDB のバージョンを返します。

サーバが XDM/RD E2 の場合，XDM/RD E2 のバージョンに 20 を加算した値を，"vv-rr"の形式で返します。例えば，XDM/RD E2 のバージョンが 11-03 の場合，"31-03"を返します。

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(t) getDefaultTransactionIsolation()

【機能】

デフォルトのトランザクション遮断レベルを返します。

【形式】

```
public int getDefaultTransactionIsolation() throws SQLException
```

【引数】

なし。

【戻り値】

int 型：

遮断レベル

【機能詳細】

常に TRANSACTION_REPEATABLE_READ を返します。

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(u) getDriverMajorVersion()

【機能】

この JDBC ドライバのメジャーバージョンを int 型で返します。例えば，バージョンが 08-00 の場合，int 型の 8 を返します。

【形式】

```
public int getDriverMajorVersion() throws SQLException
```

【引数】

なし。

【戻り値】

int 型：

JDBC ドライバのメジャーバージョン

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(v) getDriverMinorVersion()

【機能】

この JDBC ドライバのマイナーバージョンを int 型で返します。例えば，バージョンが 08-00 の場合，int 型の 0 を返します。

【形式】

```
public int getDriverMinorVersion() throws SQLException
```

【引数】

なし。

【戻り値】

int 型：

JDBC ドライバのマイナーバージョン

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(w) getDriverName()

【機能】

JDBC ドライバの名前を返します。

【形式】

```
public String getDriverName() throws SQLException
```

【引数】

なし。

【戻り値】

String オブジェクト

【機能詳細】

JDBC ドライバの名前として，次のどれかを返却します。

- JDBC2.0：HiRDB_Type4_JDBC_Driver
- JDBC4.0：HiRDB_Type4_JDBC40_Driver
- JDBC4.3：HiRDB_Type4_JDBC43_Driver

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(x) getDriverVersion()

【機能】

この JDBC ドライバのバージョンを String として返します。例えば，バージョンが 08-00 の場合，String 型の 08-00 を返します。

【形式】

```
public String getDriverVersion() throws SQLException
```

【引数】

なし。

【戻り値】

String オブジェクト

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(y) getExportedKeys (String catalog,String schema,String table)

【機能】

主キー及び外部キーに関する情報を返します。

【形式】

```
public ResultSet getExportedKeys(String catalog,String schema,String table) throws SQLExc
eption
```

【引数】

String catalog：

参照テーブルのカタログ名。このドライバでは無視します。

String schema：

参照テーブルのスキーマ名パターン。null 又は 0 長文字列を指定している場合，スキーマ名を検索の限定に使用しません。引数の値は大文字と小文字を区別します。

String table：

参照テーブルのテーブル名パターン。null 又は 0 長文字列を指定している場合，テーブル名を検索の限定に使用しません。引数の値は大文字と小文字を区別します。

【戻り値】

ResultSet オブジェクト

【機能詳細】

指定されたテーブルの主キー及び当該主キーを参照する外部キーに関する情報を返します。

システムプロパティ HiRDB_for_Java_Compatible に ALL が指定されている場合，指定されたテーブルの外部キー及び当該外部キーが参照する主キーに関する情報を返します。

返却する ResultSet の形式を次に示します。

列番号	型	SQL 型（Types）	列名	ソート※	列の意味
1	String	CHAR	PKTABLE_CAT	−	被参照テーブルのカタログ名。 常に NULL 値を返します。
2	String	VARCHAR	PKTABLE_SCHEM	−	被参照テーブルの認可識別子名。
3	String	VARCHAR	PKTABLE_NAME	−	被参照テーブルのテーブル名。

列番号	型	SQL 型（Types）	列名	ソート※	列の意味
4	String	VARCHAR	PKCOLUMN_NAME	−	主キーの列名。
5	String	CHAR	FKTABLE_CAT	−	参照テーブルのカタログ名。常に NULL 値を返します。
6	String	VARCHAR	FKTABLE_SCHEM	1（昇順）	参照テーブルの認可識別子名。
7	String	VARCHAR	FKTABLE_NAME	2（昇順）	参照テーブルのテーブル名。
8	String	VARCHAR	FKCOLUMN_NAME	−	外部キーの列名。
9	short	SMALLINT	KEY_SEQ	3（昇順）	外部キーのシーケンス番号。
10	short	SMALLINT	UPDATE_RULE	−	更新規則。 主キーを参照している外部キーが存在する場合に，主キーを更新すると，それを参照している外部キーがどのような影響を受けるのかを表します。 • importedKeyNoAction 主キーは更新できません。 • importedKeyCascade 主キーと同じ値で，外部キーを更新します。 • importedKeySetNull 外部キーの値が NULL になります。 • importedKeySetDefault 外部キーの値がデフォルト値になります。 • importedKeyRestrict 主キーは更新できません。
11	short	SMALLINT	DELETE_RULE	−	削除規則。 主キーを参照している外部キーが存在する場合に，主キーを削除すると，それを参照している外部キーがどのような影響を受けるのかを表します。 • importedKeyNoAction 主キーを削除できません。 • importedKeyCascade 外部キーがある行を削除します。 • importedKeySetNull 外部キーの値が NULL になります。 • importedKeySetDefault 外部キーの値がデフォルト値になります。 • importedKeyRestrict 主キーを削除できません。
12	String	VARCHAR	FK_NAME	−	参照制約の制約名。

列番号	型	SQL 型（Types）	列名	ソート※	列の意味
13	String	VARCHAR	PK_NAME	−	主キーのインデクス名。
14	short	SMALLINT	DEFERRABILITY	−	外部キーに対する制約の評価を，トランザクションのコミット時まで延期できるかを表します。 • importedKeyInitiallyDeferred 延期できます。 • importedKeyInitiallyImmediate 現時点では即時に評価されますが，延期できるように変更できます。 • importedKeyNotDeferrable 延期できません。

（凡例）

−：該当しません。

注※

ソートのキー列の優先順位を示します。

【発生する例外】

次のどちらかの場合，例外 SQLException を投入します。

- このメソッド実行前に Connection オブジェクトに対して close()を実行した場合
- データベースアクセスエラーが発生した場合

(z) getExtraNameCharacters()

【機能】

引用符で囲まれていない SQL 識別名に使用できる特殊文字を返します。ただし，a-z，A-Z，0-9，及び_は除きます。

【形式】

```
public String getExtraNameCharacters() throws SQLException
```

【引数】

なし。

【戻り値】

String オブジェクト

【機能詳細】

常に¥，@，及び#を返します。

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(aa)　getIdentifierQuoteString()

【機能】

SQL 識別子を引用するために使用する文字列を返します。

【形式】

```
public String getIdentifierQuoteString() throws SQLException
```

【引数】

なし。

【戻り値】

String オブジェクト

【機能詳細】

常に二重引用符（"）を返します。

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(ab)　getImportedKeys (String catalog,String schema,String table)

【機能】

主キー及び外部キーに関する情報を返します。

【形式】

```
public ResultSet getImportedKeys(String catalog,String schema,String table) throws SQLExc
eption
```

【引数】

String catalog：

被参照テーブルのカタログ名。このドライバでは無視します。

String schema：

被参照テーブルのスキーマ名パターン。null 又は 0 長文字列を指定している場合，スキーマ名を検索の限定に使用しません。引数の値は大文字と小文字を区別します。

String table：

被参照テーブルのテーブル名パターン。null 又は 0 長文字列を指定している場合，テーブル名を検索の限定に使用しません。引数の値は大文字と小文字を区別します。

【戻り値】

ResultSet オブジェクト

【機能詳細】

指定されたテーブルの外部キー及び当該外部キーが参照する主キーに関する情報を返します。

システムプロパティ HiRDB_for_Java_Compatible に ALL が指定されている場合，指定されたテーブルの主キー及び当該主キーを参照する外部キーに関する情報を返します。

返却する ResultSet の形式を次に示します。

列番号	型	SQL 型（Types）	列名	ソート※	列の意味
1	String	CHAR	PKTABLE_CAT	−	被参照テーブルのカタログ名。 常に NULL 値を返します。
2	String	VARCHAR	PKTABLE_SCHEM	1（昇順）	被参照テーブルの認可識別子名。
3	String	VARCHAR	PKTABLE_NAME	2（昇順）	被参照テーブルのテーブル名。
4	String	VARCHAR	PKCOLUMN_NAME	−	主キーの列名。
5	String	CHAR	FKTABLE_CAT	−	参照テーブルのカタログ名。 常に NULL 値を返します。
6	String	VARCHAR	FKTABLE_SCHEM	−	参照テーブルの認可識別子名。
7	String	VARCHAR	FKTABLE_NAME	−	参照テーブルのテーブル名。
8	String	VARCHAR	FKCOLUMN_NAME	−	外部キーの列名。
9	short	SMALLINT	KEY_SEQ	3（昇順）	外部キーのシーケンス番号。
10	short	SMALLINT	UPDATE_RULE	−	更新規則。 主キーを参照している外部キーが存在する場合に，主キーを更新すると，それを参照している外部キーがどのような影響を受けるのかを表します。 • importedKeyNoAction 主キーは更新できません。 • importedKeyCascade 主キーと同じ値で，外部キーを更新します。 • importedKeySetNull 外部キーの値が NULL になります。 • importedKeySetDefault 外部キーの値がデフォルト値になります。 • importedKeyRestrict 主キーは更新できません。
11	short	SMALLINT	DELETE_RULE	−	削除規則。 主キーを参照している外部キーが存在する場合に，主キーを削除すると，それを参照している外部キーがどのような影響を受けるのかを表します。 • importedKeyNoAction 主キーを削除できません。 • importedKeyCascade

列番号	型	SQL 型（Types）	列名	ソート※	列の意味
					外部キーがある行を削除します。 • importedKeySetNull 外部キーの値が NULL になります。 • importedKeySetDefault 外部キーの値がデフォルト値になります。 • importedKeyRestrict 主キーを削除できません。
12	String	VARCHAR	FK_NAME	−	参照制約の制約名。
13	String	VARCHAR	PK_NAME	−	主キーのインデクス名。
14	short	SMALLINT	DEFERRABILITY	−	外部キーに対する制約の評価を，トランザクションのコミット時まで延期できるかを表します。 • importedKeyInitiallyDeferred 延期できます。 • importedKeyInitiallyImmediate 現時点では即時に評価されるが，延期できるように変更できます。 • importedKeyNotDeferrable 延期できません。

（凡例）

−：該当しません。

注※

ソートのキー列の優先順位を示します。

【発生する例外】

次のどちらかの場合，例外 SQLException を投入します。

- このメソッド実行前に Connection オブジェクトに対して close()を実行した場合
- データベースアクセスエラーが発生した場合

(ac) getIndexInfo(String catalog, String schema, String table, boolean unique, boolean approximate)

【機能】

指定されたテーブルのインデクスに関する記述を返します。

【形式】

```
public ResultSet getIndexInfo(String catalog, String schema, String table, boolean unique
, boolean approximate) throws SQLException
```

【引数】

String catalog：

カタログ名。この指定値は未使用です。

String schema：

スキーマ名パターン。引数の値は大文字と小文字を区別します。

String table：

テーブル名パターン。引数の値は大文字と小文字を区別します。

boolean unique：

ユニーク属性。

true：ユニークインデクスだけを返します。

false：ユニークであるかどうかに関係なく，インデクス情報を返します。

boolean approximate：

結果属性。この指定値は未使用です。

【戻り値】

ResultSet オブジェクト

【機能詳細】

指定されたテーブルのインデクスに関する記述を返します。返却する ResultSet の形式を次の表に示します。

表 17–48　getIndexInfo で返却する ResultSet の形式

列番号	型	SQL 型	列名	列の意味
1	String	CHAR	TABLE_CAT	カタログ名 常に NULL 値を返します。
2	String	VARCHAR	TABLE_SCHEM	認可識別子名
3	String	VARCHAR	TABLE_NAME	テーブル名
4	boolean	BIT	NON_UNIQUE	インデクスを定義するキー値（インデクスとして定義する一つ又は複数の列全体の値）が，すべての行で異なっている場合に false，そうでない場合に true を返します。
5	String	CHAR	INDEX_QUALIFIER	インデクスのカタログ名 常に NULL 値を返します。
6	String	VARCHAR	INDEX_NAME	インデクス識別子
7	short	SMALLINT	TYPE	インデクス種別によって次の値を返します。 • クラスタキーの場合：DatabaseMetaData.tableIndexClustered(1) • ハッシュインデクスの場合：DatabaseMetaData.tableIndexHashed(2)

列番号	型	SQL 型	列名	列の意味
				• その他の場合： DatabaseMetaData.tableIndexOther(3)
8	short	SMALLINT	ORDINAL_POSITION	単一列インデクスの場合，1 を返します。複数列インデクスの場合，インデクスを構成する列の順序（インデクスを構成する列名順を識別する番号で，1 から始まる整数）を返します。
9	String	VARCHAR	COLUMN_NAME	列名
10	String	VARCHAR	ASC_OR_DESC	インデクスを昇順に定義する場合，"A"を返します。 インデクスを降順に定義する場合，"D"を返します。 プラグインインデクスの場合，NULL 値を返します。
11	int	INTEGER	CARDINALITY	インデクス内のユニークな値の数 常に NULL 値を返します。
12	int	INTEGER	PAGES	インデクスで使用しているページ数 常に NULL 値を返します。
13	String	CHAR	FILTER_CONDITION	フィルタ条件 常に NULL 値を返します。

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(ad)　getJDBCMajorVersion()

【機能】

ドライバの JDBC メジャーバージョンを返します。

【形式】

```
public int getJDBCMajorVersion() throws SQLException
```

【引数】

なし。

【戻り値】

int 型：

JDBC メジャーバージョン

【機能詳細】

ドライバの JDBC メジャーバージョンを返します。2 を返します。

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(ae)　getJDBCMinorVersion()

【機能】

ドライバの JDBC マイナーバージョンを返します。

【形式】

```
public int getJDBCMinorVersion () throws SQLException
```

【引数】

なし。

【戻り値】

int 型：

JDBC マイナーバージョン

【機能詳細】

ドライバの JDBC マイナーバージョンを返します。1 を返します。

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(af)　getMaxBinaryLiteralLength()

【機能】

バイナリリテラル中に入れられる 16 進数の最大文字数を返します。

【形式】

```
public int getMaxBinaryLiteralLength() throws SQLException
```

【引数】

なし。

【戻り値】

int 型：

最大文字数

【機能詳細】

常に 64000 を返します。

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(ag)　getMaxCatalogNameLength()

【機能】

カタログ名の最大文字数を返します。

【形式】

```
public int getMaxCatalogNameLength() throws SQLException
```

【引数】

なし。

【戻り値】

int 型：

カタログ名に許可される最大文字数。0 は制限がないか，制限が不明であることを示します。

【機能詳細】

常に 0 を返します。

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(ah)　getMaxCharLiteralLength()

【機能】

キャラクタリテラルの最大文字数を返します。

【形式】

```
public int getMaxCharLiteralLength() throws SQLException
```

【引数】

なし。

【戻り値】

int 型：

キャラクタリテラルに許可される最大文字数。0 は制限がないか，制限が不明であることを示します。

【機能詳細】

常に 32000 を返します。

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(ai)　getMaxColumnNameLength()

【機能】

列名の最大文字数を返します。

【形式】

```
public int getMaxColumnNameLength() throws SQLException
```

【引数】

なし。

【戻り値】

int 型：

列名に許可される最大文字数

【機能詳細】

常に 30 を返します。

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(aj)　getMaxColumnsInGroupBy()

【機能】

GROUP BY 節中の列数の最大値を返します。

【形式】

```
public int getMaxColumnsInGroupBy() throws SQLException
```

【引数】

なし。

【戻り値】

int 型：

許可される列の最大数

【機能詳細】

常に 255 を返します。

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(ak)　getMaxColumnsInIndex()

【機能】

インデクス中で許される列数の最大値を返します。

【形式】

```
public int getMaxColumnsInIndex() throws SQLException
```

【引数】

なし。

【戻り値】

int 型：

許可される列の最大数

【機能詳細】

インデクス中で許される列数の最大値を返します。返す値はクライアント及び接続先サーバの組み合わせによって異なります。クライアント及び接続先サーバの組み合わせによる返却値を次の表に示します。内部ドライバの場合，16 を返します。

表 17–49　クライアント及び接続先サーバの組み合わせによる返却値

クライアント	接続先サーバ		
	HiRDB		XDM/RD E2
	バージョン 09-50 より前	バージョン 09-50 以降	
バージョン 09-50 より前	16	16	16
バージョン 09-50 以降	16	64	16

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(al)　getMaxColumnsInOrderBy()

【機能】

ORDER BY 節中の列数の最大値を返します。

【形式】

```
public int getMaxColumnsInOrderBy() throws SQLException
```

【引数】

なし。

【戻り値】

int 型：

許可される列の最大数

【機能詳細】

常に 255 を返します。

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(am)　getMaxColumnsInSelect()

【機能】

SELECT リスト中の列数の最大値を返します。

【形式】

```
public int getMaxColumnsInSelect() throws SQLException
```

【引数】

なし。

【戻り値】

int 型：

許可される列の最大数

【機能詳細】

常に 30000 を返します。

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(an)　getMaxColumnsInTable()

【機能】

テーブル中の列数の最大値を返します。

【形式】

```
public int getMaxColumnsInTable() throws SQLException
```

【引数】

なし。

【戻り値】

int 型：

許可される列の最大数

【機能詳細】

常に 30000 を返します。

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(ao)　getMaxConnections()

【機能】

現在の接続の最大数を返します。

【形式】

```
public int getMaxConnections() throws SQLException
```

【引数】

なし。

【戻り値】

int 型：

一度に接続できるアクティブな接続の最大数。0 は制限がないか，制限が不明を示します。

【機能詳細】

常に 0 を返します。

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(ap)　getMaxCursorNameLength()

【機能】

カーソル名の最大文字数を返します。

【形式】

```
public int getMaxCursorNameLength() throws SQLException
```

【引数】

なし。

【戻り値】

int 型：

カーソル名に許可される最大文字数

【機能詳細】

常に 30 を返します。

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(aq)　getMaxIndexLength()

【機能】

インデクスの全部分を含む，インデクスの最大長を返します。

【形式】

```
public int getMaxIndexLength() throws SQLException
```

【引数】

なし。

【戻り値】

int 型：

許可されるインデクスの最大長

【機能詳細】

常に 4036 を返します。

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(ar)　getMaxProcedureNameLength()

【機能】

プロシジャ名の最大文字数を返します。

【形式】

```
public int getMaxProcedureNameLength() throws SQLException
```

【引数】

なし。

【戻り値】

int 型：

プロシジャ名の最大長

【機能詳細】

常に 30 を返します。

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(as)　getMaxRowSize()

【機能】

1 行の最大バイト数を返します。

【形式】

```
public int getMaxRowSize() throws SQLException
```

【引数】

なし。

【戻り値】

int 型：

許可される 1 行の最大バイト数。0 は制限がないか，制限が不明であることを示します。

【機能詳細】

常に 0 を返します。

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(at)　getMaxSchemaNameLength()

【機能】

スキーマ名の最大文字数を返します。

【形式】

```
public int getMaxSchemaNameLength() throws SQLException
```

【引数】

なし。

【戻り値】

int 型：

スキーマ名の最大長

【機能詳細】

スキーマ名の最大文字数を返します。返す値はクライアント及び接続先サーバの組み合わせによって異なります。クライアント及び接続先サーバの組み合わせによる返却値を次に示します。内部ドライバの場合，8 を返します。

表 17–50　クライアント及び接続先サーバの組み合わせによる返却値

クライアント	接続先サーバ		
	HiRDB		XDM/RD E2
	バージョン 09-50 より前	バージョン 09-50 以降	
バージョン 09-50 より前	8	8	30
バージョン 09-50 以降	8	30	30

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(au)　getMaxStatementLength()

【機能】

SQL 文の最大長を返します。

【形式】

```
public int getMaxStatementLength() throws SQLException
```

【引数】

なし。

【戻り値】

int 型：

SQL 文の最大長

【機能詳細】

サーバが HiRDB の場合 2,000,000 を返します。

サーバが XDM/RD E2 の場合 30,000 を返します。

注意：

サーバが XDM/RD E2 の場合，Database Connection Server のサーバ空間の起動制御文の RDB 句で MAX SQL LENGTH を指定すると，SQL 文の最大長を 2,000,000 まで拡張できます。しかし，JDBC ドライバは，MAX SQL LENGTH 指定の情報をサーバから取得していないので，このメソッドは，

MAX SQL LENGTH が指定されていない場合のデフォルト値（30,000）を返します。JDBC ドライバ自身は最大 2,000,000 バイトの SQL 文を処理できます。

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(av)　getMaxStatements()

【機能】

アクティブにできる SQL 文の最大数を返します。

【形式】

```
public int getMaxStatements() throws SQLException
```

【引数】

なし。

【戻り値】

int 型：

SQL 文の最大数

【機能詳細】

常に 4095 を返します。

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(aw)　getMaxTableNameLength()

【機能】

テーブル名の最大文字数を返します。

【形式】

```
public int getMaxTableNameLength() throws SQLException
```

【引数】

なし。

【戻り値】

int 型：

テーブル名の最大長

【機能詳細】

常に 30 を返します。

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(ax)　getMaxTablesInSelect()

【機能】

SELECT 文中の最大テーブル数を返します。

【形式】

```
public int getMaxTablesInSelect() throws SQLException
```

【引数】

なし。

【戻り値】

int 型：

SELECT 文で許可されるテーブルの最大数

【機能詳細】

クライアント及び接続先サーバの組み合わせによって，次の値を返します。内部ドライバの場合，64 を返します。

クライアント	接続先サーバ		
	HiRDB		XDM/RD E2
	バージョン 09-65-02 より前	バージョン 09-65-02 以降	
バージョン 09-65-02 より前	64	64	256
バージョン 09-65-02 以降	64	128	256

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(ay)　getMaxUserNameLength()

【機能】

ユーザ名の最大文字数を返します。

【形式】

```
public int getMaxUserNameLength() throws SQLException
```

【引数】

なし。

【戻り値】

int 型：

ユーザ名に許可される最大長

【機能詳細】

ユーザ名の最大文字数を返します。返す値はクライアント及び接続先サーバの組み合わせによって異なります。クライアント及び接続先サーバの組み合わせによる返却値を次に示します。内部ドライバの場合，8 を返します。

表 17–51　クライアント及び接続先サーバの組み合わせによる返却値

クライアント	接続先サーバ		
	HiRDB		XDM/RD E2
	バージョン 09-50 より前	バージョン 09-50 以降	
バージョン 09-50 より前	8	8	7
バージョン 09-50 以降	8	30	7

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(az)　getNumericFunctions()

【機能】

数学関数を，コンマで区切ったリストで返します。

【形式】

```
public String getNumericFunctions() throws SQLException
```

【引数】

なし。

【戻り値】

String オブジェクト

【機能詳細】

サーバが HiRDB の場合，次のリストを返します。

ABS,ACOS,ASIN,ATAN,ATAN2,CEILING,COS,DEGREES,EXP,FLOOR,LOG,LOG10,MOD,PI,POWER,RADIANS,ROUND,SIGN,SIN,SQRT,TAN,TRUNCATE

サーバが XDM/RD E2 の場合，次のリストを返します。

ABS,CEILING,EXP,FLOOR,LOG,MOD,POWER,SQRT

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(ba)　getPrimaryKeys(String catalog, String schema, String table)

【機能】

指定されたテーブルの主キー列の記述を返します。

【形式】

```
public ResultSet getPrimaryKeys(String catalog, String schema, String table) throws SQLException
```

【引数】

String catalog：

カタログ名。この指定値は未使用です。

String schema：

スキーマ名パターン。引数の値は大文字と小文字を区別します。

String table：

テーブル名パターン。引数の値は大文字と小文字を区別します。

【戻り値】

ResultSet オブジェクト

【機能詳細】

指定されたテーブルの主キー列の記述を返します。返却する ResultSet の形式を次の表に示します。

表 17–52　getPrimaryKeys で返却する ResultSet の形式

列番号	型	SQL 型	列名	列の意味
1	String	CHAR	TABLE_CAT	カタログ名 常に NULL 値を返します。
2	String	VARCHAR	TABLE_SCHEM	認可識別子名
3	String	VARCHAR	TABLE_NAME	テーブル名
4	String	VARCHAR	COLUMN_NAME	列名
5	short	SMALLINT	KEY_SEQ	インデクスを構成する列の順序（インデクスを構成する列名順を識別する番号で，1 から始まる整数）を返します。
6	String	VARCHAR	PK_NAME	プライマリインデクス識別子

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(bb) getProcedureColumns(String catalog,String schemaPattern,String procedureNamePattern, String columnNamePattern)

【機能】

ストアドプロシジャパラメタに関する記述を返します。

【形式】

```
public ResultSet getProcedureColumns(String catalog, String schemaPattern, String procedu
reNamePattern, String columnNamePattern) throws SQLException
```

【引数】

String catalog：

カタログ名。このドライバでは無視します。

String schemaPattern：

スキーマ名パターン。null 又は 0 長文字列を指定している場合，スキーマ名を検索の限定に使用しません。引数の値は大文字と小文字を区別します。

String procedureNamePattern：

プロシジャ名パターン。null 又は 0 長文字列を指定している場合，プロシジャ名を検索の限定に使用しません。引数の値は大文字と小文字を区別します。

String columnNamePattern：

パラメタ名パターン。null 又は 0 長文字列を指定している場合，パラメタ名を検索の限定に使用しません。引数の値は大文字と小文字を区別します。

【戻り値】

ResultSet オブジェクト

【機能詳細】

JDBC2.0 の場合

指定したストアドプロシジャ及び関数のパラメタに関する記述を返します。HiRDB サーバに SQL_ROUTINE_PARAMS 表が定義されていない場合，SQLException を返します。

JDBC4.0 以降の場合

指定したストアドプロシジャのパラメタに関する記述を返します。HiRDB サーバに SQL_ROUTINE_PARAMS 表が定義されていない場合，検索結果行数 0 の ResultSet を返却します。

返却する ResultSet の形式を次に示します。

列番号	型	SQL 型（Types）	列名	ソート※	列の意味	ドライバが返却する列	
						JDBC2.0	JDBC4.0 以降
1	String	CHAR	PROCEDURE_CAT	−	カタログ名 常に NULL 値を返します。	○	○

列番号	型	SQL 型（Types）	列名	ソート※	列の意味	ドライバが返却する列	
						JDBC2.0	JDBC4.0 以降
2	String	VARCHAR	PROCEDURE_SCHEM	1（昇順）	認可識別子名	○	○
3	String	VARCHAR	PROCEDURE_NAME	2（昇順）	プロシジャ名	○	○
4	String	VARCHAR	COLUMN_NAME	−	パラメタ名	○	○
5	short	SMALLINT	COLUMN_TYPE	−	パラメタの種類 • procedureColumnUnknown 不明 • procedureColumnIn IN パラメタ • procedureColumnInOut INOUT パラメタ • procedureColumnOut OUT パラメタ • procedureColumnReturn プロシジャ戻り値 • procedureColumnResult ResultSet の結果列 HiRDB，XDM/RD E2 共に，procedureColumnReturn，procedureColumnResult は設定しません。	○	○
6	int	INTEGER	DATA_TYPE	−	パラメタの SQL 型 （java.sql.Types で定義されている値）	○	○
7	String	VARCHAR	TYPE_NAME	−	パラメタの SQL 型名 （型名を文字列で表現したもの）	○	○
8	int	INTEGER	PRECISION	−	パラメタの精度	○	○

列番号	型	SQL 型（Types）	列名	ソート※	列の意味	ドライバが返却する列	
						JDBC2.0	JDBC4.0 以降
					JDBC2.0： パラメタの SQL 型名が DECIMAL 以外の場合，0 を返します。 JDBC4.0 以降： • 数値データの場合：最大精度 • 文字データの場合：文字数 • 日時データ型の場合：String 表現の文字数 • バイナリデータの場合：バイト数		
9	int	INTEGER	LENGTH	−	パラメタのサイズ JDBC2.0： NCHAR 及び NVARCHAR：文字数 DECIMAL 型：精度 INTERVAL 型：DATA_LENGTH_CODE の値 上記以外：バイト数 JDBC4.0 以降： バイト数を返します。 DECIMAL 型は精度に符号とピリオドを加えるため，位取りなしの場合は精度+1，位取りありの場合は精度+2 となります。	○	○
10	short	SMALLINT	SCALE	−	パラメタの位取り（小数部分のけた数） パラメタの SQL 型名が DECIMAL,TIMESTAMP 以外の場合は，0 を返します。	○	○
11	short	SMALLINT	RADIX	−	パラメタの基数	○	○

列番号	型	SQL 型（Types）	列名	ソート※	列の意味	ドライバが返却する列	
						JDBC2.0	JDBC4.0 以降
					• 概数値：2 • 真数値：10 • 数値以外：0		
12	short	SMALLINT	NULLABLE	−	NULL 値の可否 • procedureNoNulls NULL 値を許しません。 • procedureNullable NULL 値を許します。 • procedureNullableUnknown NULL 値を許すかどうかは不明です。 常に，procedureNullable を返します。	○	○
13	String	VARCHAR	REMARKS	−	パラメタに関するコメント 常に NULL 値を返します。	○	○
14	String	VARCHAR	COLUMN_DEF	−	列のデフォルト値 常に NULL 値を返します。	×	○
15	int	INTEGER	SQL_DATA_TYPE	−	将来予約用 0 を返します。	×	○
16	int	INTEGER	SQL_DATE_TIME	−	将来予約用 0 を返します。	×	○
17	int	INTEGER	CHAR_OCTET_LENGTH	−	バイナリと文字ベースのパラメタ又は列の最大長	×	○
18	int	INTEGER	ORDINAL_POSITION	4（昇順）	入力及び出力パラメタの１から始まる順番	×	○
19	String	CHAR	IS_NULLABLE	−	パラメタ又は列で NULL 値を許可するかどうか • YES NULL 値を許します。	×	○

列番号	型	SQL 型（Types）	列名	ソート※	列の意味	ドライバが返却する列	
						JDBC2.0	JDBC4.0 以降
					• NO NULL 値を許しません。 • 空白 NULL 値を許すかどうか不明です。 常に YES を返します。		
20	String	VARCHAR	SPECIFIC_NAME	3（昇順）	このプロシジャを一意に識別する名前	×	○

（凡例）

－：該当しません。

○：返却されます。

×：返却されません。

注※

ソートのキー列の優先順位を示します。

【発生する例外】

次のどちらかの場合，例外 SQLException を投入します。

- このメソッド実行前に Connection オブジェクトに対して close()を実行した場合
- データベースアクセスエラーが発生した場合

(bc) getProcedures(String catalog,String schemaPattern,String procedureNamePattern)

【機能】

ストアドプロシジャに関する記述を返します。

【形式】

```
public ResultSet getProcedures(String catalog, String schemaPattern, String procedureNamePattern) throws SQLException
```

【引数】

String catalog：

カタログ名。このドライバでは無視します。

String schemaPattern：

スキーマ名パターン。null 又は 0 長文字列を指定している場合，スキーマ名を検索の限定に使用しません。引数の値は大文字と小文字を区別します。

String procedureNamePattern：

プロシジャ名パターン。null 又は 0 長文字列を指定している場合，プロシジャ名を検索の限定に使用しません。引数の値は大文字と小文字を区別します。

【戻り値】

ResultSet オブジェクト

【機能詳細】

JDBC2.0 の場合：

指定されたストアドプロシジャ及び関数に関する記述を返します。HiRDB サーバに SQL_ROUTINES 表が定義されていない場合，SQLException を返します。

JDBC4.0 以降の場合：

指定されたストアドプロシジャに関する記述を返します。HiRDB サーバに SQL_ROUTINES 表が定義されていない場合，検索結果行数 0 の ResultSet を返却します。

返却する ResultSet の形式を次に示します。

列番号	型	SQL 型（Types）	列名	ソート※	列の意味	ドライバが返却する列	
						JDBC2.0	JDBC4.0 以降
1	String	CHAR	PROCEDURE_CAT	−	カタログ名 常に NULL 値を返します。	○	○
2	String	VARCHAR	PROCEDURE_SCHEM	1（昇順）	認可識別子名	○	○
3	String	VARCHAR	PROCEDURE_NAME	2（昇順）	プロシジャ名	○	○
4	String	VARCHAR	RESERVE1	−	予約 常に NULL 値を返します。	○	○
5	String	VARCHAR	RESERVE2	−	予約 常に NULL 値を返します。	○	○
6	String	VARCHAR	RESERVE3	−	予約 常に NULL 値を返します。	○	○
7	String	VARCHAR	REMARKS	−	プロシジャの説明文	○	○
8	short	SMALLINT	PROCEDURE_TYPE	−	プロシジャの種類 • procedureResultUnknown 結果を返す可能性があります。 • procedureNoResult 結果を返しません。 • procedureReturnsResult 結果を返します。	○	○

列番号	型	SQL 型（Types）	列名	ソート※	列の意味	ドライバが返却する列	
						JDBC2.0	JDBC4.0 以降
9	String	VARCHAR	SPECIFIC_NAME	3（昇順）	このプロシジャを一意に識別する名前	×	○

（凡例）

－：該当しません。

○：返却されます。

×：返却されません。

注※

ソートのキー列の優先順位を示します。

【発生する例外】

次のどちらかの場合，例外 SQLException を投入します。

- このメソッド実行前に Connection オブジェクトに対して close()を実行した場合
- データベースアクセスエラーが発生した場合

(bd)　getProcedureTerm()

【機能】

"procedure"に対する推奨用語を返します。

【形式】

```
public String getProcedureTerm() throws SQLException
```

【引数】

なし。

【戻り値】

String オブジェクト

【機能詳細】

常に"procedure"を返します。

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(be)　getResultSetHoldability()

【機能】

ResultSet オブジェクトの保持機能を返します。

【形式】

```
public int getResultSetHoldability() throws SQLException
```

【引数】

なし。

【戻り値】

ResultSet オブジェクトの保持機能。Connection.commit メソッドが呼び出されたときに ResultSet オブジェクトがクローズされるかどうかを示します。

ResultSet.HOLD_CURSORS_OVER_COMMIT：クローズされません。

ResultSet.CLOSE_CURSORS_AT_COMMIT：クローズされます。

【機能詳細】

有効な ResultSet オブジェクトの保持機能を返します。

Connection インスタンス生成時のカーソル動作モード（HIRDB_CURSOR）で，ホールダブルカーソルを有効とした場合は ResultSet.HOLD_CURSORS_OVER_COMMIT，無効とした場合は ResultSet.CLOSE_CURSORS_AT_COMMIT を返します。

カーソル動作モードの指定方法については，「ユーザプロパティ」を参照してください。

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(bf)　getSchemas()

【機能】

使用できるスキーマ名に関する情報を返します。

【形式】

```
public ResultSet getSchemas() throws SQLException
```

【引数】

なし。

【戻り値】

ResultSet オブジェクト

【機能詳細】

JDBC2.0 の場合：

表を所有するスキーマ名だけを返します。

JDBC4.0 以降の場合：

HiRDB サーバに存在するすべてのスキーマ名に関する情報を返します。

システム共通定義に pd_security_host_group を指定し，指定値に UAP が動作するクライアントマシンのホストが含まれていない場合，SQLException を投入します。

接続先が XDM/RD E2 の場合，表を所有するスキーマ名だけを返します。

返却する ResultSet の形式を次の表に示します。

表 17–53　getSchemas で返却する ResultSet の形式

列番号	型	SQL 型	列名	列の意味
1	String	VARCHAR	TABLE_SCHEM	認可識別子名
2	String	CHAR	TABLE_CATALOG	常に NULL 値を返します。

【発生する例外】

次の場合，SQLException を投入します。

- このメソッド実行前に Connection オブジェクトに対して close()を実行している
- データベースアクセスエラーが発生した
- システム共通定義に pd_security_host_group を指定し，指定値に UAP が動作するクライアントマシンのホストが含まれていない（JDBC4.0 以降の場合だけ）

(bg)　getSchemaTerm()

【機能】

"schema"に対する推奨用語を返します。

【形式】

```
public String getSchemaTerm() throws SQLException
```

【引数】

なし。

【戻り値】

String オブジェクト

【機能詳細】

常に"schema"を返します。

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(bh)　getSearchStringEscape()

【機能】

ワイルドカード文字をエスケープするために使用する文字列を返します。

【形式】

```
public String getSearchStringEscape() throws SQLException
```

【引数】

なし。

【戻り値】

String オブジェクト

【機能詳細】

常に"¥"を返します。

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(bi)　getSQLKeywords()

【機能】

データベース固有の SQL キーワードであり，かつ SQL92 のキーワードではないすべてのキーワードを，コンマで区切ったリストで返します。

【形式】

```
public String getSQLKeywords() throws SQLException
```

【引数】

なし。

【戻り値】

String オブジェクト

【機能詳細】

サーバの種類によって，返却するリストに違いがあります。

サーバが HiRDB の場合

ABS,ACCESS,AFTER,ALIAS,AMOUNT,ANDNOT,ANSI,ARRAY,ASSIGN,
ASYNC,AUTO,BASE,BEFORE,BINARY,BIT_AND_TEST,BLOB,
BOOLEAN,BREADTH,BTREE,BUFFER,BYTE,CALL,CHANGE,CLUSTER,
COLUMNS,COMMENT,COMPLETION,COMPRESSED,CONDITION,
CONFIGURATION,CONST,CONSTRUCTOR,CONTIGUOUS,
COUNT_FLOAT,CURAID,CURRENT_ROLE,CYCLE,DATA,DATABASE,DAYS,DBA,DEFER,
DEMOTING,DEPTH,DEVICE,DICTIONARY,DIGITS,DIRECT,DO,
DOUBLE_PRECISION,EACH,EDIT,ELSEIF,ENCRYPT,EQUALS,
ESTIMATED,EXCLUSIVE,EXIT,EXTERN,FILE,FIX,FIXED,FLAT,FORCE,
FREE,FUNCTION,GENERAL,GET_JAVA_STORED_ROUTINE_SOURCE,
HANDLER,HASH,HELP,HEX,HiRDB,HOURS,HUGE,IDENTIFIED,IF,
IGNORE,INDEX,INOUT,IS_USER_CONTAINED_IN_HDS_GROUP,LARGE,

LEAVE,LENGTH,LESS,LIMIT,LINES,LINK,LIST,LOCATOR,LOCK,
LOCKS,LOGID,LOGNAME,LONG,LOOP,MASTER,MAXUSAGES,MCHAR,
MINUTES,MOD,MODE,MODIFY,MONTHS,MOVE,MVARCHAR,NEW,
NONE,NOWAIT,NULLABLE,NVARCHAR,OBJECT,OFF,OFFSET,OID,
OLD,OPERATION,OPERATORS,OPTIMIZE,OTHERS,OUT,OVER,
OVERFLOW,OWN,PAGE,PARAMETERS,PARTITIONED,PCTFREE,
PENDANT,PIC,PICTURE,PREALLOCATED,PREFERRED,PREORDER,
PRIVATE,PROGRAM,PROTECTED,PURGE,RANDOM,RD,RDAREA,
RECOMPILE,RECOVERABLE,RECOVERY,RECURSIVE,REF,
REFERENCING,REGLIKE,RELEASE,RELEASING,RENAME,
RESIGNAL,RESTART,RETURN,RETURNS,ROLE,ROOT,ROUTINE,ROW,
ROWID,SAVEPOINT,SCALE,SCAN,SCHEMAS,SCOPE,SD,SEARCH,
SECONDS,SEGMENT,SENSITIVE,SEPARATE,SEPARATOR,SEQUENCE,
SFLIKE,SHARE,SHORT,SIGNAL,SIMILAR,SLOCK,SMALLFLT,SPLIT,
SQLCODE_OF_LAST_CONDITION,SQLCODE_TYPE,SQLCOUNT,SQLDA,
SQLERRM,SQLERRM_OF_LAST_CONDITION,SQLERRMC,SQLERRML,
SQLEXCEPTION,SQLNAME,SQL_STANDARD,SQLWARN,
SQLWARNING,START,STATIC,STOP,STOPPING,STRUCTURE,SUBSTR,
SUPPRESS,SYNONYM,TEST,TEXT,THERE,TIMESTAMP_FORMAT,
TREAT,TRIGGER,TYPE,UAMT,UBINBUF,UCHAR,UDATE,UHANT,
UHDATE,UNDER,UNIFY_2000,UNIONALL,UNLIMITED,UNLOCK,UNTIL,
USE,UTIME,UTXTBUF,VARCHAR_FORMAT,VARIABLE,VIRTUAL,
VISIBLE,VOLATILE,VOLUME,VOLUMES,WAIT,WHILE,WITHOUT,
XLIKE,XLOCK,XML,XMLAGG,XMLEXISTS,XMLPARSE,XMLQUERY,
XMLSERIALIZE,YEARS

サーバが XDM/RD E2 の場合

ABS,ANDNOT,ARRAY,ASSIGN,AUDIT,BINARY,BLOB,BOOLEAN,CALL,
CARDINALITY,CHANGE,CLOB,CLUSTER,COMMENT,COMPRESSED,
CORR,COUNT,COVAR_POP,COVAR_SAMP,CUBE,CUME_DIST,
CURRENT_ROLE,DATA,DAYS,DBA,DENSE_RANK,DIGITS,DO,EACH,
ELSEIF,EVERY,EXCLUSIVE,FALSE,FILTER,FIX,FLAT,FORCE,
FUNCTION,GROUPING,HEX,HOURS,IDENTIFIED,IF,INDEX,INOUT,
ITERATE,LABEL,LARGE,LEAVE,LENGTH,LIMIT,LIST,LOCK,LONG,
LOOP,MCHAR,MICROSECOND,MICROSECONDS,MINUTES,MOD,
MODE,MONTHS,MVARCHAR,NEW,NONLOCAL,NOWAIT,NVARCHAR,
OLD,OPTIMIZE,OUT,OVER,OVERLAY,OWN,PARTITION,PCTFREE,

PERCENT_RANK,PERCENTILE_CONT,PERCENTILE_DISC,PRIVATE,
PROGRAM,PROTECTED,PURGE,RANGE,RANK,RDAREA,RDNODE,
RECURSIVE,REFERENCING,REGR_AVGX,REGR_AVGY,REGR_COUNT,
REGR_INTERCEPT,REGR_R2,REGR_SLOPE,REGR_SXX,REGR_SXY,
REGR_SYY,RELEASE,REPEAT,RESERVED,RETURN,RETURNS,ROLLUP,
ROUTINE,ROW,ROW_NUMBER,ROWID,SECONDS,SHARE,SIGNAL,
SMALLFLT,SPECIFIC,SQLCOUNT,SQLDA,SQLERRM,SQLERRMC,
SQLERRML,SQLNAME,SQLWARN,STDDEV_POP,STDDEV_SAMP,
STOPPING,SUBSTR,TIMESTAMP_FORMAT,TRIGGER,TYPE,
UNBOUNDED,UNDER,UNTIL,USER_AUDIT,USER_GROUP,
USER_LEVEL,VAR_POP,VAR_SAMP,VARCHAR_FORMAT,WAIT,WHILE,
WINDOW,WITHIN,WITHOUT,YEARS

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(bj)　getSQLStateType()

【機能】

SQLException.getSQLState によって返される SQLSTATE が，X/Open の SQL CLI であるか SQL99 であるかを返します。

【形式】

```
public int getSQLStateType() throws SQLException
```

【引数】

なし。

【戻り値】

int 型：

SQLSTATE の型。sqlStateXOpen 又は sqlStateSQL99。

【機能詳細】

- サーバが HiRDB の場合

  クライアント環境変数 PDSTANDARDSQLSTATE に YES を設定している場合，又はクライアント環境変数 PDSTANDARDSQLSTATE を省略してシステム共通定義 pd_standard_sqlstate が Y の場合，SQLSTATE には SQL2003 の値を設定するため，戻り値には sqlStateSQL99 を設定します。

  上記以外の場合，SQLSTATE には HiRDB 独自の値を設定するため，戻り値には 0 を設定します。
- サーバが XDM/RD E2 の場合

  戻り値に sqlStateSQL99 を設定します。

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(bk) getStringFunctions()

【機能】

文字列関数を，コンマで区切ったリストで返します。

【形式】

```
public String getStringFunctions() throws SQLException
```

【引数】

なし。

【戻り値】

String オブジェクト

【機能詳細】

サーバが HiRDB の場合，次のリストを返します。

ASCII,CHAR,INSERT,LCASE,LEFT,LENGTH,LOCATE,LTRIM,REPLACE,RIGHT,RTRIM,SUBSTRING,UCASE

サーバが XDM/RD E2 の場合，次のリストを返します。

LCASE,LENGTH,LOCATE,LTRIM,POSITION,REPLACE,RTRIM,SUBSTRING,UCASE

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(bl) getSuperTables(String catalog,String schemaPattern,String tableNamePattern)

【機能】

特定のスキーマで定義されているテーブル階層の説明を返します。

【形式】

```
public ResultSet getSuperTables(String catalog,String schemaPattern,String tableNamePatte
rn) throws SQLException
```

【引数】

String catalog：

カタログ名

String schemaPattern：

スキーマ名パターン

String tableNamePattern：

テーブル名パターン

【戻り値】

ResultSet オブジェクト

【機能詳細】

HiRDB，XDM/RD E2 では，テーブル階層は未サポートのため，常に検索結果行数 0 の ResultSet を返します。すべての引数について妥当性チェックを行いません。返却する ResultSet の形式を次に示します。

列番号	型	SQL 型（Types）	列名	列の意味
1	String	CHAR	TABLE_CAT	カタログ名
2	String	VARCHAR	TABLE_SCHEM	認可識別子
3	String	VARCHAR	TABLE_NAME	型名
4	String	VARCHAR	SUPERTABLE_NAME	スーパタイプ名

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(bm) getSuperTypes(String catalog,String schemaPattern,String typeNamePattern)

【機能】

特定のスキーマで定義されているユーザ定義型階層の説明を返します。

【形式】

```
public ResultSet getSuperTypes(String catalog,String schemaPattern,String typeNamePattern
) throws SQLException
```

【引数】

String catalog：

カタログ名

String schemaPattern：

スキーマ名パターン

String typeNamePattern：

ユーザ定義型名パターン

【戻り値】

ResultSet オブジェクト

【機能詳細】

TYPE4 JDBC ドライバでは，ユーザ定義型は未サポートのため，常に検索結果行数 0 の ResultSet を返します。すべての引数について妥当性チェックを行いません。返却する ResultSet の形式を次に示します。

列番号	型	SQL 型（Types）	列名	列の意味
1	String	CHAR	TYPE_CAT	ユーザ定義型のカタログ名
2	String	VARCHAR	TYPE_SCHEM	ユーザ定義型のスキーマ名
3	String	VARCHAR	TYPE_NAME	ユーザ定義型の型名
4	String	CHAR	SUPERTYPE_CAT	直接のスーパタイプのカタログ名
5	String	VARCHAR	SUPERTYPE_SCHEM	直接のスーパタイプのスキーマ名
6	String	VARCHAR	SUPERTYPE_NAME	直接のスーパタイプ名

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(bn)　getSystemFunctions()

【機能】

使用できるシステム関数を返します。

【形式】

```
public String getSystemFunctions() throws SQLException
```

【引数】

なし。

【戻り値】

String オブジェクト

【機能詳細】

常に"USER"を返します。

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(bo) getTablePrivileges(String catalog,String schemaPattern,String tableNamePattern)

【機能】

テーブルのアクセス権に関する記述を返します。

【形式】

```
public ResultSet getTablePrivileges(String catalog, String schemaPattern, String tableNam
ePattern)throws SQLException
```

【引数】

String catalog：

カタログ名。このドライバでは無視します。

String schemaPattern：

スキーマ名パターン。null 又は 0 長文字列を指定している場合，スキーマ名を検索の限定に使用しません。引数の値は大文字と小文字を区別します。

String tableNamePattern：

テーブル名パターン。null 又は 0 長文字列を指定している場合，テーブル名を検索の限定に使用しません。引数の値は大文字と小文字を区別します。

【戻り値】

ResultSet オブジェクト

【機能詳細】

指定されたテーブルに対するアクセス権に関する記述を返します。返却する ResultSet の形式を次に示します。

列番号	型	SQL 型（Types）	列名	ソート※	列の意味
1	String	CHAR	TABLE_CAT	−	カタログ名 常に NULL 値を返します。
2	String	VARCHAR	TABLE_SCHEM	1（昇順）	認可識別子名
3	String	VARCHAR	TABLE_NAME	2（昇順）	テーブル名
4	String	VARCHAR	GRANTOR	−	アクセス権の付与者
5	String	VARCHAR	GRANTEE	−	アクセス権の被付与者
6	String	VARCHAR	PRIVILEGE	3（昇順）	許可されているアクセス権限名。複数のアクセス権限を許可されている場合は，コンマ（,）で区切って返します。 SELECT：SELECT 権限 INSERT：INSERT 権限

列番号	型	SQL 型（Types）	列名	ソート※	列の意味
					UPDATE：UPDATE 権限 DELETE：DELETE 権限
7	String	VARCHAR	IS_GRANTABLE	−	アクセス権の被付与者が，別のユーザにアクセス権を与えることができるかを表します。 YES：与えることができます。 NO：与えることができません。 null：与えることができるかどうか不明です。 HiRDB,XDM/RD E2 共に,null は返しません。PRIVILEGE に設定されたアクセス権限が複数ある場合，その内の一つ以上のアクセス権限を別のユーザに与えることができれば，YES を設定します。

（凡例）

−：該当しません。

注※

ソートのキー列の優先順位を示します。

【発生する例外】

次のどちらかの場合，例外 SQLException を投入します。

- このメソッド実行前に Connection オブジェクトに対して close()を実行した場合
- データベースアクセスエラーが発生した場合

(bp) getTables(String catalog, String schemaPattern, String tableNamePattern, String[] types)

【機能】

テーブルに関する記述を返します。

【形式】

```
public ResultSet getTables(String catalog, String schemaPattern, String tableNamePattern,
 String[] types) throws SQLException
```

【引数】

String catalog：

未使用。

String schemaPattern：

スキーマ名パターン。引数の値は大文字と小文字を区別します。

String tableNamePattern：

テーブル名パターン。引数の値は大文字と小文字を区別します。

String[] types：

テーブルの型のリスト。getTableTypes()メソッドによって返されるテーブルの型を指定できます。引数の値は大文字と小文字を区別します。null の場合，"TABLE"，"VIEW"，"SYSTEM TABLE"，及び"GLOBAL TEMPORARY"のすべてが指定されたものとします。

【戻り値】

ResultSet オブジェクト

【機能詳細】

テーブルに関する記述を返します。返却する ResultSet の形式を次の表に示します。引数 String[] types に，getTableTypes()メソッドによって返されるテーブルの型のどれにも該当しないテーブルの型を指定した場合，その指定値を無視します。有効な型が 1 つも指定されていない場合は検索結果行数 0 の ResultSet を返却します。

表 17–54　getTables で返却する ResultSet の形式

列番号	型	SQL 型	列名	列の意味
1	String	CHAR	TABLE_CAT	常に NULL 値を返します。
2	String	VARCHAR	TABLE_SCHEM	認可識別子名
3	String	VARCHAR	TABLE_NAME	テーブル名
4	String	VARCHAR	TABLE_TYPE	表の種別 • TABLE：永続実表（DABroker for Java 互換で"BASE TABLE"と指定されても"TABLE"と同様の動作をします） • VIEW：ビュー表 • SYSTEM TABLE：ディクショナリ表 • GLOBAL TEMPORARY：一時表
5	String	VARCHAR	REMARKS	コメント
6	String	CHAR	TYPE_CAT	常に NULL 値を返します。
7	String	CHAR	TYPE_SCHEM	常に NULL 値を返します。
8	String	CHAR	TYPE_NAME	常に NULL 値を返します。
9	String	CHAR	SELF_REFERENCING_COL_NAME	常に NULL 値を返します。
10	String	CHAR	REF_GENERATION	常に NULL 値を返します。

【発生する例外】

次の場合，SQLException を投入します。

- このメソッド実行前に Connection オブジェクトに対して close()を実行している場合
- 引数 String[] types の一つ以上の要素が null である場合

(bq)　getTableTypes()

【機能】

使用できるテーブルの型を返します。

【形式】

```
public ResultSet getTableTypes()throws SQLException
```

【引数】

なし。

【戻り値】

ResultSet オブジェクト

【機能詳細】

使用できるテーブルの型を返します。返却する ResultSet の形式を次に示します。

列番号	型	SQL 型	列名	列の意味
1	String	VARCHAR	TABLE_TYPE	表の種別 • TABLE：永続実表 • VIEW：ビュー表 • SYSTEM TABLE：ディクショナリ表 • GLOBAL TEMPORARY：一時表

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(br)　getTimeDateFunctions()

【機能】

使用できる時間関数と日付関数をコンマで区切ったリストを返します。

【形式】

```
public String getTimeDateFunctions() throws SQLException
```

【引数】

なし。

【戻り値】

String オブジェクト

【機能詳細】

サーバが HiRDB の場合，次のリストを返します。

ADD_INTERVAL,CENTURY,CHARACTER,CURRENT_DATE,CURRENT_TIME,CURRENT_TIMESTAMP,

DATE,DATE_TIME,DAYNAME,DAYOFWEEK,DAYOFYEAR,DAY,DAYS,INTERVAL_DATETIMES,
HALF,HOUR,LAST_DAY,MIDNIGHTSECONDS,MINUTE,MONTH,MONTHNAME,MONTHS_BETWEEN,
NEXT_DAY,QUARTER,ROUNDMONTH,SECOND,TIME,TIMESTAMP,TIMESTAMP_FORMAT,TRUNCYEAR,
VARCHAR_FORMAT,WEEK,WEEKOFMONTH,YEAR,YEARS_BETWEEN

サーバが XDM/RD E2 の場合，次のリストを返します。

CURDATE,CURRENT_DATE,CURTIME,CURRENT_TIME,CURRENT_TIMESTAMP,HOUR,MINUTE,MONTH,NOW,SECOND,YEAR

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(bs)　getTypeInfo()

【機能】

標準 SQL の型に関する記述を返します。

【形式】

```
public ResultSet getTypeInfo() throws SQLException
```

【引数】

なし。

【戻り値】

ResultSet オブジェクト

【機能詳細】

標準 SQL の型に関する記述を返します。返却する ResultSet の形式を次の表に示します。

表 17–55　getTypeInfo で返却する ResultSet の形式

列番号	型	SQL 型	列名	列の意味
1	String	VARCHAR	TYPE_NAME	型名
2	short	SMALLINT	DATA_TYPE	java.sql.Types の SQL データ型
3	int	INTEGER	PRECISION	最大の精度
4	String	VARCHAR	LITERAL_PREFIX	リテラルを引用するのに使用する接頭辞
5	String	VARCHAR	LITERAL_SUFFIX	リテラルを引用するのに使用する接尾辞
6	String	VARCHAR	CREATE_PARAMS	型の作成に使用するパラメタ
7	short	SMALLINT	NULLABLE	この型に NULL を使用できるかどうかを返します。 • typeNoNulls：NULL 値を使用できません。

列番号	型	SQL 型	列名	列の意味
				• typeNullable：NULL 値を使用できます。 • typeNullableUnknown：NULL 値を使用できるか不明です。 常に typeNullableUnknown を返します。
8	boolean	BIT	CASE_SENSITIVE	大文字と小文字を区別するかどうかを返します。 • true：文字列系データ型 • false：その他のデータ型
9	short	SMALLINT	SEARCHABLE	この型に"WHERE"を使用できるかどうかを返します。 • typePredNone：使用できません。 • typePredChar：WHERE..LIKE だけ使用できます。 • typePredBasic：WHERE..LIKE 以外を使用できます。 • typeSearchable：すべての WHERE..を使用できます。 常に typeSearchable を返します。
10	boolean	BIT	UNSIGNED_ATTRIBUTE	符号無し属性かどうかを返します。 数値データは符号有りのため false，その他のデータ型は符号の有無に関係ないため false を返します。
11	boolean	BIT	FIXED_PREC_SCALE	通貨の値になれるかどうかを返します。 常に false を返します。
12	boolean	BIT	AUTO_INCREMENT	自動インクリメントの値に使用できるかどうかを返します。 常に false を返します。
13	String	VARCHAR	LOCAL_TYPE_NAME	型名の地域対応されたバージョン 型名と同じものを返します。
14	short	SMALLINT	MINIMUM_SCALE	サポートされる最小スケール
15	short	SMALLINT	MAXIMUM_SCALE	サポートされる最大スケール
16	int	INTEGER	SQL_DATA_TYPE	常に NULL 値を返します。
17	int	INTEGER	SQL_DATETIME_SUB	常に NULL 値を返します。
18	int	INTEGER	NUM_PREC_RADIX	数値データは 2 又は 10， それ以外は 0

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(bt) getUDTs(String catalog,String schemaPattern,String typeNamePattern,int[] types)

【機能】

ユーザ定義型に関する情報を返します。

【形式】

```
public ResultSet getUDTs(String catalog, String schemaPattern, String typeNamePattern, int[] types) throws SQLException
```

【引数】

String catalog：

カタログ名

String schemaPattern：

スキーマ名パターン

String typeNamePattern：

型名パターン

int[] types：

ユーザ定義型のリスト

【戻り値】

ResultSet オブジェクト

【機能詳細】

Type4 JDBC ドライバでは，ユーザ定義型は未サポートのため，常に検索結果行数 0 の ResultSet を返します。すべての引数について妥当性チェックを行いません。返却する ResultSet の形式を次に示します。

列番号	型	SQL 型（Types）	列名	列の意味
1	String	CHAR	TYPE_CAT	ユーザ定義型のカタログ名
2	String	VARCHAR	TYPE_SCHEM	ユーザ定義型のスキーマ名
3	String	VARCHAR	TYPE_NAME	ユーザ定義型の型名
4	String	VARCHAR	CLASS_NAME	Java クラス名
5	int	INTEGER	DATA_TYPE	java.sql.Types で定義されている型値
6	String	VARCHAR	REMARKS	型に関する説明
7	short	SMALLINT	BASE_TYPE	DISTINCT 型のソースの型の型コード，又は java.sql.Types で定義される構造型の SELF_REFERENCING_COLUMN のユーザ生成参照型を実装する型の型コード

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(bu)　getURL()

【機能】

HiRDB 又は XDM/RD E2 に接続した URL を返します。URL がない場合は NULL を返します。

【形式】

```
public String getURL() throws SQLException
```

【引数】

なし。

【戻り値】

String オブジェクト

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(bv)　getUserName()

【機能】

HiRDB 又は XDM/RD E2 に接続する際に使用したユーザ名を返します。

OS ログインユーザの簡易認証機能を使用している場合は，簡易認証キーワードを返します。

【形式】

```
public String getUserName() throws SQLException
```

【引数】

なし。

【戻り値】

String オブジェクト

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(bw)　getVersionColumns(String catalog,String schema,String table)

【機能】

テーブル中の行が更新された場合に，自動的に更新されるテーブルの列に関する記述を返します。

【形式】

```
public ResultSet getVersionColumns(String catalog,String schema,String table) throws SQLE
xception
```

【引数】

String catalog：

カタログ名

String schema：

スキーマ名

String table：

テーブル名

【戻り値】

ResultSet オブジェクト

【機能詳細】

HiRDB，XDM/RD E2 では，自動的に列を更新する機能は未サポートのため，常に検索結果行数 0 の ResultSet を返します。すべての引数について妥当性チェックを行いません。返却する ResultSet の形式を次に示します。

列番号	型	SQL 型（Types）	列名	列の意味
1	short	SMALLINT	SCOPE	未使用
2	String	VARCHAR	COLUMN_NAME	列名
3	int	INTEGER	DATA_TYPE	SQL 型
4	String	VARCHAR	TYPE_NAME	データソース依存の型名
5	int	INTEGER	COLUMN_SIZE	データの精度
6	int	INTEGER	BUFFER_LENGTH	データのバイト数
7	short	SMALLINT	DECIMAL_DIGITS	スケール
8	short	SMALLINT	PSEUDO_COLUMN	列が擬似列かどうかを表します。

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(bx)　insertsAreDetected(int type)

【機能】

ResultSet.rowInserted()メソッドを呼び出すことによって，可視の行が挿入されたことを検出できるかどうかを返します。

【形式】

```
public boolean insertsAreDetected(int type) throws SQLException
```

【引数】

int type：

ResultSet の型。次のどれかです。

- ResultSet.TYPE_FORWARD_ONLY
- ResultSet.TYPE_SCROLL_INSENSITIVE
- ResultSet.TYPE_SCROLL_SENSITIVE

【戻り値】

boolean 型：

true：指定された結果セットの型によって変更が検出されます。

false：指定された結果セットの型によって変更が検出されません。

【機能詳細】

常に false を返します。

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(by)　isCatalogAtStart()

【機能】

完全修飾されたテーブル名の開始部分（又は終了部分）に，カタログが現れるかどうかを返します。

【形式】

```
public boolean isCatalogAtStart() throws SQLException
```

【引数】

なし。

【戻り値】

boolean 型：

true：完全修飾されたテーブル名の開始部分にカタログ名が現れます。

false：完全修飾されたテーブル名の開始部分にカタログ名が現れません。

【機能詳細】

常に false を返します。

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(bz)　isReadOnly()

【機能】

データベースが読み込み専用モードかどうかを返します。

【形式】

```
public boolean isReadOnly() throws SQLException
```

【引数】

なし。

【戻り値】

boolean 型：

true：読み込み専用モードです。

false：読み込み専用モードではありません。

【機能詳細】

参照 DB の場合は true，参照 DB でない場合は false を返します。

注意：

サーバが XDM/RD E2 の場合，Database Connection Server の起動制御文，又は運用コマンドで，データベースのアクセスモードを READ に設定すると読み込み専用となりますが，JDBC ドライバはアクセスモードの情報をサーバから取得していないので，アクセスモードが UPDATE であると仮定して処理します。

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(ca)　locatorsUpdateCopy()

【機能】

LOB への変更が，コピーに対して行われたのか，LOB に直接行われたのかを示します。

【形式】

```
public boolean locatorsUpdateCopy() throws SQLException
```

【引数】

なし。

【戻り値】

boolean 型：

true ：変更が LOB のコピーに対して行われました。

false：変更が LOB に直接行われました。

【機能詳細】

常に false を返します。

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(cb)　nullPlusNonNullIsNull()

【機能】

NULL 値と非 NULL 値の連結を，NULL とするかどうかを返します。

【形式】

```
public boolean nullPlusNonNullIsNull() throws SQLException
```

【引数】

なし。

【戻り値】

boolean 型：

true：NULL 値と非 NULL 値の連結を NULL とします。

false：NULL 値と非 NULL 値の連結を NULL としません。

【機能詳細】

常に true を返します。

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(cc)　nullsAreSortedAtEnd()

【機能】

NULL 値が，終了時にソート順に関係なくソートされるかどうかを返します。

【形式】

```
public boolean nullsAreSortedAtEnd() throws SQLException
```

【引数】

なし。

【戻り値】

boolean 型：

true：ソートされます。

false：ソートされません。

【機能詳細】

常に false を返します。

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(cd)　nullsAreSortedAtStart()

【機能】

NULL 値が，開始時にソート順に関係なくソートされるかどうかを返します。

【形式】

```
public boolean nullsAreSortedAtStart() throws SQLException
```

【引数】

なし。

【戻り値】

boolean 型：

true：ソートされます。

false：ソートされません。

【機能詳細】

常に false を返します。

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(ce)　nullsAreSortedHigh()

【機能】

NULL 値が高位にソートされるかどうかを返します。

【形式】

```
public boolean nullsAreSortedHigh() throws SQLException
```

【引数】

なし。

【戻り値】

boolean 型：

true：ソートされます。

false：ソートされません。

【機能詳細】

常に true を返します。

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(cf)　nullsAreSortedLow()

【機能】

NULL 値が下位にソートされるかどうかを返します。

【形式】

```
public boolean nullsAreSortedLow() throws SQLException
```

【引数】

なし。

【戻り値】

boolean 型：

true：ソートされます。

false：ソートされません。

【機能詳細】

常に false を返します。

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(cg)　othersDeletesAreVisible(int type)

【機能】

ほかで行われた削除が可視かどうかを返します。

【形式】

```
public boolean othersDeletesAreVisible(int type) throws SQLException
```

【引数】

int type：

ResultSet の型。次のどれかです。

- ResultSet.TYPE_FORWARD_ONLY
- ResultSet.TYPE_SCROLL_INSENSITIVE
- ResultSet.TYPE_SCROLL_SENSITIVE

【戻り値】

boolean 型：

true：可視です。

false：可視ではありません。

【機能詳細】

常に false を返します。

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(ch)　othersInsertsAreVisible(int type)

【機能】

ほかで行われた挿入が可視かどうかを返します。

【形式】

```
public boolean othersInsertsAreVisible(int type) throws SQLException
```

【引数】

int type：

ResultSet の型。次のどれかです。

- ResultSet.TYPE_FORWARD_ONLY
- ResultSet.TYPE_SCROLL_INSENSITIVE
- ResultSet.TYPE_SCROLL_SENSITIVE

【戻り値】

boolean 型：

true：可視です。

false：可視ではありません。

【機能詳細】

常に false を返します。

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(ci)　othersUpdatesAreVisible(int type)

【機能】

ほかで行われた削除が可視かどうかを返します。

【形式】

```
public boolean othersUpdatesAreVisible(int type) throws SQLException
```

【引数】

int type：

ResultSet の型。次のどれかです。

- ResultSet.TYPE_FORWARD_ONLY
- ResultSet.TYPE_SCROLL_INSENSITIVE
- ResultSet.TYPE_SCROLL_SENSITIVE

【戻り値】

boolean 型：

true：可視です。

false：可視ではありません。

【機能詳細】

常に false を返します。

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(cj)　ownDeletesAreVisible(int type)

【機能】

結果セット自身の削除が可視かどうかを返します。

【形式】

```
public boolean ownDeletesAreVisible(int type) throws SQLException
```

【引数】

int type：

ResultSet の型。次のどれかです。

- ResultSet.TYPE_FORWARD_ONLY
- ResultSet.TYPE_SCROLL_INSENSITIVE
- ResultSet.TYPE_SCROLL_SENSITIVE

【戻り値】

boolean 型：

true：可視です。

false：可視ではありません。

【機能詳細】

常に false を返します。

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(ck)　ownInsertsAreVisible(int type)

【機能】

結果セット自身の挿入が可視かどうかを返します。

【形式】

```
public boolean ownInsertsAreVisible(int type) throws SQLException
```

【引数】

int type：

ResultSet の型。次のどれかです。

- ResultSet.TYPE_FORWARD_ONLY
- ResultSet.TYPE_SCROLL_INSENSITIVE
- ResultSet.TYPE_SCROLL_SENSITIVE

【戻り値】

boolean 型：

true：可視です。

false：可視ではありません。

【機能詳細】

常に false を返します。

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(cl)　ownUpdatesAreVisible(int type)

【機能】

結果セット自身の更新が可視かどうかを返します。

【形式】

```
public boolean ownUpdatesAreVisible(int type) throws SQLException
```

【引数】

int type：

ResultSet の型。次のどれかです。

- ResultSet.TYPE_FORWARD_ONLY
- ResultSet.TYPE_SCROLL_INSENSITIVE
- ResultSet.TYPE_SCROLL_SENSITIVE

【戻り値】

boolean 型：

true：可視です。

false：可視ではありません。

【機能詳細】

常に false を返します。

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(cm)　storesLowerCaseIdentifiers()

【機能】

大文字と小文字が混在する引用符なしの SQL 識別子を，大文字と小文字を区別しないで処理し，小文字で格納するかどうかを返します。

【形式】

```
public boolean storesLowerCaseIdentifiers() throws SQLException
```

【引数】

なし。

【戻り値】

boolean 型：

true：大文字と小文字を区別しないで，小文字で格納します。

false：大文字と小文字を区別して格納します。

【機能詳細】

常に false を返します。

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(cn)　storesLowerCaseQuotedIdentifiers()

【機能】

大文字と小文字が混在する引用符付きの SQL 識別子を，大文字と小文字を区別しないで処理し，小文字で格納するかどうかを返します。

【形式】

```
public boolean storesLowerCaseQuotedIdentifiers() throws SQLException
```

【引数】

なし。

【戻り値】

boolean 型：

true：大文字と小文字を区別しないで，小文字で格納します。

false：上記ではありません。

【機能詳細】

常に false を返します。

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(co)　storesMixedCaseIdentifiers()

【機能】

大文字と小文字が混在する引用符なしの SQL 識別子を，大文字と小文字を区別しないで処理し，大文字と小文字を混在して格納するかどうかを返します。

【形式】

```
public boolean storesMixedCaseIdentifiers() throws SQLException
```

【引数】

なし。

【戻り値】

boolean 型：

true：大文字と小文字を区別しないで処理し，大文字と小文字を混在して格納します。

false：上記ではありません。

【機能詳細】

常に false を返します。

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(cp)　storesMixedCaseQuotedIdentifiers()

【機能】

大文字と小文字が混在する引用符付きの SQL 識別子を，大文字と小文字を区別しないで処理し，大文字と小文字を混在して格納するかどうかを返します。

【形式】

```
public boolean storesMixedCaseQuotedIdentifiers() throws SQLException
```

【引数】

なし。

【戻り値】

boolean 型：

true：大文字と小文字を区別しないで処理し，大文字と小文字を混在して格納します。

false：上記ではありません。

【機能詳細】

常に true を返します。

注意：

1. 実際には大文字と小文字を区別して処理しますが，結果として大文字と小文字を混在して格納するので，true を返します。大文字と小文字を区別して処理するかどうかは，supportsMixedCaseQuotedIdentifiers メソッドで判定できます。
2. サーバが XDM/RD E2 の場合，XDM/RD E2 の RD 環境定義で，DELIMITED ID UPPER オペランドに YES（デフォルトは NO）を指定しているときは大文字で格納しますが，JDBC ドライバは DELIMITED ID UPPER オペランドの情報をサーバから取得していないので，デフォルトの NO であると仮定して処理します。

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(cq)　storesUpperCaseIdentifiers()

【機能】

大文字と小文字が混在する引用符なしの SQL 識別子を，大文字と小文字を区別しないで処理し，大文字で格納するかどうかを返します。

【形式】

```
public boolean storesUpperCaseIdentifiers() throws SQLException
```

【引数】

なし。

【戻り値】

boolean 型：

true：大文字と小文字を区別しないで処理し，大文字で格納します。

false：上記ではありません。

【機能詳細】

常に true を返します。

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(cr)　storesUpperCaseQuotedIdentifiers()

【機能】

大文字と小文字が混在する引用符付きの SQL 識別子を，大文字と小文字を区別しないで処理し，大文字で格納するかどうかを返します。

【形式】

```
public boolean storesUpperCaseQuotedIdentifiers() throws SQLException
```

【引数】

なし。

【戻り値】

boolean 型：

true：大文字と小文字を区別しないで処理し，大文字で格納します。

false：上記ではありません。

【機能詳細】

常に false を返します。

注意：

サーバが XDM/RD E2 の場合，XDM/RD E2 の RD 環境定義で，DELIMITED ID UPPER オペランドに YES（デフォルトは NO）を指定しているときは大文字で格納しますが，JDBC ドライバからは DELIMITED ID UPPER オペランドの情報を取得できないので，JDBC ドライバはデフォルトの NO であると仮定して処理します。

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(cs)　supportsAlterTableWithAddColumn()

【機能】

追加列のある ALTER TABLE が，サポートされているかどうかを返します。

【形式】

```
public boolean supportsAlterTableWithAddColumn() throws SQLException
```

【引数】

なし。

【戻り値】

boolean 型：

true：サポートしています。

false：サポートしていません。

【機能詳細】

常に true を返します。

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(ct)　supportsAlterTableWithDropColumn()

【機能】

ドロップ列のある ALTER TABLE が，サポートされているかどうかを返します。

【形式】

```
public boolean supportsAlterTableWithDropColumn() throws SQLException
```

【引数】

なし。

【戻り値】

boolean 型：

true：サポートしています。

false：サポートしていません。

【機能詳細】

常に true を返します。

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(cu)　supportsANSI92EntryLevelSQL()

【機能】

ANSI92 エントリレベルの SQL 文法が，サポートされるかどうかを返します。

【形式】

```
public boolean supportsANSI92EntryLevelSQL() throws SQLException
```

【引数】

なし。

【戻り値】

boolean 型：

true：サポートしています。

false：サポートしていません。

【機能詳細】

常に true を返します。

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(cv)　supportsANSI92FullSQL()

【機能】

ANSI92 完全レベルの SQL 文法が，サポートされるかどうかを返します。

【形式】

```
public boolean supportsANSI92FullSQL() throws SQLException
```

【引数】

なし。

【戻り値】

boolean 型：

true：サポートしています。

false：サポートしていません。

【機能詳細】

常に false を返します。

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(cw)　supportsANSI92IntermediateSQL()

【機能】

ANSI92 中間レベルの SQL 文法が，サポートされるかどうかを返します。

【形式】

```
public boolean supportsANSI92IntermediateSQL() throws SQLException
```

【引数】

なし。

【戻り値】

boolean 型：

true：サポートしています。

false：サポートしていません。

【機能詳細】

常に false を返します。

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(cx)　supportsBatchUpdates()

【機能】

バッチ更新をサポートしているかどうかを返します。

【形式】

```
public boolean supportsBatchUpdates() throws SQLException
```

【引数】

なし。

【戻り値】

boolean 型：

true：サポートしています。

false：サポートしていません。

【機能詳細】

常に true を返します。

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(cy)　supportsCatalogsInDataManipulation()

【機能】

データ操作文でカタログ名を使用できるかどうかを返します。

【形式】

```
public boolean supportsCatalogsInDataManipulation() throws SQLException
```

【引数】

なし。

【戻り値】

boolean 型：

true：カタログ名を使用できます。

false：カタログ名を使用できません。

【機能詳細】

常に false を返します。

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(cz)　supportsCatalogsInIndexDefinitions()

【機能】

インデクス定義文でカタログ名を使用できるかどうかを返します。

【形式】

```
public boolean supportsCatalogsInIndexDefinitions() throws SQLException
```

【引数】

なし。

【戻り値】

boolean 型：

true：カタログ名を使用できます。

false：カタログ名を使用できません。

【機能詳細】

常に false を返します。

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(da)　supportsCatalogsInPrivilegeDefinitions()

【機能】

特権定義文でカタログ名を使用できるかどうかを返します。

【形式】

```
public boolean supportsCatalogsInPrivilegeDefinitions() throws SQLException
```

【引数】

なし。

【戻り値】

boolean 型：

true：カタログ名を使用できます。

false：カタログ名を使用できません。

【機能詳細】

常に false を返します。

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(db)　supportsCatalogsInProcedureCalls()

【機能】

プロシジャ呼び出し文でカタログ名を使用できるかどうかを返します。

【形式】

```
public boolean supportsCatalogsInProcedureCalls() throws SQLException
```

【引数】

なし。

【戻り値】

boolean 型：

true：カタログ名を使用できます。

false：カタログ名を使用できません。

【機能詳細】

常に false を返します。

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(dc)　supportsCatalogsInTableDefinitions()

【機能】

テーブル定義文でカタログ名を使用できるかどうかを返します。

【形式】

```
public boolean supportsCatalogsInTableDefinitions() throws SQLException
```

【引数】

なし。

【戻り値】

boolean 型：

true：カタログ名を使用できます。

false：カタログ名を使用できません。

【機能詳細】

常に false を返します。

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(dd)　supportsColumnAliasing()

【機能】

カラムの別名をサポートしているかどうかを返します。

【形式】

```
public boolean supportsColumnAliasing() throws SQLException
```

【引数】

なし。

【戻り値】

boolean 型：

true：サポートしています。

false：サポートしていません。

【機能詳細】

常に true を返します。

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(de)　supportsConvert()

【機能】

SQL タイプ間の CONVERT 関数をサポートしているかどうかを返します。

【形式】

```
public boolean supportsConvert() throws SQLException
```

【引数】

なし。

【戻り値】

boolean 型：

true：サポートしています。

false：サポートしていません。

【機能詳細】

常に true を返します。

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(df)　supportsConvert(int fromType, int toType)

【機能】

与えられた SQL タイプ間の CONVERT 関数をサポートしているかどうかを返します。

【形式】

```
public boolean supportsConvert(int fromType, int toType) throws SQLException
```

【引数】

int fromType：

変換元の型。java.sql.Types クラスの型コードです。

int toType：

変換先の型。java.sql.Types クラスの型コードです。

【戻り値】

boolean 型：

true：サポートしています。

false：サポートしていません。

【機能詳細】

変換元の型と変換先の型の組み合わせによって値を返します。変換元の型と変換先の型の組み合わせでの戻り値を次の表に示します。

表 17–56　変換元の型と変換先の型での組み合わせでの戻り値（その 1）

変換元の型 java.sql.Types の型	変換先の型 java.sql.Types の型												
	BIT	TINYINT	SMALLINT	INTEGER	BIGINT	FLOAT	REAL	DOUBLE	NUMERIC	DECIMAL	CHAR	VARCHAR	LONGVARCHAR
BIT	1	1	1	1	1	1	1	1	1	1	1	1	1
TINYINT	1	1	1	1	1	1	1	1	1	1	1	1	1
SMALLINT	1	1	1	1	1	1	1	1	1	1	1	1	1
INTEGER	1	1	1	1	1	1	1	1	1	1	1	1	1
BIGINT	1	1	1	1	1	1	1	1	1	1	1	1	1
FLOAT	1	1	1	1	1	1	1	1	1	1	1	1	1
REAL	1	1	1	1	1	1	1	1	1	1	1	1	1
DOUBLE	1	1	1	1	1	1	1	1	1	1	1	1	1
NUMERIC	1	1	1	1	1	1	1	1	1	1	1	1	1
DECIMAL	1	1	1	1	1	1	1	1	1	1	1	1	1
CHAR	1	1	1	1	1	1	1	1	1	1	1	1	1
VARCHAR	1	1	1	1	1	1	1	1	1	1	1	1	1
LONGVARCHAR	1	1	1	1	1	1	1	1	1	1	1	1	1
DATE	0	0	0	0	0	0	0	0	0	0	1	1	1
TIME	0	0	0	0	0	0	0	0	0	0	1	1	1
TIMESTAMP	0	0	0	0	0	0	0	0	0	0	1	1	1
BINARY	0	0	0	0	0	0	0	0	0	0	0	0	0
VARBINARY	0	0	0	0	0	0	0	0	0	0	0	0	0
LONGVARBINARY	0	0	0	0	0	0	0	0	0	0	0	0	0
JAVA_OBJECT	0	0	0	0	0	0	0	0	0	0	0	0	0
STRUCT	0	0	0	0	0	0	0	0	0	0	0	0	0

変換元の型 java.sql.Types の型	変換先の型 java.sql.Types の型												
	BIT	TINYINT	SMALLINT	INTEGER	BIGINT	FLOAT	REAL	DOUBLE	NUMERIC	DECIMAL	CHAR	VARCHAR	LONGVARCHAR
ARRAY	0	0	0	0	0	0	0	0	0	0	0	0	0
BLOB	0	0	0	0	0	0	0	0	0	0	0	0	0
CLOB	0	0	0	0	0	0	0	0	0	0	0	0	0
REF	0	0	0	0	0	0	0	0	0	0	0	0	0

（凡例）

0：false　1：true

表 17–57　変換元の型と変換先の型での組み合わせでの戻り値（その 2）

変換元の型 java.sql.Types の型	変換先の型 java.sql.Types の型											
	DATE	TIME	TIMESTAMP	BINARY	VARBINARY	LONGVARBINARY	JAVA_OBJECT	STRUCT	ARRAY	BLOB	CLOB	REF
BIT	0	0	0	0	0	0	0	0	0	0	0	0
TINYINT	0	0	0	0	0	0	0	0	0	0	0	0
SMALLINT	0	0	0	0	0	0	0	0	0	0	0	0
INTEGER	0	0	0	0	0	0	0	0	0	0	0	0
BIGINT	0	0	0	0	0	0	0	0	0	0	0	0
FLOAT	0	0	0	0	0	0	0	0	0	0	0	0
REAL	0	0	0	0	0	0	0	0	0	0	0	0
DOUBLE	0	0	0	0	0	0	0	0	0	0	0	0
NUMERIC	0	0	0	0	0	0	0	0	0	0	0	0
DECIMAL	0	0	0	0	0	0	0	0	0	0	0	0
CHAR	1	1	1	0	0	0	0	0	0	0	0	0
VARCHAR	1	1	1	0	0	0	0	0	0	0	0	0
LONGVARCHAR	1	1	1	0	0	0	0	0	0	0	0	0
DATE	1	0	1	0	0	0	0	0	0	0	0	0
TIME	0	1	0	0	0	0	0	0	0	0	0	0
TIMESTAMP	1	0	1	0	0	0	0	0	0	0	0	0

変換元の型 java.sql.Types の型	変換先の型 java.sql.Types の型											
	DATE	TIME	TIMESTAMP	BINARY	VARBINARY	LONGVARBINARY	JAVA_OBJECT	STRUCT	ARRAY	BLOB	CLOB	REF
BINARY	0	0	0	1	1	1	0	0	0	0	0	0
VARBINARY	0	0	0	1	1	1	0	0	0	0	0	0
LONGVARBINARY	0	0	0	1	1	1	0	0	0	0	0	0
JAVA_OBJECT	0	0	0	0	0	0	0	0	0	0	0	0
STRUCT	0	0	0	0	0	0	0	0	0	0	0	0
ARRAY	0	0	0	0	0	0	0	0	0	0	0	0
BLOB	0	0	0	0	0	0	0	0	0	0	0	0
CLOB	0	0	0	0	0	0	0	0	0	0	0	0
REF	0	0	0	0	0	0	0	0	0	0	0	0

（凡例）

0：false　1：true

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(dg)　supportsCoreSQLGrammar()

【機能】

ODBC Core SQL 文法をサポートしているかどうかを返します。

【形式】

```
public boolean supportsCoreSQLGrammar() throws SQLException
```

【引数】

なし。

【戻り値】

boolean 型：

true：サポートしています。

false：サポートしていません。

【機能詳細】

常に true を返します。

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(dh)　supportsCorrelatedSubqueries()

【機能】

照合関連サブクエリーをサポートしているかどうかを返します。

【形式】

```
public boolean supportsCorrelatedSubqueries() throws SQLException
```

【引数】

なし。

【戻り値】

boolean 型：

true：サポートしています。

false：サポートしていません。

【機能詳細】

常に true を返します。

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(di)　supportsDataDefinitionAndDataManipulationTransactions()

【機能】

トランザクションで，データ定義文とデータ操作文の両方をサポートしているかどうかを返します。

【形式】

```
public boolean supportsDataDefinitionAndDataManipulationTransactions() throws SQLExceptio
n
```

【引数】

なし。

【戻り値】

boolean 型：

true：サポートしています。

false：サポートしていません。

【機能詳細】

サーバが HiRDB の場合，false を返します。

サーバが XDM/RD E2 の場合，true を返します。

注意：

サーバが XDM/RD E2 の場合，データ操作文を実行したトランザクションが決着していない状態で，データ定義文を実行できます。ただし，データ定義文がトランザクションを強制的にコミットさせます。

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(dj)　supportsDataManipulationTransactionsOnly()

【機能】

トランザクションで，データ操作文だけをサポートしているかどうかを返します。

【形式】

```
public boolean supportsDataManipulationTransactionsOnly() throws SQLException
```

【引数】

なし。

【戻り値】

boolean 型：

true：サポートしています。

false：サポートしていません。

【機能詳細】

常に false を返します。

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(dk)　supportsDifferentTableCorrelationNames()

【機能】

テーブル相互関連名がサポートされている場合，テーブルの名前と異なる名前にするという制限があるかどうかを返します。

【形式】

```
public boolean supportsDifferentTableCorrelationNames() throws SQLException
```

【引数】

なし。

【戻り値】

boolean 型：

true：制限があります。

false：制限がありません。

【機能詳細】

サーバが HiRDB の場合，false を返します。

サーバが XDM/RD E2 の場合，true を返します。

注意：

サーバが XDM/RD E2 の場合，XDM/RD E2 の RD 環境定義で，RANGE VARIABLE オペランドに USE（デフォルトは NOUSE）を指定しているときは，テーブルの名前と同じ相関名を使用できます。しかし，JDBC ドライバは，RANGE VARIABLE オペランドの情報をサーバから取得していないので，このメソッドは，デフォルトの NOUSE であると仮定して true を返します。JDBC ドライバ自身は，テーブルの名前と同じ相関名の使用を制限していません。

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(dl)　supportsExpressionsInOrderBy()

【機能】

"ORDER BY"リスト中の式をサポートしているかどうかを返します。

【形式】

```
public boolean supportsExpressionsInOrderBy() throws SQLException
```

【引数】

なし。

【戻り値】

boolean 型：

true：サポートしています。

false：サポートしていません。

【機能詳細】

サーバが HiRDB の場合，false を返します。

サーバが XDM/RD E2 の場合，true を返します。

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(dm)　supportsExtendedSQLGrammar()

【機能】

ODBC Extended SQL 文法をサポートしているかどうかを返します。

【形式】

```
public boolean supportsExtendedSQLGrammar() throws SQLException
```

【引数】

なし。

【戻り値】

boolean 型：

true：サポートしています。

false：サポートしていません。

【機能詳細】

常に false を返します。

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(dn)　supportsFullOuterJoins()

【機能】

完全ネストの外部結合をサポートしているかどうかを返します。

【形式】

```
public boolean supportsFullOuterJoins() throws SQLException
```

【引数】

なし。

【戻り値】

boolean 型：

true：サポートしています。

false：サポートしていません。

【機能詳細】

常に true を返します。

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(do)　supportsGetGeneratedKeys()

【機能】

文が実行された後に自動生成キーを取得できるかどうかを返します。

【形式】

```
public boolean supportsGetGeneratedKeys() throws SQLException
```

【引数】

なし。

【戻り値】

boolean 型：

true：自動生成キーを取得できます。

false：自動生成キーを取得できません。

【機能詳細】

Type4 JDBC ドライバでは，自動生成キーは未サポートのため，常に false を返します。

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(dp)　supportsGroupBy()

【機能】

"GROUP BY"節のフォームをサポートしているかどうかを返します。

【形式】

```
public boolean supportsGroupBy() throws SQLException
```

【引数】

なし。

【戻り値】

boolean 型：

true：サポートしています。

false：サポートしていません。

【機能詳細】

常に true を返します。

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(dq)　supportsGroupByBeyondSelect()

【機能】

SELECT 中のすべてのカラムを指定するという条件で，"GROUP BY"節が SELECT 中にないカラムを使用できるかどうかを返します。

【形式】

```
public boolean supportsGroupByBeyondSelect() throws SQLException
```

【引数】

なし。

【戻り値】

boolean 型：

true：使用できます。

false：使用できません。

【機能詳細】

常に true を返します。

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(dr)　supportsGroupByUnrelated()

【機能】

"GROUP BY"節が，SELECT 中にないカラムを使用できるかどうかを返します。

【形式】

```
public boolean supportsGroupByUnrelated() throws SQLException
```

【引数】

なし。

【戻り値】

boolean 型：

true：使用できます。

false：使用できません。

【機能詳細】

常に true を返します。

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(ds)　supportsIntegrityEnhancementFacility()

【機能】

SQL Integrity Enhancement Facility をサポートしているかどうかを返します。

【形式】

```
public boolean supportsIntegrityEnhancementFacility() throws SQLException
```

【引数】

なし。

【戻り値】

boolean 型：

true：サポートしています。

false：サポートしていません。

【機能詳細】

常に false を返します。

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(dt)　supportsLikeEscapeClause()

【機能】

"LIKE"節で，エスケープ文字をサポートしているかどうかを返します。

【形式】

```
public boolean supportsLikeEscapeClause() throws SQLException
```

【引数】

なし。

【戻り値】

boolean 型：

true：サポートしています。

false：サポートしていません。

【機能詳細】

常に true を返します。

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(du)　supportsLimitedOuterJoins()

【機能】

外部結合に関し，制限されたサポートがあるかどうかを返します。

【形式】

```
public boolean supportsLimitedOuterJoins() throws SQLException
```

【引数】

なし。

【戻り値】

boolean 型：

true：サポートしています。

false：サポートしていません。

【機能詳細】

常に true を返します。

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(dv)　supportsMinimumSQLGrammar()

【機能】

ODBC Minimum SQL 文法をサポートしているかどうかを返します。

【形式】

```
public boolean supportsMinimumSQLGrammar() throws SQLException
```

【引数】

なし。

【戻り値】

boolean 型：

true：サポートしています。

false：サポートしていません。

【機能詳細】

常に true を返します。

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(dw)　supportsMixedCaseIdentifiers()

【機能】

大文字と小文字が混在する引用符なしの SQL 識別子を，大文字と小文字を区別して処理し，結果として大文字と小文字を混在して格納するかどうかを返します。

【形式】

```
public boolean supportsMixedCaseIdentifiers() throws SQLException
```

【引数】

なし。

【戻り値】

boolean 型：

true：結果として大文字と小文字を混在して格納します。

false：結果として大文字と小文字を混在して格納しません。

【機能詳細】

常に false を返します。

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(dx)　supportsMixedCaseQuotedIdentifiers()

【機能】

大文字と小文字が混在する引用符付きの SQL 識別子を，大文字と小文字を区別して処理し，結果として大文字と小文字を混在して格納するかどうかを返します。

【形式】

```
public boolean supportsMixedCaseQuotedIdentifiers() throws SQLException
```

【引数】

なし。

【戻り値】

boolean 型：

true：結果として大文字と小文字を混在して格納します。

false：結果として大文字と小文字を混在して格納しません。

【機能詳細】

常に true を返します。

注意：

サーバが XDM/RD E2 の場合，XDM/RD E2 の RD 環境定義で，DELIMITED ID UPPER オペランドに YES（デフォルトは NO）を指定しているときは大文字で格納しますが，JDBC ドライバは，DELIMITED ID UPPER オペランドの情報をサーバから取得していないので，デフォルトの NO であると仮定して処理します。

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(dy)　supportsMultipleOpenResults()

【機能】

CallableStatement オブジェクトから同時に返された複数の ResultSet オブジェクトを持つことが可能かどうかを返します。

【形式】

```
public boolean supportsMultipleOpenResults() throws SQLException
```

【引数】

なし。

【戻り値】

boolean 型：

true：CallableStatement オブジェクトが複数の ResultSet オブジェクトを同時に返すことができます。

false：CallableStatement オブジェクトが複数の ResultSet オブジェクトを同時に返すことができません。

【機能詳細】

サーバが HiRDB の場合，true を返します。

サーバが XDM/RD E2 の場合，false を返します。

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(dz)　supportsMultipleResultSets()

【機能】

単一の execute メソッド実行からの複数 ResultSet をサポートしているかどうかを返します。

【形式】

```
public boolean supportsMultipleResultSets() throws SQLException
```

【引数】

なし。

【戻り値】

boolean 型：

true：サポートしています。

false：サポートしていません。

【機能詳細】

常に false を返します。

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(ea)　supportsMultipleTransactions()

【機能】

一度に複数のトランザクションを（異なった接続に関し）オープンできるかどうかを返します。

【形式】

```
public boolean supportsMultipleTransactions() throws SQLException
```

【引数】

なし。

【戻り値】

boolean 型：

true：オープンできます。

false：オープンできません。

【機能詳細】

常に true を返します。

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(eb)　supportsNamedParameters()

【機能】

callable 文への名前付きパラメータがサポートされるかどうかを返します。

【形式】

```
public boolean supportsNamedParameters() throws SQLException
```

【引数】

なし。

【戻り値】

boolean 型：

true：サポートしています。

false：サポートしていません。

【機能詳細】

常に true を返します。

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(ec)　supportsNonNullableColumns()

【機能】

列を非 null として定義できるかどうかを返します。

【形式】

```
public boolean supportsNonNullableColumns() throws SQLException
```

【引数】

なし。

【戻り値】

boolean 型：

true：定義できます。

false：定義できません。

【機能詳細】

常に true を返します。

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(ed)　supportsOpenCursorsAcrossCommit()

【機能】

カーソルをコミット間でオープンされた状態のままにできるかどうかを返します。

【形式】

```
public boolean supportsOpenCursorsAcrossCommit() throws SQLException
```

【引数】

なし。

【戻り値】

boolean 型：

true：カーソルをコミット間でオープンされた状態のままにできます。

false：カーソルをコミット間でオープンされた状態のままにできません。

【機能詳細】

Connection インスタンス生成時のカーソル動作モード（HIRDB_CURSOR）で，ホールダブルカーソルを有効とした場合は true，無効とした場合は false を返します。カーソル動作モードの指定方法については，「ユーザプロパティ」を参照してください。

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(ee)　supportsOpenCursorsAcrossRollback()

【機能】

カーソルをロールバック間でオープンされた状態のままにできるかどうかを返します。

【形式】

```
public boolean supportsOpenCursorsAcrossRollback() throws SQLException
```

【引数】

なし。

【戻り値】

boolean 型：

true：カーソルをロールバック間でオープンされた状態のままにできます。

false：カーソルをロールバック間でオープンされた状態のままにできません。

【機能詳細】

常に false を返します。

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(ef)　supportsOpenStatementsAcrossCommit()

【機能】

文をコミット間でオープンされた状態のままにできるかどうかを返します。

【形式】

```
public boolean supportsOpenStatementsAcrossCommit() throws SQLException
```

【引数】

なし。

【戻り値】

boolean 型：

true：文をコミット間でオープンされた状態のままにできます。

false：文をコミット間でオープンされた状態のままにできません。

【機能詳細】

Connection インスタンス生成時の URL 指定項目 STATEMENT_COMMIT_BEHAVIOR で，TRUE を指定するか同指定項目を省略した場合は true，同指定項目で FALSE を指定した場合は false を返します。URL 指定項目 STATEMENT_COMMIT_BEHAVIOR の指定方法については，「ユーザプロパティ」を参照してください。

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(eg)　supportsOpenStatementsAcrossRollback()

【機能】

文をロールバック間でオープンされた状態のままにできるかどうかを返します。

【形式】

```
public boolean supportsOpenStatementsAcrossRollback() throws SQLException
```

【引数】

なし。

【戻り値】

boolean 型：

true：文をロールバック間でオープンされた状態のままにできます。

false：文をロールバック間でオープンされた状態のままにできません。

【機能詳細】

Connection インスタンス生成時の URL 指定項目 STATEMENT_COMMIT_BEHAVIOR で，TRUE を指定するか同指定項目を省略した場合は true，同指定項目で FALSE を指定した場合は false を返します。URL 指定項目 STATEMENT_COMMIT_BEHAVIOR の指定方法については，「ユーザプロパティ」を参照してください。

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(eh)　supportsOrderByUnrelated()

【機能】

"ORDER BY"節が，SELECT 中にない列を使用できるかどうかを返します。

【形式】

```
public boolean supportsOrderByUnrelated() throws SQLException
```

【引数】

なし。

【戻り値】

boolean 型：

true：使用できます。

false：使用できません。

【機能詳細】

常に treu を返します。

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(ei)　supportsOuterJoins()

【機能】

何らかの外部結合のフォームをサポートしているかどうかを返します。

【形式】

```
public boolean supportsOuterJoins() throws SQLException
```

【引数】

なし。

【戻り値】

boolean 型：

true：サポートしています。

false：サポートしていません。

【機能詳細】

常に true を返します。

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(ej)　supportsPositionedDelete()

【機能】

位置決めされた DELETE をサポートしているかどうかを返します。

【形式】

```
public boolean supportsPositionedDelete() throws SQLException
```

【引数】

なし。

【戻り値】

boolean 型：

true：サポートしています。

false：サポートしていません。

【機能詳細】

常に false を返します。

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(ek)　supportsPositionedUpdate()

【機能】

位置決めされた UPDATE をサポートしているかどうかを返します。

【形式】

```
public boolean supportsPositionedUpdate() throws SQLException
```

【引数】

なし。

【戻り値】

boolean 型：

true：サポートしています。

false：サポートしていません。

【機能詳細】

常に false を返します。

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(el)　supportsResultSetConcurrency(int type, int concurrency)

【機能】

指定された ResultSet タイプと並行処理タイプの組み合わせをサポートしているかどうかを返します。

【形式】

```
public boolean supportsResultSetConcurrency(int type, int concurrency) throws SQLExceptio
n
```

【引数】

int type：

結果セットタイプ java.sql.ResultSet で定義されている型

int concurrency：

並行処理タイプ java.sql.ResultSet で定義されている型

【戻り値】

boolean 型：

true：サポートしています。

false：サポートしていません。

【機能詳細】

type が TYPE_FORWARD_ONLY 又は TYPE_SCROLL_INSENSITIVE，かつ concurrency が CONCUR_READ_ONLY の場合，true を返します。

上記以外は false を返します。

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(em)　supportsResultSetHoldability(int holdability)

【機能】

指定された ResultSet オブジェクトの保持機能を，サポートしているかどうかを返します。

【形式】

```
public boolean supportsResultSetHoldability(int holdability) throws SQLException
```

【引数】

int holdability：

定数。

ResultSet.HOLD_CURSORS_OVER_COMMIT：Connection.commit メソッドが呼び出されたときに ResultSet オブジェクトがクローズされません。

ResultSet.CLOSE_CURSORS_AT_COMMIT：Connection.commit メソッドが呼び出されたときに ResultSet オブジェクトがクローズされます。

【戻り値】

boolean 型：

true：サポートしています。

false：サポートしていません。

【機能詳細】

指定された ResultSet オブジェクトの保持機能を，サポートしているかどうかを返します。次の値を返します。

引数 holdability	HIRDB_CURSOR※	
	有効	無効
HOLD_CURSORS_OVER_COMMIT	true を返します。	false を返します。
CLOSE_CURSORS_AT_COMMIT	false を返します。	true を返します。

注※

Connection インスタンス生成時のカーソル動作モード（HIRDB_CURSOR）

カーソル動作モードの指定方法については，「ユーザプロパティ」を参照してください。

【発生する例外】

次のどちらかの場合，例外 SQLException を投入します。

- このメソッド実行前に Connection オブジェクトに対して close()を実行した場合
- 引数 holdability の指定値が不正な場合

(en)　supportsResultSetType(int type)

【機能】

指定された ResultSet タイプをサポートしているかどうかを返します。

【形式】

```
public boolean supportsResultSetType(int type) throws SQLException
```

【引数】

int type：

結果セットタイプ java.sql.ResultSet で定義されている型結果セットタイプ

【戻り値】

boolean 型：

true：サポートしています。

false：サポートしていません。

【機能詳細】

type が TYPE_FORWARD_ONLY 又は TYPE_SCROLL_INSENSITIVE の場合，true を返します。

上記以外は false を返します。

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(eo)　supportsSavepoints()

【機能】

セーブポイントをサポートしているかどうかを返します。

【形式】

```
public boolean supportsSavepoints() throws SQLException
```

【引数】

なし。

【戻り値】

boolean 型：

true：サポートしています。

false：サポートしていません。

【機能詳細】

HiRDB，XDM/RD E2 ではセーブポイントは未サポートのため，常に false を返します。

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(ep)　supportsSchemasInDataManipulation()

【機能】

データ操作文でスキーマ名を使用できるかどうかを返します。

【形式】

```
public boolean supportsSchemasInDataManipulation() throws SQLException
```

【引数】

なし。

【戻り値】

boolean 型：

true：スキーマ名を使用できます。

false：スキーマ名を使用できません。

【機能詳細】

常に true を返します。

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(eq)　supportsSchemasInIndexDefinitions()

【機能】

インデクス定義文でスキーマ名を使用できるかどうかを返します。

【形式】

```
public boolean supportsSchemasInIndexDefinitions() throws SQLException
```

【引数】

なし。

【戻り値】

boolean 型：

true：スキーマ名を使用できます。

false：スキーマ名を使用できません。

【機能詳細】

常に true を返します。

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(er)　supportsSchemasInPrivilegeDefinitions()

【機能】

特権定義文でスキーマ名を使用できるかどうかを返します。

【形式】

```
public boolean supportsSchemasInPrivilegeDefinitions() throws SQLException
```

【引数】

なし。

【戻り値】

boolean 型：

true：スキーマ名を使用できます。

false：スキーマ名を使用できません。

【機能詳細】

常に true を返します。

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(es)　supportsSchemasInProcedureCalls()

【機能】

プロシジャ呼び出し文でスキーマ名を使用できるかどうかを返します。

【形式】

```
public boolean supportsSchemasInProcedureCalls() throws SQLException
```

【引数】

なし。

【戻り値】

boolean 型：

true：スキーマ名を使用できます。

false：スキーマ名を使用できません。

【機能詳細】

常に true を返します。

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(et)　supportsSchemasInTableDefinitions()

【機能】

テーブル定義文でスキーマ名を使用できるかどうかを返します。

【形式】

```
public boolean supportsSchemasInTableDefinitions() throws SQLException
```

【引数】

なし。

【戻り値】

boolean 型：

true：スキーマ名を使用できます。

false：スキーマ名を使用できません。

【機能詳細】

常に true を返します。

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(eu)　supportsSelectForUpdate()

【機能】

UPDATE 用の SELECT をサポートしているかどうかを返します。

【形式】

```
public boolean supportsSelectForUpdate() throws SQLException
```

【引数】

なし。

【戻り値】

boolean 型：

true：サポートしています。

false：サポートしていません。

【機能詳細】

常に false を返します。

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(ev)　supportsStatementPooling()

【機能】

文のプールをサポートしているかどうかを返します。

【形式】

```
public boolean supportsStatementPooling() throws SQLException
```

【引数】

なし。

【戻り値】

boolean 型：

true：サポートしています。

false：サポートしていません。

【機能詳細】

Type4 JDBC ドライバでは未サポートのため，常に false を返します。

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(ew)　supportsStoredProcedures()

【機能】

ストアドプロシジャコールをサポートしているかどうかを返します。

【形式】

```
public boolean supportsStoredProcedures() throws SQLException
```

【引数】

なし。

【戻り値】

boolean 型：

true：サポートしています。

false：サポートしていません。

【機能詳細】

常に true を返します。

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(ex)　supportsSubqueriesInComparisons()

【機能】

比較式中でサブクエリーをサポートしているかどうかを返します。

【形式】

```
public boolean supportsSubqueriesInComparisons() throws SQLException
```

【引数】

なし。

【戻り値】

boolean 型：

true：スキーマ名を使用できます。

false：スキーマ名を使用できません。

【機能詳細】

常に true を返します。

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(ey)　supportsSubqueriesInExists()

【機能】

"exists"式中でサブクエリーをサポートしているかどうかを返します。

【形式】

```
public boolean supportsSubqueriesInExists() throws SQLException
```

【引数】

なし。

【戻り値】

boolean 型：

true：サポートしています。

false：サポートしていません。

【機能詳細】

常に true を返します。

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(ez)　supportsSubqueriesInIns()

【機能】

"in"文中でサブクエリーをサポートしているかどうかを返します。

【形式】

```
public boolean supportsSubqueriesInIns() throws SQLException
```

【引数】

なし。

【戻り値】

boolean 型：

true：スキーマ名を使用できます。

false：スキーマ名を使用できません。

【機能詳細】

常に true を返します。

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(fa)　supportsSubqueriesInQuantifieds()

【機能】

定量化された式中でサブクエリーをサポートしているかどうかを返します。

【形式】

```
public boolean supportsSubqueriesInQuantifieds() throws SQLException
```

【引数】

なし。

【戻り値】

boolean 型：

true：サポートしています。

false：サポートしていません。

【機能詳細】

常に true を返します。

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(fb)　supportsTableCorrelationNames()

【機能】

テーブル相互関連名をサポートしているかどうかを返します。

【形式】

```
public boolean supportsTableCorrelationNames() throws SQLException
```

【引数】

なし。

【戻り値】

boolean 型：

true：スキーマ名を使用できます。

false：スキーマ名を使用できません。

【機能詳細】

常に true を返します。

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(fc)　supportsTransactionIsolationLevel(int level)

【機能】

与えられたトランザクションアイソレーションレベルをサポートしているかどうかを返します。

【形式】

```
public boolean supportsTransactionIsolationLevel(int level) throws SQLException
```

【引数】

int level：

java.sql.Connection で定義されているトランザクション遮断レベル

【戻り値】

boolean 型：

true：サポートしています。

false：サポートしていません。

【機能詳細】

level が TRANSACTION_REPEATABLE_READ，TRANSACTION_READ_COMMITTED，TRANSACTION_READ_UNCOMMITTED の場合に true を返します。

上記以外の場合は false を返します。

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(fd)　supportsTransactions()

【機能】

トランザクションをサポートしているかどうかを返します。

【形式】

```
public boolean supportsTransactions() throws SQLException
```

【引数】

なし。

【戻り値】

boolean 型：

true：スキーマ名を使用できます。

false：スキーマ名を使用できません。

【機能詳細】

常に true を返します。

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(fe)　supportsUnion()

【機能】

SQL UNION をサポートしているかどうかを返します。

【形式】

```
public boolean supportsUnion() throws SQLException
```

【引数】

なし。

【戻り値】

boolean 型：

true：サポートしています。

false：サポートしていません。

【機能詳細】

常に true を返します。

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(ff)　supportsUnionAll()

【機能】

SQL UNION ALL をサポートしているかどうかを返します。

【形式】

```
public boolean supportsUnionAll() throws SQLException
```

【引数】

なし。

【戻り値】

boolean 型：

true：サポートしています。

false：サポートしていません。

【機能詳細】

常に true を返します。

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(fg)　updatesAreDetected(int type)

【機能】

指定された ResultSet タイプで，ResultSet に行われた更新を ResultSet.rowUpdated メソッドによって検出できるかどうかを返します。

【形式】

```
public boolean updatesAreDetected(int type) throws SQLException
```

【引数】

int type：

ResultSet の型。次のどれかです。

- ResultSet.TYPE_FORWARD_ONLY
- ResultSet.TYPE_SCROLL_INSENSITIVE
- ResultSet.TYPE_SCROLL_SENSITIVE

【戻り値】

boolean 型：

true：検出できます。

false：検出できません。

【機能詳細】

常に false を返します。

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(fh)　usesLocalFilePerTable()

【機能】

各テーブルにファイルを使用するかどうかを返します。

【形式】

```
public boolean usesLocalFilePerTable() throws SQLException
```

【引数】

なし。

【戻り値】

boolean 型：

true：使用します。

false：使用しません。

【機能詳細】

常に false を返します。

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(fi)　usesLocalFiles()

【機能】

ローカルファイルにテーブルを格納するかどうかを返します。

【形式】

```
public boolean usesLocalFiles() throws SQLException
```

【引数】

なし。

【戻り値】

boolean 型：

true：格納します。

false：格納しません。

【機能詳細】

常に false を返します。

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(fj)　getRowIdLifetime()

【機能】

RowId 型をサポートしているかどうかを返します。RowId 型をサポートしている場合は，RowId オブジェクトが有効になっている寿命を返します。

【形式】

```
public RowIdLifetime getRowIdLifetime() throws SQLException
```

【引数】

なし。

【戻り値】

RowIdLifeTime 型：

ROWID_UNSUPPORTED：ROWID 型をサポートしていません。

ROWID_VALID_OTHER：ROWID の寿命が不確定です。

ROWID_VALID_SESSION：ROWID の寿命がセッション中です。

ROWID_VALID_TRANSACTION：ROWID の寿命がトランザクション中です。

ROWID_VALID_FOREVER：ROWID の寿命が無期限です。

【機能詳細】

常に ROWID_UNSUPPORTED を返します。

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(fk)　getSchemas(String catalog, String schemaPattern)

【機能】

使用できるスキーマ名に関する情報を返します。

【形式】

```
public ResultSet getSchemas(String catalog,String schemaPattern) throws SQLException
```

【引数】

String catalog：

カタログ名。このドライバでは無視します。

String schemaPattern：

スキーマ名パターン。null 又は 0 長文字を指定している場合，スキーマ名を検索の限定に使用しません。引数の値は大文字と小文字を区別します。

【戻り値】

ResultSet オブジェクト

【機能詳細】

HiRDB サーバに存在するすべてのスキーマ名に関する情報を返します。システム共通定義に pd_security_host_group を指定し，指定値に UAP が動作するクライアントマシンのホストが含まれていない場合，SQLException を投入します。

接続先が XDM/RD E2 の場合，表を所有するスキーマ名だけを返します。

返却する ResultSet の形式を次の表に示します。

表 17–58　返却する ResultSet の形式（getSchemas メソッド）

型	SQL 型	列名	列の意味
String	VARCHAR	TABLE_SCHEM	認可識別子名。
String	CHAR	TABLE_CATALOG	常に NULL 値を返します。

【発生する例外】

次の場合，SQLException を投入します。

- このメソッド実行前に Connection オブジェクトに対して close()を実行している
- データベースアクセスエラーが発生した
- システム共通定義に pd_security_host_group を指定し，指定値に UAP が動作するクライアントマシンのホストが含まれていない

(fl)　supportsStoredFunctionsUsingCallSyntax()

【機能】

プロシジャエスケープ構文を使用した，ユーザ定義関数又はベンダー関数の呼び出しをサポートするかどうかを返します。

【形式】

```
public boolean supportsStoredFunctionsUsingCallSyntax() throws SQLException
```

【引数】

なし。

【戻り値】

boolean 型：

true：サポートします。

false：サポートしません。

【機能詳細】

常に false を返します。

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(fm)　autoCommitFailureClosesAllResultSets()

【機能】

オートコミット有効時に SQLException が発生した場合，オープンされたすべての ResultSet がクローズされるかどうかを返します。

【形式】

```
public boolean autoCommitFailureClosesAllResultSets() throws SQLException
```

【引数】

なし。

【戻り値】

boolean 型：

true：クローズします。

false：クローズしません。

【機能詳細】

常に false を返します。

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(fn)　getClientInfoProperties()

【機能】

JDBC ドライバがサポートするクライアント情報プロパティのリスト情報を取得します。

【形式】

```
public ResultSet getClientInfoProperties() throws SQLException
```

【引数】

なし。

【戻り値】

ResultSet オブジェクト

【機能詳細】

JDBC ドライバがサポートするクライアント情報プロパティのリスト情報を返します。常に検索結果行数 0 の ResultSet を返します。返却する ResultSet の形式を次の表に示します。

表 17–59　返却する ResultSet の形式（getClientInfoProperties メソッド）

列番号	型	SQL 型	列名	列の意味
1	String	CHAR	NAME	クライアント情報プロパティの名前
2	int	INTEGER	MAX_LEN	プロパティの値の最大長
3	String	CHAR	DEFAULT_VALUE	プロパティのデフォルト値
4	String	CHAR	DESCRIPTION	プロパティの記述

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(fo)　getFunctions(String catalog, String schemaPattern, String functionNamePattern)

【機能】

関数に関する情報を返します。

【形式】

```
public ResultSet getFunctions(String catalog,
                              String schemaPattern,
                              String functionNamePattern)
                              throws SQLException
```

【引数】

String catalog：

カタログ名。このドライバでは無視します。

String schemaPattern：

スキーマ名パターン。null 又は 0 長文字を指定している場合，スキーマ名を検索の限定に使用しません。引数の値は大文字と小文字を区別します。

String functionNamePattern：

関数名パターン。関数名と一致しなければなりません。引数の値は大文字と小文字を区別します。

【戻り値】

ResultSet オブジェクト

【機能詳細】

指定されたユーザ定義関数に関する情報を返します。HiRDB サーバに SQL_ROUTINES 表が定義されていない場合，検索結果行数 0 の ResultSet を返します。返却する ResultSet の形式を次の表に示します。

表 17–60　返却する ResultSet の形式（getFunctions メソッド）

列番号	型	SQL 型（Types）	列名	ソート※	列の意味
1	String	CHAR	FUNCTION_CAT	–	カタログ名 常に null 値を返します。
2	String	VARCHAR	FUNCTION_SCHEM	1（昇順）	認可識別子名
3	String	VARCHAR	FUNCTION_NAME	2（昇順）	関数名
4	String	VARCHAR	REMARKS	–	関数に関する説明 常に null 値を返します。
5	Short	SMALLINT	FUNCTION_TYPE	–	関数の種類 • functionResultUnknown 戻り値又はテーブルが返されるかどうか判断できません。 • functionNoTable テーブルが返されません。 • functionReturnsTable テーブルが返されます。 常に functionNoTable を返します。
6	String	VARCHAR	SPECIFIC_NAME	3（昇順）	この関数をそのスキーマ内で一意に識別する名前

（凡例）

–：該当しません。

注※

ソートのキー列の優先順位を示します。

【発生する例外】

次の場合，SQLException を投入します。

- このメソッド実行前に Connection オブジェクトに対して close()を実行している場合
- データベースアクセスエラーが発生した場合

(fp) getFunctionColumns(String catalog,String schemaPattern,String functionNamePattern,String columnNamePattern)

【機能】

関数のパラメタと返される型に関する情報を返します。

【形式】

```
public ResultSet getFunctionColumns(String catalog,
                                    String schemaPattern,
                                    String functionNamePattern,
                                    String columnNamePattern)
                                    throws SQLException
```

【引数】

String catalog：

カタログ名。このドライバでは無視します。

String schemaPattern：

スキーマ名パターン。null 又は 0 長文字を指定している場合，スキーマ名を検索の限定に使用しません。引数の値は大文字と小文字を区別します。

String functionNamePattern：

関数名パターン。null 又は 0 長文字を指定している場合，関数名を検索の限定に使用しません。引数の値は大文字と小文字を区別します。

String columnNamePattern：

パラメタ名パターン。null 又は 0 長文字を指定している場合，パラメタ名を検索の限定に使用しません。引数の値は大文字と小文字を区別します。

【戻り値】

ResultSet オブジェクト

【機能詳細】

指定されたユーザ定義関数のパラメタと返される型に関する情報を返します。HiRDB サーバに SQL_ROUTINE_PARAMS 表が定義されていない場合，検索結果行数 0 の ResultSet を返します。返却する ResultSet の形式を次の表に示します。

表 17–61　返却する ResultSet の形式（getFunctionColumns メソッド）

列番号	型	SQL 型（Types）	列名	ソート※	列の意味
1	String	CHAR	FUNCTION_CAT	−	カタログ名 常に null 値を返します。
2	String	VARCHAR	FUNCTION_SCHEM	1（昇順）	認可識別子名
3	String	VARCHAR	FUNCTION_NAME	2（昇順）	関数名
4	String	VARCHAR	COLUMN_NAME	−	列/パラメタ名

列番号	型	SQL 型（Types）	列名	ソート※	列の意味
5	short	SMALLINT	COLUMN_TYPE	−	列の種類/パラメタ • functionColumnUnknown 不明 • functionColumnIn IN パラメタ • functionColumnInOut INOUT パラメタ • functionColumnOut OUT パラメタ • functionReturn 関数の戻り値 • functionColumnResult パラメタ又は列が ResultSet の列 functionColumnIn，functionColumnInOut，functionColumnOut，及びfunctionColumnResult については設定しません。
6	int	INTEGER	DATA_TYPE	−	パラメタの SQL 型（java.sql.Types で定義されている値）
7	String	CHAR	TYPE_NAME	−	パラメタの SQL 型名（型名を文字列で表現したもの）
8	int	INTEGER	PRECISION	−	パラメタの精度 • 数値データの場合 最大精度 • 文字データの場合 文字数 • 日時データ型の場合 String 表現の文字数 • バイナリデータの場合 バイト数
9	int	INTEGER	LENGTH	−	パラメタのサイズ バイト数を返します。 DECIMAL 型は精度に符号とピリオドを加えるため，位取りなしの場合は精度+1，位取りありの場合は精度+2 となります。
10	short	SMALLINT	SCALE	−	パラメタの位取り（小数部分のけた数） パラメタの SQL 型名が DECIMAL 又は TIMESTAMP 以外の場合は，0 を返します。
11	short	SMALLINT	RADIX	−	パラメタの基数

列番号	型	SQL 型（Types）	列名	ソート※	列の意味
					• 概数値：2 • 真数値：10 • 数値以外：0
12	short	SMALLINT	NULLABLE	−	null 値可否 • functionNoNulls null 値を許しません。 • functionNullable null 値を許します。 • functionNullableUnknown null 値を許すかどうかは不明です。 常に，functionNullable を返します。
13	String	VARCHAR	REMARKS	−	パラメタに関するコメント 常に null 値を返します。
14	int	INTEGER	CHAR_OCTET_LENGTH	−	バイナリと文字ベースのパラメタ又は列の最大長
15	int	INTEGER	ORDINAL_POSITION	4（昇順）	入力及び出力パラメタの 1 から始まる順番 関数の戻り値の場合は 0 を返します。
16	String	CHAR	IS_NULLABLE	−	パラメタ又は列で null 値を許可するかどうか • YES null 値を許します。 • NO null 値を許しません。 • 空白 null 値を許すかどうかは不明です。 常に YES を返します。
17	String	VARCHAR	SPECIFIC_NAME	3（昇順）	この関数をスキーマ内で一意に識別する名前

（凡例）

−：該当しません。

注※

ソートのキー列の優先順位を示します。

【発生する例外】

次の場合，SQLException を投入します。

- このメソッド実行前に Connection オブジェクトに対して close()を実行している場合
- データベースアクセスエラーが発生した場合

(fq)　getPseudoColumns(String catalog, String schemaPattern, String tableNamePattern, String columnNamePattern)

【機能】

擬似列，隠し列の説明を返します。

【形式】

```
public ResultSet getPseudoColumns(String catalog,
                                  String schemaPattern,
                                  String tableNamePattern,
                                  String columnNamePattern)
                                  throws SQLException
```

【引数】

String catalog：

カタログ名

String schemaPattern：

スキーマ名パターン

String tableNamePattern：

表名パターン

String columnNamePattern：

パラメタ名パターン

【戻り値】

ResultSet オブジェクト

【機能詳細】

常に検索結果行数 0 の ResultSet を返します。返却する ResultSet の形式を次の表に示します。

表 17–62　返却する ResultSet の形式（getPseudoColumns メソッド）

列番号	型	SQL 型（Types）	列名	列の意味
1	String	CHAR	TABLE_CAT	カタログ名
2	String	VARCHAR	TABLE_SCHEM	認可識別子名
3	String	VARCHAR	TABLE_NAME	表名
4	String	VARCHAR	COLUMN_NAME	列名
5	int	INTEGER	DATA_TYPE	パラメタの SQL 型
6	int	INTEGER	COLUMN_SIZE	列の定義長
7	int	INTEGER	DECIMAL_DIGITS	小数点以下の桁数
8	int	INTEGER	NUM_PREC_RADIX	パラメタの基数
9	String	VARCHAR	COLUMN_USAGE	列の許可された使用法

列番号	型	SQL 型（Types）	列名	列の意味
10	String	VARCHAR	REMARKS	列を記述するコメント
11	int	INTEGER	CHAR_OCTET_LENGTH	バイナリと文字ベースのパラメタ又は列の最大長(バイト数)
12	String	CHAR	IS_NULLABLE	パラメタ又は列で null 値を許可するかどうか

【発生する例外】

次の場合，SQLException を投入します。

- このメソッド実行前に Connection オブジェクトに対して close()を実行している場合
- データベースアクセスエラーが発生した場合

(fr)　generatedKeyAlwaysReturned()

【機能】

自動生成キーの列を表す指定された列名又はインデックスが有効であり，SQL 文が成功した場合に生成されたキーが常に返されるかどうかを返します。

【形式】

```
public boolean generatedKeyAlwaysReturned() throws SQLException
```

【引数】

なし。

【戻り値】

boolean 型：

true：キーを返します。

false：キーを返しません。

【機能詳細】

常に false を返します。

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(fs)　getMaxLogicalLobSize()

【機能】

このデータベースで LOB の論理サイズとして許容される最大バイト数を返します。

【形式】

```
public boolean getMaxLogicalLobSize() throws SQLException
```

【引数】

なし。

【戻り値】

最大バイト数

【機能詳細】

常に 2147483647 を返します。

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(ft)　supportsRefCursors()

【機能】

REF CURSOR のサポート有無を返却します。

【形式】

```
public boolean supportsRefCursors() throws SQLException
```

【引数】

なし。

【戻り値】

boolean 型：

true：サポートします。

false：サポートしません。

【機能詳細】

常に false を返します。

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(fu)　supportsSharding()

【機能】

シャーディングのサポート有無を返却します。

【形式】

```
public boolean supportsSharding() throws SQLException
```

【引数】

なし。

【戻り値】

boolean 型：

true：サポートします。

false：サポートしません。

【機能詳細】

常に false を返します。

【発生する例外】

このメソッド実行前に Connection オブジェクトに対して close()を実行している場合，SQLException を投入します。

(3)　パッケージ名称及びクラス名称

このインタフェースを実装するパッケージ名称とクラス名称を次に示します。

パッケージ名称：**JP.co.Hitachi.soft.HiRDB.JDBC**

クラス名称：**PrdbDatabaseMetaData**

(4)　注意事項

(a)　1 コネクション当たりのステートメントオブジェクト数の上限

ResultSet を返却する DatabaseMetaData のメソッドでは，内部的にステートメントオブジェクトを使用しているため，ステートメントオブジェクト数が 4096 以上となった場合，SQLException を投入します。詳細は「Statement インタフェース」の「注意事項」を参照してください。

17.4.8　ResultSetMetaData インタフェース

(1)　概要

ResultSetMetaData インタフェースでは，主に次の機能が提供されます。

- ResultSet（結果セット）の各列に対する，データ型及びデータ長などのメタ情報の返却

(2)　メソッド

ResultSetMetaData インタフェースのメソッド一覧を次の表に示します。なお，表に記載されていないメソッドはサポートしていません。サポートしていないメソッドを指定すると，SQLException を投入します。

表 17–63　ResultSetMetaData インタフェースのメソッド一覧

メソッド	提供ドライバ			機能
	JDBC2.0	JDBC4.0	JDBC4.3	
getCatalogName(int column)	○	○	○	指定された列番号のテーブルでのカタログ名を返します。
getColumnClassName(int column)	○	○	○	カラムに対する Java クラスの完全指定された名前を返します。
getColumnCount()	○	○	○	この ResultSet オブジェクトの列数を返します。
getColumnDisplaySize(int column)	○	○	○	指定された列の通常の最大幅を，文字数で返します。
getColumnLabel(int column)	○	○	○	印刷や表示に使用する列の推奨タイトルを返します。
getColumnName(int column)	○	○	○	指定した列の名前を返します。
getColumnType(int column)	○	○	○	指定した列の SQL データ型を返します。
getColumnTypeName(int column)	○	○	○	指定された列の，データベース固有の形名を返します。
getPrecision(int column)	○	○	○	指定された列のサイズを返します。
getScale(int column)	○	○	○	指定された列の，小数点以下のけた数を返します。
getSchemaName(int column)	○	○	○	指定された列のテーブルスキーマを返します。
getTableName(int column)	○	○	○	指定された列のテーブル名を返します。
isAutoIncrement(int column)	○	○	○	指定された列が自動的に番号付けされて，読み込み専用として扱われるかどうかを返します。
isCaseSensitive(int column)	○	○	○	列の大文字と小文字が区別されるかどうかを返します。
isCurrency(int column)	○	○	○	指定された列が，通貨の値かどうか返します。
isDefinitelyWritable(int column)	○	○	○	指定された列の書き込みが，成功するかどうかを返します。
isNullable(int column)	○	○	○	指定した列に NULL をセットできるかどうかを返します。
isReadOnly(int column)	○	○	○	指定された列が，読み取り専用かどうかを返します。

メソッド	提供ドライバ			機能
	JDBC2.0	JDBC4.0	JDBC4.3	
isSearchable(int column)	○	○	○	指定された列を where 節で使用できるかどうかを返します。
isSigned(int column)	○	○	○	指定された列の値が，符号付数値かどうかを返します。
isWritable(int column)	○	○	○	指定された列への書き込みを，成功させることができるかどうかを返します。

（凡例）

○：提供されます。

(a)　getCatalogName(int column)

【機能】

指定された列番号のテーブルでのカタログ名を返します。

【形式】

```
public synchronized String getCatalogName(int column) throws SQLException
```

【引数】

int column：

1 から始まる列番号

【戻り値】

String オブジェクト

【機能詳細】

常に null を返します。

【発生する例外】

column 指定値が 0 以下，又は列数より大きい場合，SQLException を投入します。

(b)　getColumnClassName(int column)

【機能】

カラムに対する Java クラスの完全指定された名前を返します。

【形式】

```
public synchronized String getColumnClassName(int column) throws SQLException
```

【引数】

int column：

1 から始まる列番号

【戻り値】

String オブジェクト

【機能詳細】

列に対して ResultSet#getObject メソッドを実行した結果，返却する Java クラスの型を String 型で返します。列のデータ型と返却値を次の表に示します。ただし，繰返し列の場合は"java.sql.Array"を返却します。

表 17–64　getColumnClassName で返却する文字列

列の HiRDB のデータ型	返却する文字列
BLOB	"java.lang.Object"
BINARY	"java.lang.Object"
INTEGER	"java.lang.Integer"
SMALLINT	"java.lang.Integer"
FLOAT DOUBLE PRECISION	"java.lang.Double"
SMALLFLT REAL	"java.lang.Float"
DECIMAL NUMERIC	"java.math.BigDecimal"
CHAR	"java.lang.String"
MCHAR	"java.lang.String"
NCHAR	"java.lang.String"
VARCHAR	"java.lang.String"
MVARCHAR	"java.lang.String"
NVARCHAR	"java.lang.String"
DATE	"java.sql.Date"
TIME	"java.sql.Time"
TIMESTAMP	"java.sql.Timestamp"
BOOLEAN（DatabaseMetaData から生成した ResultSet にだけ存在する列）	"java.lang.Boolean"

【発生する例外】

column 指定値が 0 以下，又は列数より大きい場合，SQLException を投入します。

(c)　getColumnCount()

【機能】

この ResultSet オブジェクトの列数を返します。

【形式】

```
public synchronized int getColumnCount()
```

【引数】

なし。

【戻り値】

int 型：

列数

【発生する例外】

なし。

(d)　getColumnDisplaySize(int column)

【機能】

指定された列の通常の最大幅を，文字数で返します。

【形式】

```
public synchronized int getColumnDisplaySize(int column) throws SQLException
```

【引数】

int column：

1 から始まる列番号

【戻り値】

int 型：

最大文字数

【機能詳細】

指定された列の通常の最大幅を，文字数で返します。getColumnDisplaySize で返却する最大文字数を次の表に示します。

表 17–65　getColumnDisplaySize で返却する最大文字数

列の HiRDB のデータ型	返却する最大文字数
BLOB(n) BINARY(n)	n
INTEGER	11
SMALLINT	6

列の HiRDB のデータ型	返却する最大文字数
FLOAT DOUBLE PRECISION	23
SMALLFLT REAL	13
DECIMAL(n,m) NUMERIC(n,m)	n ＋ 2
CHAR(n) MCHAR(n) NCHAR(n) VARCHAR(n) MVARCHAR(n) NVARCHAR(n)	n
DATE	10
TIME	8
TIMESTAMP(n)	n ＞ 0 の場合：19 ＋(n ＋ 1) n ＝ 0 の場合：19
BOOLEAN（DatabaseMetaData から生成した ResultSet にだけ存在する列）	5

【発生する例外】

column 指定値が 0 以下，又は列数より大きい場合，SQLException を投入します。

(e)　getColumnLabel(int column)

【機能】

印刷や表示に使用する列の推奨タイトルを返します。

PrdbResultSetMetaData#getColumnName での返却値と同じ値（列名）を返します。詳細については，「getColumnName(int column)」を参照してください。

【形式】

```
public synchronized String getColumnLabel(int column) throws SQLException
```

【引数】

int column：

1 から始まる列番号

【戻り値】

String オブジェクト

【発生する例外】

column 指定値が 0 以下，又は列数より大きい場合，SQLException を投入します。

(f)　getColumnName(int column)

【機能】

指定した列の名前を返します。

【形式】

```
public synchronized String getColumnName(int column) throws SQLException
```

【引数】

int column：

1 から始まる列番号

【戻り値】

String オブジェクト

【機能詳細】

指定した列の名前を，HiRDB の列名記述領域の内容で返します。HiRDB サーバからこのドライバに送られる，列名記述領域（SQLCNDA）中の SQLNAME から，検索項目の名称を取得（かつ Java の内部コードに変換）して返します。ただし，SELECT 文で集合関数，値式，WRITE 指定を使用した場合，28 バイトまで返し，29 バイト目以降は，サーバ，クライアント間の通信時に切り捨てられます。また，HiRDB サーバから受け取った次の先頭文字を，このドライバは無視します。

- △△
- △■

（凡例）△：半角空白　■：0xff

指定した列に対するこのメソッドの返却値の内容については，「列名記述領域の構成と内容」を参照してください。

Array オブジェクトから取得した ResultSet クラスの 1 列目は，JDBC ドライバが生成する列であり，列名として"JDBC_Array_Index"を返します。

【発生する例外】

column 指定値が 0 以下，又は列数より大きい場合，SQLException を投入します。

(g)　getColumnType(int column)

【機能】

指定した列の SQL データ型を返します。列のデータ型と返却値の対応については，「SQL データ型のマッピング」を参照してください。ただし，繰返し列の場合は"java.sql.Array"を返却します。

【形式】

```
public synchronized int getColumnType(int column) throws SQLException
```

【引数】

int column：

1 から始まる列番号

【戻り値】

int 型：

java.sql.Types からの SQL 型

【発生する例外】

column 指定値が 0 以下，又は列数より大きい場合，SQLException を投入します。

(h)　getColumnTypeName(int column)

【機能】

指定された列の，データベース固有の形名を返します。

【形式】

```
public synchronized String getColumnTypeName(int column) throws SQLException
```

【引数】

int column：

1 から始まる列番号

【戻り値】

String オブジェクト

【機能詳細】

指定された列の，データベース固有の形名を返します。列のデータ型と返却値を次の表に示します。

表 17–66　getColumnTypeName で返却する文字列

列の HiRDB のデータ型	返却する文字列
BLOB	"BLOB"
BINARY	"BINARY"
INTEGER	"INTEGER"
SMALLINT	"SMALLINT"
FLOAT	"FLOAT"
REAL	"REAL"
DECIMAL	"DECIMAL"
CHAR	"CHAR"
MCHAR	"MCHAR"
NCHAR	"NCHAR"

列の HiRDB のデータ型	返却する文字列
VARCHAR	"VARCHAR"
MVARCHAR	"MVARCHAR"
NVARCHAR	"NVARCHAR"
DATE	"DATE"
TIME	"TIME"
TIMESTAMP	"TIMESTAMP"
BOOLEAN（DatabaseMetaData から生成した ResultSet にだけ存在する列）	"BOOLEAN"

【発生する例外】

column 指定値が 0 以下，又は列数より大きい場合，SQLException を投入します。

(i)　getPrecision(int column)

【機能】

指定された列のサイズを返します。

【形式】

```
public synchronized int getPrecision(int column) throws SQLException
```

【引数】

int column：

1 から始まる列番号

【戻り値】

int 型：

列のサイズ

【機能詳細】

指定された列のサイズを返します。返却する値を次の表に示します。

表 17–67　getPrecision の返却値

列の HiRDB のデータ型	返却する値	返却値の単位
BLOB(n)，BINARY(n)	n	バイト数
INTEGER	10	10 進けた数
SMALLINT	5	10 進けた数
FLOAT，DOUBLE PRECISION	15	10 進けた数
SMALLFLT，REAL	7	10 進けた数

列の HiRDB のデータ型	返却する値	返却値の単位
DECIMAL(n,m)，NUMERIC(n,m)	n	10 進けた数
CHAR(n)，MCHAR(n)，VARCHAR(n)，MVARCHAR(n)	n	文字数
NCHAR(n)，NVARCHAR(n)	JDBC2.0：n×2	バイト数
	JDBC4.0 以降：n	文字数
DATE	10	文字数
TIME	8	文字数
TIMESTAMP(n)	n ＞ 0 の場合：19 ＋（n ＋ 1） n ＝ 0 の場合：19	文字数
BOOLEAN（DatabaseMetaData から生成した ResultSet にだけ存在する列）	1	バイト数

【発生する例外】

column 指定値が 0 以下，又は列数より大きい場合，SQLException を投入します。

(j)　getScale(int column)

【機能】

指定された列の，小数点以下のけた数を返します。

【形式】

```
public synchronized int getScale(int column) throws SQLException
```

【引数】

int column：

1 から始まる列番号

【戻り値】

int 型：

列の 10 進けた数

【機能詳細】

指定された列の，小数点以下のけた数を返します。列のデータ型と返却値を次の表に示します。

表 17–68　getScale で返却する値

列の HiRDB のデータ型	返却する値
DECIMAL NUMERIC	小数点以下のけた数
TIMESTAMP	ミリ秒以下のけた数

列の HiRDB のデータ型	返却する値
上記以外	0

【発生する例外】

column 指定値が 0 以下，又は列数より大きい場合，SQLException を投入します。

(k) getSchemaName(int column)

【機能】

指定された列のテーブルスキーマを返します。

【形式】

```
public synchronized String getSchemaName(int column) throws SQLException
```

【引数】

int column：

1 から始まる列番号

【戻り値】

String オブジェクト

【機能詳細】

常に null を返します。

【発生する例外】

column 指定値が 0 以下，又は列数より大きい場合，SQLException を投入します。

(l) getTableName(int column)

【機能】

指定された列のテーブル名を返します。

【形式】

```
public synchronized String getTableName(int column) throws SQLException
```

【引数】

int column：

1 から始まる列番号

【戻り値】

String オブジェクト

【機能詳細】

常に null を返します。

【発生する例外】

column 指定値が 0 以下，又は列数より大きい場合，SQLException を投入します。

(m)　isAutoIncrement(int column)

【機能】

指定された列が自動的に番号付けされて，読み込み専用として扱われるかどうかを返します。

【形式】

```
public synchronized boolean isAutoIncrement(int column) throws SQLException
```

【引数】

int column：

1 から始まる列番号

【戻り値】

boolean 型：

true：指定された列は自動的に番号付けされて，読み込み専用として扱われます。

false：指定された列は自動的に番号付けされないか，又は読み込み専用として扱われません。

【機能詳細】

常に false を返します。

【発生する例外】

column 指定値が 0 以下，又は列数より大きい場合，SQLException を投入します。

(n)　isCaseSensitive(int column)

【機能】

列の大文字と小文字が区別されるかどうかを返します。

【形式】

```
public synchronized boolean isCaseSensitive(int column) throws SQLException
```

【引数】

int column：

1 から始まる列番号

【戻り値】

boolean 型：

true：列の大文字と小文字が区別されます。

false：列の大文字と小文字が区別されません。

【機能詳細】

常に false を返します。

【発生する例外】

column 指定値が 0 以下，又は列数より大きい場合，SQLException を投入します。

(o) isCurrency(int column)

【機能】

指定された列が，通貨の値かどうか返します。

【形式】

```
public synchronized boolean isCurrency(int column) throws SQLException
```

【引数】

int column：

1 から始まる列番号

【戻り値】

boolean 型：

true：列が通貨の値を示します。

false：列が通貨の値を示しません。

【機能詳細】

常に false を返します。

【発生する例外】

column 指定値が 0 以下，又は列数より大きい場合，SQLException を投入します。

(p) isDefinitelyWritable(int column)

【機能】

指定された列の書き込みが，成功するかどうかを返します。

【形式】

```
public synchronized boolean isDefinitelyWritable(int column) throws SQLException
```

【引数】

int column：

1 から始まる列番号

【戻り値】

boolean 型：

true：列の書き込みが成功します。

false：列の書き込みが成功するとは限りません。

【機能詳細】

常に false を返します。

【発生する例外】

column 指定値が 0 以下，又は列数より大きい場合，SQLException を投入します。

(q)　isNullable(int column)

【機能】

指定した列に NULL をセットできるかどうかを返します。

【形式】

```
public synchronized int isNullable(int column) throws SQLException
```

【引数】

int column：

1 から始まる列番号

【戻り値】

int 型：

ResultSetMetaData.columnNoNulls：NULL 値をセットできません。

ResultSetMetaData.columnNullable：NULL 値をセットできます。

【発生する例外】

column 指定値が 0 以下，又は列数より大きい場合，SQLException を投入します。

(r)　isReadOnly(int column)

【機能】

指定された列が，読み取り専用かどうかを返します。

【形式】

```
public synchronized boolean isReadOnly(int column) throws SQLException
```

【引数】

int column：

1 から始まる列番号

【戻り値】

boolean 型：

true：読み取り専用です。

false：読み取り専用とは限りません。

【機能詳細】

常に false を返します。

【発生する例外】

column 指定値が 0 以下，又は列数より大きい場合，SQLException を投入します。

(s)　isSearchable(int column)

【機能】

指定された列を where 節で使用できるかどうかを返します。

【形式】

```
public synchronized boolean isSearchable(int column) throws SQLException
```

【引数】

int column：

1 から始まる列番号

【戻り値】

boolean 型：

true：where 節で使用できます。

false：where 節で使用できません。

【機能詳細】

DatabaseMetaData から生成した ResultSet の場合，false を返します。

DatabaseMetaData から生成した ResultSet ではない場合，true を返します。

【発生する例外】

column 指定値が 0 以下，又は列数より大きい場合，SQLException を投入します。

(t)　isSigned(int column)

【機能】

指定された列の値が，符号付数値かどうかを返します。

【形式】

```
public synchronized boolean isSigned(int column) throws SQLException
```

【引数】

int column：

1 から始まる列番号

【戻り値】

boolean 型：

true：符号付数値です。

false：符号付数値ではありません。

【機能詳細】

指定された列の値が，符号付数値かどうかを返します。列のデータ型と返却値を次の表に示します。

表 17–69　isSigned で返却する値

列の HiRDB のデータ型	返却する値
INTEGER SMALLINT FLOAT DOUBLE PRECISION REAL SMALLFLT DECIMAL NUMERIC	true
上記以外	false

【発生する例外】

column 指定値が 0 以下，又は列数より大きい場合，SQLException を投入します。

(u)　isWritable(int column)

【機能】

指定された列への書き込みを，成功させることができるかどうかを返します。

【形式】

```
public synchronized boolean isWritable(int column) throws SQLException
```

【引数】

int column：

1 から始まる列番号

【戻り値】

boolean 型：

true：書き込みを成功させることができます。

false：書き込みを成功させることができません。

【機能詳細】

常に false を返します。

【発生する例外】

column 指定値が 0 以下，又は列数より大きい場合，SQLException を投入します。

(3)　パッケージ名称及びクラス名称

このインタフェースを実装するパッケージ名称とクラス名称を次に示します。

パッケージ名称：**JP.co.Hitachi.soft.HiRDB.JDBC**

クラス名称：**PrdbResultSetMetaData**

(4)　注意事項

(a)　getColumnName，getColumnLabel メソッド

getColumnName，及び getColumnLabel メソッドは，HiRDB サーバから JDBC ドライバに送られる列名記述領域（SQLCNDA）中の SQLNAME から検索項目の名称を取得し，Java の内部コードに変換して返却します。指定した列に対するこれらのメソッドの返却値の内容は，「列名記述領域の構成と内容」を参照してください。

17.4.9　Blob インタフェース

(1)　概要

Blob インタフェースでは，主に次の機能が提供されます。

- バイナリデータの取得
- バイナリデータ長の取得
- パターンに一致する位置の取得

JDBC ドライバは，Blob インタフェースを PrdbBlob クラスで実装します。

PrdbBlob クラスのオブジェクトは，ResultSet や CallableStatement の getBlob メソッドの返却値として，JDBC ドライバが生成します。

(2)　メソッド

Blob インタフェースのメソッド一覧を次の表に示します。なお，表に記載されていないメソッドはサポートしていません。サポートしていないメソッドを指定すると，SQLException を投入します。

表 17–70　Blob インタフェースのメソッド一覧

メソッド	提供ドライバ			機能
	JDBC2.0	JDBC4.0	JDBC4.3	
getBinaryStream()	○	○	○	BLOB 又は BINARY 値をストリーム（PrdbDataStream オブジェクト）として返します。
getBytes(long pos, int length)	○	○	○	BLOB 又は BINARY 値の全部又は一部をバイト配列として返します。

メソッド	提供ドライバ			機能
	JDBC2.0	JDBC4.0	JDBC4.3	
length()	○	○	○	この PrdbBlob オブジェクトによって指定された BLOB 又は BINARY 値のバイト数を返します。
position(Blob pattern, long start)	○	○	○	PrdbBlob オブジェクトによって指定された BLOB 又は BINARY 値内で，pattern が始まるバイト位置を返します。
position(byte[] pattern, long start)	○	○	○	この Blob オブジェクトが表す BLOB 値内で，指定されたバイト pattern が始まるバイト位置を返します。
free()	×	○	○	Blob オブジェクトが保持しているリソースを解放します。

（凡例）

○：提供されます。

×：提供されません。

(a) getBinaryStream()

【機能】

BLOB 又は BINARY 値をストリーム（PrdbDataStream オブジェクト）として返します。

【形式】

```
public InputStream getBinaryStream() throws SQLException
```

【引数】

なし。

【戻り値】

InputStream を派生した PrdbDataStream を返します。

【機能詳細】

BLOB 又は BINARY 値をストリーム（PrdbDataStream オブジェクト）として返します。

位置付け子機能使用時は，PrdbResultSet オブジェクトから位置付け子を取得し，取得した位置付け子を PrdbDataStream クラスのコンストラクタに渡して PrdbDataStream オブジェクトを生成します。

位置付け子機能使用時は，PrdbDataStream オブジェクト内部に BLOB 又は BINARY 値を保持していないため，データ取得要求が行われるたびに HiRDB サーバに対してデータ取得要求を行います。

位置付け子機能を使用していない場合は PrdbResultSet オブジェクトから BLOB 又は BINARY 値を取得し，取得した値を PrdbDataStream クラスのコンストラクタに渡して PrdbDataStream オブジェク

トを生成します。この場合は，BLOB 又は BINARY 値を保持しているため，HiRDB サーバに対してデータ取得要求は行いません。

【発生する例外】

次の場合，SQLException を投入します。

- 位置付け子機能使用時に，この PrdbBlob オブジェクトに関連している PrdbConnection オブジェクトが close されている場合
- トランザクションの決着によって PrdbBlob オブジェクトが無効になった場合
- 位置付け子機能使用時に，通信エラーなどでデータが取得できなかった場合
- free メソッドが呼び出し済みの場合

(b) getBytes(long pos, int length)

【機能】

BLOB 又は BINARY 値の全部又は一部をバイト配列として返します。バイト配列には，pos の位置から length の連続するバイト数を格納します。

【形式】

```
public byte[] getBytes(long pos,int length) throws SQLException
```

【引数】

long pos：

BLOB 値内の最初に抽出されるバイトの位置（序数）

最初のバイトの位置は 1

int length：

コピー対象の連続するバイトの数

【戻り値】

BLOB 又は BINARY 値内の，pos の位置から length の連続するバイト数が格納されている配列

【機能詳細】

BLOB 又は BINARY 値の全部又は一部をバイト配列として返します。バイト配列には，pos の位置から length の連続するバイト数を格納します。

pos と length の指定値と返されるデータを次に示します。

pos	length	実長（X）※1	制限値※2 と length	制限値※2 と実長（X）※1	返されるデータ
1 ＜＝ pos ＜＝実長（X）※1	＜ 0	－	－	－	SQLException
	＞＝ 0	＜＝ length	－	Y ＜制限値	実長（Y）の長さの BLOB 又は BINARY 値
			－	Y ＞＝制限値	制限値の長さの BLOB 又は BINARY 値

pos	length	実長（X）※1	制限値※2 と length	制限値※2 と実長（X）※1	返されるデータ
		Y ＞ length	length ＜制限値	−	length で指定された長さの BLOB 又は BINARY 値
			length ＞＝制限値	−	制限値の長さの BLOB 又は BINARY 値
＞実長（X）※1	−	−	−	−	0 長データ
上記以外	−	−	−	−	SQLException

（凡例）

−：該当しません。

注※1

実長（X）とは，取得可能な BLOB 又は BINARY 値の実長（BLOB 又は BINARY 値の実長−引数 pos ＋ 1）を示します。

注※2

制限値は，MaxFieldSize 又は HiRDB_for_Java_MAXBINARYSIZE が該当します。制限値として有効になる値を次に示します。

MaxFieldSize	HiRDB_for_Java_MAXBINARYSIZE	HiRDB のデータ型	取得できるデータの最大長（制限値）
0（デフォルト）	0（デフォルト）	全データ型	定義長（デフォルト）
0（デフォルト）	＞ 0（指定有り）	BLOB 又は BINARY 型	HiRDB_for_Java_MAXBINARYSIZE
		BLOB 又は BINARY 型以外※	定義長（デフォルト）
＞ 0（指定有り）	0（デフォルト）	全データ型	MaxFieldSize
＞ 0（指定有り）	＞ 0（指定有り）	全データ型	MaxFieldSize

注※

文字型（HiRDB データ型の CHAR，VARCHAR，NCHAR.NVARCHAR，MCHAR，MVARCHAR が該当）

【発生する例外】

次の場合，SQLException を投入します。

- 引数 pos が＜ 1 又は引数 length が＜ 0 の場合
- トランザクションの決着によって PrdbBlob が無効の場合
- 位置付け子機能使用時に，この PrdbBlob オブジェクトに関連している PrdbConnection オブジェクトが close されている場合
- 位置付け子機能使用時に，通信エラーなどでデータが取得できなかった場合
- free メソッドが呼び出し済みの場合

(c) length()

【機能】

この PrdbBlob オブジェクトによって指定された BLOB 又は BINARY 値のバイト数を返します。

【形式】

```
public long length() throws SQLException
```

【引数】

なし。

【戻り値】

BLOB 又は BINARY 値の長さ（バイト単位）

【発生する例外】

次の場合，SQLException を投入します。

- 位置付け子機能使用時に，PrdbBlob オブジェクトに関連した PrdbConnection オブジェクトが close されている場合
- 位置付け子機能使用時に，トランザクションの決着によって PrdbBlob が無効になった場合
- 位置付け子機能使用時に，通信エラーなどでデータが取得できなかった場合
- free メソッドが呼び出し済みの場合

(d) position(Blob pattern, long start)

【機能】

PrdbBlob オブジェクトによって指定された BLOB 又は BINARY 値内で，pattern が始まるバイト位置を返します。pattern の検索は，start の位置から開始します。

【形式】

```
public long position(Blob pattern, long start) throws SQLException
```

【引数】

Blob pattern：

検索対象の BLOB 又は BINARY 値を指定する Blob オブジェクト

long start：

検索を開始する BLOB 又は BINARY 値内の位置。最初の位置は 1 です。

【戻り値】

pattern で指定されたデータが開始する位置

【発生する例外】

次の場合，SQLException を投入します。

- 位置付け子機能使用時に，PrdbBlob オブジェクトに関連した PrdbConnection オブジェクトが close されている場合
- 位置付け子機能使用時に，トランザクションの決着によって PrdbBlob が無効になった場合
- 位置付け子機能使用時に，通信エラーなどでデータが取得できなかった場合
- free メソッドが呼び出し済みの場合

(e) position(byte[] pattern, long start)

【機能】

この Blob オブジェクトが表す BLOB 値内で，指定されたバイト pattern が始まるバイト位置を返します。pattern の検索は，start の位置から開始します。

【形式】

```
public long position(byte[] pattern, long start) throws SQLException
```

【引数】

byte[] pattern：

検索対象の byte[]

long start：

検索を開始する BLOB 値内の位置。最初の位置は 1 です。

【戻り値】

pattern で指定されたデータが開始する位置

【発生する例外】

次の場合，SQLException を投入します。

- 位置付け子機能使用時に，PrdbBlob オブジェクトに関連した PrdbConnection オブジェクトが close されている場合
- 位置付け子機能使用時に，トランザクションの決着によって PrdbBlob が無効になった場合
- 位置付け子機能使用時に，通信エラーなどでデータが取得できなかった場合
- free メソッドが呼び出し済みの場合

(f) free()

【機能】

Blob オブジェクトが保持しているリソースを解放します。

【形式】

```
void free() throws SQLException
```

【引数】

なし。

【戻り値】

なし。

【機能詳細】

Blob オブジェクトが保持しているリソースを解放します。位置付け子機能使用時は，HiRDB サーバに対して位置付け子の解放要求を行います。

このメソッドを呼び出した後はオブジェクトが無効となるため，free メソッド以外のメソッドを呼び出した場合，KFPJ20023-E の SQLException が投入されます。

このメソッドを複数回呼び出した場合，2 回目以降の呼び出しは無視されます。

【発生する例外】

次の場合，SQLException を投入します。

- 位置付け子機能使用時に，PrdbBlob オブジェクトに関連した PrdbConnection オブジェクトが close されている場合
- 位置付け子機能使用時に，通信エラーなどでデータが取得できなかった場合

(3)　パッケージ名称及びクラス名称

このインタフェースを実装するパッケージ名称とクラス名称を次に示します。

パッケージ名称：**JP.co.Hitachi.soft.HiRDB.JDBC**

クラス名称：**PrdbBlob**

17.4.10　Array インタフェース

(1)　概要

Array インタフェースでは，繰返し列のアクセス手段として主に次の機能が提供されます。

- SQL Array 値の取得
- SQL Array 値を格納した結果セットの取得

JDBC ドライバは，Array インタフェースを PrdbArray クラスで実装します。

PrdbArray クラスのオブジェクトは，ResultSet の getArray メソッドの返却値として，JDBC ドライバが生成します。

(2)　メソッド

Array インタフェースのメソッド一覧を次の表に示します。なお，表に記載されていないメソッドはサポートしていません。サポートしていないメソッドを指定すると，SQLException を投入します。

表 17–71　Array インタフェースのメソッド一覧

メソッド	提供ドライバ			機能
	JDBC2.0	JDBC4.0	JDBC4.3	
getArray()	○	○	○	繰返し列の全要素を Object 配列として取得します。
getArray(long index, int count)	○	○	○	繰返し列の要素の一部を取り出します。
getBaseType()	○	○	○	PrdbArray オブジェクトが表す繰返し列の JDBC の型を，java.sql.Types クラスの定数として取得します。
getBaseTypeName()	○	○	○	PrdbArray オブジェクトが表す繰返し列のデータ型名を取得します。
getResultSet()	○	○	○	繰返し列の要素を保持する ResultSet オブジェクトを返します。
getResultSet(long index,int count)	○	○	○	繰返し列の要素を保持する ResultSet オブジェクトを返します。

（凡例）

○：提供されます。

(a)　getArray()

【機能】

繰返し列の全要素を Object 配列として取得します。

【形式】

```
public Object getArray() throws SQLException
```

【引数】

なし。

【戻り値】

繰返し列の全要素

【機能詳細】

繰返し列の全要素を Object 配列として取得します。

PrdbArray オブジェクト内に保持している繰返し列の要素を Object 配列の形式で返します。返される配列のデータ型は次のとおりです。

HiRDB のデータ型	返される配列
SMALLINT	java.lang.Short[]
INTEGR	java.lang.Integer[]
SMALLFLT,REAL	java.lang.Float[]
FLOAT,DOUBLE PRECISION	java.sql.Double[]
DECIMAL	java.math.BigDecimal[]
CHAR,NCHAR,MCHAR	java.lagn.String[]
VARCHAR,NVARCHAR,MVARCHAR	java.lagn.String[]
DATE	java.sql.Date[]
TIME	java.sql.Time[]
TIMESTAMP	java.sql.Timestamp[]
BINARY,BLOB	java.io.InputStream[]※

注※

HiRDB では BLOB，BINARY の繰返し列は未サポートであり，PrdbArray オブジェクトを作ることができないため，実際に返されることはありません。

【発生する例外】

次の場合，SQLException を投入します。

- この PrdbArray オブジェクトを生成した PrdbResultSet オブジェクトが close されている場合
  PrdbResultSet オブジェクトを生成した PrdbStatement オブジェクトを close したことによって，このドライバが PrdbResultSet オブジェクトを close した場合を含みます。
- この PrdbArray オブジェクトを生成した PrdbResultSet オブジェクトを生成した PrdbStatement オブジェクトを生成した Connection オブジェクトが close されている場合
- CltResultSet#getXXX()でエラーが発生した場合

(b) getArray(long index, int count)

【機能】

繰返し列の要素の一部を取り出します。指定されたインデクスから，最大で指定された要素数分 Object 配列として取得します。

【形式】

```
public Object getArray(long index, int count) throws SQLException
```

【引数】

long index：

最初に取り出す要素のインデクス（最初の要素は 1）

int count：

取得する連続する配列要素の数

【戻り値】

指定されたインデクスから count で指定された要素数分の Object 配列。

【機能詳細】

繰返し列の要素の一部を取り出します。指定されたインデクスから，最大で指定された要素数分 Object 配列として取得します。

引数 index と引数 count に指定された値による返される配列の関係を次に示します。

index	count	（index ＋ count）	返される配列
0 ＜ index ＜＝要素数	0 ＜＝ count	（index ＋ count）−1 ＜＝要素数	長さが count 分の配列
		（index ＋ count）−1 ＞要素数	要素数−（index−1）分の長さを持つ配列
	count ＜ 0	−	SQLException を投入
index ＞要素数	−	−	SQLException を投入
index ＜ 1	−	−	SQLException を投入

（凡例）

−：該当しません。

【発生する例外】

次の場合，SQLException を投入します。

- この PrdbArray オブジェクトを生成した PrdbResultSet オブジェクトが close されている場合
  PrdbResultSet オブジェクトを生成した PrdbStatement オブジェクトを close したことによって，このドライバが PrdbResultSet オブジェクトを close した場合を含みます。
- この PrdbArray オブジェクトを生成した PrdbResultSet オブジェクトを生成した PrdbStatement オブジェクトを生成した Connection オブジェクトが close されている場合
- CltResultSet#getXXX()でエラーが発生した場合
- 指定された引数の値が，index ＜ 1 又は count ＜ 0 の場合
- 指定された引数の値が，index ＞要素数の場合

(c) getBaseType()

【機能】

PrdbArray オブジェクトが表す繰返し列の JDBC の型を，java.sql.Types クラスの定数として取得します。

【形式】

```
public int getBaseType() throws SQLException
```

【引数】

なし。

【戻り値】

PrdbArray オブジェクトが表す繰返し列の JDBC の型を表す，java.sql.Types クラスの定数

【発生する例外】

なし。

(d) getBaseTypeName()

【機能】

PrdbArray オブジェクトが表す繰返し列のデータ型名を取得します。

【形式】

```
public String getBaseTypeName() throws SQLException
```

【引数】

なし。

【戻り値】

HiRDB のデータ型名

【発生する例外】

なし。

(e) getResultSet()

【機能】

繰返し列の要素を保持する ResultSet オブジェクトを返します。

【形式】

```
public ResultSet getResultSet()
```

【引数】

なし。

【戻り値】

ResultSet オブジェクト

【発生する例外】

次の場合，SQLException を投入します。

- この PrdbArray オブジェクトを生成した PrdbResultSet オブジェクトが close されている場合
  PrdbResultSet オブジェクトを生成した PrdbStatement オブジェクトを close したことによって，このドライバが PrdbResultSet オブジェクトを close した場合を含みます。

- この PrdbArray オブジェクトを生成した PrdbResultSet オブジェクトを生成した PrdbStatement オブジェクトを生成した Connection オブジェクトが close されている場合
- JDBC ドライバでエラーが発生した場合

(f) getResultSet(long index,int count)

【機能】

繰返し列の要素を保持する ResultSet オブジェクトを返します。

【形式】

```
public ResultSet getResultSet(long index,int count) throws SQLException
```

【引数】

long index：

最初に取り出す要素のインデクス（最初の要素は 1）

int count：

取得する連続する配列要素の数

【戻り値】

ResultSet オブジェクト。

【機能詳細】

繰返し列の要素を保持する ResultSet オブジェクトを返します。

ResultSet オブジェクトは，指定されたインデクスで始まり，最大でカウント数分の連続した繰返し列の要素を持ちます。

引数 index と引数 count に指定された値による返される ResultSet オブジェクトの関係を次に示します。

index	count	（index ＋ count）	返される ResultSet
0 ＜ index ＜＝要素数	0 ＜＝ count	（index ＋ count）−1 ＜＝要素数	行数が count 分の結果集合
		（index−1 ＋ count）−1 ＞要素数	要素数−（index−1）分の行数を持つ結果集合
	count ＜ 0	−	SQLException を投入
index ＞要素数	−	−	SQLException を投入
index ＜ 1	−	−	SQLException を投入

（凡例）

−：該当しません。

【発生する例外】

次の場合，SQLException を投入します。

- この PrdbArray オブジェクトを生成した PrdbResultSet オブジェクトが close されている場合

PrdbResultSet オブジェクトを生成した PrdbStatement オブジェクトを close したことによって，このドライバが PrdbResultSet オブジェクトを close した場合を含みます。

- この PrdbArray オブジェクトを生成した PrdbResultSet オブジェクトを生成した PrdbStatement オブジェクトを生成した Connection オブジェクトが close されている場合
- 指定された引数の値が，index ＜ 1 又は count ＜ 0 の場合
- 指定された引数の値が，index ＞要素数の場合

(3) パッケージ名称及びクラス名称

このインタフェースを実装するパッケージ名称とクラス名称を次に示します。

パッケージ名称：**JP.co.Hitachi.soft.HiRDB.JDBC**

クラス名称：**PrdbArray**

17.4.11 SQLException インタフェース

SQLException は，java.sql パッケージの SQLException クラスを直接利用します。SQLException インタフェースが提供する各メソッドの詳細，使用方法については，JavaSoft が提供する JDBC 関連ドキュメントを参照してください。

17.4.12 SQLWarning インタフェース

(1) 概要

SQLWarning インタフェースでは，主に次の機能が提供されます。

- データベースアクセスの警告に関する情報の提供

SQLWarning オブジェクトは，警告が報告される原因となったメソッドのオブジェクトに，例外での通知なしで蓄積されます。

(2) 注意事項

(a) 蓄積された SQLWarning オブジェクトの解放

SQLWarning オブジェクトは，警告が報告される原因となったメソッドのオブジェクト（Connection，Statement，PreparedStatement，CallableStatement，及び ResultSet）から，チェーンによって蓄積されます。

蓄積された SQLWarning オブジェクトを明示的に解放するには，警告が報告される原因となったメソッドのオブジェクトに対して clearWarnings メソッドを実行してください。

(b)　SQLWarning オブジェクトの生成条件

SQL の実行で発生した警告が，JDBC ドライバ内で保持することを，警告保持レベルで指定している場合，SQLWarning オブジェクトを生成して警告情報を保持します。また，Connection オブジェクトについては，プロパティなどで警告保持の指定ができます。

SQLWarning オブジェクトの生成条件を次の表に示します。

表 17–72　SQLWarning オブジェクトの生成条件

SQL の実行結果		Connection オブジェクトでの警告保持指定					
		警告を保持する			警告を保持しない		
		警告保持レベル			警告保持レベル		
		IGNORE	SQLWARN	ALLWARN	IGNORE	SQLWARN	ALLWARN
SQLCODE>0，かつ SQLCODE が 100，110，及び 120 以外	Connection オブジェクト以外で発生	×	×	○	×	×	○
	Connection オブジェクトで発生	×	×	○	×	×	×
SQL 連絡領域の SQLWARN0 が W（SQLWARN6 が W である場合を除く）	Connection オブジェクト以外で発生	×	○	○	×	○	○
	Connection オブジェクトで発生	×	○	○	×	×	×
JDBC ドライバ内での警告発生	Connection オブジェクト以外で発生	×	○	○	×	○	○
	Connection オブジェクトで発生	×	○	○	×	×	×

（凡例）

○：生成します。

×：生成しません。

注

Connection オブジェクトでの警告保持指定は，URL の SQLWARNING_IGNORE，ユーザプロパティの SQLWARNING_IGNORE，又はメソッドの setSQLWarningIgnore で指定できます。デフォルトは false です。

また，警告保持レベルは，URL の SQLWARNING_LEVEL，プロパティの HiRDB_for_Java_SQLWARNING_LEVEL，又はメソッドの setSQLWarningLevel で指定できます。デフォルトは SQLWARN です。

17.4.13　サポートしていないインタフェース

サポートしていないインタフェースはありません。

17.5　JDBC2.1 コア API

JDBC2.1 コア API で追加された機能のうち，サポートする機能とドライバを次の表に示します。

表 17–73　JDBC2.1 コアでの追加機能とサポートするドライバ

機能		サポートするドライバ		
		JDBC2.0	JDBC4.0	JDBC4.3
結果セットの拡張	スクロール	○	○	○
	並行処理	○	○	○
バッチ更新	Statement クラスでのバッチ更新	○	○	○
	PreparedStatement クラスでのバッチ更新	○	○	○
	CallableStatement クラスでのバッチ更新	○	○	○

（凡例）

○：サポートされます。

17.5.1　結果セットの拡張

JDBC2.1 コア API では，結果セット（ResultSet クラス）の拡張機能として「スクロール」と「並行処理」が追加されました。

(1)　スクロールタイプ

結果セットのスクロールタイプには，次の 3 種類があります。

- 順方向専用型
- スクロール非反映型
- スクロール反映型

このうち，順方向専用型，及びスクロール非反映型をサポートしています。

(2)　並行処理タイプ

結果セットの並行処理タイプには，次の 2 種類があります。

- 読み取り専用型
- 更新可能型

このうち，読み取り専用型をサポートしています。

(3)　注意事項

(a)　サポートしていない結果セットタイプ又は並行処理タイプが指定された場合の注意点

サポートしていない結果セットタイプ又は並行処理タイプが指定された場合，エラーになりません。指定された結果セットタイプ又は並行処理タイプに最も近い結果セットを仮定して，Statement クラス又はそのサブクラスのインスタンスを生成します。なお，その際に，警告（SQLWarning オブジェクト）を生成して，Connection クラスのインスタンスに関連づけます。

(b)　スクロール型結果セット使用時の注意点

スクロール型結果セットでは，すべての検索データを JDBC ドライバ内でキャッシングします。そのため，データ量が多い場合は，メモリ不足や性能劣化となる可能性が高くなります。したがって，スクロール型結果セットを使用する場合は，「SQL に条件を付加する」など，検索データ量をあらかじめ抑制しておいてください。

17.5.2　バッチ更新

JDBC2.1 コア API では，Statement クラス及び PreparedStatement クラスにバッチ更新機能が追加されました。バッチ更新機能によって，複数の SQL，又は複数のパラメタ値を登録し，一括して実行できるようになります。

バッチ更新を実行する場合，HiRDB の配列を使用した機能が使用できます。

配列を使用した機能は，HiRDB に対して大量のデータを高速に更新したい場合に有効です。なお，配列を使用した機能については，「配列を使用した機能」を参照してください。

(1)　Statement クラスでのバッチ更新

Statement クラスでのバッチ更新の留意点を次に示します。

- 複数の更新系 SQL を，addBatch メソッドで登録します。
- 登録した更新系 SQL を，executeBatch メソッドで一括して実行します。
- 一括実行の結果として，それぞれの更新系 SQL で更新された行数の配列を返却します。
- 一括実行の途中でエラーが発生した場合，BatchUpdateException を投入します。
- 登録した SQL 中に検索系 SQL がある場合，executeBatch メソッド呼び出し時に BatchUpdateException を投入します。

なお，JDBC ドライバでは，複数の SQL を一括実行できないため，登録された SQL を逐次実行することになります。

(2) PreparedStatement クラスでのバッチ更新

PreparedStatement クラスでのバッチ更新の留意点を次に示します。

- PreparedStatement インスタンス生成時に指定した更新系 SQL に対する？パラメタを，通常の手順（setXXX メソッド）で設定します。
- addBatch メソッドで？パラメタのセットを登録します。
- 登録した複数セットの？パラメタを，executeBatch メソッドで一括して実行します。
- 一括実行の結果として，それぞれの？パラメタのセットで更新した行数の配列を返却します。
- 一括実行の途中でエラーが発生した場合，BatchUpdateException を投入します。
- PreparedStatement インスタンス生成時に指定した SQL が検索系 SQL の場合，executeBatch メソッド呼び出し時に BatchUpdateException を投入します。

なお，JDBC ドライバでは，次の場合に逐次実行します。

- パラメタと SQL 文の addBatch の混在時でのバッチ更新
- HiRDB の BINARY 型に対する？パラメタを含む SQL 文でのバッチ更新（BINARY 型の？パラメタに設定するデータの長さが 32,001 バイト以上の場合）
- HiRDB の BLOB 型に対する？パラメタを含む SQL 文でのバッチ更新

＜注意事項＞

2 件目以降の addBatch で，setXXX メソッドで指定するパラメタ数が不足していた場合，前回セットした値が引き継がれるため注意が必要です。

INTEGER 型列が 2 列（列 1，列 2）ある場合の例を次に示します。

[指定例]

```
prepstmt.setInt(1,100);
prepstmt.setInt(2,100);
prepstmt.addBatch();
prepstmt.setInt(1,200);
prepstmt.addBatch();
prepstmt.executeBatch();
```

[説明]

- 1 件目の addBatch で設定される値は，列 1 ＝ 100，列 2 ＝ 100 となります。
  1 件目の addBatch でパラメタ数が不足している場合は，エラーが発生します。
- 2 件目の addBatch で設定される値は，列 1 ＝ 200，列 2 ＝ 100 となります。
  2 件目の addBatch で，列 2 の情報が更新されていないため，1 件目の addBatch の情報が引き継がれます。

(3) CallableStatement クラスでのバッチ更新

CallableStatement クラスでのバッチ更新の留意点を次に示します。

- CallableStatement インスタンス生成時に指定した Java ストアドルーチンに対する入力パラメタを，通常の手順（setXXX メソッド）で設定します。
- addBatch メソッドで入力パラメタのセットを登録します。
- 登録した複数セットの入力パラメタを，executeBatch メソッドで一括して実行します。
- 一括実行の結果として，それぞれの入力パラメタのセットで実行した Java ストアドルーチンの返却値（更新行数）の配列を返却します。
- 一括実行の途中でエラーが発生した場合，BatchUpdateException を投入します。
- CallableStatement インスタンス生成時に指定した Java ストアドルーチンが，更新行数を返却するルーチンでない場合，executeBatch メソッド呼び出し時に BatchUpdateException を投入します。
- CallableStatement インスタンス生成時に指定した Java ストアドルーチンが，出力パラメタ及び入出力パラメタを持つ場合，addBatch メソッド呼び出し時に BatchUpdateException を投入します。

なお，JDBC ドライバでは，複数行のストアドプロシジャの？パラメタを一括実行できないため，複数行のストアドプロシジャの？パラメタを逐次実行することになります。

注意事項：

結果セット(ResultSet)を返すストアドプロシジャは，バッチ更新ではストアドプロシジャを実行するまで結果セットを返すかどうか分からないため，ストアドプロシジャ内で更新をしている場合，更新が反映されることがあるので注意してください。例えば，更新後にその結果を検索し取得するストアドプロシジャを，バッチ更新で実行すると，BatchUpdateException が発生するが，更新は反映されてしまうことがあります。

(4) 注意事項

(a) HiRDB サーバによる暗黙的コミット

SQL 文のバッチ更新機能を使用する場合，addBatch した SQL 文に次の SQL 文が含まれるとき，その SQL 文を実行した時点で HiRDB サーバが暗黙的にコミットするため，注意が必要です。

- PURGE TABLE 文
- クライアント環境定義 PDCMMTBFDDL に YES を設定した状態での定義系 SQL

(b) パラメタと SQL 文の addBatch の混在時でのバッチ更新機能

パラメタと SQL 文の addBatch との混在時は，一括更新をしないで逐次実行します。例を次に示します。

```
PreparedStatement pstmt = con.prepareStatement("UPDATE T1 SET C1=? WHERE C2=?");
pstmt.setInt(1, 1);
pstmt.setInt(2, 1);
pstmt.addBatch();
pstmt.setInt(1, 2);
pstmt.setInt(2, 2);
pstmt.addBatch();
pstmt.addBatch("INSERT INTO T2 VALUES(1,2,3)");
```

```
pstmt.setInt(1, 3);
pstmt.setInt(2, 4);
pstmt.addBatch();
pstmt.setInt(1, 4);
pstmt.setInt(2, 4);
pstmt.addBatch();
pstmt.executeBatch();
```

この UAP を実行すると，パラメタと SQL 文の addBatch が混在しているため，各 addBatch 単位での SQL 実行となります。そのため，次の UAP を実行した場合と同じ結果となります。

```
PreparedStatement pstmt = con.prepareStatement("UPDATE T1 SET C1=? WHERE C2=?");
pstmt.setInt(1, 1);
pstmt.setInt(2, 1);
pstmt.executeUpdate();
pstmt.setInt(1, 2);
pstmt.setInt(2, 2);
pstmt.executeUpdate();
pstmt.executeUpdate("INSERT INTO T2 VALUES(1,2,3)");
pstmt.setInt(1, 3);
pstmt.setInt(2, 4);
pstmt.executeUpdate();
pstmt.setInt(1, 4);
pstmt.setInt(2, 4);
pstmt.executeUpdate();
```

なお，パラメタと SQL 文の addBatch とが混在するバッチ更新機能を使用する場合，Connection クラスの自動コミットモードを無効にすることを推奨します。

(c) HiRDB の BINARY 型に対する？パラメタを含む SQL 文でのバッチ更新

HiRDB の BINARY 型に対する？パラメタを含む SQL 文でバッチ更新をする場合，次の条件に該当するとき，一括更新をしないで逐次実行します。

- ？パラメタに対して setXXX メソッドで設定するデータ長が 32,001 バイト以上である（setString メソッドなどで文字データを指定している場合は，HiRDB に渡すデータにエンコードした後のデータ長が 32,001 バイト以上である）。

(d) HiRDB の BLOB 型に対する？パラメタを含む SQL 文でのバッチ更新

HiRDB の BLOB 型に対する？パラメタを含む SQL 文でバッチ更新をする場合，一括更新をしないで逐次実行します。

(e) addBatch メソッドによって多数のパラメタを登録する場合

JDBC ドライバは，addBatch メソッドで登録したすべてのパラメタを，executeBatch メソッドを実行するまでドライバ内に蓄積します。そのため，多数のパラメタを登録する際は，メモリ使用量に注意してください。

また，HiRDB の配列を使用した機能でバッチ更新をする場合，HiRDB サーバに対して JDBC ドライバが要求できる実行回数は最大で 30,000 回です。30,001 個以上のパラメタを登録している場合，1 回のバッチ更新で用いるパラメタを 30,000 個単位に分割し，HiRDB サーバに対して SQL の実行を要求します。この場合も，JDBC ドライバ内のメモリを消費するため，バッチ更新による性能向上効果が下がるおそれがあります。そのため，30,001 回以上の SQL 実行が必要な場合は，30,000 回以下の単位で executeBatch メソッドを実行することを推奨します。

(f)　例外 BatchUpdateException で通知する更新カウント

バッチ更新実行時に発生する例外 BatchUpdateException の，getUpdateCounts メソッドの戻り値で通知する更新カウント（int 型の配列）の内容を次の表に示します。

表 17–74　getUpdateCounts メソッドの戻り値で通知する更新カウントの内容

バッチ更新の実行形態	BATCHEXCEPTION_BEHAVIOR※1 の設定値	
	TRUE	FALSE
配列機能を利用した一括実行	要素数 0 の配列	要素数 n の配列 n：何回目の addBatch で登録したパラメタでエラーになったかを示します。 エラーが発生するとロールバックされるため，すべての配列要素には Statement.EXECUTE_FAILED が設定されます。
JDBC ドライバによる逐次実行	要素数が実行した SQL 数の配列 各配列要素には更新行数※2 を設定	要素数 n の配列 n：何回目の addBatch で登録したパラメタ又は SQL でエラーになったかを示します。 配列要素には次の値が設定されます。 要素 0〜n-2：更新行数※2 要素 n-1：Statement.EXECUTE_FAILED

注※1

次のどれかで指定します。

- HiRDB 接続時のユーザプロパティ HiRDB_for_Java_BATCHEXCEPTION_BEHAVIOR
- URL の BATCHEXCEPTION_BEHAVIOR
- DataSource 系インタフェースの setBatchExceptionBehavior メソッド

なお，接続先がバージョン 08-01 以前の HiRDB の場合，"TRUE"が指定されたものとして動作します。

注※2

CallableStatement クラスの executeBatch メソッドで CALL 文を実行した場合，Statement.SUCCESS_NO_INFO を設定します。

更新カウントの例を次に示します。

≪配列機能を利用した一括実行のプログラム例≫

```
PreparedStatement pstmt = con.prepareStatement("INSERT INTO T1 VALUES(?,?)");
pstmt.setInt(1, 1);
pstmt.setString(2,"aaaa");
pstmt.addBatch();
pstmt.setInt(1, 2);
pstmt.setString(2,"bbbbbbbb");
pstmt.addBatch();.....[A]
pstmt.setInt(1, 3);
pstmt.setString(2,"cccc");
pstmt.addBatch();
pstmt.executeBatch();
```

≪JDBC ドライバによる逐次実行のプログラム例≫

```
Statement stmt = con.createStatement();
stmt.addBatch("INSERT INTO T1 VALUES(1,'aaaa')");
stmt.addBatch("INSERT INTO T1 VALUES(2,'bbbbbbbb')");...[A]
stmt.addBatch("INSERT INTO T1 VALUES(3,'cccc')");
stmt.executeBatch();
```

プログラム例を実行し，[A]で登録したパラメタ，又は SQL の処理でエラーになった場合，getUpdateCounts メソッドで返却する更新カウントの内容を次に示します。

バッチ更新の実行形態	BATCHEXCEPTION_BEHAVIOR の設定値	
	TRUE	FALSE
配列機能を利用した一括実行	要素数 0 の int 型の配列	要素数 2 の int 型の配列 **要素 0 の値**：Statement.EXECUTE_FAILED **要素 1 の値**：Statement.EXECUTE_FAILED
JDBC ドライバによる逐次実行	要素数 1 の int 型の配列 **要素 0 の値**：更新行数	要素数 2 の int 型の配列 **要素 0 の値**：更新行数 **要素 1 の値**：Statement.EXECUTE_FAILED

17.5.3　追加されたデータ型

JDBC2.1 コア API では，幾つかの新たな JDBC SQL タイプが追加されました。次の JDBC SQL タイプが追加されましたが，JDBC ドライバでは使用できません。

- BLOB
- CLOB
- ARRAY
- REF
- DISTINCT
- STRUCT

- JAVA OBJECT

17.5.4　サポートしていないインタフェース

次のインタフェースはサポートしていません。

- Clob
- Ref
- SQLData
- SQLInput
- SQLOutput
- Struct

17.6　JDBC2.0 Optional Package

JDBC2.0 Optional Package では，次の機能が追加されました。

- JNDI 対応
- 接続プール
- 分散トランザクション
- RowSets

ただし，JDBC ドライバでは RowSets を使用できません。

JDBC2.0 Optional Package の機能，サポートするドライバ，及びインタフェースを次の表に示します。

表 17–75　JDBC2.0 Optional Package の機能とインタフェース

機能	サポートするドライバ			インタフェース
	JDBC2.0	JDBC4.0	JDBC4.3	
JNDI 対応	○	○	○	DataSource インタフェース
接続プール	○	○	○	ConnectionPoolDataSource インタフェース PooledConnection インタフェース
分散トランザクション	○	○	○	XAConnection インタフェース XADataSource インタフェース XAResource インタフェース XAException インタフェース

（凡例）
　○：サポートされます。

17.6.1　DataSource インタフェース

DataSource インタフェースで提供される各メソッドの詳細，使用方法については，JDBC の関連ドキュメントを参照してください。ここでは，JDBC ドライバがサポートする DataSource インタフェースのメソッドを示します。

(1)　メソッド

DataSource インタフェースのメソッド一覧を次の表に示します。

表 17–76　DataSource インタフェースのメソッド一覧

メソッド	提供ドライバ			機能
	JDBC2.0	JDBC4.0	JDBC4.3	
getConnection()	○	○	○	データソースに設定した接続情報によって，データベース接続を試みます。
getConnection(String username, String password)	○	○	○	データソースに設定した接続情報によって，データベース接続を試みます。
getLoginTimeout()	○	○	○	setLoginTimeout メソッドで指定された値を返します。
getLogWriter()	○	○	○	DataSource オブジェクトのログライターを取得します。
setLoginTimeout(int seconds)	○	○	○	データベースへの接続試行中に待機する最長時間（秒）を指定します。
setLogWriter(PrintWriter out)	○	○	○	DataSource オブジェクトのログライターを設定します。

（凡例）

○：提供されます。

(a)　getConnection()

【機能】

データソースに設定した接続情報によって，データベース接続を試みます。

【形式】

```
public Connection getConnection() throws SQLException
```

【引数】

なし。

【戻り値】

Connection オブジェクト

【機能詳細】

DataSource オブジェクトに事前に設定された接続情報を基に，HiRDB サーバとの接続を行い，接続された Connection オブジェクトを返します。ユーザ名称，パスワードの各設定方法での優先順位については，「接続情報の優先順位」を参照してください。

【発生する例外】

次の場合，SQLException を投入します。

- データベースアクセスエラーが発生した場合

- 指定した接続情報が不正である場合
  各接続情報が不正である条件については，「URL の構文」，及び「ユーザプロパティ」を参照してください。

(b)　getConnection(String username, String password)

【機能】

データソースに設定した接続情報によって，データベース接続を試みます。

【形式】

```
public Connection getConnection(String username, String password) throws SQLException
```

【引数】

String username：

接続時のユーザ名

String password：

接続時のパスワード

【戻り値】

Connection オブジェクト

【機能詳細】

引数で指定された情報，及び DataSource オブジェクトに事前に設定された接続情報を基に，HiRDB サーバとの接続を行い，接続された Connection オブジェクトを返します。

引数 username 又は引数 password が null の場合，ユーザ名称又はパスワードを，この引数で指定しなかったことを示します。また，引数 password が長さ 0 の文字列の場合，パスワードを指定しなかったことを示します。引数 username と ConnectionProperty 中にユーザ ID を設定した場合，引数 username の指定値を優先します。同様に，パスワードも引数 password の指定値を優先します。引数 username 及び password を指定しない場合については，「接続情報の優先順位」を参照してください。

【発生する例外】

次の場合，SQLException を投入します。

- データベースアクセスエラーが発生した場合
- 指定した接続情報が不正である場合
  各接続情報が不正である条件については，「URL の構文」，及び「ユーザプロパティ」を参照してください。
- 引数に指定したユーザ名が長さ 0 の文字列の場合

(c)　getLoginTimeout()

【機能】

setLoginTimeout メソッドで指定された値を返します。

【形式】

```
public synchronized int getLoginTimeout()
```

【引数】

なし。

【戻り値】

int 型：

setLoginTimeout メソッドで指定された値。setLoginTimeout メソッドで指定されていない場合は，0 を返します。

【発生する例外】

なし。

(d) getLogWriter()

【機能】

DataSource オブジェクトのログライターを取得します。

【形式】

```
public synchronized PrintWriter getLogWriter() throws SQLException
```

【引数】

なし。

【戻り値】

PrdbDataSource オブジェクトのログライターを返します。ログライターが設定されていない場合は NULL 値を返します。

【発生する例外】

なし。

(e) setLoginTimeout(int seconds)

【機能】

データベースへの接続試行中に待機する最長時間（秒）を指定します。

【形式】

```
public synchronized void setLoginTimeout(int seconds) throws SQLException
```

【引数】

int seconds：

接続待ち時間（秒）

【戻り値】

なし。

【機能詳細】

getConnection メソッドで Connection オブジェクトを取得する際に行う，HiRDB サーバとの物理接続時に使用します。0 を指定した場合，又は setLoginTimeout を実行していない場合は，PDCONNECTWAITTIME で指定した時間が，HiRDB サーバとの物理接続時の，HiRDB サーバに対する最大待ち時間になります。

getConnection メソッドでの接続処理は，大きく分けて次の二つから成ります。

1. TCP/IP のコネクション確立処理
2. HiRDB サーバとのユーザ認証などのネゴシエーション

このメソッド及びクライアント環境定義 PDCONNECTWAITTIME の指定値は，2.を監視対象としており，1.は監視対象に含みません。1.を監視する場合，クライアント環境定義 PDNBLOCKWAITTIME を指定する必要があります。

監視範囲の詳細については「クライアント環境定義の設定内容」のクライアント環境定義 PDCONNECTWAITTIME を参照してください。

【発生する例外】

引数 seconds が 0 未満，又は 301 以上の場合，SQLException を投入します。

(f)　setLogWriter(PrintWriter out)

【機能】

DataSource オブジェクトのログライターを設定します。

【形式】

```
public synchronized void setLogWriter(PrintWriter out) throws SQLException
```

【引数】

PrintWriter out：ログライター

【戻り値】

なし。

【発生する例外】

なし。

(2)　パッケージ名称及びクラス名称

このインタフェースを直接使用する場合に必要なパッケージ名称とクラス名称を次に示します。

パッケージ名称：**JP.co.Hitachi.soft.HiRDB.JDBC**

クラス名称：**PrdbDataSource**

17.6.2　ConnectionPoolDataSource インタフェース

ConnectionPoolDataSource インタフェースで提供される各メソッドの詳細，使用方法については，JDBC の関連ドキュメントを参照してください。ここでは，JDBC ドライバがサポートする ConnectionPoolDataSource インタフェースのメソッドを示します。

(1)　メソッド

ConnectionPoolDataSource インタフェースのメソッド一覧を次の表に示します。

表 17–77　ConnectionPoolDataSource インタフェースのメソッド一覧

メソッド	提供ドライバ			機能
	JDBC2.0	JDBC4.0	JDBC4.3	
getLoginTimeout()	○	○	○	setLoginTimeout メソッドで指定された値を返します。
getLogWriter()	○	○	○	ConnectionPoolDataSource オブジェクトのログライターを取得します。
getPooledConnection()	○	○	○	データソースに設定した接続情報によって，プールされた接続として使用できる PooledConnection オブジェクトを取得します。
getPooledConnection(String user, String password)	○	○	○	データソースに設定した接続情報によって，プールされた接続として使用できる PooledConnection オブジェクトを取得します。
setLoginTimeout(int seconds)	○	○	○	データベースへの接続試行中に待機する最長時間（秒）を指定します。
setLogWriter(PrintWriter out)	○	○	○	ConnectionPoolDataSource オブジェクトのログライターを設定します。

（凡例）

○：提供されます。

(a)　getLoginTimeout()

【機能】

setLoginTimeout メソッドで指定された値を返します。

【形式】

```
public synchronized int getLoginTimeout() throws SQLException
```

【引数】

なし。

【戻り値】

int 型：

setLoginTimeout メソッドで指定された値を返します。setLoginTimeout メソッドで指定されていない場合は，0 を返します。

【発生する例外】

なし。

(b) getLogWriter()

【機能】

ConnectionPoolDataSource オブジェクトのログライターを取得します。

【形式】

```
public synchronized PrintWriter getLogWriter() throws SQLException
```

【引数】

なし。

【戻り値】

PrdbConnectionPoolDataSource オブジェクトのログライターを返します。ログライターが設定されていない場合は，NULL 値を返します。

【発生する例外】

なし。

(c) getPooledConnection()

【機能】

データソースに設定した接続情報によって，プールされた接続として使用できる PooledConnection オブジェクトを取得します。

【形式】

```
public synchronized PooledConnection getPooledConnection() throws SQLException
```

【引数】

なし。

【戻り値】

PooledConnection オブジェクト

【機能詳細】

DataSource オブジェクトに事前に設定された接続情報を基に，プールされた接続として使用できる PooledConnection オブジェクトを返します。ユーザ名称，パスワードの各設定方法での優先順位は，「接続情報の優先順位」を参照してください。

【発生する例外】

データベースアクセスエラーが発生した場合，SQLException を投入します。

(d) getPooledConnection(String user, String password)

【機能】

データソースに設定した接続情報によって，プールされた接続として使用できる PooledConnection オブジェクトを取得します。

【形式】

```
public synchronized PooledConnection getPooledConnection(String user, String password) th
rows SQLException
```

【引数】

String user：接続時のユーザ名

String password：接続時のパスワード

【戻り値】

PooledConnection オブジェクト

【機能詳細】

引数で指定された情報，及び DataSource オブジェクトに事前に設定された接続情報を基に，プールされた接続として使用できる PooledConnection オブジェクトを返します。

引数 user 又は引数 password が null の場合，ユーザ名称又はパスワードを，この引数で指定しなかったことを示します。また，引数 password が長さ 0 の文字列の場合，パスワードを指定しなかったことを示します。引数 user と ConnectionProperty 中にユーザ ID を設定した場合，引数 user の指定値を優先します。同様に，パスワードも引数 password の指定値を優先します。引数 user 及び password を指定しない場合については，「接続情報の優先順位」を参照してください。

【発生する例外】

次の場合，SQLException を投入します。

- データベースアクセスエラーが発生した場合
- 引数に指定したユーザ名が長さ 0 の文字列の場合

(e) setLoginTimeout(int seconds)

【機能】

データベースへの接続試行中に待機する最長時間（秒）を指定します。

【形式】

```
public synchronized void setLoginTimeout(int seconds) throws SQLException
```

【引数】

int seconds：接続待ち時間（秒）

【戻り値】

なし。

【機能詳細】

PooledConnection インタフェースの getConnection メソッドで Connection オブジェクトを取得する際に行う，HiRDB サーバとの物理接続時に使用します。0 を指定した場合，又は setLoginTimeout を実行していない場合は，PDCONNECTWAITTIME で指定した時間が，HiRDB サーバとの物理接続時の，HiRDB サーバに対する最大待ち時間になります。

getConnection メソッドでの接続処理は，大きく分けて次の二つから成ります。

1. TCP/IP のコネクション確立処理
2. HiRDB サーバとのユーザ認証などのネゴシエーション

このメソッド及びクライアント環境定義 PDCONNECTWAITTIME の指定値は，2.だけを監視対象としており，1.は監視対象に含みません。1.を監視する場合，クライアント環境定義 PDNBLOCKWAITTIME を指定する必要があります。

監視範囲の詳細については「クライアント環境定義の設定内容」のクライアント環境定義 PDCONNECTWAITTIME を参照してください。

【発生する例外】

引数 seconds が 0 未満，又は 301 以上の場合，SQLException を投入します。

(f)　setLogWriter(PrintWriter out)

【機能】

ConnectionPoolDataSource オブジェクトのログライターを設定します。

【形式】

```
public synchronized void setLogWriter(PrintWriter out)
```

【引数】

PrintWriter out：ログライター

【戻り値】

なし。

【発生する例外】

なし。

(2)　パッケージ名称及びクラス名称

このインタフェースを直接使用する場合に必要なパッケージ名称とクラス名称を次に示します。

パッケージ名称：**JP.co.Hitachi.soft.HiRDB.JDBC**

クラス名称：**PrdbConnectionPoolDataSource**

17.6.3　PooledConnection インタフェース

PooledConnection インタフェースで提供される各メソッドの詳細，使用方法については，JDBC の関連ドキュメントを参照してください。ここでは，JDBC ドライバがサポートする PooledConnection インタフェースのメソッドを示します。

(1)　メソッド

PooledConnection インタフェースのメソッド一覧を次の表に示します。

表 17–78　PooledConnection インタフェースのメソッド一覧

メソッド	提供ドライバ			機能
	JDBC2.0	JDBC4.0	JDBC4.3	
getConnection()	○	○	○	Connection オブジェクトは，HiRDB サーバとの物理接続と 1 対 1 の関係となり，必要に応じて物理接続が行われ，Connection オブジェクトが返されます。
addConnectionEventListener(ConnectionEventListener listener)	○	○	○	指定したイベントリスナを登録して，この PooledConnection オブジェクトでイベントが発生したときに通知されるようにします。
close()	○	○	○	物理接続の切断をします。
removeConnectionEventListener(ConnectionEventListener listener)	○	○	○	指定したイベントリスナを，この PooledConnection オブジェクトでイベントが発生したときに通知されるコンポーネントリストから削除します。
addStatementEventListener(StatementEventListener listener)	×	○	○	引数で指定した StatementEventListener を PooledConnection に登録します。
removeStatementEventListener(StatementEventListener listener)	×	○	○	引数で指定した StatementEventListener を PooledConnection から削除します。

（凡例）

○：提供されます。

×：提供されません。

(a) getConnection()

【機能】

Connection オブジェクトは，HiRDB サーバとの物理接続と 1 対 1 の関係となり，必要に応じて物理接続が行われ，Connection オブジェクトが返されます。

【形式】

```
public synchronized Connection getConnection() throws SQLException
```

【引数】

なし。

【戻り値】

Connection オブジェクト

【機能詳細】

Connection オブジェクトは，HiRDB サーバとの物理接続と 1 対 1 の関係となり，必要に応じて物理接続が行われ，Connection オブジェクトが返されます。一度接続した物理接続は，このクラスオブジェクトがクローズされるまで切断されません。Connection オブジェクトに対して close メソッドを実行しても，物理接続の切断はしないで，このクラスオブジェクトが物理接続を保持します。保持した物理接続は，次のこのメソッド呼び出しによる接続要求で再利用します（setLoginTimeout やクライアント環境定義 PDCONNECTWAITTIME で指定した待ち時間は発生しません）。

【発生する例外】

次の場合は SQLException を投入します。

- データベースアクセスエラーが発生した場合
- 指定した接続情報が不正の場合

(b) addConnectionEventListener(ConnectionEventListener listener)

【機能】

指定したイベントリスナを登録して，この PooledConnection オブジェクトでイベントが発生したときに通知されるようにします。

【形式】

```
public synchronized void addConnectionEventListener(ConnectionEventListener listener)
```

【引数】

ConnectionEventListener listener：

ConnectionEventListener インタフェースを実装し，接続が閉じたかエラーが発生したときに通知されるようにするコンポーネント。通常は接続プール管理プログラムです。

【戻り値】

なし。

【機能詳細】

指定したイベントリスナを登録して，この PooledConnection オブジェクトでイベントが発生したときに通知されるようにします。追加するリスナが null であれば登録しません。

addConnectionEventListener メソッドで登録したイベントリスナから，このドライバのメソッドを呼び出すことはできません。呼び出した場合，（デッドロックによって）このドライバからの応答がなくなることがあります。

【発生する例外】

なし。

(c) close()

【機能】

物理接続の切断をします。

【形式】

```
public synchronized void close()
```

【引数】

なし。

【戻り値】

なし。

【機能詳細】

プールされたすべての接続に対し，物理接続の切断をします。Connection オブジェクトを取得してデータベースアクセスを行っている最中でも，PooledConnection.close()の実行で物理切断を試みます。

【発生する例外】

なし。

(d) removeConnectionEventListener(ConnectionEventListener listener)

【機能】

指定したイベントリスナを，この PooledConnection オブジェクトでイベントが発生したときに通知されるコンポーネントリストから削除します。

【形式】

```
public synchronized void removeConnectionEventListener(ConnectionEventListener listener)
```

【引数】

ConnectionEventListener listener：

ConnectionEventListener インタフェースを実装し，リスナとして登録されたコンポーネント。通常は接続プール管理プログラムです。

【戻り値】

なし。

【発生する例外】

なし。

(e) addStatementEventListener(StatementEventListener listener)

【機能】

引数で指定した StatementEventListener を PooledConnection に登録します。

【形式】

```
public synchronized void addStatementEventListener(StatementEventListener listener)
```

【引数】

StatementEventListener listener：

StatementEventListener インタフェースを実装し，前処理結果が無効となるエラーが発生したときに通知を受けるためのオブジェクトです。

【戻り値】

なし。

【機能詳細】

引数で指定した StatementEventListener を PooledConnection に登録して，PreparedStatement（又は CallableStatement）オブジェクトが無効となる SQLException が発生したときに通知を受けられるようにします。引数 listener が null の場合，何も行いません。

【発生する例外】

なし。

【注意事項】

このメソッドの最初の呼び出し前に生成した PreparedStatement（又は CallableStatement）オブジェクトは通知の対象になりません。イベント通知対象となる PreparedStatement の例を次に示します。

```
PooledConnection pcon = ConnectionPoolDataSource.getPooledConnection();
Connection con = pcon.getConnection();
PreparedStatement pstmt1 = con.prepareStatement(<SQL文A>);   ←イベント通知対象外
pcon.addStatementEventListener(listener);
PreparedStatement pstmt2 = con.prepareStatement(<SQL文B>);   ←イベント通知対象
```

(f)　removeStatementEventListener(StatementEventListener listener)

【機能】

引数で指定した StatementEventListener を PooledConnection から削除します。

【形式】

```
public synchronized void removeStatementEventListener(StatementEventListener listener)
```

【引数】

StatementEventListener listener：

addStatementEventListener メソッドで PooledConnection に登録済みのオブジェクトです。

【戻り値】

なし。

【機能詳細】

addStatementEventListener メソッドで登録済みの StatementEventListener を，PooledConnection が持つリストから削除します。引数 listener が null の場合，及び対応する StatementEventListener が存在しない場合は，何も行いません。

【発生する例外】

なし。

(2)　パッケージ名称及びクラス名称

このインタフェースを直接使用する場合に必要なパッケージ名称とクラス名称を次に示します。

パッケージ名称：**JP.co.Hitachi.soft.HiRDB.JDBC**

クラス名称：**PrdbPooledConnection**

17.6.4　XAConnection インタフェース

XAConnection インタフェースで提供される各メソッドの詳細，使用方法については，JDBC の関連ドキュメントを参照してください。ここでは，JDBC ドライバがサポートする XAConnection インタフェースのメソッドを示します。

(1)　メソッド

XAConnection インタフェースのメソッド一覧を次の表に示します。

表 17–79　XAConnection インタフェースのメソッド一覧

メソッド	提供ドライバ			備考
	JDBC2.0	JDBC4.0	JDBC4.3	
getXAResource()	○	○	○	–

（凡例）

○：提供されます。

–：特にありません。

(2)　パッケージ名称及びクラス名称

このインタフェースを直接使用する場合に必要なパッケージ名称とクラス名称を次に示します。

パッケージ名称：**JP.co.Hitachi.soft.HiRDB.JDBC**

クラス名称：**PrdbXAConnection**

17.6.5　XADataSource インタフェース

XADataSource インタフェースで提供される各メソッドの詳細，使用方法については，JDBC の関連ドキュメントを参照してください。ここでは，JDBC ドライバがサポートする XADataSource インタフェースのメソッドを示します。

(1)　メソッド

XADataSource インタフェースのメソッド一覧を次の表に示します。

表 17–80　XADataSource インタフェースのメソッド一覧

メソッド	提供ドライバ			備考
	JDBC2.0	JDBC4.0	JDBC4.3	
getLoginTimeout()	○	○	○	setLoginTimeout メソッドで指定されている値を返却します。setLoginTimeout メソッドで指定されていない場合は，0 を返却します。
getLogWriter()	○	○	○	–
getXAConnection()	○	○	○	認可識別子及びパスワードの各設定方法の優先順位については，「接続情報の優先順位」を参照してください。
getXAConnection(String username,String password)	○	○	○	引数 user 又は引数 password がナル値の場合は，認可識別子又はパスワードをこの引数で指定しなかったことを示します。

メソッド	提供ドライバ			備考
	JDBC2.0	JDBC4.0	JDBC4.3	
				引数 password が長さ 0 の文字列の場合は，パスワードを指定しなかったことを示します。 指定しない場合の設定値については，「接続情報の優先順位」を参照してください。 引数 user が長さ 0 の文字列の場合は，SQLException を投入します。
setLoginTimeout(int seconds)	○	○	○	この指定は XAConnection インタフェースの getConnection メソッドで Connection オブジェクトを取得する際に行う，HiRDB サーバとの物理接続時にだけ使用されます。0 を指定した場合，又は setLoginTimeout メソッドを実行していない場合は，クライアント環境定義 PDCONNECTWAITTIME で指定した時間が HiRDB サーバに対する最大待ち時間となります。 getConnection メソッドでの接続処理は，大きく分けて次の二つから成ります。 1. TCP/IP のコネクション確立処理 2. HiRDB サーバとのユーザ認証などのネゴシエーション このメソッド及びクライアント環境定義 PDCONNECTWAITTIME の指定値は，2.だけを監視対象としており，1.は監視対象に含みません。1.を監視する場合，クライアント環境定義 PDNBLOCKWAITTIME を指定する必要があります。 監視範囲の詳細については「クライアント環境定義の設定内容」のクライアント環境定義 PDCONNECTWAITTIME を参照してください。 0～300 の範囲外の値を設定すると，SQLException を投入します。
setLogWriter(PrintWriter out)	○	○	○	－

（凡例）

○：提供されます。

－：特にありません。

(2)　パッケージ名称及びクラス名称

このインタフェースを直接使用する場合に必要なパッケージ名称とクラス名称を次に示します。

パッケージ名称：**JP.co.Hitachi.soft.HiRDB.JDBC**

クラス名称：**PrdbXADataSource**

17.6.6　XAResource インタフェース

XAResource インタフェースで提供される各メソッドの詳細，使用方法については，JDBC の関連ドキュメントを参照してください。ここでは，JDBC ドライバがサポートする XAResource インタフェースのメソッドを示します。

(1)　メソッド

XAResource インタフェースのメソッド一覧を次の表に示します。

表 17–81　XAResource インタフェースのメソッド一覧

メソッド	提供ドライバ			備考
	JDBC2.0	JDBC4.0	JDBC4.3	
commit(Xid xid, boolean onePhase)	○	○	○	–
end(Xid xid, int flags)	○	○	○	–
getTransactionTimeout()	○	○	○	無条件に 0 を返却します。
isSameRM(XAResource xares)	○	○	○	–
prepare(Xid xid)	○	○	○	–
recover(int flag)	○	○	○	–
rollback(Xid xid)	○	○	○	–
setTransactionTimeout(int seconds)	○	○	○	トランザクションタイムアウト値の設定はしません。このメソッドは，トランザクションタイムアウト値が正常に設定されなかったことを示す false を返却します。
start(Xid xid, int flags)	○	○	○	–

（凡例）

○：提供されます。

–：特にありません。

(2)　パッケージ名称及びクラス名称

このインタフェースを直接使用する場合に必要なパッケージ名称とクラス名称を次に示します。

パッケージ名称：**JP.co.Hitachi.soft.HiRDB.JDBC**

クラス名称：**PrdbXAResource**

17.6.7　XAException インタフェース

XAException インタフェースは，javax.transaction.xa パッケージの XAException クラスを直接利用します。XAException インタフェースで提供される各メソッドの詳細，使用方法については，JDBC の関連ドキュメントを参照してください。

17.6.8　サポートしていないインタフェース

次のインタフェースはサポートしていません。

- RowSet
- RowSetInternal
- RowSetListner
- RowSetMetaData
- RowSetReader

17.7　JDBC3.0 API

JDBC3.0 API で追加された機能のうち，サポートする機能とドライバを次の表に示します。

表 17–82　JDBC3.0 での追加機能とサポートするドライバ

機能	サポートするドライバ		
	JDBC2.0	JDBC4.0	JDBC4.3
パラメタメタデータ	×	○	○
ホールダブルカーソル	○	○	○
データベースメタデータ追加 API	○	○	○

（凡例）

○：サポートされます。

×：サポートされません。

17.7.1　ParameterMetaData インタフェース

(1)　概要

ParameterMetaData インタフェースでは，PreparedStatement インタフェースの各パラメタに対するメタ情報を返す機能が提供されます。

(2)　メソッド

ParameterMetaData インタフェースのメソッド一覧を次の表に示します。なお，表に記載されていないメソッドはサポートしていません。サポートしていないメソッドを指定すると，SQLException を投入します。

表 17–83　ParameterMetaData インタフェースのメソッド一覧

メソッド	提供ドライバ			機能
	JDBC2.0	JDBC4.0	JDBC4.3	
getParameterCount()	×	○	○	PreparedStatement オブジェクトのパラメタ数を返します。
isNullable(int param)	×	○	○	指定されたパラメタに対応する列に対して，null 値が許可されているかどうかを返します。
isSigned(int param)	×	○	○	指定されたパラメタに対応する列のデータ型が符号付き数値かどうかを返します。

メソッド	提供ドライバ			機能
	JDBC2.0	JDBC4.0	JDBC4.3	
getPrecision(int param)	×	○	○	指定されたパラメタに対応する列のサイズを返します。
getScale(int param)	×	○	○	指定されたパラメタに対応する列の小数点以下のけた数を返します。
getParameterType(int param)	×	○	○	指定されたパラメタに対応する列の SQL データ型を返します。
getParameterTypeName(int param)	×	○	○	指定されたパラメタに対応する列のデータベース固有の形名を返します。
getParameterClassName(int param)	×	○	○	指定されたパラメタを PreparedStatement.setObject で設定する場合に適用される，Java クラスの完全指定された名前を返します。
getParameterMode(int param)	×	○	○	指定されたパラメタのモードを返します。

（凡例）

○：提供されます。

×：提供されません。

(a)　getParameterCount()

【機能】

PreparedStatement オブジェクトのパラメタ数を返します。

【形式】

```
public int getParameterCount() throws SQLException
```

【引数】

なし。

【戻り値】

int 型：

パラメタ数

【発生する例外】

なし。

(b)　isNullable(int param)

【機能】

指定されたパラメタに対応する列に対して，null 値が許可されているかどうかを返します。

【形式】

```
public int isNullable(int param) throws SQLException
```

【引数】

int param：

1から始まるパラメタ番号

【戻り値】

int 型：

- ParameterMetaData.parameterNullable：null 値が許可されています。
- ParameterMetaData.parameterNoNulls：null 値が許可されていません。

【発生する例外】

param に指定された値が 0 以下，又はパラメタ数より大きい場合，メッセージ KFPJ20413-E で SQLException を投入します。

(c)　isSigned(int param)

【機能】

指定されたパラメタに対応する列のデータ型が符号付き数値かどうかを返します。

【形式】

```
public boolean isSigned(int param) throws SQLException
```

【引数】

int param：

1から始まるパラメタ番号

【戻り値】

boolean 型：

- true：データ型が符号付き数値です。
- false：データ型が符号付き数値ではありません。

【機能詳細】

指定されたパラメタのデータ型が符号付き数値かどうかを返します。列の HiRDB のデータ型と返す値について次の表に示します。

表 17–84　列の HiRDB のデータ型と返す値（isSigned メソッド）

列の HiRDB のデータ型	返す値
INTEGER，SMALLINT，FLOAT，DOUBLE PRECISION，REAL，SMALLFLT，DECIMAL，NUMERIC	true
上記以外の型	false

【発生する例外】

param に指定された値が 0 以下，又はパラメタ数より大きい場合，メッセージ KFPJ20413-E で SQLException を投入します。

(d)　getPrecision(int param)

【機能】

指定されたパラメタに対応する列のサイズを返します。

【形式】

```
publc int getPrecision(int param) throws SQLException
```

【引数】

int param：

１から始まるパラメタ番号

【戻り値】

int 型：

パラメタに対応する列のサイズ

【機能詳細】

指定されたパラメタに対応する列のサイズを返します。列の HiRDB のデータ型と返す値について，次の表に示します。

表 17–85　列の HiRDB のデータ型と返す値（getPrecision メソッド）

列の HiRDB のデータ型	返す値	返却値の単位
BLOB(n)，BINARY(n)	n	バイト数
INTEGER	10	10 進けた数
SMALLINT	5	
FLOAT，DOUBLE PRECISION	15	
SMALLFLT，REAL	7	
DECIMAL(n,m)，NUMERIC(n,m)	n	
CHAR(n)，MCHAR(n)，VARCHAR(n)，MVARCHAR(n)	n	文字数
NCHAR(n)，NVARCHAR(n)	n	
DATE	10	
TIME	8	
TIMESTAMP(n)	n>0 の場合：19+(n+1) n=0 の場合：19	

列の HiRDB のデータ型	返す値	返却値の単位
ANY	0	−

【発生する例外】

param に指定された値が 0 以下，又はパラメタ数より大きい場合，メッセージ KFPJ20413-E で SQLException を投入します。

(e)　getScale(int param)

【機能】

指定されたパラメタに対応する列の小数点以下のけた数を返します。

【形式】

```
public int getScale(int param) throws SQLException
```

【引数】

int param：

１から始まるパラメタ番号

【戻り値】

int 型：

パラメタに対応する列の小数点以下のけた数

【機能詳細】

指定された列の小数点以下のけた数を返します。列の HiRDB のデータ型と返す値について次の表に示します。

表 17–86　列の HiRDB のデータ型と返す値（getScale メソッド）

列の HiRDB のデータ型	返す値
DECIMAL，NUMERIC	小数点以下のけた数
TIMESTAMP	ミリ秒以下のけた数
上記以外	0

【発生する例外】

param に指定された値が 0 以下，又はパラメタ数より大きい場合，メッセージ KFPJ20413-E で SQLException を投入します。

(f)　getParameterType(int param)

【機能】

指定されたパラメタに対応する列の SQL データ型を返します。

【形式】

```
public int getParameterType(int param) throws SQLException
```

【引数】

int param：

１から始まるパラメタ番号

【戻り値】

int 型：

java.sql.Types クラスからの SQL 型

【機能詳細】

指定したパラメタに対応する列の SQL データ型を返します。SQL データ型と返す値の対応については，「データ型」を参照してください。

ただし，繰返し列の場合は java.sql.Types.Array を返します。

【発生する例外】

param に指定された値が 0 以下，又はパラメタ数より大きい場合，メッセージ KFPJ20413-E で SQLException を投入します。

(g)　getParameterTypeName(int param)

【機能】

指定されたパラメタに対応する列のデータベース固有の形名を返します。

【形式】

```
public String getParameterTypeName(int param) throws SQLException
```

【引数】

int param：

１から始まるパラメタ番号

【戻り値】

String オブジェクト

【機能詳細】

指定されたパラメタに対応する列のデータベース固有の形名を返します。列の HiRDB のデータ型と返却する文字列について次の表に示します。

表 17–87　列の HiRDB のデータ型と返却する文字列（getParameterTypeName メソッド）

列の HiRDB のデータ型	返却する文字列
BLOB	"BLOB"
BINARY	"BINARY"

列の HiRDB のデータ型	返却する文字列
INTEGER	"INTEGER"
SMALLINT	"SMALLINT"
FLOAT，DOUBLE PRECISION	"FLOAT"
SMALLFLT，REAL	"REAL"
DECIMAL，NUMERIC	"DECIMAL"
CHAR	"CHAR"
MCHAR	"MCHAR"
NCHAR	"NCHAR"
VARCHAR	"VARCHAR"
MVARCHAR	"MVARCHAR"
NVARCHAR	"NVARCHAR"
DATE	"DATE"
TIME	"TIME"
TIMESTAMP	"TIMESTAMP"
ANY	"ANY"

【発生する例外】

param に指定された値が 0 以下，又はパラメタ数より大きい場合，メッセージ KFPJ20413-E で SQLException を投入します。

(h)　getParameterClassName(int param)

【機能】

指定されたパラメタを PreparedStatement.setObject で設定する場合に適用される，Java クラスの完全指定された名前を返します。

【形式】

```
public String getParameterClassName(int param) throws SQLException
```

【引数】

int param：

１から始まるパラメタ番号

【戻り値】

String オブジェクト

【機能詳細】

パラメタを PreparedStatement.setObject で設定する場合に適用される，Java クラスの完全指定された名前を String 型で返します。setObject に渡せるオブジェクトの型は複数存在するため，代表的と考えられるもの（推奨する型があるものはその型）を返します。

列の HiRDB のデータ型と返却する文字列について次の表に示します。ただし，繰返し列の場合は"java.sql.Array"を返却します。

表 17–88　列の HiRDB のデータ型と返却する文字列（getParameterClassName メソッド）

列の HiRDB のデータ型	返却する文字列
BLOB	"java.sql.Blob"
BINARY	"java.sql.Blob"
INTEGER	"java.lang.Integer"
SMALLINT	"java.lang.Integer"
FLOAT，DOUBLE PRECISION	"java.lang.Double"
SMALLFLT，REAL	"java.lang.Float"
DECIMAL，NUMERIC	"java.math.BigDecimal"
CHAR	"java.lang.String"
MCHAR	"java.lang.String"
NCHAR	"java.lang.String"
VARCHAR	"java.lang.String"
MVARCHAR	"java.lang.String"
NVARCHAR	"java.lang.String"
DATE	"java.sql.Date"
TIME	"java.sql.Time"
TIMESTAMP	"java.sql.Timestamp"
ANY	"java.lang.Object"

【発生する例外】

param に指定された値が 0 以下，又はパラメタ数より大きい場合，メッセージ KFPJ20413-E で SQLException を投入します。

(i)　getParameterMode(int param)

【機能】

指定されたパラメタのモードを返します。

【形式】

```
public int getParameterMode(int param) throws SQLException
```

【引数】

int param：

１から始まるパラメタ番号

【戻り値】

int 型：

- ParameterMetaData.parameterModeIn：パラメタのモードが IN です。
- ParameterMetaData.parameterModeOut：パラメタのモードが OUT です。
- ParameterMetaData.parameterModeInOut：パラメタのモードが INOUT です。

【発生する例外】

param に指定された値が 0 以下，又はパラメタ数より大きい場合，メッセージ KFPJ20413-E で SQLException を投入します。

17.7.2　サポートしていないインタフェース

次のインタフェースはサポートしていません。

- Savepoint

17.8　JDBC4.0 API

17.8.1　JDBC4.0 API での追加機能

JDBC4.0 API で追加された機能のうち，サポートする機能とドライバを次の表に示します。

表 17–89　JDBC4.0 での追加機能とサポートするドライバ

機能	サポートするドライバ		
	JDBC2.0	JDBC4.0	JDBC4.3
java.sql.Driver 自動ローディング	×	○	○
BLOB 機能拡張	×	○	○
ラッパーパターン	×	○	○
接続管理	×	○	○
SQLException 拡張	×	○	○
スカラ関数追加	○	○	○
StatementEventListener	×	○	○

（凡例）

○：サポートされます。

×：サポートされません。

(1)　java.sql.Driver 自動ローディング

JDBC4.0 より前の規格では，Java 仮想マシンへの Driver クラスの登録が必要でした。JDBC4.0 以降は，Driver クラスの登録が不要になります。Driver クラスの登録については，「Driver クラスの登録」を参照してください。なお，JDBC4.0 以降で Driver クラスを登録しても，動作に影響はありません。

(2)　BLOB 機能拡張

JDBC4.0 では，BLOB に関するメソッドが複数追加されました。このうち，Blob インタフェースの free() メソッドをサポートします。free()メソッドについては，「free()」を参照してください。

(3)　ラッパーパターン

Connection，DatabaseMetaData，DataSource，ResultSet，ResultSetMetaData，Statement，PreparedStatement，及び CallableStatement の各インタフェースが Wrapper インタフェースを継承します。Wrapper インタフェースについては「Wrapper インタフェース」を参照してください。

(4)　接続管理

JDBC4.0 では，コネクションの状態を確認するためのメソッド，及びコネクションプーリングを制御するためのメソッドが追加されています。このうち，コネクションの状態を確認するメソッド（Connection インタフェースの isValid(int timeout)メソッド）をサポートします。isValid(int timeout)メソッドについては「isValid(int timeout)」を参照してください。

(5)　SQLException 拡張

JDBC4.0 では，SQLException のサブクラスとして複数の例外クラスが追加されています。追加された例外クラスと，サポートの可否を次の表に示します。ただし，これらの例外クラスは，HiRDB サーバで詳細な SQLSTATE が設定されている場合にだけ返却されます。SQLSTATE が詳細に設定されていない場合は，JDBC ドライバ内で検知するエラーを除き，JDBC2.0 と同様に SQLException が返却されます。

表 17–90　追加された例外クラスと，サポートの可否

クラス名	サポート可否
SQLNonTransientException	○
SQLFeatureNotSupportedException	○
SQLNonTransientConnectionException	○
SQLDataException	○
SQLIntegrityConstraintViolationException	○
SQLInvalidAuthorizationException	○
SQLSyntaxErrorException	○
SQLTransientException	○
SQLTransientConnectionException	○
SQLTransactionRollbackException	○
SQLTimeoutException	○
SQLRecoverableException	○
SQLClientInfoException	○

（凡例）

○：サポートします。

詳細は，「SQLException 拡張機能」を参照してください。

(6)　スカラ関数追加

JDBC4.0 では，スカラ関数が複数追加されました。追加されたスカラ関数と，サポートの可否を次の表に示します。

表 17–91　追加されたスカラ関数と，サポートの可否

スカラ関数	サポート可否
CHARA_LENGTH， CHARACTER_LENGTH	×
CURRENT_DATE	○
CURRENT_TIME	○
CURRENT_TIMESTAMP	○
EXTRACT	×
OCTET_LENGTH	×
POSITION	○

（凡例）

○：サポートします。

×：サポートしません。

(7)　StatementEventListener

StatementEventListener は Statement プールを管理する外部プログラムが使用する機能です。Statement プール機能を実装しているアプリケーションサーバ向けに提供します。

アプリケーションサーバは，PreparedStatement が無効となった場合に JDBC ドライバから通知を受けるため，該当インタフェースを実装して JDBC ドライバに登録します。PreparedStatement が無効となると，該当インタフェースを通して JDBC ドライバからアプリケーションサーバへ通知を行い，アプリケーションサーバは無効となった PreparedStatement を Statement プールから削除します。

なお，上記の PreparedStatement は，CallableStatement（PreparedStatement を継承）も含むものとします（以降も同様です）。

StatementEventListener のメソッドの呼び出し契機を次の表に示します。

表 17–92　StatementEventListener のメソッドの呼び出し契機

メソッド	呼び出し契機
statementClosed	なし
statementErrorOccurred	SQL 実行時にエラーとなり，PreparedStatement が無効となった時点

PreparedStatement が無効になる契機と statementErrorOccurred メソッド呼び出し契機を次の表に示します。

表 17–93　PreparedStatement が無効になる契機と statementErrorOccurred メソッド呼び出し契機

<table>
<tr><th colspan="2" rowspan="2">メソッド</th><th colspan="2">statementErrorOccurred メソッド</th></tr>
<tr><th>呼び出し契機</th><th>対象 PreparedStatement</th></tr>
<tr><td rowspan="2">SQLException
発生時</td><td>接続障害を示す SQLCODE（-720，-722，-723，-728，-732，-735，-932，-1700）※1</td><td rowspan="2">該当 SQLException 投入直前</td><td rowspan="2">SQLException が発生した Connection で生成したすべての PreparedStatement</td></tr>
<tr><td>SQL 実行エラーによる暗黙的ロールバック</td></tr>
<tr><td rowspan="5">トランザクション決着時（STATEMENT_COMMIT_BEHAVIOR=FALSE の場合※2）</td><td>Connection.commit，XAResource.commit 実行時</td><td rowspan="6">無効となった PreparedStatement を使用したことで発生する SQLException（SQLCODE=-901，又は-1512）投入直前</td><td rowspan="6">SQLException 発生時に処理対象としている PreparedStatement</td></tr>
<tr><td>XAResource.prepare でのトランザクション決着時</td></tr>
<tr><td>Connection.rollback，XAResource.rollback 実行時</td></tr>
<tr><td>CALL 文実行時（プロシジャ内でトランザクション決着）</td></tr>
<tr><td>PURGE TABLE 実行時</td></tr>
<tr><td colspan="2">定義系 SQL 実行時（PDDDLDEAPRPEXE=YES，又は PDDDLDEAPRP=YES の場合※3）</td></tr>
<tr><td colspan="2">SET SESSION AUTHORIZATION 実行時</td><td colspan="2"></td></tr>
<tr><td rowspan="4">SQLException を伴わない接続障害発生時</td><td>XAResource のメソッド実行時に接続障害によって XAException 投入</td><td rowspan="4">無効となった Connection を使用したことで発生する SQLException（SQLCODE=-563）投入直前</td><td rowspan="4">接続障害が発生した Connection で生成したすべての PreparedStatement</td></tr>
<tr><td>InputStream によるデータ取得時に接続障害によって IOException 投入</td></tr>
<tr><td>PrdbConnection.checkSession 実行時に SESSION_ALIVE 以外が返却</td></tr>
<tr><td>Connection.isValid で false が返却</td></tr>
</table>

注※1

ConnectionEventListener も登録されている場合，StatementEventListener の呼び出しを先行します。

注※2

STATEMENT_COMMIT_BEHAVIOR の指定方法，及び優先順位については，「接続情報の優先順位」の「ステートメントのコミット実行後の状態」を参照してください。

注※3

PDDDLDEAPRPEXE,PDDDLDEAPRP の指定方法，及び優先順位については，「接続情報の優先順位」の「そのほかのクライアント環境定義の優先順位」を参照してください。

17.8.2　Wrapper インタフェース

(1)　概要

Wrapper インタフェースでは，JDBC での規定以外のメソッドを呼び出すための標準化された仕組みが提供されます。Connection，DatabaseMetaData，DataSource，ResultSet，ResultSetMetaData，Statement，PrepareStatement，CallableStatement，及び ParameterMetaData の各インタフェースが Wrapper インタフェースを継承します。

(2)　メソッド

Wrapper インタフェースのメソッド一覧を次の表に示します。なお，表に記載されていないメソッドはサポートしていません。サポートしていないメソッドを指定すると，SQLException を投入します。

表 17–94　Wrapper インタフェースのメソッド一覧

メソッド	提供ドライバ			機能
	JDBC2.0	JDBC4.0	JDBC4.3	
isWrapperFor(Class<?> iface)	×	○	○	指定されたクラスのオブジェクトを unwrap メソッドで返却できるかどうかを返します。
unwrap(Class<T> iface)	×	○	○	指定されたクラスのオブジェクトを返します。

（凡例）

○：提供されます。

×：提供されません。

Wrapper インタフェースを継承したインタフェースに対して，unwrap で指定できるクラスの一覧を次の表に示します。

表 17–95　unwrap で指定できるクラス

項番	インタフェース	unwrap で指定できるクラス
1	java.sql.Connection	JP.co.Hitachi.soft.HiRDB.JDBC.PrdbConnection
2	java.sql.DatabaseMetaData	JP.co.Hitachi.soft.HiRDB.JDBC.PrdbDatabaseMetaData
3	javax.sql.DataSource	JP.co.Hitachi.soft.HiRDB.JDBC.PrdbDataSource
4	java.sql.ResultSet	JP.co.Hitachi.soft.HiRDB.JDBC.PrdbResultSet

項番	インタフェース	unwrap で指定できるクラス
5	java.sql.ResultSetMetaData	JP.co.Hitachi.soft.HiRDB.JDBC.PrdbResultSetMetaData
6	java.sql.Statement	JP.co.Hitachi.soft.HiRDB.JDBC.PrdbStatement
7	java.sql.PreparedStatement	JP.co.Hitachi.soft.HiRDB.JDBC.PrdbPreparedStatement
8	java.sql.CallableStatement	JP.co.Hitachi.soft.HiRDB.JDBC.PrdbCallableStatement
9	java.sql.ParameterMetaData	JP.co.Hitachi.soft.HiRDB.JDBC.PrdbParameterMetaData

(a) isWrapperFor(Class<?> iface)

【機能】

指定されたクラスのオブジェクトを unwrap で返却できるかどうかを返します。

【形式】

```
public boolean isWrapperFor(Class<?> iface) throws SQLException
```

【引数】

Class iface：

チェック対象のクラス

【戻り値】

boolean 型：

- true：指定されたクラスのオブジェクトを unwrap で返却できます。
- false：指定されたクラスのオブジェクトを unwrap で返却できません。

【機能詳細】

指定されたクラスのオブジェクトを unwrap で返却できるかどうかを返します。このメソッドで true が返された場合，このメソッドで指定した引数と同じ引数で unwrap を実行すれば，必ず成功します。

【発生する例外】

なし。

(b) unwrap(Class<T> iface)

【機能】

指定されたクラスのオブジェクトを返します。

【形式】

```
public <T> T unwrap(Class<T> iface) throws SQLException
```

【引数】

Class<T> iface：

取得オブジェクトのクラス

【戻り値】

<T> T：指定されたクラスのオブジェクト

【発生する例外】

指定されたクラスのオブジェクトを返却できない場合，メッセージ KFPJ20022-E で SQLException を投入します。

17.8.3　SQLException 拡張機能

エラーが発生した場合，JDBC2.0 では SQLException が投入されましたが，JDBC4.0 以降では JDBC4.0 で追加された SQLException の下位分類の例外クラスが投入されます。ただし，HiRDB サーバで詳細な SQLSTATE が設定されていない場合は，JDBC ドライバ内で検知できるエラーを除き，JDBC2.0 と同様に SQLException が投入されます。

(1)　サポートクラス一覧

JDBC4.0 以降で返却される例外クラスの一覧と各クラスの説明を次の表に示します。

表 17–96　JDBC4.0 で追加された例外クラス一覧

クラス名	内容	エラー後の接続
SQLNonTransientException	一時的ではないエラーを示します。失敗した SQL を再試行しても正常に実行できない場合に投入されます。	有効
SQLFeatureNotSupportedException	SQLSTATE のクラス値が 0A（サポートされていない機能）の場合に投入されます。	
SQLNonTransientConnectionException	SQLSTATE のクラス値が 08（コネクション違反）の場合に投入されます。	無効（接続時のエラー）
SQLDataException	SQLSTATE のクラス値が 22（データ例外）の場合に投入されます。	有効
SQLIntegrityConstraintViolationException	SQLSTATE のクラス値が 23（整合性制約違反）の場合に投入されます。	
SQLInvalidAuthorizationException	SQLSTATE のクラス値が 28（認可識別子の指定が正しくない）の場合に投入されます。	無効（接続時のエラー）
SQLSyntaxErrorException	SQLSTATE のクラス値が 42（構文誤り又はアクセス規則違反）の場合に投入されます。	有効
SQLTransientException	一時的なエラーを示します。失敗した SQL を再試行すれば成功する可能性がある場合に投入されます。	
SQLTransientConnectionException	SQLSTATE のクラス値が 08（コネクション違反）の場合に投入されます（HiRDB が開始又は終了中である場合など）。	無効（接続時のエラー）

クラス名	内容	エラー後の接続
SQLTransactionRollbackException	SQLSTATE のクラス値が 40（トランザクションがロールバックした）の場合に投入されます。	有効
SQLTimeoutException	タイムアウトが発生した場合に投入されます。	
SQLRecoverableException	接続を再確保した上で，失敗したトランザクションを再試行すれば成功する可能性がある場合に投入されます。	無効
SQLClientInfoException	設定できないクライアントのプロパティが一つ以上ある場合に，Connection.setClientInfo メソッドによって投入されます。	−

（凡例）

−：該当しません（接続状態は関係しません）。

JDBC4.0 で追加された例外クラスの継承関係を次に示します。

```
java.sql.SQLException
   |
   ├ java.sql.SQLNonTransientException
   |      ├ java.sql.SQLFeatureNotSupportedException
   |      ├ java.sql.SQLNonTransientConnectionException
   |      ├ java.sql.SQLDataException
   |      ├ java.sql.SQLIntegrityConstraintViolationException
   |      ├ java.sql.SQLInvalidAuthorizationException
   |      ├ java.sql.SQLSyntaxErrorException
   |
   ├ java.sql.SQLTransientException
   |      ├ java.sql.SQLTransientConnectionException
   |      ├ java.sql.SQLTransactionRollbackException
   |      ├ java.sql.SQLTimeoutException
   |
   ├ java.sql.SQLRecoverableException
   └ java.sql.SQLClientInfoException
```

(2)　例外クラス投入の条件

Type4 JDBC ドライバがサポートする例外クラスは，詳細な SQLSTATE の設定有無によって投入の有無が異なります。詳細な SQLSTATE の設定と例外クラス投入の有無を次の表に示します。

表 17–97　詳細な SQLSTATE の設定と例外クラス投入の有無

JDBC 規格	クラス名	詳細な SQLSTATE	
		設定あり	設定なし
1.0	SQLException	×	○
2.0	BatchUpdateException	○	○
	XAException	○	○

JDBC 規格	クラス名	詳細な SQLSTATE	
		設定あり	設定なし
4.0	SQLNonTransientException	○	○※
	SQLFeatureNotSupportedException	○	○※
	SQLNonTransientConnectionException	○	×
	SQLDataException	○	○※
	SQLIntegrityConstraintViolationException	○	×
	SQLInvalidAuthorizationException	○	×
	SQLSyntaxErrorException	○	○※
	SQLTransientException	○	×
	SQLTransientConnectionException	○	○※
	SQLTransactionRollbackException	○	○
	SQLTimeoutException	○	×
	SQLRecoverableException	○	○
	SQLClientInfoException	○	○

（凡例）

○：投入されます。

×：投入されません。

注※

JDBC ドライバ内で検知できるエラーだけ投入されます。

SQLSTATE の値を詳細に出力するかどうかの設定については，「PDSTANDARDSQLSTATE」を参照してください。

(3)　メソッド詳細

例外クラスが提供する各メソッドの詳細及び使用方法については，JDBC の関連ドキュメントを参照してください。なお，各例外クラスは java.sql パッケージのクラスを直接利用するため，独自の仕様はありません。

(4)　HiRDB のエラーと各例外クラスの対応

HiRDB のエラー（JDBC ドライバ内のエラーを含む）と各例外クラスの対応については，マニュアル「HiRDB メッセージ」の「SQLSTATE」を参照してください。

(5) 注意事項

executeQuery()などの SQL 実行メソッドで，setQueryTimeout()又は PDCWAITTIME で設定した時間でタイムアウトした場合，JDBC 規格では SQLTimeoutException が投入されます。しかし，Type4 JDBC ドライバでは，タイムアウト時に HiRDB サーバと必ず切断されるため，SQLRecoverableException が投入されます。

17.8.4 サポートしていないインタフェース

次のインタフェースはサポートしていません。

- NClob
- RowId
- SQLXML

17.9　JDBC4.3API

17.9.1　JDBC4.3 API での追加機能

JDBC4.3 API で追加された機能のうち，サポートする機能とドライバを次の表に示します。

表 17–98　JDBC4.3 での追加機能とサポートするドライバ

機能	サポートするドライバ	
	JDBC2.0 JDBC4.0	JDBC4.3
try-with-resource 文	×	○
メタデータの追加	×	○
スキーマ指定	×	○
ステートメント自動クローズ	×	○
大きい更新カウント(2147483647)	×	○
SQL 識別子判定	×	○

（凡例）

○：サポートされます。

×：サポートされません。

(1)　try-with-resource 文

Connection インタフェース，ResultSet インタフェース，及び Statement インタフェースで try-with-resource 文が使用できるようになりました。

(2)　メタデータの追加

擬似列，自動生成キーの返却有無，LOB の最大論理サイズ，REF CURSOR のサポート有無，及び Sharding サポート有無を返すメソッドが DatabaseMetaData インタフェースに追加されました。これらのメソッドについては DatabeseMetaData インタフェースのメソッド「getPseudoColumns(String catalog, String schemaPattern, String tableNamePattern, String columnNamePattern)」，「generatedKeyAlwaysReturned()」，「getMaxLogicalLobSize()」，「supportsRefCursors()」，及び「supportsSharding()」を参照してください。

(3)　スキーマ指定

Connection インタフェースにスキーマ（デフォルトの認可識別子）を設定するメソッドが追加されました。これらのメソッドについては Connection インタフェースのメソッド「setSchema(String schema)」及び「getSchema()」を参照してください。

(4)　ステートメント自動クローズ

Statement インタフェースに，ResultSet オブジェクトクローズ時に Statement オブジェクトを自動でクローズするメソッドが追加されました。これらのメソッドについては Statement インタフェースの「closeOnCompletion()」及び「isCloseOnCompletion()」を参照してください。

(5)　大きい更新カウント

更新系 SQL 実行時に更新行数が 2147483647 を超える場合のメソッドが追加されました。これらのメソッドについては Statement インタフェースの「executeLargeBatch()」，「executeLargeUpdate(String sql)」，「setLargeMaxRows(long max)」，「getLargeMaxRows()」，「getLargeUpdateCount()」，及び PreparedStatement インタフェースの「executeLargeUpdate()」を参照してください。

(6)　SQL 識別子判定

Statement インタフェースに SQL 識別子（文字列定数表記，各国文字列定数表記，名前の指定）を判断，返却するメソッドが追加されました。これらのメソッドについては Statement インタフェースの「enquoteLiteral(String val)」，「enquoteIdentifier(String identifier, boolean alwaysQuote)」，「isSimpleIdentifier(String identifier)」，「enquoteNCharLiteral(String val)」を参照してください。

17.9.2　サポートしていないインタフェース

次のインタフェースはサポートしていません。

- JDBCType
- SQLType
- DriverAction
- ConnectionBuilder
- ShardingBuilder
- PooledConnectionBuilder
- XAConnectionBuilder

17.10　システムプロパティの設定

ここでは，システムプロパティの設定について説明します。

なお，クライアント環境定義とシステムプロパティの対応については，「指定できるクライアント環境定義」を参照してください。

DABroker for Java 互換機能に関するシステムプロパティについては，「DABroker for Java 互換機能に関するシステムプロパティ」を参照してください。

Exception トレースログを取得するためのシステムプロパティの設定については，「Exception トレースログを取得するための設定」の「システムプロパティの設定」を参照してください。

不正電文トレースを取得するためのシステムプロパティの設定については，「不正電文トレースを取得するための設定」の「システムプロパティの設定」を参照してください。

17.10.1　ステートメントに関する接続情報の設定

指定できるシステムプロパティを次の表に示します。

表 17–99　ステートメントに関するシステムプロパティ

プロパティ名	指定内容
HiRDB_for_Java_STATEMENT_CLOSE_BEHAVIOR	ステートメント（Statement クラス，PreparedStatement クラス，及び CallableStatement クラス）の close メソッド実行時に前処理結果を無効にするかどうかを指定します。
HiRDB_for_Java_UPDATECOUNT_BEHAVIOR	更新系 SQL 発行時の更新行数が 0 の場合，java.sql.Statement の getUpdateCount メソッドの戻り値として-1 を返却するかどうかを指定します。

(1)　HiRDB_for_Java_STATEMENT_CLOSE_BEHAVIOR

ステートメント（Statement クラス，PreparedStatement クラス，及び CallableStatement クラス）の close メソッド実行時に前処理結果を無効にするかどうかを指定します。なお，指定した値について大文字小文字の区別はしません。

【指定値】

TRUE：前処理結果を無効にします。

FALSE：前処理結果を無効にしません。

上記以外：指定が省略されたものとします。

【省略値】

この指定を省略した場合は，ほかの設定方法で指定した値を使用します。使用する値は，「接続情報の優先順位」の「ステートメントの close メソッド実行時の SQL 前処理の破棄」を参照してください。どの指定もない場合は，FALSE が指定されたものとします。

このプロパティの指定は，「getConnection メソッドによる HiRDB への接続」の「HiRDB_for_Java_STATEMENT_CLOSE_BEHAVIOR」の設定と同等になります。

(2)　HiRDB_for_Java_UPDATECOUNT_BEHAVIOR

更新系 SQL 発行時の更新行数が 0 であった場合，java.sql.Statement の getUpdateCount メソッドの戻り値として-1 を返却するかどうかを指定します。なお，指定した値について大文字小文字の区別はしません。

【指定値】

TRUE：更新系 SQL 発行時の更新行数が 0 の場合，-1 を返却します（JDBC 規格）。

FALSE：更新系 SQL 発行時の更新行数が 0 の場合，0 を返却します。

上記以外：指定が省略されたものとします。

【省略値】

この指定を省略した場合は，ほかの設定方法で指定した値を使用します。使用する値は，「接続情報の優先順位」の「ステートメントの更新行数が JDBC 規格に準拠するかどうか」を参照してください。どの指定もない場合は，FALSE が指定されたものとします。

このプロパティの指定は，「getConnection メソッドによる HiRDB への接続」の「HiRDB_for_Java_UPDATECOUNT_BEHAVIOR」の設定と同等になります。

17.10.2　互換維持に関する接続情報の設定

指定できるシステムプロパティを次の表に示します。

表 17–100　互換維持に関するシステムプロパティ

項番	プロパティ名	指定内容
1	HiRDB_for_Java_Compatible	Type4 JDBC ドライバのメソッドを，JDBC の規格に準拠した動作とするか，HiRDB 独自の動作とするかを指定します。

(1)　HiRDB_for_Java_Compatible

Type4 JDBC ドライバのメソッドを，JDBC の規格に準拠した動作とするか，HiRDB 独自の動作とするかを指定します。なお，指定した値について大文字小文字の区別はしません。

【指定値】

NO：次の表に示すすべてのメソッドが JDBC の規格に準拠した動作をします。

ALL：次の表に示すすべてのメソッドが HiRDB 独自の動作をします。

上記以外：指定が省略されたものとします。

【省略値】

この指定を省略した場合は，NO を仮定します。

HiRDB_for_Java_Compatible プロパティの指定値とメソッドの動作を次の表に示します。

表 17–101　HiRDB_for_Java_Compatible プロパティの指定値によるメソッドの動作

項番	対象クラス	対象メソッド	指定値による動作	
			返却する情報	
			NO（JDBC 規格）	ALL（HiRDB 独自）
1	DatabaseMetaData	getImportedKeys	指定されたテーブルの外部キー及び当該外部キーが参照する主キー	指定されたテーブルの主キー及び当該主キーを参照する外部キー
		getExportedKeys	指定されたテーブルの主キー及び当該主キーを参照する外部キー	指定されたテーブルの外部キー及び当該外部キーが参照する主キー

17.10.3　DatabaseMetaData クラスのメソッドの動作に関する接続情報の設定

指定できるシステムプロパティを次の表に示します。

表 17–102　DatabaseMetaData クラスのメソッドの動作に関するシステムプロパティ

項番	プロパティ名	指定内容
1	HiRDB_for_Java_METADATA_BEHAVIOR	DatabaseMetaData クラスのメソッドの動作を変更する場合に指定します。

(1)　HiRDB_for_Java_METADATA_BEHAVIOR

DatabaseMetaData クラスのメソッドの動作を変更する場合に指定します。なお，指定した値について大文字小文字の区別はしません。

【指定値】

CAT_SEP_EMPTY：次の表に示すメソッドの動作を変更します。

上記以外：指定が省略されたものとします。

【省略値】

この指定を省略した場合は，DatabaseMetaData クラスのメソッドの動作を変更しません。

HiRDB_for_Java_METADATA_BEHAVIOR プロパティの指定値によるメソッドの動作を次の表に示します。

表 17–103　HiRDB_for_Java_METADATA_BEHAVIOR プロパティの指定値によるメソッドの動作

項番	対象クラス	対象メソッド	指定値による動作	
			CAT_SEP_EMPTY	指定なし
1	DatabaseMetaData	getCatalogSeparator	空文字("")を返します。	null を返します。

17.11　接続情報設定／取得インタフェース

JDBC2.0 Optional Package で提供する，DataSource，ConnectionPoolDataSource，及び XADataSource の各クラスでは，JDBC2.0 Optional Package 規格で定められたメソッドのほかに，DB 接続に必要となる接続情報設定／取得用のメソッドを提供します。

接続情報設定／取得メソッドの一覧を次の表に示します。

表 17–104　接続情報設定／取得メソッドの一覧

メソッド	機能
setDescription	データベースへの接続に必要な情報（接続付加情報）を設定します。
getDescription	データベースへの接続に必要な情報（接続付加情報）を取得します。
setDBHostName	接続する HiRDB のホスト名を設定します。
getDBHostName	接続する HiRDB のホスト名を取得します。
setJDBC_IF_TRC	JDBC インタフェースメソッドトレースの取得の有無を設定します。
getJDBC_IF_TRC	JDBC インタフェースメソッドトレースの取得の有無を取得します。
setTRC_NO	JDBC インタフェースメソッドトレースのエントリ数を設定します。
getTRC_NO	JDBC インタフェースメソッドトレースのエントリ数を取得します。
setUapName	UAP 名称を設定します。
getUapName	UAP 名称を取得します。
setUser	データベース接続時の認可識別子を設定します。
getUser	データベース接続時の認可識別子を取得します。
setPassword	データベース接続時のパスワードを設定します。
getPassword	データベース接続時のパスワードを取得します。
setXAOpenString	XA オープン文字列を設定します。
getXAOpenString	XA オープン文字列を取得します。
setXACloseString	XA クローズ文字列を設定します。
getXACloseString	XA クローズ文字列を取得します。
setLONGVARBINARY_Access	JDBC SQL タイプ LONGVARBINARY（HiRDB のデータ型である BLOB 型，及び BINARY 型）のデータへのアクセス方法を設定します。
getLONGVARBINARY_Access	JDBC SQL タイプ LONGVARBINARY（HiRDB のデータ型である BLOB 型，及び BINARY 型）のデータへのアクセス方法を取得します。
setSQLInNum	実行する SQL の入力？パラメタの最大数を設定します。
getSQLInNum	実行する SQL の入力？パラメタの最大数を取得します。
setSQLOutNum	実行する SQL の検索項目の最大数を設定します。

メソッド	機能
getSQLOutNum	実行する SQL の検索項目の最大数を取得します。
setSQLWarningLevel	SQL 実行時に発生した警告保持レベルを設定します。
getSQLWarningLevel	setSQLWarningLevel メソッドで設定した警告保持レベルを取得します。
setXALocalCommitMode	XA 接続時，トランザクションが分散トランザクションでない場合，オートコミット機能を有効にするかどうかを設定します。
getXALocalCommitMode	XA 接続時，トランザクションが分散トランザクションでない場合，オートコミット機能を有効にするかどうかの設定情報を取得します。
setSQLWarningIgnore	データベースから返される警告を Connection クラスで保持するかどうかを設定します。
getSQLWarningIgnore	データベースから返される警告を Connection クラスで保持するかどうかの設定情報を取得します。
setHiRDBCursorMode	HiRDB がコミットした場合に ResultSet クラスのオブジェクトを有効とするかどうかを設定します。
getHiRDBCursorMode	HiRDB がコミットした場合に ResultSet クラスのオブジェクトを有効とするかどうかの設定情報を取得します。
setNotErrorOccurred	ConnectionEventListener.connectionErrorOccurred の発生を抑止するかどうかを設定します。
getNotErrorOccurred	ConnectionEventListener.connectionErrorOccurred の発生を抑止するかどうかの設定情報を取得します。
setEnvironmentVariables	HiRDB のクライアント環境定義を設定します。
getEnvironmentVariables	HiRDB のクライアント環境定義を取得します。
setEncodeLang	データ変換時の変換文字セット名称を設定します。
getEncodeLang	設定された変換文字セット名称を取得します。
setMaxBinarySize	JDBC SQL タイプ LONGVARBINARY 型データ取得時のデータサイズの上限を設定します。
getMaxBinarySize	JDBC SQL タイプ LONGVARBINARY 型データ取得時のデータサイズの上限を取得します。
setStatementCommitBehavior	コミットした場合，ステートメントオブジェクトをコミット実行後も有効とするかどうかを設定します。
getStatementCommitBehavior	コミットした場合，ステートメントオブジェクトをコミット実行後も有効とするかどうかの設定情報を取得します。
setLONGVARBINARY_AccessSize	HiRDB サーバに対して一度に要求する JDBC SQL タイプ LONGVARBINARY 型データの長さを設定します。
getLONGVARBINARY_AccessSize	HiRDB サーバに対して一度に要求する JDBC SQL タイプ LONGVARBINARY 型データの長さを取得します。
setLONGVARBINARY_TruncError	JDBC SQL タイプ LONGVARBINARY 型データ取得時に切り捨てが発生した場合に，例外を投入するかどうかを設定します。
getLONGVARBINARY_TruncError	JDBC SQL タイプ LONGVARBINARY 型データ取得時に切り捨てが発生した場合に，例外を投入するかどうかの設定情報を取得します。

メソッド	機能
setStatementCloseBehavior	ステートメント（Statement クラス，PreparedStatement クラス，及び CallableStatement クラス）の close メソッド実行時に前処理結果を無効にするかどうかを設定します。
getStatementCloseBehavior	ステートメント（Statement クラス，PreparedStatement クラス，及び CallableStatement クラス）の close メソッド実行時に前処理結果を無効にするかどうかの設定情報を取得します。
setHiRDBINI	HiRDB.INI ファイルのディレクトリを指定します。
getHiRDBINI	HiRDB.INI ファイルのディレクトリを取得します。
setBatchExceptionBehavior	java.sql.BatchUpdateException の getUpdateCounts メソッドの戻り値に，JDBC 規格に準拠した更新カウントを設定するかどうかを指定します。
getBatchExceptionBehavior	java.sql.BatchUpdateException の getUpdateCounts メソッドの戻り値に，JDBC 規格に準拠した更新カウントを設定するかどうかの設定情報を取得します。
setUpdateCountBehavior	更新系 SQL 発行時の更新行数が 0 の場合，java.sql.Statement の getUpdateCount メソッドの戻り値として-1 を返却するかどうかを指定します。
getUpdateCountBehavior	更新系 SQL 発行時の更新行数が 0 の場合，java.sql.Statement の getUpdateCount メソッドの戻り値として-1 を返却するかどうかの設定情報を取得します。

17.11.1 setDescription

(1) 機能

データベースへの接続に必要な情報（接続付加情報）を設定します。

(2) 形式

```
public void setDescription ( String description ) throws SQLException
```

(3) 引数

String description：

接続付加情報を指定します。ナル値の場合は，このメソッドによって現在までに設定した接続付加情報を無効とし，初期状態に戻します。

(4) 戻り値

なし。

(5) 機能詳細

このメソッドで設定する接続付加情報を次に示します。

設定内容	設定する内容	設定の要否
HiRDB のポート番号	HiRDB のポート番号を文字列で設定します。 HiRDB のポート番号の各設定方法での優先順位については，「接続情報の優先順位」を参照してください。	任意
HiRDB クライアントの環境変数グループ名	HiRDB クライアントの環境変数グループ名を，「@HIRDBENVGRP=」に続けて絶対パス名で指定します。次の点に注意してください。 • 「@HIRDBENVGRP=,」のように，"="の後に何も指定しないと，この項目による指定がないものとみなされます。 • 環境変数グループ名は大文字と小文字を区別するため，注意してください。なお，環境変数グループ名は OS に依存します。 • 環境変数グループ名に半角スペース，又は半角@文字を含む場合，半角引用符（"）で囲んでください。環境変数グループ名を半角引用符で囲んだ場合，最後の半角引用符から文字終端までの文字は無視されます。また，半角引用符，半角コンマを含む環境変数グループ名は指定できません。 • Windows の場合，HiRDB クライアント環境変数登録ツールで登録した環境変数グループ名は指定できません。	任意
HiRDB の環境変数グループ識別子	HiRDB の環境変数グループ識別子を，英数字だけの 4 文字で設定します。	XA 接続時は必要

注 1

指定例を次に示します。なお，指定例では，PrdbDataSource クラスのインスタンスの参照を持つ変数名を「ds」とします。また，△は半角スペースを示します。

例 1：HiRDB のポート番号を指定する場合

```
ds.setDescription (”22200”);
```

例 2：環境変数グループ名のパスが「C:¥HiRDB_P¥Client¥HiRDB.ini」の場合

```
ds.setDescription (”@HIRDBENVGRP=C:¥¥HiRDB_P¥¥Client¥¥HiRDB.ini”);
```

例 3：環境変数グループ名のパスが「C:¥Program△Files¥HITACHI¥HiRDB¥HiRDB.ini」の場合

```
ds.setDescription (”@HIRDBENVGRP=¥”C:¥¥Program△Files¥¥HITACHI¥¥HiRDB¥HiRDB.ini¥””);
```

例 4：環境変数グループ名のパスが/HiRDB_P/Client/HiRDB.ini の場合

```
ds.setDescription (”@HIRDBENVGRP=/HiRDB_P/Client/HiRDB.ini”);
```

例 5：XA 接続時，HiRDB の環境変数グループ識別子を指定する場合

```
ds.setDescription (”HDB1”);ds.setXAOpenString (”HDB1+C:¥¥Program△Files¥¥HITACHI¥¥HiRDB¥¥HiRDB.ini”);
```

注 2

環境変数グループ名を指定する場合は，指定内容に半角スペースを含めないでください。エラーになる指定例を次に示します。

```
@△HIRDBENVGRP=/HiRDB_P/Client/HiRDB.ini
@HIRDBENVGRP△=/HiRDB_P/Client/HiRDB.ini
@HIRDBENVGRP=△/HiRDB_P/Client/HiRDB.ini
@HIRDBENVGRP=/HiRDB_P/Client/HiRDB.ini△
```

注　△は半角スペースを示します。

(6)　発生する例外

半角@文字から始まる環境変数グループ名を指定する場合，半角@文字より後の指定内容に半角スペースがあるときは，SQLException を投入します。

17.11.2　getDescription

(1)　機能

データベースへの接続に必要な情報（接続付加情報）を取得します。

(2)　形式

```
public String getDescription() throws SQLException
```

(3)　引数

なし。

(4)　戻り値

String：

接続付加情報（設定されていない場合，ナル値を返却します）

(5)　機能詳細

setDescription メソッドで設定された，データベースへの接続に必要な情報（接続付加情報）を返却します。

(6)　発生する例外

なし。

17.11.3　setDBHostName

(1)　機能

接続する HiRDB のホスト名を設定します。

(2)　形式

```
public void setDBHostName ( String db_host_name ) throws SQLException
```

(3)　引数

String db_host_name：

HiRDB のホスト名を指定します。

ナル値の場合は，このメソッドによって現在までに設定したホスト名を無効とし，初期状態に戻します。

(4)　戻り値

なし。

(5)　機能詳細

接続する HiRDB のホスト名を設定します。

HiRDB のホスト名の各設定方法での優先順位は，「接続情報の優先順位」を参照してください。

(6)　発生する例外

なし。

17.11.4　getDBHostName

(1)　機能

接続する HiRDB のホスト名を取得します。

(2)　形式

```
public String getDBHostName() throws SQLException
```

(3) 引数

なし。

(4) 戻り値

String：

HiRDB のホスト名（設定されていない場合，ナル値を返却します）

(5) 機能詳細

setDBHostName メソッドで設定された，接続する HiRDB のホスト名を返却します。

(6) 発生する例外

なし。

17.11.5 setJDBC_IF_TRC

(1) 機能

JDBC インタフェースメソッドトレースの取得の有無を設定します。

(2) 形式

```
public void setJDBC_IF_TRC ( boolean flag ) throws SQLException
```

(3) 引数

boolean flag：トレースの取得の有無を指定します。

true：取得します。

false：取得しません。

(4) 戻り値

なし。

(5) 機能詳細

JDBC インタフェースメソッドトレースの取得の有無を設定します。

このメソッドが呼び出されない場合のデフォルト値は false（取得しない）です。なお，別途 setLogWriter メソッドで有効な出力先を設定しておいてください。JDBC インタフェースメソッドトレースの詳細は，「JDBC インタフェースメソッドトレース」を参照してください。

(6) 発生する例外

なし。

(7) 注意事項

JDBC インタフェースメソッドトレースの取得の有無は，インスタンス単位では設定できません。このメソッドで設定した JDBC インタフェースメソッドトレースの取得の有無は，設定時点，及び設定以降に存在するすべての DataSource，ConnectionPoolDataSource，及び XADataSource のインスタンスに影響します。

17.11.6 getJDBC_IF_TRC

(1) 機能

JDBC インタフェースメソッドトレースの取得の有無を取得します。

(2) 形式

```
public boolean getJDBC_IF_TRC() throws SQLException
```

(3) 引数

なし。

(4) 戻り値

boolean：

トレース取得の有無です。

true：取得します。

false：取得しません。

(5) 機能詳細

setJDBC_IF_TRC メソッドで設定された，JDBC インタフェースメソッドトレースの取得の有無を返却します。

JDBC インタフェースメソッドトレースの詳細は，「JDBC インタフェースメソッドトレース」を参照してください。

(6)　発生する例外

なし。

17.11.7　setTRC_NO

(1)　機能

JDBC インタフェースメソッドトレースのエントリ数を設定します。

(2)　形式

```
public void setTRC_NO ( int trc_no ) throws SQLException
```

(3)　引数

int trc_no：

JDBC インタフェースメソッドトレースのエントリ数を指定します。

(4)　戻り値

なし。

(5)　機能詳細

JDBC インタフェースメソッドトレースのエントリ数を，10～1,000 の範囲で設定します。

このメソッドが呼び出されない場合，デフォルトの JDBC インタフェースメソッドトレースのエントリ数は 500 です。

JDBC インタフェースメソッドトレースの詳細は，「JDBC インタフェースメソッドトレース」を参照してください。

(6)　発生する例外

10～1,000 以外の値を設定した場合は，SQLException を投入します。

17.11.8　getTRC_NO

(1)　機能

JDBC インタフェースメソッドトレースのエントリ数を取得します。

(2)　形式

```
public int getTRC_NO() throws SQLException
```

(3)　引数

なし。

(4)　戻り値

int：

JDBC インタフェースメソッドトレースのエントリ数（設定されていない場合，デフォルトである 500 を返却します）

(5)　機能詳細

setTRC_NO メソッドで設定された，JDBC インタフェースメソッドトレースのエントリ数を返却します。

JDBC インタフェースメソッドトレースの詳細は，「JDBC インタフェースメソッドトレース」を参照してください。

(6)　発生する例外

なし。

17.11.9　setUapName

(1)　機能

UAP 名称を設定します。

(2)　形式

```
public void setUapName ( String uap_name ) throws SQLException
```

(3) 引数

String uap_name：

UAP 名称を指定します。

ナル値の場合は，このメソッドによって現在までに設定した UAP 名称を無効とし，初期状態に戻します。

(4) 戻り値

なし。

(5) 機能詳細

UAP 名称を設定します。

指定された UAP 名称は，次の箇所で使用されます。

- 各種トレース情報への出力情報
- pdls コマンドに-d prc オプションを付与して実行したときに出力される UAP の識別名称

次のどちらかの場合は，このメソッドによる UAP 名称の設定がないものとみなします。設定がない場合の扱いについては，「接続情報の優先順位」を参照してください。

- 引数 uap_name にナル値を指定した場合
- 引数 uap_name に長さ 0 の文字列，又は半角スペースだけの文字列を指定した場合

(6) 発生する例外

なし。

(7) 注意事項

このメソッドで指定した UAP は，setEncodeLang メソッドで指定された変換文字セットでエンコードされ，エンコード後の UAP 名称の先頭から 30 バイトが HiRDB サーバに転送されます（30 バイト目が文字の途中であっても，30 バイトで打ち切られます）。そのため，HiRDB サーバで取得できる UAP 名称は，エンコード後の先頭 30 バイトまでです。

17.11.10 getUapName

(1) 機能

UAP 名称を取得します。

(2) 形式

```
public String getUapName() throws SQLException
```

(3) 引数

なし。

(4) 戻り値

String：

UAP 名称

(5) 機能詳細

setUapName メソッドで設定された UAP 名称を返却します。

UAP 名称が設定されていない場合，又は null が設定されている場合，次のどれかを返却します。

- JDBC2.0：HiRDB_Type4_JDBC_Driver
- JDBC4.0：HiRDB_Type4_JDBC40_Driver
- JDBC4.3：HiRDB_Type4_JDBC43_Driver

(6) 発生する例外

なし。

17.11.11 setUser

(1) 機能

データベース接続時の認可識別子を設定します。

(2) 形式

```
public void setUser ( String user ) throws SQLException
```

(3) 引数

String user：

認可識別子を指定します。

ナル値の場合は，このメソッドによって現在までに設定した認可識別子を無効とし，初期状態に戻します。

(4)　戻り値

なし。

(5)　機能詳細

認可識別子を設定します。

次のメソッドの実行時に，setUser メソッド，及び setPassword メソッドで指定された認可識別子及びパスワードを使用し，データベースへの物理接続を確保します。

- DataSource インタフェースの getConnection メソッド（引数なし）
- ConnectionPoolDataSource インタフェースの getPooledConnection メソッド
- XADataSource インタフェースの getXAConnection メソッド

なお，引数 user がナル値の場合は，このメソッドによる認可識別子の設定がないものとみなします。

設定がない場合の扱いについては，「接続情報の優先順位」を参照してください。

(6)　発生する例外

引数 user で指定した文字列の長さが 0 の場合，SQLException を投入します。

17.11.12　getUser

(1)　機能

データベース接続時の認可識別子を取得します。

(2)　形式

```
String void getUser() throws SQLException
```

(3)　引数

なし。

(4) 戻り値

String：

認可識別子

(5) 機能詳細

setUser メソッドで設定された認可識別子を返却します。設定されていない場合は，ナル値を返却します。OS ログインユーザの簡易認証機能を使用している場合は，簡易認証キーワードを返却します。

(6) 発生する例外

なし。

17.11.13 setPassword

(1) 機能

データベース接続時のパスワードを設定します。

(2) 形式

```
public void setPassword ( String password ) throws SQLException
```

(3) 引数

String password：

パスワードを指定します。

ナル値の場合は，このメソッドによって現在までに設定したパスワードを無効とし，初期状態に戻します。

(4) 戻り値

なし。

(5) 機能詳細

パスワードを設定します。

次のメソッドの実行時に，setUser メソッド，及び setPassword メソッドで指定された認可識別子及びパスワードを使用し，データベースへの物理接続を確保します。

- DataSource インタフェースの getConnection メソッド（引数なし）
- ConnectionPoolDataSource インタフェースの getPooledConnection メソッド
- XADataSource インタフェースの getXAConnection メソッド

引数 password がナル値又は長さ 0 の文字列の場合，このメソッドによるパスワードの設定がないものとみなします。設定がない場合の扱いについては，「接続情報の優先順位」を参照してください。

(6)　発生する例外

なし。

17.11.14　getPassword

(1)　機能

データベース接続時のパスワードを取得します。

(2)　形式

```
public String getPassword() throws SQLException
```

(3)　引数

なし。

(4)　戻り値

String：

パスワード

(5)　機能詳細

setPassword メソッドで指定されたパスワードを返却します。

(6)　発生する例外

なし。

17.11.15　setXAOpenString

(1)　機能

XA オープン文字列を設定します。

(2)　形式

```
public void setXAOpenString ( String xa_string ) throws SQLException
```

(3)　引数

String xa_string：

XA オープン文字列を指定します。

ナル値の場合は，このメソッドによって現在までに設定した XA オープン文字列を無効とし，初期状態に戻します。

(4)　戻り値

なし。

(5)　機能詳細

XA オープン文字列を設定します。このメソッドは，XADataSource インタフェースでだけ提供されます。XA オープン文字列は，次の形式で指定します。

[形式]

```
HiRDBの環境変数グループ識別子＋HiRDBクライアントの環境変数グループ名
```

HiRDB の環境変数グループ識別子は，setDescription メソッドでの設定と同一にしてください。なお，setDescription メソッドで HiRDB クライアントの環境変数グループ名を指定する場合と異なり，HiRDB クライアントの環境変数グループ名中に半角@文字や半角スペースを含んでいても，引用符で囲む必要はありません。

設定例を次に示します。

[設定例 1]

HiRDB クライアントの環境変数グループ名のパスが「/HiRDB/HiRDB.ini」の場合

```
ds.setDescription("HDB1");
ds.setXAOpenString("HDB1+/HiRDB/HiRDB.ini");
```

[設定例 2]

HiRDB クライアントの環境変数グループ名のパスが「C:¥Program△Files¥HITACHI¥HiRDB¥HiRDB.ini」の場合（△は半角スペース）

```
ds.setDescription(”HDB1”);
ds.setXAOpenString(”HDB1+C:¥¥Program△Files¥¥HITACHI¥¥HiRDB¥¥HiRDB.ini”);
```

(6) 発生する例外

なし。

17.11.16 getXAOpenString

(1) 機能

XA オープン文字列を取得します。

(2) 形式

```
public String getXAOpenString() throws SQLException
```

(3) 引数

なし。

(4) 戻り値

String：

XA オープン文字列（設定されていない場合，ナル値を返却します）

(5) 機能詳細

setXAOpenString メソッドで設定された，XA オープン文字列を返却します。このメソッドは，XADataSource インタフェースでだけ提供されます。

(6) 発生する例外

なし。

17.11.17　setXACloseString

(1)　機能

XA クローズ文字列を設定します。

(2)　形式

```
public void setXACloseString ( String xa_string ) throws SQLException
```

(3)　引数

String xa_string：

XA クローズ文字列を指定します。

ナル値の場合は，このメソッドによって現在までに設定した XA クローズ文字列を無効とし，初期状態に戻します。

(4)　戻り値

なし。

(5)　機能詳細

XA クローズ文字列を設定します。このメソッドは，XADataSource インタフェースでだけ提供されます。

(6)　発生する例外

なし。

17.11.18　getXACloseString

(1)　機能

XA クローズ文字列を取得します。

(2)　形式

```
public String getXACloseString() throws SQLException
```

(3) 引数

なし。

(4) 戻り値

String：

XA クローズ文字列（設定されていない場合，ナル値を返却します）

(5) 機能詳細

setXACloseString メソッドで設定された，XA クローズ文字列を返却します。このメソッドは，XADataSource インタフェースでだけ提供されます。

(6) 発生する例外

なし。

17.11.19 setLONGVARBINARY_Access

(1) 機能

JDBC SQL タイプ LONGVARBINARY（HiRDB のデータ型である BLOB 型，及び BINARY 型）のデータへのアクセス方法を設定します。

(2) 形式

```
public void setLONGVARBINARY_Access ( String mode ) throws SQLException
```

(3) 引数

String mode：

JDBC SQL タイプ LONGVARBINARY（HiRDB のデータ型である BLOB 型，及び BINARY 型）のデータへのアクセス方法を指定します。

このメソッドでは，引数に指定した内容について大文字と小文字を区別しません。

REAL：

実データでアクセスします。

LOCATOR：

HiRDB の位置付け子機能を使用してアクセスします。

ナル値の場合は，現在までに設定したデータへのアクセス方法を無効とし，初期状態に戻します。

(4) 戻り値

なし。

(5) 機能詳細

JDBC SQL タイプ LONGVARBINARY（HiRDB のデータ型である BLOB 型，及び BINARY 型）のデータへのアクセス方法を設定します。このメソッドが呼び出されない場合のデフォルト値は"REAL"です。

このメソッドでの設定は，「ユーザプロパティ」で示したプロパティ LONGVARBINARY_ACCESS の設定と同等になります。

(6) 発生する例外

引数 mode に"REAL"及び"LOCATOR"以外の値を指定した場合は，SQLException を投入します。

(7) 注意事項

「LONGVARBINARY_ACCESS」の注意事項を参照してください。

17.11.20 getLONGVARBINARY_Access

(1) 機能

JDBC SQL タイプ LONGVARBINARY（HiRDB のデータ型である BLOB 型，及び BINARY 型）のデータへのアクセス方法を取得します。

(2) 形式

```
public String getLONGVARBINARY_Access()
```

(3) 引数

なし。

(4) 戻り値

String：

JDBC SQL タイプ LONGVARBINARY（HiRDB のデータ型である BLOB 型，及び BINARY 型）のデータへのアクセス方法の設定情報

REAL：

実データでアクセスします。

LOCATOR：

HiRDB の位置付け子機能を使用してアクセスします。

(5) 機能詳細

setLONGVARBINARY_Access メソッドで設定された情報を返却します。

(6) 発生する例外

なし。

17.11.21 setSQLInNum

(1) 機能

実行する SQL の入力？パラメタの最大数を設定します。

(2) 形式

```
public void setSQLInNum ( int inNum ) throws SQLException
```

(3) 引数

int inNum：

実行する SQL の入力？パラメタの最大数を指定します。指定値は 1～30,000 です。

(4) 戻り値

なし。

(5) 機能詳細

SQL の前処理時に取得する入力？パラメタ情報の数を設定します。

実際の？パラメタの数がこのメソッドの指定値よりも多い場合，SQL の前処理の後に入力？パラメタ情報を取得します。このメソッドが呼び出されない場合のデフォルト値は 300 です。

このメソッドで指定した値は，データベース接続時に「ユーザプロパティ」で示したプロパティ HiRDB_for_Java_SQL_IN_NUM の値となります。

(6) 発生する例外

引数に 1～30,000 の以外の値を指定した場合，SQLException を投入します。

(7) 注意事項

入力？パラメタのある SQL 文を実行しない場合，引数は 1 を指定することを推奨します。

17.11.22 getSQLInNum

(1) 機能

実行する SQL の入力？パラメタの最大数を取得します。

(2) 形式

```
public int getSQLInNum() throws SQLException
```

(3) 引数

なし。

(4) 戻り値

int：

setSQLInNum で設定した，実行する SQL の入力？パラメタの最大数（設定されていない場合，デフォルトである 300 を返却します）

(5) 機能詳細

setSQLInNum で設定した，実行する SQL の入力？パラメタの最大数を取得します。

(6) 発生する例外

なし。

17.11.23 setSQLOutNum

(1) 機能

実行する SQL の検索項目の最大数を設定します。

(2) 形式

```
public void setSQLOutNum ( int outNum ) throws SQLException
```

(3) 引数

int outNum：

実行する SQL の検索項目の最大数を指定します。指定値は 1～30,000 です。

(4) 戻り値

なし。

(5) 機能詳細

実行する SQL の検索項目の最大数を設定します。

この指定は，SQL の前処理時に取得する出力項目数の数となります。このメソッドが呼び出されない場合のデフォルト値は 300 です。

実際の出力項目数がこのメソッドの指定値よりも多い場合，SQL の前処理の後に出力項目情報を取得します。

このメソッドで指定した値は，データベース接続時に「ユーザプロパティ」で示したプロパティ HiRDB_for_Java_SQL_OUT_NUM の値となります。

(6) 発生する例外

引数が 1～30,000 の範囲外である場合，SQLException を投入します。

(7) 注意事項

検索項目がない場合，引数は 1 を指定することを推奨します。

17.11.24 getSQLOutNum

(1) 機能

実行する SQL の検索項目の最大数を取得します。

(2) 形式

```
public int getSQLOutNum() throws SQLException
```

(3) 引数

なし。

(4) 戻り値

int：

setSQLOutNum で設定した実行する SQL の検索項目の最大数（設定されていない場合，デフォルトである 300 を返却します）

(5) 機能詳細

setSQLOutNum で設定した，実行する SQL の検索項目の最大数を取得します。

(6) 発生する例外

なし。

17.11.25 setSQLWarningLevel

(1) 機能

SQL 実行時に発生した警告保持レベルを設定します。

(2) 形式

```
public void setSQLWarningLevel ( String warningLevel ) throws SQLException
```

(3) 引数

String warningLevel：

SQL 実行時に発生した警告情報の保持レベルを指定します。

指定できる値を次に示します。指定値と保持する警告の関係は，「SQLWarning オブジェクトの生成条件」を参照してください。

- IGNORE
- SQLWARN
- ALLWARN

なお，このメソッドでは，引数に指定した内容について大文字と小文字を区別しません。ナル値の場合は，このメソッドによって現在までに設定した警告保持レベルを無効とし，初期状態に戻します。

(4)　戻り値

なし。

(5)　機能詳細

SQL 実行時に発生した警告情報の保持レベルを設定します。このメソッドが呼び出されない場合のデフォルト値は"SQLWARN"です。

このメソッドで指定した値は，データベース接続時に「ユーザプロパティ」で示したプロパティ HiRDB_for_Java_SQLWARNING_LEVEL の値となります。

(6)　発生する例外

引数が上記の指定値以外である場合，SQLException を投入します。

17.11.26　getSQLWarningLevel

(1)　機能

setSQLWarningLevel メソッドで設定した警告保持レベルを取得します。

(2)　形式

```
public String getSQLWarningLevel() throws SQLException
```

(3)　引数

なし。

(4)　戻り値

String：

setSQLWarningLevel で設定した警告保持レベルを返却します。返却値と保持する警告の関係は，「SQLWarning オブジェクトの生成条件」を参照してください。

(5)　機能詳細

setSQLWarningLevel メソッドで設定された情報を返却します。設定されていない場合，デフォルト値である"SQLWARN"を返却します。

(6)　発生する例外

なし。

17.11.27　setXALocalCommitMode

(1)　機能

XA 接続時，トランザクションが分散トランザクションでない場合，オートコミット機能を有効にするかどうかを設定します。

(2)　形式

```
public void setXALocalCommitMode ( boolean autoCommitMode ) throws SQLException
```

(3)　引数

boolean autoCommitMode：

オートコミット機能を設定します。

true：オートコミット機能を有効に設定します。

false：オートコミット機能を無効に設定します。

(4)　戻り値

なし。

(5)　機能詳細

XA 接続時にオートコミット機能を設定します。デフォルト値は false（オートコミット機能は無効）です。このメソッドの指定値と，JDBC ドライバの動作の関係を次に示します。

指定値	条件	JDBC ドライバの動作
true	Connetion オブジェクト生成時の自動コミットのデフォルト	オートコミット有効
	con.commit メソッド及び con.rollback メソッドによるトランザクション終了	正常に受け付け
	setAutoCommit(true)の実行	オートコミットを有効
	setAutoCommit(false)の実行	オートコミットを無効
false（デフォルト）	Connetion オブジェクト生成時の自動コミットのデフォルト	オートコミット無効
	con.commit メソッド及び con.rollback メソッドによるトランザクション終了	SQLException

指定値	条件	JDBC ドライバの動作
	setAutoCommit(true)の実行	SQLException
	setAutoCommit(false)の実行	正常終了（オートコミットを有効にできないので何もしない）

(6)　発生する例外

なし。

17.11.28　getXALocalCommitMode

(1)　機能

XA 接続時，トランザクションが分散トランザクションでない場合，オートコミット機能を有効にするかどうかの設定情報を取得します。

(2)　形式

```
public boolean getXALocalCommitMode() throws SQLException
```

(3)　引数

なし。

(4)　戻り値

boolean：

オートコミット機能の設定です。

true：オートコミット機能は有効です。

false：オートコミット機能は無効です。

(5)　機能詳細

オートコミット機能の設定を取得します。

(6)　発生する例外

なし。

17.11.29　setSQLWarningIgnore

(1)　機能

データベースから返される警告を Connection クラスで保持するかどうかを設定します。

(2)　形式

```
public void setSQLWarningIgnore ( boolean mode )
```

(3)　引数

boolean mode：

警告を保持するかどうかを設定します。

true：警告を保持しません。

false：警告を保持します。

(4)　戻り値

なし。

(5)　機能詳細

Connection クラスで発生した警告を Connection クラスで保持するかどうかを設定します。デフォルト値は"false"です。

このメソッドの実行は，「URL の構文」で示した項目 SQLWARNING_IGNORE，及び「ユーザプロパティ」で示したプロパティ SQLWARNING_IGNORE の設定と同等です。

指定値と保持する警告の関係については，「SQLWarning オブジェクトの生成条件」を参照してください。

(6)　発生する例外

なし。

17.11.30　getSQLWarningIgnore

(1)　機能

データベースから返される警告を Connection クラスで保持するかどうかの設定情報を取得します。

(2) 形式

```
public boolean getSQLWarningIgnore()
```

(3) 引数

なし。

(4) 戻り値

boolean：

警告を保持するかどうかの設定情報です。

true：警告を保持しません。

false：警告を保持します。

(5) 機能詳細

Connection クラスで発生した警告を Connection クラスで保持するかどうかの設定情報を取得します。

setSQLWarningIgnore メソッドで設定した情報を返却します。設定されていない場合，デフォルト値である"false"を返却します。

返却値と保持する警告の関係については，「SQLWarning オブジェクトの生成条件」を参照してください。

(6) 発生する例外

なし。

17.11.31 setHiRDBCursorMode

(1) 機能

HiRDB がコミットした場合に ResultSet クラスのオブジェクトを有効とするかどうかを設定します。

(2) 形式

```
public void setHiRDBCursorMode ( boolean mode )
```

(3) 引数

boolean mode：

次の値を指定します。

true：コミットした場合でも ResultSet クラスのオブジェクトを有効とします。なお，true を指定した場合，次のクラスのオブジェクトについても，コミット後も有効となります。

- Statement クラス
- PreparedStatement クラス
- CallableStatement クラス

false：コミットした場合，ResultSet クラスのオブジェクトを無効とします。

(4) 戻り値

なし。

(5) 機能詳細

HiRDB がコミットした場合に，ResultSet クラスのオブジェクトを有効とするかどうかを設定します。このメソッドが呼び出されない場合のデフォルトは false です。

無効となった ResultSet オブジェクトで close メソッド呼び出し以外の操作を行った場合，SQLException を投入します。

このメソッドの実行は，「URL の構文」で示した項目 HIRDB_CURSOR の設定と同等です。

(6) 発生する例外

なし。

(7) 注意事項

「HIRDB_CURSOR 及び STATEMENT_COMMIT_BEHAVIOR 指定時の注意事項」を参照してください。

17.11.32 getHiRDBCursorMode

(1) 機能

HiRDB がコミットした場合に ResultSet クラスのオブジェクトを有効とするかどうかの設定情報を取得します。

(2) 形式

```
public boolean getHiRDBCursorMode()
```

(3) 引数

なし。

(4) 戻り値

boolean：

HiRDB がコミットした場合に，ResultSet クラスのオブジェクトを有効とするかどうかの設定情報です。

true：コミットした場合でも ResultSet クラスのオブジェクトが有効です。

false：コミットした場合，ResultSet クラスのオブジェクトは無効です。

(5) 機能詳細

HiRDB がコミットした場合に ResultSet クラスのオブジェクトを有効とするかどうかの設定情報を取得します。

(6) 発生する例外

なし。

(7) 注意事項

なし。

17.11.33 setNotErrorOccurred

(1) 機能

ConnectionEventListener.connectionErrorOccurred の発生を抑止するかどうかを設定します。

(2) 形式

```
public void setNotErrorOccurred ( boolean mode )
```

(3) 引数

boolean mode：

connectionErrorOccurred の発生を抑止するかどうかを指定します。

true：connectionErrorOccurred の発生を抑止します。

false：connectionErrorOccurred の発生を抑止しません（デフォルト）。

(4) 戻り値

なし。

(5) 機能詳細

ConnectionPoolDataSource，XADataSource を使用している場合，致命的な接続エラーが発生したときに呼ばれる ConnectionEventListener.connectionErrorOccurred の，呼び出しを抑止するための設定をします。

設定していない場合，connectionErrorOccurred を呼びます。通常は未設定にするか，又は false を設定します。

(6) 発生する例外

なし。

17.11.34 getNotErrorOccurred

(1) 機能

ConnectionEventListener.connectionErrorOccurred の発生を抑止するかどうかの設定情報を取得します。

(2) 形式

```
public boolean getNotErrorOccurred()
```

(3) 引数

なし。

(4) 戻り値

boolean：

ConnectionEventListener.connectionErrorOccurred を発生させるかどうかの設定情報です。

true：connectionErrorOccurred は発生しません。

false：connectionErrorOccurred が発生します（デフォルト）。

(5)　機能詳細

ConnectionPoolDataSource，XADataSource を使用している場合，致命的な接続エラーが発生したときに，ConnectionEventListener.connectionErrorOccurred が呼ばれるかどうかの設定情報を取得します。設定していない場合は，false を返却します。

(6)　発生する例外

なし。

17.11.35　setEnvironmentVariables

(1)　機能

HiRDB のクライアント環境定義を設定します。

(2)　形式

```
public void setEnvironmentVariables ( String variables ) throws SQLException
```

(3)　引数

String variables：

HiRDB のクライアント環境定義を，次の形式で指定します。

[形式]

```
”変数名＝値;変数名＝値; ..〈省略〉.. ;変数名＝値”
```

指定例を次に示します。

[指定例]

```
setEnvironmentVariables (”PDFESHOST=FES1;PDCWAITTIME=0”);
```

ナル値の場合は，このメソッドによって現在までに設定したクライアント環境定義を無効とし，初期状態に戻します。

(4)　戻り値

なし。

(5)　機能詳細

HiRDB クライアント環境定義を設定します。

JDBC ドライバで指定できるクライアント環境定義は，「指定できるクライアント環境定義」を参照してください。変数名に JDBC ドライバで指定できないクライアント環境定義が指定された場合，指定を無視します。なお，変数名は大文字と小文字を区別するため，注意してください。

複数の設定方法を持つ接続情報の優先順位については，「接続情報の優先順位」を参照してください。

なお，このメソッドでは，クライアント環境定義の各指定値をチェックしません。データベース接続時に指定値をチェックし，不正であれば SQLException を投入します。

(6) 発生する例外

なし。

17.11.36 getEnvironmentVariables

(1) 機能

HiRDB のクライアント環境定義を取得します。

(2) 形式

```
String void getEnvironmentVariables()
```

(3) 引数

なし。

(4) 戻り値

String：

HiRDB のクライアント環境定義（未指定の場合，ナル値を返却します）

(5) 機能詳細

HiRDB のクライアント環境定義を取得します。

(6) 発生する例外

なし。

17.11.37　setEncodeLang

(1)　機能

データ変換時の変換文字セット名称を設定します。

(2)　形式

```
public void setEncodeLang ( String encode_lang ) throws SQLException
```

(3)　引数

String encode_lang：

変換文字セットを指定します。「Java 2 SDK, Standard Edition ドキュメント」の「国際化」で示されるエンコーディング一覧から選択してください。

HiRDB の文字コードと対応する変換文字セットを次に示します。

HiRDB の文字コード （pdntenv 又は pdsetup コマンドで設定した文字コード）	指定する変換文字セット
lang-c	ISO8859_1
sjis	SJIS 又は MS932※
ujis	EUC_JP
utf-8	UTF-8
utf-8_ivs	UTF-8
chinese	EUC_CN
chinese-gb18030	GB18030

注※

SJIS か MS932 の指定は，アプリケーションでの Windows 特殊文字の扱いによります。

"OFF"を指定すると，HiRDB の文字コードに対して上記の表の変換文字セットが指定されたものとして動作します。なお，HiRDB の文字コードが sjis の場合，JDBC ドライバが動作する OS によって変換文字セットは次のようになります。

UNIX の場合：**SJIS**

Windows の場合：**MS932**

ナル値の場合は，現在までに設定したこのメソッドによる変換文字セット名称を無効とし，初期状態に戻します。

"OFF"以外を指定した場合は，大文字と小文字を区別します。

(4) 戻り値

なし。

(5) 機能詳細

次のデータ変換時に，このメソッドで指定された変換文字セット名称での変換を行います。

- HiRDB から取得したデータをアプリケーションが String で取得する場合の，文字データ（Unicode）への変換
- アプリケーションが String で HiRDB へ値を設定する場合の，バイナリデータへの変換

このメソッドを指定しない場合，JDBC ドライバは前記対応表に合わせた変換文字セットで文字の変換をします。ただし，次の項目は，Java 仮想マシンのデフォルトの変換文字セットで変換されます。

- setUapName の指定値
- 認可識別子又はパスワード（setUser，setPassword，getConnection などで指定するもの）
- setEnvironmentVariables で指定したクライアント環境定義の指定値
- HiRDB クライアントの環境変数グループ名で指定した各環境変数の指定値

(6) 発生する例外

Java 仮想マシンがサポートしない変換文字セットを指定した場合，SQLException を投入します。

17.11.38 getEncodeLang

(1) 機能

設定された変換文字セット名称を取得します。

(2) 形式

```
public String getEncodeLang()
```

(3) 引数

なし。

(4) 戻り値

String：

変換文字セット名称（setEncodeLang メソッドで変換文字セット名称が指定されていない場合は，ナル値を返却します）

(5) 機能詳細

setEncodeLang メソッドで設定された変換文字セット名称を返却します。

(6) 発生する例外

なし。

17.11.39 setMaxBinarySize

(1) 機能

JDBC SQL タイプ LONGVARBINARY 型データ取得時のデータサイズの上限を設定します。

(2) 形式

```
public void setMaxBinarySize ( int size ) throws SQLException
```

(3) 引数

int size：

上限となるデータサイズです。0～2,147,483,647 の範囲で設定します。

0 を指定した場合は，取得対象データの定義長を上限とします。

(4) 戻り値

なし。

(5) 機能詳細

JDBC SQL タイプ LONGVARBINARY 型データ取得時のデータサイズの上限を，バイト単位で設定します。

JDBC SQL タイプ LONGVARBINARY 型データを取得する際，JDBC ドライバはデータを取得するまで実際のデータ長を認識できないため，定義長分のメモリを確保します。そのため，定義長に HiRDB のデータ型である BINARY 型，及び BLOB 型の最大長である 2,147,483,647 バイトのように長大なサイズを指定した

列の値を取得する場合は，その定義長である 2,147,483,647 バイトのメモリを確保しようとします。そのため，実行環境によってはメモリ不足が発生することがあります。

したがって，このメソッドで，実際に格納されているデータの最大長を指定してください。取得対象となる HiRDB のデータ型である BINARY 型，及び BLOB 型データの定義長が，このメソッドで指定したサイズよりも大きい場合，取得データをこのメソッドで指定したサイズに切り捨てます。実際にデータを切り捨てた場合，ResultSet の next メソッド実行時に，JDBC ドライバは HiRDB サーバから警告を受け取ります。受け取った警告に対しては，setLONGVARBINARY_TruncError の指定値に従って SQLException の投入，SQLWarning の生成（又は無視）をします。

このメソッドで上限を設定していない場合は，取得対象データの定義長を上限とします。

(6) 発生する例外

負の値を指定した場合，SQLException を投入します。

(7) 注意事項

setLONGVARBINARY_Access メソッドの引数 mode に LOCATOR を指定している場合，このメソッドの指定値は無効です。実際のデータ長に基づいて領域を確保し，全データを取得します。

17.11.40 getMaxBinarySize

(1) 機能

JDBC SQL タイプ LONGVARBINARY 型データ取得時のデータサイズの上限を取得します。

(2) 形式

```
public int getMaxBinarySize()
```

(3) 引数

なし。

(4) 戻り値

int：

上限となるデータサイズの設定値

(5)　機能詳細

setMaxBinarySize で設定した，JDBC SQL タイプ LONGVARBINARY 型データ取得時のデータサイズの上限を返却します。

setMaxBinarySize でデータサイズの上限を設定していない場合は，0 を返却します。

(6)　発生する例外

なし。

17.11.41　setStatementCommitBehavior

(1)　機能

コミットを行った場合，ステートメントオブジェクトをコミット実行後も有効とするかどうかを設定します。なお，ここでのステートメントオブジェクトとは，次のクラスのことを指します。

- Statement クラス
- PreparedStatement クラス
- CallableStatement クラス

(2)　形式

```
public void setStatementCommitBehavior ( boolean mode ) throws SQLException
```

(3)　引数

boolean mode：

ステートメントオブジェクトが，コミットなどによるトランザクションの終了の前後にわたって有効かを指定します。

true：トランザクション終了後もステートメントのオブジェクトを有効にします。

false：トランザクション終了後はステートメントのオブジェクトを無効にします。

(4)　機能詳細

コミットを行った場合，ステートメントオブジェクトをコミット実行後も有効とするかを設定します。このメソッドが呼び出されない場合のデフォルトは true です。

このメソッドの実行は，「URL の構文」で示した項目 STATEMENT_COMMIT_BEHAVIOR の設定と同等です。

(5)　発生する例外

なし。

(6)　注意事項

「HIRDB_CURSOR 及び STATEMENT_COMMIT_BEHAVIOR 指定時の注意事項」を参照してください。

17.11.42　getStatementCommitBehavior

(1)　機能

コミットを行った場合，ステートメントオブジェクトをコミット実行後も有効とするかどうかの設定情報を取得します。なお，ここでのステートメントオブジェクトとは，次のクラスのことを指します。

- Statement クラス
- PreparedStatement クラス
- CallableStatement クラス

(2)　形式

```
public boolean getStatementCommitBehavior() throws SQLException
```

(3)　引数

なし。

(4)　戻り値

boolean：

ステートメントオブジェクトが，コミットなどによるトランザクションの終了の前後にわたって有効となるかどうかです。

true：有効となります。

false：有効となりません。

(5)　機能詳細

コミットを行った場合に，次のクラスのオブジェクトをコミット実行後も有効とするかどうかの設定情報を取得します。

- Statement クラス

- PreparedStatement クラス
- CallableStatement クラス

このメソッドは，setStatementCommitBehavior メソッドの設定値を返却します。なお，設定していない場合は true を返却します。

(6) 発生する例外

なし。

(7) 注意事項

なし。

17.11.43 setLONGVARBINARY_AccessSize

(1) 機能

HiRDB サーバに対して一度に要求する JDBC SQL タイプ LONGVARBINARY 型データの長さを設定します。

(2) 形式

```
public void setLONGVARBINARY_AccessSize ( int access_size ) throws SQLException
```

(3) 引数

int access_size：

要求するデータ長をキロバイト単位で指定します。指定値は，0～2,097,151 です（デフォルトは 0）。0 を指定した場合は，データ全体を一度に要求します。

(4) 戻り値

なし。

(5) 機能詳細

HiRDB サーバに対して一度に要求する JDBC SQL タイプ LONGVARBINARY 型データの長さを設定します。

例えば，引数 access_size に 20 を指定した場合，データベースに格納している 100 キロバイトの JDBC SQL タイプ LONGVARBINARY 型データを ResultSet の getBytes メソッドで取得しようとすると，データを 20 キロバイトずつ 5 回に分けて取得します。

setLONGVARBINARY_Access メソッドの引数 mode に LOCATOR 以外を指定している場合は，この指定値は有効となりません。

このメソッドでの設定は，「ユーザプロパティ」で示したプロパティ HiRDB_for_Java_LONGVARBINARY_ACCESS_SIZE の設定と同等になります。

(6) 発生する例外

引数 access_size に 0～2,097,151 以外の値を指定した場合は，SQLException を投入します。

(7) 注意事項

「LONGVARBINARY_ACCESS」の注意事項を参照してください。

17.11.44 getLONGVARBINARY_AccessSize

(1) 機能

HiRDB サーバに対して一度に要求する JDBC SQL タイプ LONGVARBINARY 型データの長さを取得します。

(2) 形式

```
public int getLONGVARBINARY_AccessSize() throws SQLException
```

(3) 引数

なし。

(4) 戻り値

int：

一度に要求するデータ長（単位：キロバイト）（設定していない場合は 0 を返却します）

(5) 機能詳細

HiRDB サーバに対して一度に要求する JDBC SQL タイプ LONGVARBINARY 型データの長さを取得します。このメソッドは，setLONGVARBINARY_AccessSize メソッドの設定値を返却します。

(6) 発生する例外

なし。

17.11.45　setLONGVARBINARY_TruncError

(1)　機能

JDBC SQL タイプ LONGVARBINARY 型データ取得時に切り捨てが発生した場合に，例外を投入するかどうかを設定します。

(2)　形式

```
public void setLONGVARBINARY_TruncError ( boolean mode ) throws SQLException
```

(3)　引数

boolean mode：

切り捨てが発生した場合に例外を投入するかどうかを指定します。

true：

例外を投入します。

false：

例外を投入しません。

(4)　戻り値

なし。

(5)　機能詳細

JDBC SQL タイプ LONGVARBINARY 型データ取得時に切り捨てが発生した場合に例外を投入するかどうかを設定します。設定がない場合，true が指定されたものとして動作します。

setSQLWarningLevel メソッドの引数 warningLevel に IGNORE を指定している場合は，このメソッドの指定値は無効です。false が指定されたものとして動作します。

なお，JDBC SQL タイプ LONGVARBINARY 型データ取得時に発生する切り捨てとは，次の条件式を満たした場合のことを指します。

```
SQLの実行で得られるJDBC SQLタイプLONGVARBINARY型データの実際の長さ ＞ setMaxBinarySizeで指定したデータ長
```

(6)　発生する例外

なし。

17.11.46　getLONGVARBINARY_TruncError

(1)　機能

JDBC SQL タイプ LONGVARBINARY 型データ取得時に切り捨てが発生した場合に，例外を投入するかどうかの設定情報を取得します。

(2)　形式

```
public boolean getLONGVARBINARY_TruncError()
```

(3)　引数

なし。

(4)　戻り値

boolean：

切り捨てが発生した場合に例外を投入するかどうかの設定情報です。

true：

例外を投入します。

false：

例外を投入しません。

(5)　機能詳細

JDBC SQL タイプ LONGVARBINARY 型データ取得時に切り捨てが発生した場合に例外を投入するかどうかの設定情報を取得します。

(6)　発生する例外

なし。

17.11.47　setStatementCloseBehavior

(1)　機能

ステートメント（Statement クラス，PreparedStatement クラス，及び CallableStatement クラス）の close メソッド実行時に前処理結果を無効にするかどうかを設定します。

(2) 形式

```
public synchronized void setStatementCloseBehavior(boolean mode)
```

(3) 引数

boolean mode：

ステートメントの close メソッド実行時に前処理結果を無効にするかどうかを設定します。

true：

前処理結果を無効にします。

false：

前処理結果を無効にしません。

(4) 戻り値

なし。

(5) 機能詳細

ステートメント（Statement クラス，PreparedStatement クラス，及び CallableStatement クラス）の close メソッド実行時に前処理結果を無効にするかどうかを設定します。

設定がない場合，false が指定されたものとして動作します。

このメソッドでの設定は，「ユーザプロパティ」で示したプロパティ HiRDB_for_Java_STATEMENT_CLOSE_BEHAVIOR の設定と同等になります。

(6) 発生する例外

なし。

17.11.48 getStatementCloseBehavior

(1) 機能

ステートメント（Statement クラス，PreparedStatement クラス，及び CallableStatement クラス）の close メソッド実行時に前処理結果を無効にするかどうかの設定情報を取得します。

(2) 形式

```
public boolean getStatementCloseBehavior()
```

(3) 引数

なし。

(4) 戻り値

true：

前処理結果を無効にします。

false：

前処理結果を無効にしません。

(5) 機能詳細

ステートメント（Statement クラス，PreparedStatement クラス，及び CallableStatement クラス）の close メソッド実行時に前処理結果を無効にするかどうかの設定情報を取得します。このメソッドは setStatementCloseBehavior メソッドの設定値を返却します。設定していない場合は false を返却します。

(6) 発生する例外

なし。

17.11.49 setHiRDBINI

(1) 機能

HiRDB.INI ファイル内に記述されているクライアント環境変数を有効にする場合に，HiRDB.INI ファイルのディレクトリを指定します。

(2) 形式

```
public synchronized void setHiRDBINI (String dir)
```

(3) 引数

string dir：

HiRDB.INI ファイルが存在するディレクトリの絶対パスを指定します。null 指定の場合は，未指定の状態とします。

(4) 戻り値

なし。

(5)　機能詳細

HiRDB.INI ファイル内に記述されているクライアント環境変数を有効にする場合に，HiRDB.INI ファイルのディレクトリの絶対パスを指定します。このメソッドの詳細については，「URL の各項目の説明」の HiRDB_INI の説明を参照してください。

(6)　発生する例外

なし。

17.11.50　getHiRDBINI

(1)　機能

setHiRDBINI メソッドで設定したディレクトリを取得します。

(2)　形式

```
public String getHiRDBINI()
```

(3)　引数

なし。

(4)　戻り値

String：

setHiRDBINI メソッドで設定したディレクトリの絶対パスです。未設定の場合は null を返します。

(5)　機能詳細

setHiRDBINI メソッドで設定したディレクトリの絶対パスを取得します。

(6)　発生する例外

なし。

17.11.51　setBatchExceptionBehavior

(1)　機能

java.sql.BatchUpdateException の getUpdateCounts メソッドの戻り値に，JDBC 規格に準拠した更新カウントを設定するかどうかを指定します。

(2)　形式

```
public synchronized void setBatchExceptionBehavior(boolean mode)
```

(3)　引数

boolean mode：

java.sql.BatchUpdateException の getUpdateCounts メソッドの戻り値に，JDBC 規格に準拠した更新カウントを設定するかどうかを指定します。

true：JDBC 規格に準拠した更新カウントを設定します。

false：HiRDB 独自の更新カウントを設定します。

(4)　戻り値

なし。

(5)　機能詳細

java.sql.BatchUpdateException の getUpdateCounts メソッドの戻り値に，JDBC 規格に準拠した更新カウントを設定するかどうかを指定します。このメソッドが呼び出されない場合のデフォルト値は"TRUE"です。

このメソッドでの設定は，「ユーザプロパティ」で示したプロパティ HiRDB_for_Java_BATCHEXCEPTION_BEHAVIOR の設定と同等になります。

(6)　発生する例外

なし。

17.11.52　getBatchExceptionBehavior

(1)　機能

setBatchExceptionBehavior で設定した java.sql.BatchUpdateException の getUpdateCounts メソッドの戻り値に，JDBC 規格に準拠した更新カウントを設定するかどうかの設定情報を取得します。

(2) 形式

```
public synchronized boolean getBatchExceptionBehavior()
```

(3) 引数

なし。

(4) 戻り値

boolean：

setBatchExceptionBehavior で設定した java.sql.BatchUpdateException の getUpdateCounts メソッドの戻り値に，JDBC 規格に準拠した更新カウントを設定するかどうかを返却します。未設定の場合は，デフォルト値"TRUE"を返却します。

true：JDBC 規格に準拠した更新カウントを設定します。

false：HiRDB 独自の更新カウントを設定します。

(5) 機能詳細

setBatchExceptionBehavior で設定した java.sql.BatchUpdateException の getUpdateCounts メソッドの戻り値に，JDBC 規格に準拠した更新カウントを設定するかどうかの設定情報を取得します。

(6) 発生する例外

なし。

17.11.53 setUpdateCountBehavior

(1) 機能

更新系 SQL 発行時の更新行数が 0 の場合，java.sql.Statement の getUpdateCount メソッドの戻り値として-1 を返却するかどうかを指定します。

(2) 形式

```
public synchronized void setUpdateCountBehavior(boolean mode)
```

(3) 引数

boolean mode：

更新系 SQL 発行時の更新行数が 0 の場合，java.sql.Statement の getUpdateCount メソッドの戻り値として-1 を返却するかどうかを指定します。

true：更新系 SQL 発行時の更新行数が 0 の場合，-1 を返却します（JDBC 規格）。

false：更新系 SQL 発行時の更新行数が 0 の場合，0 を返却します。

(4) 戻り値

なし。

(5) 機能詳細

更新系 SQL 発行時の更新行数が 0 の場合，java.sql.Statement の getUpdateCount メソッドの戻り値として-1 を返却するかどうかを指定します。

このメソッドが呼び出されない場合のデフォルト値は"false"です。

このメソッドでの設定は，「ユーザプロパティ」で示したプロパティ HiRDB_for_Java_UPDATECOUNT_BEHAVIOR の設定と同等になります。

(6) 発生する例外

なし。

17.11.54 getUpdateCountBehavior

(1) 機能

更新系 SQL 発行時の更新行数が 0 の場合，java.sql.Statement の getUpdateCount メソッドの戻り値として-1 を返却するかどうかの設定情報を取得します。

(2) 形式

```
public synchronized boolean getUpdateCountBehavior()
```

(3) 引数

なし。

(4) 戻り値

boolean：

setUpdateCountBehavior で設定した，更新系 SQL 発行時の更新行数が 0 の場合，java.sql.Statement の getUpdateCount メソッドの戻り値として-1 を返却するかどうかの設定情報を返却します。未設定の場合は，デフォルト値"FALSE"を返却します。

true：更新系 SQL 発行時の更新行数が 0 の場合，-1 を返却します（JDBC 規格）。

false：更新系 SQL 発行時の更新行数が 0 の場合，0 を返却します。

(5) 機能詳細

更新系 SQL 発行時の更新行数が 0 の場合，java.sql.Statement の getUpdateCount メソッドの戻り値として-1 を返却するかどうかの設定情報を取得します。

(6) 発生する例外

なし。

17.12　データ型

17.12.1　SQL データ型のマッピング

HiRDB の SQL データ型と JDBC の SQL データ型は，完全には一致しません。そのため，JDBC ドライバでは，JDBC の SQL データ型と，接続する HiRDB の SQL データ型とのマッピング（変換）を実行します。マッピングできない SQL データ型を使用してアクセスしようとすると，SQLException を投入します。SQL データ型のマッピングは，ResultSet，PreparedStatement，CallableStatement クラスの getXXX メソッド及び setXXX メソッドで実行します。なお，SQL データ型と，getXXX メソッド及び setXXX メソッドとのマッピング規則については，JDBC1.0 規格又は JDBC2.0 基本規格のドキュメントを参照してください。

HiRDB と JDBC の SQL データ型の対応を次の表に示します。

表 17–105　HiRDB と JDBC の SQL データ型の対応（Type4 JDBC ドライバ）

HiRDB の SQL データ型	JDBC の SQL データ型
INTEGER	INTEGER
SMALLINT	SMALLINT
DECIMAL，NUMERIC	DECIMAL（，NUMERIC）※1
FLOAT，DOUBLE PRECISION	FLOAT（，DOUBLE）※1
SMALLFLT，REAL	REAL
CHAR	CHAR
VARCHAR	VARCHAR（，LONGVARCHAR）※1
NCHAR	CHAR
NVARCHAR	VARCHAR（，LONGVARCHAR）※1
MCHAR	CHAR
MVARCHAR	VARCHAR（，LONGVARCHAR）※1
DATE	DATE
TIME	TIME
BLOB	LONGVARBINARY（，BINARY，VARBINARY，BLOB）※1
BINARY	LONGVARBINARY（，BINARY，VARBINARY，BLOB）※1
TIMESTAMP	TIMESTAMP
BOOLEAN※2	BIT
ANY※3	OTHER
INTERVAL YEAR TO DAY	Type4 JDBC ドライバでは未サポート

HiRDB の SQL データ型	JDBC の SQL データ型
INTERVAL HOUR TO SECOND	
ROW	

注※1

（）内のデータ型は，setNull メソッド，setObject メソッド，又は registerOutParameter メソッドの引数に JDBC の SQL データ型を指定する場合にだけ対応します。HiRDB の SQL データ型から JDBC の SQL データ型へマッピングする際には対応しません。

注※2

DatabaseMetaData の getTypeInfo メソッドなどで生成される，ResultSet オブジェクトが持つ BOOLEAN 型の列を指します。

注※3

?パラメタに NULL 述語を指定した SQL（? IS 〔NOT〕 NULL）での，パラメタのデータ型としてだけ使用されます。

17.12.2　検索データ取得時のマッピング

ResultSet クラス，及び CallableStatement クラスの getXXX メソッドと，JDBC の SQL データ型とのマッピングを次の表に示します。マッピングできない JDBC の SQL データ型に対して getXXX メソッドが呼び出された場合は，SQLException を投入します。

表 17–106　getXXX メソッドと JDBC の SQL データ型とのマッピング（1/2）

getXXX メソッド	JDBC の SQL データ型					
	SMALLINT	INTEGER	FLOAT	REAL	DECIMAL	CHAR
getByte	○	○	○	○	○	○※1
getShort	◎	○	○	○	○	○※1
getInt	○	◎	○	○	○	○※1
getLong	○	○	○	○	○	○※1
getFloat	○	○	○	◎	○	○※1
getDouble	○	○	◎	○	○	○※1
getBigDecimal	○	○	○	○	◎	○※1
getBoolean	○	○	○	○	○	○
getString	○	○	○	○	○	◎
getBytes	×	×	×	×	×	○※3

getXXX メソッド	JDBC の SQL データ型					
	SMALLINT	INTEGER	FLOAT	REAL	DECIMAL	CHAR
getDate	×	×	×	×	×	○※1
getTime	×	×	×	×	×	○※1
getTimestamp	×	×	×	×	×	○※1
getAsciiStream	×	×	×	×	×	○
getBinaryStream	×	×	×	×	×	×
getObject	○	○	○	○	○	○
getCharacterStream	×	×	×	×	×	○
getArray	×	×	×	×	×	×
getBlob	×	×	×	×	×	×

表 17–107　getXXX メソッドと JDBC の SQL データ型とのマッピング（2/2）

getXXX メソッド	JDBC の SQL データ型					
	VARCHAR	DATE	TIME	TIMESTAMP	LONGVARBINARY	ARRAY
getByte	○※1	×	×	×	×	×
getShort	○※1	×	×	×	×	×
getInt	○※1	×	×	×	×	×
getLong	○※1	×	×	×	×	×
getFloat	○※1	×	×	×	×	×
getDouble	○※1	×	×	×	×	×
getBigDecimal	○※1	×	×	×	×	×
getBoolean	○	×	×	×	×	×
getString	◎	○	○	○	○	×
getBytes	○※3	×	×	×	○	×
getDate	○※1	◎※2	×	○	×	×
getTime	○※1	×	◎	○	×	×
getTimestamp	○※1	○	○	◎	×	×
getAsciiStream	○	×	×	×	○	×
getBinaryStream	×	×	×	×	◎	×
getObject	○	○	○	○	○	×

getXXX メソッド	JDBC の SQL データ型					
	VARCHAR	DATE	TIME	TIMESTAMP	LONGVARBINARY	ARRAY
getCharacterStream	○	×	×	×	○	×
getArray	×	×	×	×	×	◎
getBlob	×	×	×	×	○	×

（凡例）

◎：マッピングすることを推奨します。

○：マッピングできます。ただし，変換元データの形式によっては，データの欠落や変換エラーとなることがあるため，注意してください。

×：マッピングできません。

注※1

このメソッドでの変換の際に，データベースから取得した文字列データの前後に半角スペースがある場合は，半角スペースを取り除きます。また，半角スペースを取り除いた後，getXXX メソッドが返却する Java のデータ型に変換します。

Java のデータ型に変換する場合の注意事項を次に示します。

- 文字列データに小数点以下の表現がある場合，getByte メソッド，getInt メソッド，getShort メソッド，又は getLong メソッドを実行すると，小数点以下を切り捨てて整数だけを変換し，返却します。
- 文字列データに全角文字が含まれている場合，変換しないで SQLException を投入します。全角文字には，HiRDB のデータ型である NCHAR 型の列に列の定義長よりも短い文字列を格納している場合に補完する全角スペースも含みます。
- 文字列データを Java のデータ型に変換した結果，オーバフローが発生する場合は，SQLException を投入します。

注※2

JDBC SQL タイプが DATE 型の場合，setDate メソッドに java.util.Calendar オブジェクトを指定して実行すると，指定した java.util.Calendar オブジェクトを使用してデータを変換し，時刻データを切り捨てて日付データだけをデータベースに格納します。このとき，時刻データを切り捨てるため，getDate メソッドに java.util.Calendar オブジェクトを指定して，setDate メソッドで格納したデータを取得しても，setDate メソッドに指定した日付と異なる日付を取得する場合があります。

（例）

日本標準時をデフォルトのタイムゾーンとする UAP で，setDate メソッド，及び getDate メソッドに世界標準時のタイムゾーンを持つ java.util.Calendar オブジェクトを指定した場合の例を次に示します。

setDate メソッドに「2005-10-03」を表す java.sql.Date オブジェクトを指定して実行した場合，JDBC ドライバは時刻部分に「00:00:00」を補完した後，タイムゾーンの違いによって 9 時間遅らせて「2005-10-02 15:00:00」とし，日付部分「2005-10-02」をデータベースに格納します。このデータを getDate メソッドで取得した場合，データベースから日付部分「2005-10-02」を取得し，時刻部分に「00:00:00」を補完した後，タイムゾーンの違いによって 9 時間進めて「2005-10-02

09:00:00」とします。これによって，getDate メソッドの戻り値の java.sql.Date オブジェクトには，「2005-10-02」が設定されるため，setDate メソッドに指定した「2005-10-03」とは異なります。

注※3

getBytes メソッドで取得する文字列型データの文字コードは，サーバで格納しているデータを文字コード変換しないで，そのままの形式で取得します。

17.12.3　？パラメタ設定時のマッピング

PreparedStatement クラス，及び CallableStatement クラスの setXXX メソッドと，マッピングされる JDBC SQL タイプを次の表に示します。使用できない JDBC SQL タイプの場合，setXXX メソッドは SQLException を投入します。

なお，setUnicodeStream メソッドが JDBC2.0 基本規格で推奨されないメソッドとなったため，代わりに setCharacterStream が追加されました。

表 17–108　setXXX メソッドと，マッピングされる JDBC SQL タイプ

PreparedStatement クラスの setXXX メソッド	マッピングされる JDBC SQL タイプ
setCharacterStream	CHAR，VARCHAR
setRef※	REF
setBlob	LONGVARBINARY
setClob※	CLOB
setArray	ARRAY

注※

JDBC ドライバでは使用できません。

PreparedStatement クラス，及び CallableStatement クラスの setXXX メソッドと各 JDBC SQL タイプとのマッピングを次の表に示します。

表 17–109　setXXX メソッドと JDBC の SQL データ型とのマッピング（1/2）

setXXX メソッド	JDBC の SQL データ型					
	SMALLINT	INTEGER	FLOAT	REAL	DECIMAL※3	CHAR
setByte	○	○	○	○	○	○
setShort	◎	○	○	○	○	○
setInt	○	◎	○	○	○	○
setLong	○	○	○	○	○	○
setFloat	○	○	○	◎	○	○

setXXX メソッド	JDBC の SQL データ型					
	SMALLINT	INTEGER	FLOAT	REAL	DECIMAL※3	CHAR
setDouble	○	○	◎	○	○	○
setBigDecimal	○	○	○	○	◎	○
setBoolean	○	○	○	○	○	○
setString	○	○	○	○	○	◎
setBytes	×	×	×	×	×	○※7
setDate	×	×	×	×	×	○
setTime	×	×	×	×	×	○
setTimestamp※1	×	×	×	×	×	○
setAsciiStream	×	×	×	×	×	○
setBinaryStream	×	×	×	×	×	×
setObject※2	○	○	○	○	○	○
setCharacterStream	×	×	×	×	×	○※4
setArray	×	×	×	×	×	×
setBlob	×	×	×	×	×	×

表 17–110　setXXX メソッドと JDBC の SQL データ型とのマッピング（2/2）

setXXX メソッド	JDBC の SQL データ型						
	VARCHAR	DATE	TIME	TIMESTAMP	LONGVARBINARY	ARRAY	OTHER※6
setByte	○	×	×	×	×	×	○
setShort	○	×	×	×	×	×	○
setInt	○	×	×	×	×	×	○
setLong	○	×	×	×	×	×	○
setFloat	○	×	×	×	×	×	○
setDouble	○	×	×	×	×	×	○
setBigDecimal	○	×	×	×	×	×	○
setBoolean	○	×	×	×	×	×	○
setString	◎	○	○	○	○	×	○
setBytes	○※7	×	×	×	○	×	○
setDate	○	◎※5	×	○	×	×	○

setXXX メソッド	JDBC の SQL データ型						
	VARCHAR	DATE	TIME	TIMESTAMP	LONGVARBINARY	ARRAY	OTHER※6
setTime	○	×	◎	○	×	×	○
setTimestamp※1	○	○	○	◎	×	×	○
setAsciiStream	○	×	×	×	○	×	○
setBinaryStream	×	×	×	×	○	×	○
setObject※2	○	○	○	○	○	×	○
setCharacterStream	○※4	×	×	×	○※4	×	○
setArray	×	×	×	×	×	○	×
setBlob	×	×	×	×	○	×	○

（凡例）

◎：マッピングすることを推奨します。

○：マッピングできます。ただし，変換元データの形式によっては，データの欠落や変換エラーとなることがあるため，注意してください。

×：マッピングできません。

注※1

HiRDB データ型である TIMESTAMP 型の？パラメタに対して setXXX メソッドで値を指定する場合に，？パラメタの小数秒精度と値の小数秒精度が一致していない場合の動作を次に示します。

- ？パラメタの小数秒精度よりも大きい場合：切り捨てます。
- ？パラメタの小数秒精度よりも小さい場合：拡張します。

注※2

setObject メソッドに InputStream クラス及び Reader クラス（サブクラスを含む）のオブジェクトは指定できません。

注※3

HiRDB データ型である DECIMAL 型の？パラメタに対して setXXX メソッドで値を指定する場合に，けたあふれが発生したときの動作を次に示します。

- 指定値の整数部のけた数が？パラメタの整数部のけた数よりも大きい場合：
  SQLException を投入します。
- 指定値の小数点以下のけた数が？パラメタの小数点以下のけた数よりも大きい場合：
  ？パラメタの小数点以下のけた数で切り捨てます。

（例）

？パラメタの型：DECIMAL(10,5)

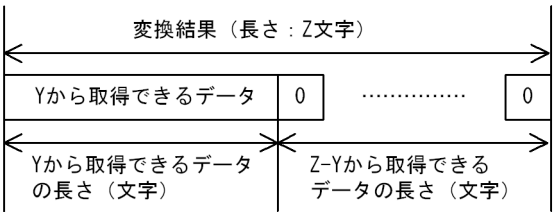
指定値が 123456.1234 の場合，SQLException を投入します。

指定値が 1234.123456 の場合，指定値を 1234.12345 として処理します。

注※4

java.io.Reader オブジェクトから取得できるデータの長さが，引数で指定した長さより短い場合，次に示すように引数で指定した長さまで 0 を補完します。

```
int X,Z;
java.io.Reader Y;
setCharacterStream(X,Y,Z)
```



注※5

JDBC SQL タイプが DATE 型の場合，setDate メソッドに java.util.Calendar オブジェクトを指定して実行すると，指定した java.util.Calendar オブジェクトを使用してデータを変換し，時刻データを切り捨てて日付データだけをデータベースに格納します。このとき，時刻データを切り捨てるため，getDate メソッドに java.util.Calendar オブジェクトを指定して，setDate メソッドで格納したデータを取得しても，setDate メソッドに指定した日付と異なる日付を取得する場合があります。

（例）

日本標準時をデフォルトのタイムゾーンとする UAP で，setDate メソッド，及び getDate メソッドに世界標準時のタイムゾーンを持つ java.util.Calendar オブジェクトを指定した場合の例を次に示します。

setDate メソッドに「2005-10-03」を表す java.sql.Date オブジェクトを指定して実行した場合，JDBC ドライバは時刻部分に「00:00:00」を補完した後，タイムゾーンの違いによって 9 時間遅らせて「2005-10-02 15:00:00」とし，日付部分「2005-10-02」をデータベースに格納します。このデータを getDate メソッドで取得した場合，データベースから日付部分「2005-10-02」を取得し，時刻部分に「00:00:00」を補完した後，タイムゾーンの違いによって 9 時間進めて「2005-10-02 09:00:00」とします。これによって，getDate メソッドの戻り値の java.sql.Date オブジェクトには，「2005-10-02」が設定されるため，setDate メソッドに指定した「2005-10-03」とは異なります。

注※6

?パラメタに NULL 述語を指定した SQL（? IS 〔NOT〕 NULL）での，パラメタのデータ型としてだけ使用されます。

注※7

setBytes メソッドで設定する文字列型データは，接続先サーバ環境に合わせる必要があるため，接続先サーバの文字コードで設定します。

17.12.4　TIME 型，DATE 型，及び TIMESTAMP 型列のデータ変換処理

(1)　setTime，setDate，setTimestamp，及び setString メソッド

setTime，setDate，setTimestamp，及び setString メソッドに，HiRDB のデータ型である TIME 型，DATE 型，又は TIMESTAMP 型のデータが設定された場合の変換処理について説明します。

HiRDB のデータ型である TIME 型，DATE 型，又は TIMESTAMP 型の列に対して，setTime，setDate，setTimestamp，及び setString メソッドを使用してデータが設定された場合，HiRDB のデータ型に応じてデータ変換が行われます。

列のデータ型とメソッドとの組み合わせによる変換処理について次の表に示します。

表 17–111　TIME 型，DATE 型，又は TIMESTAMP 型と setXXX メソッドとの組み合わせによる変換処理

setXXX メソッド	HiRDB のデータ型		
	TIME 型	DATE 型	TIMESTAMP 型
setTime(Time Obj)※1	UAP の設定値どおりにデータベースに格納します。	SQLException を投入します。	UAP の設定値 hh:mm:ss[.000000]の前に「1970-01-01」を付与したデータをデータベースに格納します。
setDate(Date Obj)※2	SQLException を投入します。	UAP の設定値どおりにデータベースに格納します。	UAP の設定値 yyyy-MM-DD の後ろに「00:00:00[.000000]」を付与したデータをデータベースに格納します。
setTimestamp(Timestamp Obj)※3	UAP の設定値の「hh:mm:ss」部分をデータベースに格納します。	UAP の設定値の「yyyy-MM-DD」部分をデータベースに格納します。	UAP の設定値どおりにデータベースに格納します。
setString(hh:mm:ss 形式の文字列)	指定された時刻を java.sql.Time.valueOf()で変換してデータベースに格納します。※5	SQLException を投入します。	SQLException を投入します。
setString(yyyy-MM-DD 形式の文字列)	SQLException を投入します。	指定された日付を java.sql.Date.valueOf()で変換してデータベースに格納します。※5	SQLException を投入します。
setString(yyyy-MM-DD△hh:mm:ss[.ffffff]形式の文字列)※4	SQLException を投入します。	SQLException を投入します。	指定された日時を java.sql.Timestamp.valueOf()で変換してデータベースに格納します。※5

注

実際に存在しない日時を指定した場合は，Java 仮想マシンが返す値となります。

注※1

「Time Obj」は，java.sql.Time オブジェクト「時：分：秒」の値を持つオブジェクトです。

注※2

「Date Obj」は，java.sql.Date オブジェクト「年−月−日」の値を持つオブジェクトです。

注※3

「Timestamp Obj」は，java.sql.Timestamp オブジェクト「年−月−日　時：分：秒．ナノ秒」の値を持つオブジェクトです。

注※4

[.ffffff]は，HiRDB の TIMESTAMP 型の精度に合わせて，小数点以下のけた数が変わります。

また，△は半角スペースを示します。

注※5

存在しない日時を指定した場合の結果は java.sql.Time.valueOf()，java.sql.Date.valueOf()，及び java.sql.Timestamp.valueOf()に依存します。

例 1：25:00:00 は，01:00:00 となります。

例 2：2000-01-32 は，2000-02-01 となります。

例 3：1582-10-05 は，1582-10-15 となります（ユリウス暦とグレゴリオ暦が切り替わります）。

(2)　getTime，getDate，及び getTimestamp メソッド

getTime，getDate，及び getTimestamp メソッドに，HiRDB のデータ型である TIME 型，DATE 型，TIMESTAMP 型，又は文字列型（CHAR 型，MCHAR 型，NCHAR 型，VARCHAR 型，MVARCHAR 型，及び NVARCHAR 型）のデータが設定された場合の変換処理について説明します。

HiRDB のデータ型である TIME 型，DATE 型，TIMESTAMP 型，又は文字列型の列に対して，getTime，getDate，及び getTimestamp メソッドを使用してデータが設定された場合，HiRDB のデータ型に応じてデータ変換が行われます。

列のデータ型とメソッドとの組み合わせによる変換処理について次の表に示します。

表 17–112　TIME 型，DATE 型，TIMESTAMP 型，又は文字列型と getXXX メソッドとの組み合わせによる変換処理

getXXX メソッド	HiRDB のデータ型			
	TIME 型	DATE 型	TIMESTAMP 型	文字列型
getTime()※2	データベースに格納されている値どおりに java.sql.Time のオブジェクトとして返却します。※1	SQLException を投入します。	データベースから取得した TIMESTAMP 型のデータの「時：分：秒」部分を java.sql.Time のオブジェクトとして返却します。※1	TIME 型の文字列表現「hh:mm:ss」だけを java.sql.Time のオブジェクトとして返却します。それ以外は SQLException を投入します。

getXXX メソッド	HiRDB のデータ型			
	TIME 型	DATE 型	TIMESTAMP 型	文字列型
getDate()※2	SQLException を投入します。	データベースに格納されている値どおりに java.sql.Date のオブジェクトとして返却します。※1	データベースから取得した TIMESTAMP 型のデータの「年−月−日」部分を java.sql.Date のオブジェクトとして返却します。※1	DATE 型の文字列表現「yyyy-MM-DD」だけを java.sql.Date のオブジェクトとして返却します。それ以外は SQLException を投入します。
getTimestamp()※2	データベースから取得した DATE 型のデータに「1970-01-01」を付加し，小数秒を 0 としたデータを java.sql.Timestamp のオブジェクトとして返却します。	データベースから取得した DATE 型のデータに「00:00:00.000000」を付加したデータを java.sql.Timestamp のオブジェクトとして返却します。	データベースに格納されている値どおりに java.sql.Timestamp のオブジェクトとして返却します。	TIMESTAMP 型の文字列表現「yyyy-MM-DD△hh:mm:ss[.ffffff]」（△は半角スペース）だけを java.sql.Timestamp のオブジェクトとして返却します。それ以外は SQLException を投入します。

（凡例）

文字列型：CHAR 型，MCHAR 型，NCHAR 型，VARCHAR 型，MVARCHAR 型，及び NVARCHAR 型

注※1

値の設定について記載していない日付項目（年−月−日）の設定値は「1970-01-01」，時間項目（時：分：秒．ミリ秒）の設定値は「00：00：00.000000」とします。

注※2

データベースに格納されている日時と java.sql.Time，java.sql.Date，及び java.sql.Timestamp から得られる日時は異なることがあります。

例 1：25:00:00 は，01:00:00 となります。

例 2：2000-01-32 は，2000-02-01 となります。

例 3：1582-10-05 と 1582-10-15 は，どちらも 1582-10-15 となります（ユリウス暦とグレゴリオ暦が切り替わります）。

17.12.5　オーバフローの扱い

ここでは，setXXX メソッドを使用して値を設定する場合，及び getXXX メソッドを使用して値を取得する場合にオーバフローするかどうかについて説明します。

(1)　setXXX メソッド（setObject メソッドを除く）

次の表に，setXXX メソッド使用時にオーバフローするかどうかを，HiRDB のデータ型ごとに示します。

表 17–113　setXXX メソッド使用時のオーバフロー有無（1/2）

setXXX メソッド	HiRDB のデータ型					
	SMALLINT	INTEGER	FLOAT	REAL	DECIMAL	文字列型
setByte	○	○	○	○	×	○
setShort	○	○	○	○	×	○
setInt	×	○	○	○	×	○
setLong	×	×	○	○	×	○
setFloat	×	×	○	○	×	○
setDouble	×	×	○	○	×	○
setBigDecimal	×	×	○	○	×	○
setBoolean	○	○	○	○	×	○
setString	×	×	○	○	×	○
setBytes	–	–	–	–	–	–
setDate	–	–	–	–	–	○
setTime	–	–	–	–	–	○
setTimestamp	–	–	–	–	–	○
setBlob	–	–	–	–	–	–
setBinaryStream	–	–	–	–	–	–
setAsciiStream	–	–	–	–	–	○
setArray	×	×	×	×	×	○
setCharacterStream	–	–	–	–	–	○

（凡例）

○：値に関係なく，オーバフローしません。

×：値によっては，オーバフローすることがあります。

–：この組み合わせでは使用できません。

文字列型：CHAR 型，MCHAR 型，NCHAR 型，VARCHAR 型，MVARCHAR 型，及び NVARCHAR 型

表 17–114　setXXX メソッド使用時のオーバフロー有無（2/2）

setXXX メソッド	HiRDB のデータ型				
	DATE※	TIME※	TIMESTAMP※	BINARY	BLOB
setByte	–	–	–	–	–
setShort	–	–	–	–	–

setXXX メソッド	HiRDB のデータ型				
	DATE※	TIME※	TIMESTAMP※	BINARY	BLOB
setInt	−	−	−	−	−
setLong	−	−	−	−	−
setFloat	−	−	−	−	−
setDouble	−	−	−	−	−
setBigDecimal	−	−	−	−	−
setBoolean	−	−	−	−	−
setString	×	○	×	−	−
setBytes	−	−	−	○	○
setDate	×	−	×	−	−
setTime	−	×	×	−	−
setTimestamp	×	−	×	−	−
setBlob	−	−	−	○	○
setBinaryStream	−	−	−	○	○
setAsciiStream	−	−	−	○	○
setArray	×	×	×	−	−
setCharacterStream	−	−	−	○	○

（凡例）

○：値に関係なく，オーバフローしません。

×：値によっては，オーバフローすることがあります。

−：この組み合わせでは使用できません。

注※

java.sql.Date，java.sql.Time，又は java.sql.Timestamp クラスの getTime メソッドで取得した値が 253,402,268,399,999 より大きいか，又は-62,135,802,000,000 より小さいオブジェクトである場合，オーバフローします。なお，getTime メソッドは，1970 年 1 月 1 日 0 時 0 分 0 秒（グリニッジ標準時）からのミリ秒数を返します。

253,402,268,399,999 は HiRDB の TIMESTAMP 型に格納できる最大値から，-62,135,802,000,000 は java.sql.Timestamp クラスで表現できる最小値から，次に示すメソッドで取得できます。

253,402,268,399,999：

Timestamp.valueOf("9999-12-31 23:59:59.999999").getTime()

-62,135,802,000,000：

Timestamp.valueOf("0001-01-01 00:00:00.0").getTime()

(2) setObject メソッド

次の表に，setObject メソッド使用時にオーバフローするかどうかを，HiRDB のデータ型ごとに示します。

表 17–115 setObject メソッド使用時のオーバフロー有無（1/2）

setObject メソッド	HiRDB のデータ型					
	SMALLINT	INTEGER	FLOAT	REAL	DECIMAL	文字列型
Byte	○	○	○	○	×	○
Short	○	○	○	○	×	○
Integer	×	○	○	○	×	○
Long	×	×	○	○	×	○
Decimal	×	×	○	○	×	○
Float	×	×	○	○	×	○
Double	×	×	○	×	×	○
Boolean	○	○	○	○	×	○
String	×	×	○	○	×	○
Date	–	–	–	–	–	○
Time	–	–	–	–	–	○
Timestamp	–	–	–	–	–	○
byte[]	–	–	–	–	–	○
Blob	–	–	–	–	–	–
Array	–	–	–	–	–	–

（凡例）

○：値に関係なく，オーバフローしません。

×：値によっては，オーバフローすることがあります。

–：この組み合わせでは使用できません。

文字列型：CHAR 型，MCHAR 型，NCHAR 型，VARCHAR 型，MVARCHAR 型，及び NVARCHAR 型

表 17–116 setObject メソッド使用時のオーバフロー有無（2/2）

setObject メソッド	HiRDB のデータ型				
	DATE※	TIME※	TIMESTAMP※	BINARY	BLOB
Byte	–	–	–	–	–
Short	–	–	–	–	–
Integer	–	–	–	–	–

setObject メソッド	HiRDB のデータ型				
	DATE※	TIME※	TIMESTAMP※	BINARY	BLOB
Long	−	−	−	−	−
Decimal	−	−	−	−	−
Float	−	−	−	−	−
Double	−	−	−	−	−
Boolean	−	−	−	−	−
String	×	○	×	−	−
Date	×	−	×	−	−
Time	−	×	−	−	−
Timestamp	×	−	×	−	−
byte[]	−	−	−	○	○
Blob	−	−	−	○	○
Array	−	−	−	−	−

（凡例）

○：値に関係なく，オーバフローしません。

×：値によっては，オーバフローすることがあります。

−：この組み合わせでは使用できません。

注※

java.sql.Date，java.sql.Time，又は java.sql.Timestamp クラスの getTime メソッドで取得した値が 253,402,268,399,999 より大きいか，又は-62,135,802,000,000 より小さいオブジェクトである場合，オーバフローします。なお，getTime メソッドは，1970 年 1 月 1 日 0 時 0 分 0 秒（グリニッジ標準時）からのミリ秒数を返します。

253,402,268,399,999 は HiRDB の TIMESTAMP 型に格納できる最大値から，-62,135,802,000,000 は java.sql.Timestamp クラスで表現できる最小値から，次に示すメソッドで取得できます。

253,402,268,399,999：

Timestamp.valueOf("9999-12-31 23:59:59.999999").getTime()

-62,135,802,000,000：

Timestamp.valueOf("0001-01-01 00:00:00.0").getTime()

(3) getXXX メソッド（getObject メソッドを除く）

次の表に，getXXX メソッド使用時にオーバフローするかどうかを，HiRDB のデータ型ごとに示します。

表 17–117　getXXX メソッド使用時のオーバフロー有無（1/2）

getXXX メソッド	HiRDB のデータ型					
	SMALLINT	INTEGER	FLOAT	REAL	DECIMAL	文字列型
getByte	×	×	×	×	×	×
getShort	○	×	×	×	×	×
getInt	○	○	×	×	×	×
getLong	○	○	×	×	×	×
getFloat	○	○	○	○	○	○
getDouble	○	○	○	○	○	○
getBigDecimal	○	○	○	○	○	○
getBoolean	○	○	○	○	○	○
getString	○	○	○	○	○	○
getBytes	–	–	–	–	–	–
getDate	–	–	–	–	–	○
getTime	–	–	–	–	–	○
getTimestamp	–	–	–	–	–	○
getAsciiStream	–	–	–	–	–	○
getBinaryStream	–	–	–	–	–	–
getCharacterStream	–	–	–	–	–	○
getArray	–	–	–	–	–	○
getBlob	–	–	–	–	–	–

（凡例）

○：値に関係なく，オーバフローしません。

×：値によっては，オーバフローすることがあります。

–：この組み合わせでは使用できません。

文字列型：CHAR 型，MCHAR 型，NCHAR 型，VARCHAR 型，MVARCHAR 型，及び NVARCHAR 型

表 17–118　getXXX メソッド使用時のオーバフロー有無（2/2）

getXXX メソッド	HiRDB のデータ型				
	DATE	TIME	TIMESTAMP	BINARY	BLOB
getByte	–	–	–	–	–
getShort	–	–	–	–	–

getXXX メソッド	HiRDB のデータ型				
	DATE	TIME	TIMESTAMP	BINARY	BLOB
getInt	－	－	－	－	－
getLong	－	－	－	－	－
getFloat	－	－	－	－	－
getDouble	－	－	－	－	－
getBigDecimal	－	－	－	－	－
getBoolean	－	－	－	－	－
getString	○	○	○	○	○
getBytes	－	－	－	○	○
getDate	○	－	○	－	－
getTime	－	○	○	－	－
getTimestamp	○	－	○	－	－
getAsciiStream	－	－	－	○	○
getBinaryStream	－	－	－	○	○
getCharacterStream	－	－	－	○	○
getArray	－	－	－	○	○
getBlob	－	－	－	○	○

（凡例）

○：値に関係なく，オーバフローしません。

－：この組み合わせでは使用できません。

(4) getObject メソッド

次の表に，getObject メソッド使用時にオーバフローするかどうかを，HiRDB のデータ型ごとに示します。

表 17–119 getObject メソッド使用時のオーバフロー有無（1/2）

getObject メソッド	HiRDB のデータ型					
	SMALLINT	INTEGER	FLOAT	REAL	DECIMAL	文字列型
Byte	×	×	×	×	×	×
Short	○	×	×	×	×	×
Int	○	○	×	×	×	×
Long	○	○	×	×	×	×

getObject メソッド	HiRDB のデータ型					
	SMALLINT	INTEGER	FLOAT	REAL	DECIMAL	文字列型
Float	○	○	×	○	×	×
Double	○	○	○	×	×	×
BigDecimal	○	○	○	×	×	×
Boolean	○	○	○	○	○	○
String	○	○	○	○	○	○
Bytes	−	−	−	−	−	−
Date	−	−	−	−	−	○
Time	−	−	−	−	−	○
Timestamp	−	−	−	−	−	○
AsciiStream	−	−	−	−	−	○
BinaryStream	−	−	−	−	−	−
Object	○	○	○	○	○	○
CharacterStream	−	−	−	−	−	○
Array	−	−	−	−	−	○
Blob	−	−	−	−	−	−

（凡例）

○：値に関係なく，オーバフローしません。

×：値によっては，オーバフローすることがあります。

−：この組み合わせでは使用できません。

文字列型：CHAR 型，MCHAR 型，NCHAR 型，VARCHAR 型，MVARCHAR 型，及び NVARCHAR 型

表 17–120　getObject メソッド使用時のオーバフロー有無（2/2）

getObject メソッド	HiRDB のデータ型				
	DATE	TIME	TIMESTAMP	BINARY	BLOB
Byte	−	−	−	−	−
Short	−	−	−	−	−
Int	−	−	−	−	−
Long	−	−	−	−	−
Float	−	−	−	−	−
Double	−	−	−	−	−

getObject メソッド	HiRDB のデータ型				
	DATE	TIME	TIMESTAMP	BINARY	BLOB
BigDecimal	−	−	−	−	−
Boolean	−	−	−	−	−
String	○	○	○	−	−
Bytes	−	−	−	○	○
Date	○	−	○	−	−
Time	−	○	○	−	−
Timestamp	○	−	○	−	−
AsciiStream	−	−	−	○	○
BinaryStream	−	−	−	○	○
Object	○	○	○	○	○
CharacterStream	−	−	−	○	○
Array	−	−	−	○	○
Blob	−	−	−	○	○

（凡例）

○：値に関係なく，オーバフローしません。

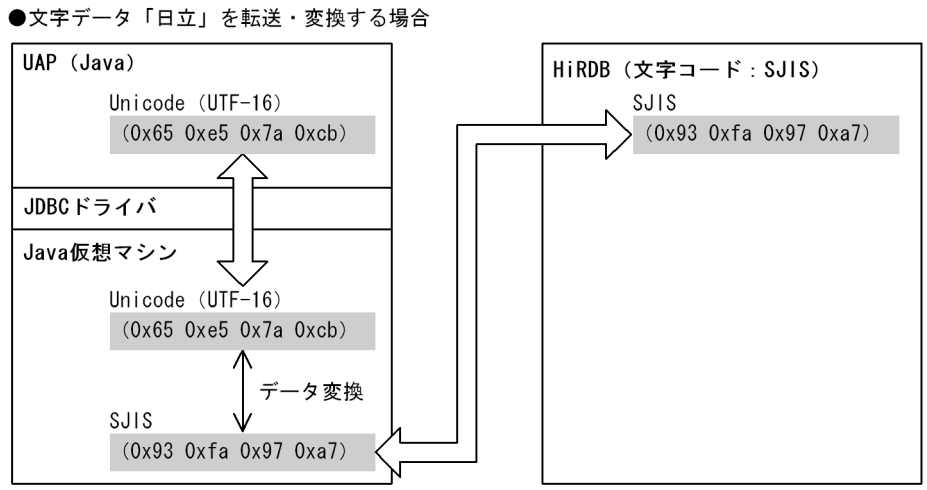
−：この組み合わせでは使用できません。

17.13　文字コード変換機能

Java プログラム内では，文字コードは Unicode で扱うため，JDBC ドライバが HiRDB の文字データと Unicode との相互文字コード変換をします。この文字コード変換処理で，JDBC ドライバは Java 仮想マシンが提供するエンコーダを利用します。

HiRDB の文字データと Unicode との相互文字コード変換の流れを次の図に示します。

図 17–2　HiRDB の文字データと Unicode との相互文字コード変換の流れ



HiRDB との文字データのやり取りの際，Java 仮想マシンのエンコーダに対して，JDBC ドライバが文字セット名称を指定します。このとき，HiRDB サーバの文字コードを取得し，それに相当する文字セット名称にします。また，接続時のプロパティ ENCODELANG や setEncodeLang メソッドによって文字セット名称が指定されている場合は，指定された文字セット名称が優先的に Java 仮想マシンのエンコーダに対して指定されます。そのため，プロパティ ENCODELANG や setEncodeLang メソッドに HiRDB サーバの文字コードと対応しない文字セット名称を指定した場合，文字コード変換に不正が生じます。

17.14　指定できるクライアント環境定義

JDBC ドライバで指定できるクライアント環境定義の一覧を次の表に示します。なお，各環境変数の詳細は「クライアント環境定義の設定内容」を参照してください。

また，指定値に引用符（"）で囲む値を含む場合の注意事項については，「クライアント環境定義の一覧」の「注※5」を参照してください。

表 17–121　JDBC ドライバで指定できるクライアント環境定義の一覧

環境変数名	対応するシステムプロパティ※1	機能	サポートドライバ			環境変数の分類
			JDBC2.0	JDBC4.0	JDBC4.3	
PDHOST	HiRDB_for_Java_PDHOST	接続する HiRDB サーバのホスト名を指定します。	○	○	○	システム構成
PDNAMEPORT	HiRDB_for_Java_PDNAMEPORT	HiRDB サーバのポート番号を指定します。	○	○	○	
PDFESHOST	HiRDB_for_Java_PDFESHOST	フロントエンドサーバのホスト名を指定します。	○	○	○	
PDSERVICEGRP	HiRDB_for_Java_PDSERVICEGRP	シングルサーバ又はフロントエンドサーバのサーバ名を指定します。	○	○	○	
PDSRVTYPE	HiRDB_for_Java_PDSRVTYPE	HiRDB サーバの種別を指定します。	○	○	○	
PDSERVICEPORT	HiRDB_for_Java_PDSERVICEPORT	高速接続用のポート番号を指定します。	○	○	○	
PDFESGRP	HiRDB_for_Java_PDFESGRP	高速接続をする場合，接続する FES グループを指定します。	○	○	○	
PDCLTRCVPORT	HiRDB_for_Java_PDCLTRCVPORT	クライアントの受信ポート番号を指定します。	○	○	○	
PDCLTRCVADDR	HiRDB_for_Java_PDCLTRCVADDR	クライアントの IP アドレス又はホスト名を指定します。	○	○	○	
PDCONTYPE	HiRDB_for_Java_PDCONTYPE	HiRDB 接続時の接続方式を指定します。	×	○	○	
PDUSER	HiRDB_for_Java_PDUSER	認可識別子，及びパスワードを指定します。UNIX 環境の場合	○	○	○	ユーザ実行環境

環境変数名	対応するシステムプロパティ※1	機能	サポートドライバ			環境変数の分類
			JDBC2.0	JDBC4.0	JDBC4.3	
		は，この環境変数を省略できます。				
PDCLTAPNAME	HiRDB_for_Java_PDCLTAPNAME	HiRDB サーバに対してアクセスする，UAP の識別情報（UAP 識別子）を指定します。	○	○	○	
PDDBLOG	HiRDB_for_Java_PDDBLOG	UAP を実行するときに，データベースの更新ログを取得するかしないかを指定します。	○	○	○	
PDEXWARN	HiRDB_for_Java_PDEXWARN	サーバから警告付きのリターンコードを受け取るかどうかを指定します。	○	○	○	
PDSUBSTRLEN	HiRDB_for_Java_PDSUBSTRLEN	1 文字を表現する最大バイト数を指定します。	○	○	○	
PDCLTGRP	HiRDB_for_Java_PDCLTGRP	クライアントグループの接続枠保証機能を使用する場合，クライアントグループ名を指定します。	○	○	○	
PDAUTORECONNECT	HiRDB_for_Java_PDAUTORECONNECT	自動再接続機能を使用するかどうかを指定します。	○	○	○	
PDRCCOUNT	HiRDB_for_Java_PDRCCOUNT	自動再接続機能での CONNECT のリトライ回数を指定します。	○	○	○	
PDRCINTERVAL	HiRDB_for_Java_PDRCINTERVAL	自動再接続機能での CONNECT のリトライ間隔を指定します。	○	○	○	
PDRCTIMING	HiRDB_for_Java_PDRCTIMING	自動再接続機能での再接続契機を指定します。	○	○	○	
PDAUTHTYPE	HiRDB_for_Java_PDAUTHTYPE	HiRDB クライアントがサーバへ接続するときの認証方式を指定します。	○	○	○	

環境変数名	対応するシステムプロパティ※1	機能	サポートドライバ			環境変数の分類
			JDBC2.0	JDBC4.0	JDBC4.3	
PDUAPENVFILE	HiRDB_for_Java_PDUAPENVFILE	UAP を個別の環境で実行する場合，実行する環境を定義した UAP 環境定義ファイルを指定します。	○	○	○	
PDDBBUFLRU	HiRDB_for_Java_PDDBBUFLRU	UAP がアクセスしたページをグローバルバッファにキャッシュするときの処理に，LRU 方式を適用するかどうかを指定します。	○	○	○	
PDHATRNQUEUING	HiRDB_for_Java_PDHATRNQUEUING	クライアントでトランザクションキューイング機能を使用しない場合に指定します。	○	○	○	
PDCLTBINDLOOPBACKADDR	HiRDB_for_Java_PDCLTBINDLOOPBACKADDR	HiRDB サーバとの通信で使用する受信ポートの生成時，ループバックアドレスで bind()するかどうかを指定します。	○	○	○	
PDDEFAULTOPTION	HiRDB_for_Java_PDDEFAULTOPTION	クライアント環境定義について，省略時の動作を指定します。	○	○	○	
PDSOCKETCRYPTO	HiRDB_for_Java_PDSOCKETCRYPTO	HiRDB クライアントと HiRDB サーバ間の通信を暗号化するかどうかを指定します。	×	○	○	
PDCWAITTIME	HiRDB_for_Java_PDCWAITTIME	HiRDB クライアントから HiRDB サーバへ要求をしてから，応答が戻ってくるまでの HiRDB クライアントの最大待ち時間を指定します。	○	○	○	システム監視
PDSWAITTIME	HiRDB_for_Java_PDSWAITTIME	HiRDB サーバが HiRDB クライアントからの要求に対する応答を返してから，次に HiRDB クライアントから要求が来るまでの HiRDB サーバ	○	○	○	

環境変数名	対応するシステムプロパティ※1	機能	サポートドライバ			環境変数の分類
			JDBC2.0	JDBC4.0	JDBC4.3	
		の最大待ち時間を指定します。 この時間監視は，トランザクション処理中の時間を対象とします。				
PDSWATCHTIME	HiRDB_for_Java_PDSWATCHTIME	HiRDB サーバが HiRDB クライアントからの要求に対する応答を返してから，次に HiRDB クライアントから要求が来るまでの HiRDB サーバの最大待ち時間を指定します。 この時間監視は，トランザクション処理以外の時間を対象とします。	○	○	○	
PDCWAITTIMEWRNPNT	HiRDB_for_Java_PDCWAITTIMEWRNPNT	SQL 実行時間警告出力機能使用時に，SQL 実行時間警告情報ファイルを出力する契機を，HiRDB クライアントの最大待ち時間に対する比率，又は時間で指定します。	○	○	○	
PDKALVL	HiRDB_for_Java_PDKALVL	HiRDB クライアントから HiRDB サーバに対して，定期的にパケットを送信する機能を使用するかどうかを指定します。	○	○	○	
PDKATIME	HiRDB_for_Java_PDKATIME	HiRDB クライアントから HiRDB サーバに対して，定期的にパケットを送信する間隔を指定します。	○	○	○	
PDNBLOCKWAITTIME	HiRDB_for_Java_PDNBLOCKWAITTIME	HiRDB サーバ，HiRDB クライアント間のコネクション接続完了を監視する場合，ノンブロックモード時のコネク	○	○	○	

環境変数名	対応するシステムプロパティ※1	機能	サポートドライバ			環境変数の分類
			JDBC2.0	JDBC4.0	JDBC4.3	
		ション確立監視時間を指定します。				
PDCONNECTWAITTIME	HiRDB_for_Java_PDCONNECTWAITTIME	HiRDB サーバとの接続時，HiRDB サーバから応答が戻ってくるまでの HiRDB クライアントの最大待ち時間を指定します。	○	○	○	
PDCLTPATH	HiRDB_for_Java_PDCLTPATH	HiRDB クライアントが作成する SQL トレースファイル及びクライアントエラーログファイルの格納先ディレクトリを指定します。	○	○	○	トラブルシュート
PDSQLTRACE※2	HiRDB_for_Java_PDSQLTRACE	UAP の SQL トレースを出力する SQL トレースファイルのサイズを，バイト単位で指定します。	○	○	○	
PDPRMTRC	HiRDB_for_Java_PDPRMTRC	SQL トレースにパラメタ情報及び検索データを出力するかどうかを指定します。	○	○	○	
PDPRMTRCSIZE	HiRDB_for_Java_PDPRMTRCSIZE	SQL トレースに出力するパラメタ情報及び検索データの最大データ長を指定します。	○	○	○	
PDUAPREPLVL	HiRDB_for_Java_PDUAPREPLVL	UAP 統計レポートの出力情報を指定します。	○	○	○	
PDREPPATH	HiRDB_for_Java_PDREPPATH	PDCLTPATH で指定したディレクトリとは別のディレクトリに，UAP 統計レポートを出力するかどうかを指定します。	○	○	○	
PDTRCPATH	HiRDB_for_Java_PDTRCPATH	動的 SQL トレースファイルの格納先ディレクトリを指定します。	○	○	○	

環境変数名	対応するシステムプロパティ※1	機能	サポートドライバ			環境変数の分類
			JDBC2.0	JDBC4.0	JDBC4.3	
PDSQLTEXTSIZE	HiRDB_for_Java_PDSQLTEXTSIZE	SQL トレースに出力する SQL 文のサイズを指定します。	○	○	○	
PDSQLEXECTIME	HiRDB_for_Java_PDSQLEXECTIME	SQL トレースに SQL 実行時間を出力するかどうかを指定します。	×	○	○	
PDRCTRACE	HiRDB_for_Java_PDRCTRACE	UAP の再接続トレースを出力するファイルのサイズを指定します。	○	○	○	
PDWRTLNPATH	HiRDB_for_Java_PDWRTLNPATH	WRITE LINE 文の値式の値を出力する，ファイルの格納先ディレクトリを指定します。	○	○	○	
PDWRTLNFILSZ	HiRDB_for_Java_PDWRTLNFILSZ	WRITE LINE 文の値式の値を出力する，ファイルの最大サイズを指定します。	○	○	○	
PDWRTLNCOMSZ	HiRDB_for_Java_PDWRTLNCOMSZ	WRITE LINE 文の値式の値の合計サイズを指定します。	○	○	○	
PDUAPEXERLOGUSE	HiRDB_for_Java_PDUAPEXERLOGUSE	拡張 SQL エラー情報出力機能を使用するかどうかを指定します。	×	○	○	
PDUAPEXERLOGPRMSZ	HiRDB_for_Java_PDUAPEXERLOGPRMSZ	拡張 SQL エラー情報出力機能を使用する場合，Exception トレースログ及び SQL エラーレポートファイルに出力するパラメタ情報の最大データ長を指定します。	×	○	○	
PDSQLTRCFMT	HiRDB_for_Java_PDSQLTRCFMT	SQL トレースの出力形式を指定します。	○	○	○	
PDSQLTRCTYPE	HiRDB_for_Java_PDSQLTRCTYPE	SQL トレース情報の出力タイミングを指定します。	×	○	○	

環境変数名	対応するシステムプロパティ※1	機能	サポートドライバ			環境変数の分類
			JDBC2.0	JDBC4.0	JDBC4.3	
PDJDBSQLTRCMEMNUM	HiRDB_for_Java_PDJDBSQLTRCMEMNUM	Type4 JDBC ドライバでメモリ上に退避する SQL トレース情報数を指定します。	×	○	○	
PDVWOPTMODE	HiRDB_for_Java_PDVWOPTMODE	アクセスパス情報ファイルを取得するかどうかを指定します。	○	○	○	アクセスパス表示ユティリティ用アクセスパス情報ファイル
PDTAAPINFPATH	HiRDB_for_Java_PDTAAPINFPATH	アクセスパス情報ファイルを HiRDB クライアント側に出力する場合に，出力先ディレクトリを指定します。この指定がない場合は出力しません。	×	○	○	HiRDB SQL Tuning Advisor 用アクセスパス情報ファイル
PDTAAPINFMODE	HiRDB_for_Java_PDTAAPINFMODE	アクセスパス情報ファイルを HiRDB クライアント側に出力する場合に，アクセスパス情報ファイルのファイル名の形式を指定します。	×	○	○	
PDTAAPINFSIZE	HiRDB_for_Java_PDTAAPINFSIZE	アクセスパス情報ファイルを HiRDB クライアント側に出力する場合に，アクセスパス情報ファイルのファイルサイズを指定します。	×	○	○	
PDSTJTRNOUT	HiRDB_for_Java_PDSTJTRNOUT	UAP に関する統計情報を，トランザクションごとに統計ログファイルに出力するかどうかを指定します。	○	○	○	UAP に関する統計情報の出力単位
PDLOCKLIMIT	HiRDB_for_Java_PDLOCKLIMIT	一つのサーバに対して UAP から発行する排他要求の上限値を指定します。	○	○	○	排他制御
PDDLKPRIO	HiRDB_for_Java_PDDLKPRIO	UAP のデッドロックプライオリティ値を指定します。	○	○	○	

環境変数名	対応するシステムプロパティ※1	機能	サポートドライバ			環境変数の分類
			JDBC2.0	JDBC4.0	JDBC4.3	
PDLOCKSKIP	HiRDB_for_Java_PDLOCKSKIP	無排他条件判定をするかどうかを指定します。	○	○	○	
PDFORUPDATEEXLOCK	HiRDB_for_Java_PDFORUPDATEEXLOCK	FOR UPDATE 句を指定した（又は仮定された）SQL の排他オプションに，WITH EXCLUSIVE LOCK を適用するかどうかを指定します。	○	○	○	
PDISLLVL	HiRDB_for_Java_PDISLLVL	SQL 文のデータ保証レベルを指定します。	○	○	○	SQL 関連
PDSQLOPTLVL	HiRDB_for_Java_PDSQLOPTLVL	データベースの状態を考慮して，最も効率的なアクセスパスを決定するための最適化の方法（SQL 最適化オプション）を指定します。	○	○	○	
PDADDITIONALOPTLVL	HiRDB_for_Java_PDADDITIONALOPTLVL	データベースの状態を考慮して，最も効率的なアクセスパスを決定するための最適化の方法（SQL 拡張最適化オプション）を指定します。	○	○	○	
PDHASHTBLSIZE	HiRDB_for_Java_PDHASHTBLSIZE	SQL の最適化で，ハッシュジョイン，副問合せのハッシュ実行を適用する場合，ハッシュ表サイズを指定します。	○	○	○	
PDDFLNVAL	HiRDB_for_Java_PDDFLNVAL	表中のデータを埋込み変数に取り出す場合，取り出した値がナル値のときに埋込み変数に既定値を設定するかどうかを指定します。	○	○	○	
PDAGGR	HiRDB_for_Java_PDAGGR	GROUP BY 処理に使用するメモリ量を決定するため，サーバごとに発生するグ	○	○	○	

環境変数名	対応するシステムプロパティ※1	機能	サポートドライバ			環境変数の分類
			JDBC2.0	JDBC4.0	JDBC4.3	
		ループ数の最大値を指定します。				
PDCMMTBFDDL	HiRDB_for_Java_PDCMMTBFDDL	操作系 SQL を実行していたトランザクションで定義系 SQL を実行する場合，自動的にコミットしてから定義系 SQL を実行するかどうかを指定します。	○	○	○	
PDPRPCRCLS	HiRDB_for_Java_PDPRPCRCLS	開いているカーソルで使用している SQL 識別子を再度 PREPARE 文で使用する場合，開いているカーソルを自動的にクローズするかどうかを指定します。	○	○	○	
PDDDLDEAPRPEXE	HiRDB_for_Java_PDDDLDEAPRPEXE	先行トランザクションの前処理結果を無効にし，定義系トランザクションの実行を優先します。	○	○	○	
PDDDLDEAPRP	HiRDB_for_Java_PDDDLDEAPRP	閉じているホールダブルカーソルで使用している表の定義情報を，トランザクション間に他 UAP からの変更を許可するかどうかを指定します。	○	○	○	
PDLCKWAITTIME	HiRDB_for_Java_PDLCKWAITTIME	排他要求が待ち状態になってから解除されるまでの最大監視時間を指定します。	○	○	○	
PDDELRSVWDFILE	HiRDB_for_Java_PDDELRSVWDFILE	SQL 予約語削除機能を使用する場合に，SQL 予約語削除ファイル名を指定します。	○	○	○	
PDCALCMDWAITTIME	HiRDB_for_Java_PDCALCMDWAITTIME	CALL COMMAND 文によってコマンド，又はユティリティを開始してから終了するまでの，HiRDB ク	○	○	○	

環境変数名	対応するシステムプロパティ※1	機能	サポートドライバ			環境変数の分類
			JDBC2.0	JDBC4.0	JDBC4.3	
		ライアントの最大待ち時間を指定します。				
PDSTANDARDSQLSTATE	HiRDB_for_Java_PDSTANDARDSQLSTATE	SQLSTATE の値を詳細に出力するかどうかを指定します。	○	○	○	
PDBLKF	HiRDB_for_Java_PDBLKF	HiRDB サーバから HiRDB クライアントに検索結果を転送するときの，一回の転送処理で送られる行数を指定します。	○	○	○	ブロック転送機能
PDBINARYBLKF	HiRDB_for_Java_PDBINARYBLKF	定義長が 32,001 バイト以上の BINARY 型の選択式がある表を検索する場合，ブロック転送機能を適用するかどうかを指定します。	○	○	○	
PDBLKBUFFSIZE	HiRDB_for_Java_PDBLKBUFFSIZE	ブロック転送機能で使用する，サーバ，クライアント間の通信バッファのサイズを指定します。	○	○	○	
PDDBACCS	HiRDB_for_Java_PDDBACCS	インナレプリカ機能を使用している場合，カレント RD エリアではない RD エリアをアクセスしたいときに，その RD エリアの世代番号を指定します。	○	○	○	インナレプリカ機能
PDDBORGUAP	HiRDB_for_Java_PDDBORGUAP	オンライン再編成閉塞のオリジナル RD エリアに対して UAP を実行する場合に指定します。	○	○	○	更新可能なオンライン再編成
PDSPACELVL	HiRDB_for_Java_PDSPACELVL	データの格納，比較，及び検索時の，空白変換レベルを指定します。	○	○	○	データの空白変換
PDCLTRDNODE	HiRDB_for_Java_PDCLTRDNODE	XDM/RD E2 接続機能使用時に，接続する XDM/RD E2 のデータ	○	○	○	XDM/RD E2 接続機能

環境変数名	対応するシステムプロパティ※1	機能	サポートドライバ			環境変数の分類
			JDBC2.0	JDBC4.0	JDBC4.3	
		ベース識別子を指定します。				
PDCNSTRNTNAME	HiRDB_for_Java_PDCNSTRNTNAME	参照制約，及び検査制約を定義する場合，制約名定義の位置を指定します。	○	○	○	参照制約及び検査制約
PDTMPTBLRDAREA	HiRDB_for_Java_PDTMPTBLRDAREA	一時表及び一時インデクスを格納する RD エリアの候補を指定します。	○	○	○	一時表
PDBESCONHOLD	HiRDB_for_Java_PDBESCONHOLD	バックエンドサーバ接続保持機能を使用するかどうかを指定します。	○	○	○	バックエンドサーバ接続保持機能
PDBESCONHTI	HiRDB_for_Java_PDBESCONHTI	バックエンドサーバ接続保持機能を使用する場合，バックエンドサーバ接続保持期間を指定します。	○	○	○	
PDPLGIXMK	HiRDB_for_Java_PDPLGIXMK	プラグインインデクスの遅延一括作成を使用するかどうかを指定します。	○	○	○	プラグイン
PDPLUGINNSUB	HiRDB_for_Java_PDPLUGINNSUB	詳細については，各プラグインマニュアルを参照してください。	○	○	○	
PDPLGPFSZ	HiRDB_for_Java_PDPLGPFSZ	プラグインの遅延一括作成用のインデクス情報ファイルの初期容量を指定します。	○	○	○	
PDPLGPFSZEXP	HiRDB_for_Java_PDPLGPFSZEXP	プラグインの遅延一括作成用のインデクス情報ファイルの増分値を指定します。	○	○	○	
PDHSICOPTIONS	HiRDB_for_Java_PDHSICOPTIONS	HSIC を使用する運用の場合に，HSIC で集積する情報を指定します。	○	○	○	HSIC 限定
PDJDBFILEDIR	－	Type4 JDBC ドライバでの Exception トレースログのログ	○	○	○	JDBC ドライバ

環境変数名	対応するシステムプロパティ※1	機能	サポートドライバ			環境変数の分類
			JDBC2.0	JDBC4.0	JDBC4.3	
		ファイル出力先を指定します。				
PDJDBFILEOUTNUM	−	Type4 JDBC ドライバでの Exception トレースログのログファイル及び不正電文トレースファイルへの出力数を指定します。	○	○	○	
PDJDBONMEMNUM	−	Type4 JDBC ドライバでの Exception トレースログのメモリ内取得情報数を指定します。	○	○	○	
PDJDBTRACELEVEL	−	Type4 JDBC ドライバでの Exception トレースログのトレース取得レベルを指定します。	○	○	○	
PDJDBFILESIZE	−	Type4 JDBC ドライバでの Exception トレースログのログファイルの最大サイズを指定します。	○	○	○	

(凡例)

−：該当しません。

○：指定できます。

×：指定できません。

注※1

クライアント環境定義と同じ意味を持つ接続情報をシステムプロパティで指定できます。指定の優先順位については，「接続情報の優先順位」を参照してください。Exception トレースログに関する接続情報のシステムプロパティの設定については，「Exception トレースログを取得するための設定」を参照してください。

なお，内部ドライバの場合，システムプロパティの指定は無効です。

注※2

出力ファイル名については，「SQL トレース機能」を参照してください。

HiRDB.INI ファイルを使用してクライアント環境変数を有効にする場合

次の場合，任意のディレクトリ下の HiRDB.INI ファイル内に設定しているクライアント環境変数を有効にできます。

- DriverManager.getConnection の Properties 引数中の HiRDB_for_Java_HiRDB_INI を指定している
- DriverManager.getConnection の URL 中に HiRDB_INI を指定している
- DataSource 系インタフェースの setHiRDBINI メソッドで HiRDB_INI ファイルの絶対パスを指定している

指定の優先順位については，「接続情報の優先順位」を参照してください。

なお，内部ドライバの場合，HiRDB.INI ファイルの指定は無効です。

17.15　接続情報の優先順位

17.15.1　接続情報の優先順位一覧

JDBC ドライバでは，意味が同じ接続情報を複数の設定方法で指定できます（例：URL 中に指定する DBHOST と，HiRDB クライアント環境定義の PDHOST）。このような複数の設定方法を持つ接続情報と，同時に複数の設定方法で設定された場合の優先順位を次の表に示します。

表 17–122　接続情報の優先順位

接続情報の意味	設定方法	優先順位		
		A	B	C
HiRDB のホスト名称	システムプロパティに設定した HiRDB_for_Java_PDHOST	1	1	1
	DriverManager.getConnection の Properties 引数中のプロパティ HiRDB_for_Java_DBHOST	2	−	−
	URL 中の DBHOST	3	−	−
	DriverManager.getConnection の Properties 引数中の HiRDB_for_Java_ENV_VARIABLES で指定した HiRDB クライアント環境定義の PDHOST	4	−	−
	DriverManager.getConnection の Properties 引数中の HiRDB_for_Java_DBID に指定した HiRDB 環境変数グループ内の PDHOST	5	−	−
	URL 中の DBID に指定した HiRDB 環境変数グループ内の PDHOST	6	−	−
	DriverManager.getConnection の Properties 引数中の HiRDB_for_Java_HiRDB_INI に指定した HiRDB.ini ファイル内の PDHOST	7	−	−
	URL 中の HiRDB_INI に指定した HiRDB.ini ファイル内の PDHOST	8	−	−
	DataSource 系インタフェースの setDBHostName メソッド	−	2	2
	DataSource 系インタフェースの setEnvironmentVariables メソッドで指定した HiRDB クライアント環境定義の PDHOST	−	3	3
	DataSource 系インタフェースの setDescription メソッドで指定した HiRDB 環境変数グループ内の PDHOST	−	4	−
	XADataSource.setXAOpenString で指定した HiRDB 環境変数グループ内の PDHOST	−	−	4
	DataSource 系インタフェースの setHiRDBINI メソッドで指定した HiRDB.ini ファイル内の PDHOST	−	5	5
HiRDB のポート番号	システムプロパティに設定した HiRDB_for_Java_ PDNAMEPORT	1	1	1

接続情報の意味	設定方法	優先順位		
		A	B	C
	DriverManager.getConnection の Properties 引数中のプロパティ HiRDB_for_Java_DBID	2	−	−
	URL 中の DBID	3	−	−
	DriverManager.getConnection の Properties 引数中の HiRDB_for_Java_ENV_VARIABLES で指定した HiRDB クライアント環境定義の PDNAMEPORT	4	−	−
	DriverManager.getConnection の Properties 引数中の HiRDB_for_Java_DBID に指定した HiRDB 環境変数グループ内の PDNAMEPORT	5	−	−
	URL 中の DBID に指定した HiRDB 環境変数グループ内の PDNAMEPORT	6	−	−
	DriverManager.getConnection の Properties 引数中の HiRDB_for_Java_HiRDB_INI に指定した HiRDB.ini ファイル内の PDNAMEPORT	7	−	−
	URL 中の HiRDB_INI に指定した HiRDB.ini ファイル内の PDNAMEPORT	8	−	−
	DataSource 系インタフェースの setDescription メソッド	−	2	−
	DataSource 系インタフェースの setEnvironmentVariables メソッドで指定した HiRDB クライアント環境定義の PDNAMEPORT	−	3	2
	DataSource 系インタフェースの setDescription メソッドで指定した HiRDB 環境変数グループ内の PDNAMEPORT	−	4	−
	XADataSource.setXAOpenString で指定した HiRDB 環境変数グループ内の PDNAMEPORT	−	−	3
	DataSource 系インタフェースの setHiRDBINI メソッドで指定した HiRDB.ini ファイル内の PDNAMEPORT	−	5	4
接続時のユーザ名，パスワード※1	システムプロパティに設定した HiRDB_for_Java_PDUSER	1	1	1
	DriverManager.getConnection の user 引数及び password 引数，又は Properties 引数中の user 及び password	2	−	−
	URL 中の USER 及び PASSWORD	3	−	−
	DriverManager.getConnection の Properties 引数中の HiRDB_for_Java_ENV_VARIABLES で指定した HiRDB クライアント環境変数内の PDUSER	4	−	−
	DriverManager.getConnection の Properties 引数中の HiRDB_for_Java_DBID に指定した HiRDB 環境変数グループ内の PDUSER	5	−	−
	URL 中の DBID に指定した HiRDB 環境変数グループ内の PDUSER	6	−	−

接続情報の意味	設定方法	優先順位		
		A	B	C
	DriverManager.getConnection の Properties 引数中の HiRDB_for_Java_HiRDB_INI に指定した HiRDB.ini ファイル内の PDUSER	7	−	−
	URL 中の HiRDB_INI に指定した HiRDB.ini ファイル内の PDUSER	8	−	−
	DataSource インタフェースの getConnection メソッドの引数，又は ConnectionPoolDataSource インタフェースの getPooledConnection メソッドの引数	−	2	−
	XADataSource インタフェースの getXAConnection メソッドの引数	−	−	2
	DataSource 系インタフェースの setUser メソッド及び setPassword メソッド	−	3	3
	DataSource 系インタフェースの setEnvironmentVariables メソッドで指定した HiRDB クライアント環境変数内の PDUSER	−	4	4
	DataSource 系インタフェースの setDescription メソッドで指定した HiRDB 環境変数グループ内の PDUSER	−	5	−
	XADataSource.setXAOpenString で指定した HiRDB 環境変数グループ内の PDUSER	−	−	5
	DataSource 系インタフェースの setHiRDBINI メソッドで指定した HiRDB.ini ファイル内の PDUSER	−	6	6
UAP 名称※2	システムプロパティに設定した HiRDB_for_Java_ PDCLTAPNAME	1	1	1
	DriverManager.getConnection の Properties 引数中のプロパティ UAPNAME	2	−	−
	URL 中の UAPNAME	3	−	−
	DriverManager.getConnection の Properties 引数中の HiRDB_for_Java_ENV_VARIABLES で指定した HiRDB クライアント環境定義の PDCLTAPNAME	4	−	−
	DriverManager.getConnection の Properties 引数中の HiRDB_for_Java_DBID に指定した HiRDB 環境変数グループ内の PDCLTAPNAME	5	−	−
	URL 中の DBID に指定した HiRDB 環境変数グループ内の PDCLTAPNAME	6	−	−
	DriverManager.getConnection の Properties 引数中の HiRDB_for_Java_HiRDB_INI に指定した HiRDB.ini ファイル内の PDCLTAPNAME	7	−	−
	URL 中の HiRDB_INI に指定した HiRDB.ini ファイル内の PDCLTAPNAME	8	−	−
	DataSource 系インタフェースの setUapName メソッド	−	2	2

接続情報の意味	設定方法	優先順位		
		A	B	C
	DataSource 系インタフェースの setEnvironmentVariables メソッドで指定した HiRDB クライアント環境定義の PDCLTAPNAME	−	3	3
	DataSource 系インタフェースの setDescription メソッドで指定した HiRDB 環境変数グループ内の PDCLTAPNAME	−	4	−
	XADataSource.setXAOpenString で指定した HiRDB 環境変数グループ内の PDCLTAPNAME	−	−	4
	DataSource 系インタフェースの setHiRDBINI メソッドで指定した HiRDB.ini ファイル内の PDCLTAPNAME	−	5	5
変換文字セット	DriverManager.getConnection の Properties 引数中のプロパティ ENCODELANG	1	−	−
	URL 中の ENCODELANG	2	−	−
	DataSource 系インタフェースの setEncodeLang	−	1	1
カーソル動作モード	DriverManager.getConnection の Properties 引数中のプロパティ HIRDB_CURSOR	1	−	−
	URL 中の HIRDB_CURSOR	2	−	−
	DataSource 系インタフェースの setHiRDBCursorMode	−	1	1
ステートメントのコミット実行後の状態	システムプロパティに設定した HiRDB_for_Java_DAB_STATEMENT_COMMIT_BEHAVIOR	1	1	1
	DriverManager.getConnection の Properties 引数中のプロパティ HiRDB_for_Java_STATEMENT_COMMIT_BEHAVIOR	2	−	−
	URL 中の STATEMENT_COMMIT_BEHAVIOR	3	−	−
	DataSource 系インタフェースの setStatementCommitBehavior	−	2	2
ログイン待ち時間	システムプロパティに設定した HiRDB_for_Java_PDCONNECTWAITTIME	1	1	1
	DriverManager.setLoginTimeout	2	−	−
	DriverManager.getConnection の Properties 引数中の HiRDB_for_Java_ENV_VARIABLES で指定した HiRDB クライアント環境定義の PDCONNECTWAITTIME	3	−	−
	DriverManager.getConnection の Properties 引数中の DBID に指定した HiRDB 環境変数グループ内の PDCONNECTWAITTIME	4	−	−
	URL 中の DBID に指定した HiRDB 環境変数グループ内の PDCONNECTWAITTIME	5	−	−
	DriverManager.getConnection の Properties 引数中の HiRDB_for_Java_HiRDB_INI に指定した HiRDB.ini ファイル内の PDCONNECTWAITTIME	6	−	−

接続情報の意味	設定方法	優先順位		
		A	B	C
	URL 中の HiRDB_INI に指定した HiRDB.ini ファイル内の PDCONNECTWAITTIME	7	−	−
	DataSource 系インタフェースの setLoginTimeout	−	2	2
	DataSource 系インタフェースの setEnvironmentVariables メソッドで指定した HiRDB クライアント環境定義の PDCONNECTWAITTIME	−	3	3
	DataSource 系インタフェースの setDescription メソッドで指定した HiRDB 環境変数グループ内の PDCONNECTWAITTIME	−	4	−
	XADataSource.setXAOpenString で指定した HiRDB 環境変数グループ内の PDCONNECTWAITTIME	−	−	4
	DataSource 系インタフェースの setHiRDBINI メソッドで指定した HiRDB.ini ファイル内の PDCONNECTWAITTIME	−	5	5
ステートメントの close メソッド実行時の SQL 前処理の破棄	システムプロパティに設定した HiRDB_for_Java_STATEMENT_CLOSE_BEHAVIOR	1	1	1
	DriverManager.getConnection の Properties 引数中のプロパティ HiRDB_for_Java_STATEMENT_CLOSE_BEHAVIOR	2	−	−
	URL 中の STATEMENT_CLOSE_BEHAVIOR	3	−	−
	DataSource 系インタフェースの setStatementCloseBehavior	−	2	2
ステートメントの更新行数が JDBC 規格に準拠するかどうか	システムプロパティに設定した HiRDB_for_Java_UPDATECOUNT_BEHAVIOR	1	1	1
	DriverManager.getConnection の Properties 引数中のプロパティ HiRDB_for_Java_UPDATECOUNT_BEHAVIOR	2	−	−
	URL 中の UPDATECOUNT_BEHAVIOR	3	−	−
	DataSource 系インタフェースの setUpdateCountBehavior	−	2	2

（凡例）

A：DriverManager を使用した接続の場合

B：DataSource 系インタフェースを使用した非 XA 接続の場合

C：XADataSource インタフェースを使用した XA 接続の場合

−：接続方法によって指定できない

注※1

ユーザ名を指定してパスワードを指定していない場合，パスワード指定がないものとみなします。ユーザ名を指定しないでパスワードを指定した場合，パスワードの指定は無効になり，次に優先順位の高い指定が有効になります。

注※2

この表で示した設定方法で設定されない場合は，次の表に示す JDBC ドライバの製品名称が設定されたものとして動作します。

表 17–123　UAP 名称を省略した場合に設定される製品名称

JDBC	Cosminexus	J2EE サーバ名長（バイト数）	製品名称
2.0 の場合	使用しない	–	HiRDB_Type4_JDBC_Driver
	使用する	23 バイト以下	JDBC20_xxxx xxxx：J2EE サーバ名
		24 バイト以上	JDBC20_xxxx xxxx：J2EE サーバ名（末尾部 23 バイト抽出）
4.0 の場合	使用しない	–	HiRDB_Type4_JDBC40_Driver
	使用する	23 バイト以下	JDBC40_xxxx xxxx：J2EE サーバ名
		24 バイト以上	JDBC40_xxxx xxxx：J2EE サーバ名（末尾部 23 バイト抽出）
4.3 の場合	使用しない	–	HiRDB_Type4_JDBC43_Driver
	使用する	23 バイト以下	JDBC43_xxxx xxxx：J2EE サーバ名
		24 バイト以上	JDBC43_xxxx xxxx：J2EE サーバ名（末尾部 23 バイト抽出）

（凡例）　–：該当しません。

17.15.2　そのほかのクライアント環境定義の優先順位

そのほかの HiRDB クライアント環境定義の優先順位を示します。

設定方法	優先順位		
	A	B	C
システムプロパティの指定	1	1	1
DriverManager.getConnection の Properties 引数中の HiRDB_for_Java_ENV_VARIABLES で指定した HiRDB クライアント環境定義	2	–	–
DriverManager.getConnection の Properties 引数中の HiRDB_for_Java_DBID に指定した HiRDB 環境変数グループファイル内のクライアント環境変数	3	–	–

設定方法	優先順位		
	A	B	C
DriverManager.getConnection の URL 中の DBID に指定した HiRDB 環境変数グループファイル内のクライアント環境定義	4	−	−
DriverManager.getConnection の Properties 引数中の HiRDB_for_Java_HiRDB_INI に指定したディレクトリ下の HiRDB.ini ファイルに指定された HiRDB クライアント環境変数	5	−	−
DriverManager.getConnection の URL 中の HiRDB_INI に指定したディレクトリ下の HiRDB.ini ファイルに指定された HiRDB クライアント環境変数	6	−	−
DataSource 系インタフェースの setEnvironmentVariables メソッドで指定したクライアント環境定義	−	2	2
DataSource 系インタフェースの setDescription メソッドで指定した HiRDB 環境変数グループファイル内のクライアント環境定義	−	3	−
XADataSource.setXAOpenString で指定した HiRDB 環境変数グループファイル内のクライアント環境定義	−	−	3
DataSource 系インタフェースの setHiRDBINI メソッドで指定した HiRDB.ini ファイル内のクライアント環境定義	−	4	4

注

- DriverManager.getConnection の Properties 引数中の HiRDB_for_Java_DBID 及び URL 中の DBID に HiRDB 環境変数グループファイルを指定した場合，URL 中の DBID の指定は無効となります。
- DriverManager.getConnection の Properties 引数中の HiRDB_for_Java_HiRDB_INI 及び URL 中の HiRDB_INI を指定した場合，URL 中の HiRDB_INI の指定は無効になります。

17.16　Type2 JDBC ドライバからの移行

ここでは，Type2 JDBC ドライバを使用して動作していた Java ストアドプロシジャを，Type4 JDBC ドライバを使用して動作させる場合について説明します。

なお，次に示すプラットフォームの場合は，Type2 JDBC ドライバから Type4 JDBC ドライバへ，プログラムを変更しないで移行できます。

- 64 ビットモードの AIX
- Linux(EM64T)
- Windows (x64)

上記以外のプラットフォームの場合に，内部ドライバを Type2 JDBC ドライバから Type4 JDBC ドライバに移行するときは，次に示すとおり設定を変更する必要があります。

変更が必要な項目	Type2 JDBC ドライバ	Type4 JDBC ドライバ
ドライバ名称	"JP.co.Hitachi.soft.HiRDB.JDBC.PrdbDriver"	"JP.co.Hitachi.soft.HiRDB.JDBC.HiRDBDriver"
HiRDB への接続時の URL で設定するプロトコル名称，サブプロトコル名称，及びサブネーム	jdbc:hitachi:PrdbDrive	jdbc:hitachi:hirdb
DataSource クラスのクラス名	JdbhDataSource	PrdbDataSource
カーソルの動作モード※	次のどれかを設定します。 • 接続時の URL の COMMIT_BEHAVIOR • 接続時に指定するプロパティの COMMIT_BEHAVIOR • DataSource クラスの setCommit_Behavior メソッド	次のどれかを設定します。 • 接続時の URL の HIRDB_CURSOR と STATEMENT_COMMIT_BEHAVIOR を組み合わせて設定 • 接続時に指定するプロパティの HIRDB_CURSOR と STATEMENT_COMMIT_BEHAVIOR を組み合わせて設定 • DataSource クラスの setStatementCommitBehavior と setHiRDBCursorMode メソッドを組み合わせて設定
HiRDB の配列更新，配列挿入，配列削除機能	次のどれかで，この機能を使用する設定をした場合に，使用できます。 • 接続時に指定するプロパティの BLOCK_UPDATE	無条件に使用できます。 ただし，JdbcDbpsvPreparedStatement クラスの setBlockUpdate メソッドは使用できません。

変更が必要な項目	Type2 JDBC ドライバ	Type4 JDBC ドライバ
	• システムプロパティの HiRDB_for_Java_BLOCK_UPDATE • DataSource クラスの，setBlockUpdate メソッド • JdbcDbpsvPreparedStatement クラスの setBlockUpdate メソッド	
実行する SQL の入力，又は入出力?パラメタの最大数	次のどれかを設定します。 • 接続時に指定するプロパティの HiRDB_for_Java_SQL_IN_NUM • システムプロパティの HiRDB_for_Java_SQL_IN_NUM • DataSource クラスの，setSQLInNum メソッド デフォルト値は，64 です。	次のどちらかを設定します。 • 接続時に指定するプロパティの HiRDB_for_Java_SQL_IN_NUM • DataSource クラスの，setSQLInNum メソッド デフォルト値は，300 です。
実行する SQL の出力項目数の最大数	次のどれかを設定します。 • 接続時に指定するプロパティの HiRDB_for_Java_SQL_OUT_NUM • システムプロパティの HiRDB_for_Java_SQL_OUT_NUM • DataSource クラスの，setSQLOutNum メソッド デフォルト値は，64 です。	次のどちらかを設定します。 • 接続時に指定するプロパティの HiRDB_for_Java_SQL_OUT_NUM • DataSource クラスの，setSQLOutNum メソッド デフォルト値は，300 です。

注※

カーソルの動作モードによってコミット実行後に ResultSet オブジェクト及びステートメントオブジェクトが有効になるかどうかが決まります。

Type2 JDBC ドライバと Type4 JDBC ドライバのカーソルの動作モードの設定の対応を次に示します。

なお，Type2 JDBC ドライバと Type4 JDBC ドライバでは，デフォルト値が異なります。

コミット実行後の ResultSet オブジェクトの状態	コミット実行後のステートメントオブジェクトの状態	Type2 JDBC ドライバ	Type4 JDBC ドライバ
無効	無効	COMMIT_BEHAVIOR="DELETE"（デフォルト値）	HIRDB_CURSOR=FALSE（デフォルト値） STATEMENT_COMMIT_BEHAVIOR=FALSE
	有効	COMMIT_BEHAVIOR="CLOSE"	HIRDB_CURSOR=FALSE（デフォルト値） STATEMENT_COMMIT_BEHAVIOR=TRUE（デフォルト値）
有効	有効	COMMIT_BEHAVIOR="PRESERVE"	HIRDB_CURSOR=TRUE STATEMENT_COMMIT_BEHAVIOR=TRUE 又は FALSE

《Java アプリケーション中のクライアント環境定義について》

Type2 JDBC ドライバの場合は，接続時の URL などを省略すると OS の環境変数に指定した PDHOST 及び PDNAMEPORT が有効となります。

Type4 JDBC ドライバの場合は，OS の環境変数は使用しません。そのため，Type2 JDBC ドライバから移行するときは，接続時の URL などにクライアント環境定義の PDHOST 及び PDNAMEPORT を指定するように，UAP を変更してください。

17.17　DABroker for Java からの移行

使用する JDBC ドライバを DABroker for Java から Type4 JDBC ドライバへ移行する場合，UAP の修正が必要になることがあります。このとき，システムプロパティを設定して Type4 JDBC ドライバを DABroker for Java 互換モードで動作させる機能を DABroker for Java 互換機能と呼びます。ここでは，DABroker for Java 互換機能について説明します。

17.17.1　DABroker for Java 互換機能に関するシステムプロパティ

DABroker for Java との互換機能に関する接続情報を，システムプロパティで指定できます。指定できるシステムプロパティを次の表に示します。

なお，内部ドライバの場合，システムプロパティの指定は無効です。

表 17–124　DABroker for Java 互換機能に関するシステムプロパティ

プロパティ名	内容	指定可否
HiRDB_for_Java_DAB_CONVERT_NULL	CallableStatement クラスの setString，getString メソッドを DABroker for Java 互換モードで動作させるかどうかを設定します。省略した場合，"FALSE"が仮定されます。設定した値については大文字と小文字を区別しません。 "TRUE"： DABroker for Java 互換モードで動作させます。 setString メソッドで，？パラメタのデータ型が CHAR 型，VARCHAR 型，NCHAR 型，NVARCHAR 型，MCHAR 型又は MVARCHAR 型で，かつ引数で指定された値が 0 長文字列の場合，null を？パラメタに設定します。 又は，getString メソッドで取得した？パラメタ値が 0 長文字列の場合，戻り値に null を設定します。 "FALSE"： DABroker for Java 互換モードで動作させません。 setString メソッドで，？パラメタのデータ型が CHAR 型，VARCHAR 型，NCHAR 型，NVARCHAR 型，MCHAR 型又は MVARCHAR 型で，かつ引数で指定された値が 0 長文字列の場合，0 長文字列を？パラメタに設定します。 又は，getString メソッドで取得した？パラメタ値が 0 長文字列の場合，戻り値に 0 長文字列を設定します。 その他： "FALSE"が設定されたものとみなします。	任意
HiRDB_for_Java_DAB_STATEMENT_COMMIT_BEHAVIOR	コミット実行時の，ステートメントのオブジェクトを無効にする処理を，DABroker for Java 互換モードで動作させるかどうかを設定します。省略した場合，"FALSE"が仮定されます。設定した値については大文字と小文字を区別しません。 "TRUE"： DABroker for Java と互換させます。 コミットした場合，ステートメントのオブジェクトを無効とします。	任意

プロパティ名	内容	指定可否
	"FALSE"： DABroker for Java と互換させません。 ほかの設定方法で設定した値を使用します。設定方法については，「接続情報の優先順位一覧」を参照してください。 その他： "FALSE"が設定されたものとみなします。	
HiRDB_for_Java_DAB_EXECUTESQL_NOCHK	ステートメントの executeQuery，executeUpdate メソッドを，DABroker for Java 互換モードで動作させるかどうかを設定します。省略した場合，"FALSE"が仮定されます。設定した値については大文字と小文字を区別しません。 "TRUE"： DABroker for Java 互換モードで動作させます。 • ステートメントの executeQuery メソッドを使用して検索系 SQL 以外の SQL を実行できます。 • ステートメントの executeUpdate メソッドを使用して検索系 SQL を実行できます。 • ステートメントの execute，executeQuery，executeUpdate メソッドで検索系 SQL 以外の SQL を実行後に getResultSet メソッドを実行した場合，0 列の ResultSet を返します。 "FALSE"： DABroker for Java 互換モードで動作させません。 • ステートメントの executeQuery メソッドを使用して検索系 SQL 以外の SQL を実行すると，SQLException を投入します。 • ステートメントの executeUpdate メソッドを使用して検索系 SQL を実行すると，SQLException を投入します。 • ステートメントの execute，executeUpdate メソッドで検索系 SQL 以外の SQL を実行後に getResultSet メソッドを実行した場合，null を返します。 その他： "FALSE"が設定されたものとみなします。	任意
HiRDB_for_Java_DAB_OUTPARMSIZE_MAX	CallableStatement クラスの registerOutParameter メソッドを，DABroker for Java 互換モードで動作させるかどうかを設定します。省略した場合，"FALSE"が仮定されます。設定した値については大文字と小文字を区別しません。 "TRUE": DABroker for Java 互換モードで動作させます。 call 文の前処理で取得したデータ型が VARCHAR，NVARCHAR，MVARCHAR の INOUT 及び OUT パラメタのデータ型を，CallableStatement クラス registerOutParameter メソッドによって java.sql.Types.CHAR 型と登録した場合，CallableStatement クラス getString メソッドでは前処理で取得したデータの最大長のサイズのデータを返します。 ストアドプロシジャが設定したデータのサイズが前処理で取得したデータの最大長より小さい場合，ストアドプロシジャが設定したデータに空白を付加し，前処理で取得したデータの最大長と同じサイズのデータにします。 "FALSE"： DABroker for Java 互換モードで動作させません。 call 文の前処理で取得したデータ型が VARCHAR，NVARCHAR，MVARCHAR の INOUT 及び OUT パラメタのデータ型を，CallableStatement クラス registerOutParameter メソッドに	任意

プロパティ名	内容	指定可否
	よって java.sql.Types.CHAR 型と登録した場合，CallableStatement クラス getString メソッドではストアドプロシジャが設定したデータをそのまま返します。ストアドプロシジャが設定したデータには空白を付加しません。 その他： "FALSE"が設定されたものとみなします。	

17.17.2　Type4 JDBC ドライバと互換性のない項目

Type4 JDBC ドライバと互換性のない項目を，対象となる DABroker for Java 及び Cosminexus のバージョンごとに，次の表に示します。

DABroker for Java 及び Cosminexus のバージョン

- DABroker for Java Version 2 02-10 以前
- Cosminexus Studio Version 5 05-05-/E 以前
- Cosminexus Application Server Version 5 05-05-/E 以前
- Cosminexus Developer Version 5 05-05-/E 以前

表 17–125　Type4 JDBC ドライバと互換性のない項目（その 1）

互換性のない項目		DABroker for Java	Type4 JDBC ドライバ
CallableStatement クラスの setString メソッド及び getString メソッドでの 0 長文字列の扱い	setString メソッド	？パラメタのデータ型が CHAR 型，VARCHAR 型，NCHAR 型，NVARCHAR 型，MCHAR 型又は MVARCHAR 型で，かつ引数で設定された値が 0 長文字列の場合，null を？パラメタに設定します。	？パラメタのデータ型が CHAR 型，VARCHAR 型，NCHAR 型，NVARCHAR 型，MCHAR 型又は MVARCHAR 型で，かつ引数で設定された値が 0 長文字列の場合，0 長文字列を？パラメタに設定します。
	getString メソッド	取得した？パラメタの値が 0 長文字列の場合，戻り値に null を設定します。	取得した？パラメタの値が 0 長文字列の場合，戻り値に 0 長文字列を設定します。
ステートメントのコミット実行後の状態		コミットした場合，ステートメントのオブジェクトを無効にします。	コミットした場合，ステートメントのオブジェクトを有効にします。

DABroker for Java 及び Cosminexus のバージョン

- DABroker for Java Version 2 02-07 以前
- Cosminexus Studio Version 5 05-05 以前
- Cosminexus Application Server Version 5 05-05 以前

- Cosminexus Developer Version 5 05-05 以前

表 17–126　Type4 JDBC ドライバと互換性のない項目（その 2）

互換性のない項目		DABroker for Java	Type4 JDBC ドライバ
• ステートメントの executeQuery メソッド，executeUpdate メソッドで実行できる SQL 種別 • 検索系 SQL 以外の SQL 実行後の getResultSet メソッドの戻り値	executeQuery メソッド	• すべての SQL を実行できます。 • 検索系 SQL 以外の SQL を実行した場合，0 列の ResultSet を返します。	検索系 SQL だけ実行できます。
	executeUpdate メソッド	• すべての SQL を実行できます。 • 検索系 SQL を実行した場合，-1 を返します。	検索系 SQL 以外の SQL だけ実行できます。
	getResultSet メソッド	ステートメントの execute メソッド，executeQuery メソッド，executeUpdate メソッドで検索系 SQL 以外の SQL を実行後に getResultSet メソッドを実行した場合，0 列の ResultSet を返します。	ステートメントの execute メソッド，executeUpdate メソッドで検索系 SQL 以外の SQL を実行後に getResultSet メソッドを実行した場合，null を返します。

DABroker for Java 及び Cosminexus のバージョン

- Cosminexus DABroker 03-00-/D 以前
- Cosminexus DABroker for Java 02-06-/B 以前
- Cosminexus Studio Version 5 05-00-/D 以前
- Cosminexus Application Server Version 5 05-00-/D 以前
- Cosminexus Developer Version 5 05-00-/D 以前

表 17–127　Type4 JDBC ドライバと互換性のない項目（その 3）

互換性のない項目	DABroker for Java	Type4 JDBC ドライバ
call 文の前処理で取得したデータ型が VARCHAR，NVARCHAR，MVARCHAR である INOUT パラメタ及び OUT パラメタのデータ型を，CallableStatement クラスの registerOutParameter メソッドによって java.sql.Types.CHAR 型と登録した場合の，CallableStatement クラスの getString メソッドで受け取るデータ	前処理で取得したデータの最大長のサイズのデータを返します。 ストアドプロシジャが設定したデータのサイズが前処理で取得したデータの最大長より小さい場合，ストアドプロシジャが設定したデータに空白を付加し，前処理で取得したデータの最大長と同じサイズのデータにします。	ストアドプロシジャが設定したデータを返します。 ストアドプロシジャが設定したデータには空白を付加しません。

17.18　JDBC インタフェースメソッドトレース

トラブルシュート情報として，JDBC インタフェースのメソッド呼び出し時に，JDBC インタフェースメソッドトレースを取得できます。

17.18.1　取得するための設定

(1)　DriverManager クラスによる接続の場合

DriverManager クラスの setLogWriter メソッドによって有効なログライタを指定し，getConnection メソッドの引数（Properties info）に，JDBC インタフェースメソッドトレースの取得を指定します。

詳細は，「JDBC_IF」及び「TRC_NO」を参照してください。

(2)　DataSource クラスによる接続の場合

DataSource，ConnectionPoolDataSource，XADataSource の各インタフェースで提供する setLogWriter メソッドによって有効なログライタを指定し，JDBC2.0 Optional Package で提供する DataSource，ConnectionPoolDataSource，XADataSource の各クラスで提供する setJDBC_IF_TRC メソッドを指定します。また，setTRC_NO メソッドに，エントリ数を指定します。

詳細は，「setJDBC_IF_TRC」及び「setTRC_NO」を参照してください。

17.18.2　取得規則

JDBC インタフェースメソッドトレースの取得規則を次に示します。

- JDBC インタフェースのメソッドの呼び出し時，及びメソッドからの戻り時に，トレース情報が取得されます。

  ただし，データベース接続前に実行できるメソッドついては，トレース情報が取得されません。

  トレース情報が取得されないメソッドを次に示します。

  Driver インタフェース

  - acceptsURL(String url)
  - getMajorVersion()
  - getMinorVersion()
  - getPropertyInfo(String url, Properties info)
  - jdbcCompliant()

  DataSource 系インタフェース

  - getLoginTimeout()

・getLogWriter()
・setLoginTimeout(int seconds)
・setLogWriter(PrintWriter out)

- トレース情報はエントリ数分保持され，Connection.close メソッドの呼び出し時（正常終了時），又はSQLException，XAException，若しくは BatchUpdateException の投入時（エラー発生時）に，指定したログライタに出力されます。
- トレース情報の数がエントリ数を超えた場合は，保持されていたトレース情報を古い順に破棄され，新しいトレース情報が保持されます。
- JDBC インタフェースメソッドトレースは，Entry 及び Return でそれぞれ 1 エントリのトレース領域を使用します。

17.18.3　出力例

JDBC インタフェースメソッドトレースの出力例を次に示します。

出力例

```
[1]                         [2]                       [3]
[HiRDB_Type4_JDBC40_Driver][JDBC Interface Entry ][PrdbStatement.executeQuery]
[HiRDB_Type4_JDBC40_Driver]                        sql=select * from pp
       [4]
[HiRDB_Type4_JDBC40_Driver][JDBC Interface Return][PrdbStatement.executeQuery]
[HiRDB_Type4_JDBC40_Driver]                        Return=JP.co.Hitachi.soft.HiRDB.JDBC.Pr
db.... [5]
[HiRDB_Type4_JDBC40_Driver][JDBC Interface Entry ][PrdbResultSet.getMetaData]
[HiRDB_Type4_JDBC40_Driver][JDBC Interface Return][PrdbResultSet.getMetaData]
[HiRDB_Type4_JDBC40_Driver]                        Return=JP.co.Hitachi.soft.HiRDB.JDBC.Pr
db.....
```

説明

1. [HiRDB_Type4_JDBC40_Driver]

   JDBC ドライバの名称

   ・JDBC2.0：HiRDB_Type4_JDBC_Driver

   ・JDBC4.0：HiRDB_Type4_JDBC40_Driver

   ・JDBC4.3：HiRDB_Type4_JDBC43_Driver

2. [JDBC Interface Entry]，[JDBC Interface Return]

   [JDBC Interface Entry]：JDBC メソッドの呼び出し

   [JDBC Interface Return]：JDBC メソッドからの戻り

3. [XXXXX.YYYYY]

   XXXXX クラスの YYYYY メソッド

4. select * from pp

   JDBC メソッドの引数。

   パスワードを示す引数については，"password=*"のように，"*"1 個を出力します。

   また，URL を示す引数にパスワードが含まれる場合は，パスワードを"*"1 個に置き換えて出力します。

5. JP.co.Hitachi.soft.HiRDB.JDBC.Prdb

   JDBC メソッドの戻り値。

   戻り値にパスワードが含まれる場合は，パスワードを"*"1 個に置き換えて出力します。

17.19　Exception トレースログ

トラブルシュート情報として，Exception トレースログを取得できます。Exception トレースログには，JDBC ドライバ内で例外による障害が発生した際の障害要因が出力されます。

出力内容を次に示します。

- 例外発生時の情報（エラーメッセージなど）
- 例外が発生するまでの，JDBC の API メソッドの実行記録

この機能を使用すると，UAP から呼び出される JDBC の API メソッドの情報が JDBC ドライバのメモリ上に蓄積され，SQLException，BatchUpdateException，又は XAException の発生を契機として，例外を投入する前に，メモリ上に蓄積された情報がファイルに出力されます。

JDBC4.0 以降では，Exception トレースログで実行記録を取得するメソッドで RuntimeException を検知した場合も同様の情報をファイルに出力します。

Exception を投入しない警告レベルの事象が発生した場合，メソッドの実行履歴，及び例外のスタックトレースを除く情報をファイルに出力します。出力情報の詳細については，「出力形式」を参照してください。

警告レベルの事象を次に示します。

事象		出力メッセージ
自動再接続処理の開始		KFPZ02445-I（AUTO_RECONNECT）
Connectoin.close 時にエラー発生		KFPA11723-E，その他
KeepAlive 処理	通信処理でエラー発生	KFPA11723-E，その他
	不正電文受信	KFPZ03000-I（KeepAlive）
System.getProperties()で SecurityException 発生		KFPZ02444-E

17.19.1　取得するメソッド，及び取得するための設定

(1)　Exception トレースログの取得対象メソッド

Exception トレースログの取得対象は，Java 2 Platform, Standard Edition, version 1.4 の API 仕様にあるパッケージ java.sql に記述されているメソッドの呼び出しと戻りです。

クライアント環境定義 PDJDBTRACELEVEL，又はシステムプロパティ HiRDB_for_Java_TraceLevel で指定した値が，次の表に示すメソッドのトレース取得レベル以上の場合に，そのメソッドを取得対象とします。

なお，ResultSet.getXXX メソッド，PreparedStatement.setXXX メソッド，Connection.isClosed メソッドなど，オブジェクト内の情報を参照して返すだけのメソッドや，オブジェクト内に情報を格納するだけのメソッドは取得対象になりません。

Exception トレースログの取得対象であるメソッドと，そのメソッドのトレース取得レベルを次の表に示します。

表 17–128　Exception トレースログの取得対象であるメソッドとトレース取得レベル

クラス	メソッド	トレース取得レベル※1			付加情報
		JDBC2.0	JDBC4.0	JDBC4.3	
Driver	Connection connect(String url, Properties info)	1	1	1	−
Connection	Statement createStatement()※2	1	1	1	−
	Statement createStatement(int resultSetType, int resultSetConcurrency)※3	1	1	1	−
	Statement createStatement(int resultSetType, int resultSetConcurrency, int resultSetHoldability)※4	1	1	1	−
	PreparedStatement prepareStatement(String sql)※2	1	1	1	−
	PreparedStatement prepareStatement(String sql, int resultSetType, int resultSetConcurrency)※3	1	1	1	−
	PreparedStatement prepareStatement(String sql, int resultSetType, int resultSetConcurrency, int resultSetHoldability)※4	1	1	1	−
	CallableStatement prepareCall(String sql)※2	1	1	1	−
	CallableStatement prepareCall(String sql, int resultSetType, int resultSetConcurrency)※3	1	1	1	−
	CallableStatement prepareCall(String sql, int resultSetType, int resultSetConcurrency, int resultSetHoldability)※4	1	1	1	−
	void setAutoCommit(boolean autoCommit)	2	2	2	−
	void commit()	2	1	1	−
	void rollback()	2	1	1	−
	void close()	1	1	1	−
	boolean isValid(int timeout)	2	1	1	−
	DatabaseMetaData getMetaData()	2	2	2	−
	void setHoldability(int holdability)	−	1	1	−
	beginRequest	−	−	1	−

クラス	メソッド	トレース取得レベル※1			付加情報
		JDBC2.0	JDBC4.0	JDBC4.3	
	endRequest	−	−	1	−
Statement	ResultSet executeQuery(String sql)	1	1	1	−
	int executeUpdate(String sql)	1	1	1	−
	boolean execute(String sql)	1	1	1	−
	void addBatch(String sql)	−	1	1	−
	int[] executeBatch()	2	1	1	−
	void clearBatch()	−	2	2	−
	void close()	1	1	1	−
	boolean isClosed()	2	2	2	−
	void cancel()	2	1	1	−
	ResultSet getResultSet()	2	2	2	−
	boolean getMoreResults()	−	1	1	−
	void setMaxFieldSize(int max)	−	2	2	−
	void setMaxRows(int max)	−	2	2	−
	void setQueryTimeout(int seconds)	−	1	1	−
	void setFetchSize(int rows)	−	1	1	−
	boolean isPoolable()	2	2	2	−
	executeLargeBatch()	−	−	1	−
	getLargeUpdateCount()	−	−	1	−
	executeLargeUpdate(String sql)	−	−	1	−
	getLargeMaxRows()	−	−	1	−
	setLargeMaxRows(long max)	−	−	1	−
	isSimpleIdentifier(String identifier)	−	−	1	−
	enquoteIdentifier(String identifier, boolean alwaysQuote)	−	−	1	−
	enquoteLiteral(String val)	−	−	1	−
	enquoteNCharLiteral(String val)	−	−	1	−
	closeOnCompletion()	−	−	1	−
	isCloseOnCompletion()	−	−	1	−

クラス	メソッド	トレース取得レベル※1			付加情報
		JDBC2.0	JDBC4.0	JDBC4.3	
PreparedStatement	ResultSet executeQuery()※2	2	1	1	[SQL]
	int executeUpdate()※2	2	1	1	[SQL]
	boolean execute()※2	2	1	1	[SQL]
	ResultSetMetaData getMetaData()	2	2	2	−
	void setBlob(int parameterIndex, Blob x)※6	−	2	2	−
	executeLargeUpdate	−	−	1	−
CallableStatement	String getString(int parameterIndex)※6	−	2	2	−
	long getLong(int parameterIndex)※6	−	2	2	−
	byte[] getBytes(int parameterIndex)※6	−	2	2	−
	Object getObject(int parameterIndex)※6	−	2	2	−
	Blob getBlob(int parameterIndex)※6	−	2	2	−
	String getString(String parameterName)※6	−	2	2	−
	long getLong(String parameterName)※6	−	2	2	−
	byte[] getBytes(String parameterName)※6	−	2	2	−
	Object getObject(String parameterName)※6	−	2	2	−
	Blob getBlob(String parameterName)※6	−	2	2	−
ResultSet	boolean next()※5	2	1	1	[COUNT]
	boolean previous()	−	2	2	−
	boolean absolute(int row)	2	1	1	−
	boolean relative(int rows)	2	1	1	−
	void beforeFirst()	2	1	1	−
	boolean first()	2	1	1	−
	boolean last()	2	1	1	−
	void afterLast()	2	1	1	−
	boolean isBeforeFirst()	2	1	1	−
	boolean isAfterLast()	2	1	1	−
	boolean isLast()	2	1	1	−
	void close()※5	2	1	1	[COUNT]
	boolean isClosed()	2	2	2	−

クラス	メソッド	トレース取得レベル※1			付加情報
		JDBC2.0	JDBC4.0	JDBC4.3	
	Statement getStatement()	2	2	2	−
	ResultSetMetaData getMetaData()	2	2	2	−
	int getHoldability()	2	2	2	−
	String getString(int columnIndex)※6	−	2	2	−
	long getLong(int columnIndex)※6	−	2	2	−
	byte[] getBytes(int columnIndex)※6	−	2	2	−
	Blob getBlob(int columnIndex)※6	−	2	2	−
	Object getObject(int columnIndex)※6	−	2	2	−
	InputStream getAsciiStream(int columnIndex)※6	−	2	2	−
	Reader getCharacterStream(int columnIndex)※6	−	2	2	−
	InputStream getBinaryStream(int columnIndex)※6	−	2	2	−
	String getString(String columnLabel)※6	−	2	2	−
	long getLong(String columnLabel)※6	−	2	2	−
	Blob getBlob(String columnLabel)※6	−	2	2	−
	Object getObject(String columnLabel)※6	−	2	2	−
	InputStream getAsciiStream(String columnLabel)※6	−	2	2	−
	Reader getCharacterStream(String columnLabel)※6	−	2	2	−
	InputStream getBinaryStream(String columnLabel)※6	−	2	2	−
Blob	long length()※6	2	2	2	−
	byte[] getBytes(long pos, int length)※6	2	2	2	−
	long position(byte[] pattern, long start)※6	2	2	2	−
	long position(Blob pattern, long start)※6	2	2	2	−
	void free()※6	1	1	1	−
DatabaseMetaData	ResultSet getProcedures(String catalog, String schemaPattern, String procedureNamePattern)	2	2	2	−
	ResultSet getProcedureColumns(String catalog, String schemaPattern, String procedureNamePattern, String columnNamePattern)	2	2	2	−

クラス	メソッド	トレース取得レベル※1			付加情報
		JDBC2.0	JDBC4.0	JDBC4.3	
	ResultSet getTables(String catalog, String schemaPattern, String tableNamePattern, String[] types)	2	2	2	−
	ResultSet getSchemas()	2	2	2	−
	ResultSet getCatalogs()	2	2	2	−
	ResultSet getTableTypes()	2	2	2	−
	ResultSet getColumns(String catalog, String schemaPattern, String tableNamePattern, String columnNamePattern)	2	2	2	−
	ResultSet getColumnPrivileges(String catalog, String schema, String table, String columnNamePattern)	2	2	2	−
	ResultSet getTablePrivileges(String catalog, String schemaPattern, String tableNamePattern)	2	2	2	−
	ResultSet getBestRowIdentifier(String catalog, String schema, String table, int scope, boolean nullable)	2	2	2	−
	ResultSet getVersionColumns(String catalog, String schema, String table)	2	2	2	−
	ResultSet getPrimaryKeys(String catalog, String schema, String table)	2	2	2	−
	ResultSet getImportedKeys(String catalog, String schema, String table)	2	2	2	−
	ResultSet getExportedKeys(String catalog, String schema, String table)	2	2	2	−
	ResultSet getCrossReference(String parentCatalog, String parentSchema, String parentTable, String foreignCatalog, String foreignSchema, String foreignTable)	2	2	2	−
	ResultSet getTypeInfo()	2	2	2	−
	ResultSet getIndexInfo(String catalog, String schema, String table, boolean unique, boolean approximate)	2	2	2	−
	ResultSet getUDTs(String catalog, String schemaPattern, String typeNamePattern, int[] types)	2	2	2	−
	Connection getConnection()	2	2	2	−
	ResultSet getSuperTypes(String catalog, String schemaPattern, String typeNamePattern)	2	2	2	−
	ResultSet getSuperTables(String catalog, String schemaPattern, String tableNamePattern)	2	2	2	−

クラス	メソッド	トレース取得レベル※1			付加情報
		JDBC2.0	JDBC4.0	JDBC4.3	
	ResultSet getAttributes(String catalog, String schemaPattern, String typeNamePattern, String attributeNamePattern)	2	2	2	−
	RowIdLifetime getRowIdLifetime()	2	2	2	−
	ResultSet getSchemas(String catalog, String schemaPattern)	2	2	2	−
	boolean supportsStoredFunctionsUsingCallSyntax()	2	2	2	−
	boolean autoCommitFailureClosesAllResultSets()	2	2	2	−
	ResultSet getClientInfoProperties()	2	2	2	−
	ResultSet getFunctions(String catalog, String schemaPattern, String functionNamePattern)	2	2	2	−
	ResultSet getFunctionColumns(String catalog, String schemaPattern, String functionNamePattern, String columnNamePattern)	2	2	2	−
	getPseudoColumns(String catalog, String schemaPattern, String tableNamePattern, String columnNamePattern)	−	2	2	−
	generatedKeyAlwaysReturned()	−	2	2	−
	getMaxLogicalLobSize()	−	2	2	−
	supportsRefCursors()	−	2	2	−
	supportsSharding()	−	2	2	−
DataSource	Connection getConnection()※2	1	1	1	−
	Connection getConnection(String username, String password)※3	1	1	1	−
ConnectionPoolDataSource	PooledConnection getPooledConnection()※2	1	1	1	−
	PooledConnection getPooledConnection(String user, String password)※3	1	1	1	−
XADataSource	XAConnection getXAConnection()※2	1	1	1	−
	XAConnection getXAConnection(String user,String password)※3	1	1	1	−
PooledConnection	Connection getConnection()	1	1	1	−
	void close()	1	1	1	−

クラス	メソッド	トレース取得レベル※1			付加情報
		JDBC2.0	JDBC4.0	JDBC4.3	
	void addConnectionEventListener(ConnectionEventListener listener)	−	2	2	−
	void removeConnectionEventListener(ConnectionEventListener listener)	−	2	2	−
	void addStatementEventListener(StatementEventListener listener)	−	2	2	−
	void removeStatementEventListener(StatementEventListener listener)	−	2	2	−
XAConnection	XAResource getXAResource()	1	1	1	−
XAResource	void start(Xid xid, int flags)	3	3	3	−
	void end(Xid xid, int flags)	3	3	3	−
	int prepare(Xid xid)	3	3	3	−
	void commit(Xid xid, boolean onePhase)	3	3	3	−
	void rollback(Xid xid)	3	3	3	−
	void forget(Xid xid)	3	3	3	−
	Xid[] recover(int flag)	3	3	3	−
	boolean isSameRM(XAResource xares)	3	3	3	−
InputStream	int read()※2	2	2	2	−
	int read(byte[] data)※3	2	2	2	−
	int read(byte[] b, int off, int len)※4	2	2	2	−
PrdbConnection	int checkSession (int waittime)	−	1	1	−

（凡例） −：出力しません。

注※1

トレース取得レベルが 5 の場合，内部呼び出しも含めて Exception トレースログを取得します。

注※2

メソッド名として，「メソッド名(1)」と出力されます。

注※3

メソッド名として，「メソッド名(2)」と出力されます。

注※4

メソッド名として，「メソッド名(3)」と出力されます。

注※5

トレース取得レベルが 1 の場合，カーソルクローズ時だけ[COUNT]付加情報（next 実行回数）付きでメソッド正常終了の履歴を出力します。トレース取得レベルが 2 以上の場合，メソッド呼び出しごとにメソッド開始・正常終了の履歴を出力し，[COUNT]付加情報は付けません。

注※6

対象が BLOB 又は BINARY 型で，位置付け子を使用している場合だけ，Exception トレースログを取得します。

(2)　Exception トレースログを取得するための設定

Exception トレースログのファイル出力先，ファイルへの出力数，メモリ内取得情報数，及びトレース取得レベルは，システムプロパティ又はクライアント環境定義で設定します。優先順位は次のとおりです。

1. システムプロパティ
2. クライアント環境定義

(a)　クライアント環境定義の設定

クライアント環境定義では，次の項目を指定します。

- PDJDBFILEDIR
- PDJDBFILEOUTNUM
- PDJDBONMEMNUM
- PDJDBTRACELEVEL
- PDJDBFILESIZE

これらは JavaVM プロセスごとでの設定となるため，同一 JavaVM プロセス上で接続ごとに異なる値を指定することはできません。指定した場合，最初の接続に対する指定が有効となります。

指定値の詳細は，「クライアント環境定義の設定内容」を参照してください。

なお，これらのクライアント環境定義に不正な値が指定されて，SQLException を投入する際に取得する Exception トレースログでは，これらのクライアント環境定義に値が指定されなかったものとみなされます。その場合は，次の表に示すデフォルトが仮定されます。

(b)　システムプロパティの設定

システムプロパティでは，次の表に示す項目を指定します。

表 17–129　Exception トレースログを取得するためのシステムプロパティの設定

項目	システムプロパティ	内容	デフォルト※
ファイル出力先	HiRDB_for_Java_FileDIR	Exception トレースログを出力するディレクトリを，絶対パスで指定します。Exception トレースログは，指定したディレクトリの直下に出力されます。 指定されたファイル出力先が存在しない，又は書き込み権限がない場合は，カレントディレクトリに出力します。	カレントディレクトリ
ファイルへの出力数	HiRDB_for_Java_FileOutNUM	1 ファイルへ出力する情報数の上限を指定します。指定できる範囲は 1〜50 です。 実際に 1 ファイルへ出力する情報数の上限は，「ファイルへの出力数」×「メモリ内取得情報数」個となります。 ファイルへの出力数は，「出力形式」に示す形式 2〜形式 4 をそれぞれ 1 個と数えます。 なお，ファイルには，メモリに蓄積した順番に出力されます。 また，上限値を超えてファイルに出力する場合は，2 ファイルでラップアラウンドします。ファイル名は次のとおりです。 • pdexc1.trc • pdexc2.trc ただし，「出力形式」に示す形式 1 と形式 2 の間で出力先のファイルが切り替わることはありません。 HiRDB_for_Java_FileSize プロパティが指定されている場合，HiRDB_for_Java_FileSize プロパティの指定を優先し，この指定は無視します。	5
メモリ内取得情報数	HiRDB_for_Java_OnMemNUM	メモリ内で蓄積する情報数の上限を指定します。指定できる範囲は 500〜10,000 です。 なお，メモリ内取得情報は，表「Exception トレースログの取得対象であるメソッドとトレース取得レベル」で示したメソッド一つを 1 個と数えます。 また，上限値を超えて情報を蓄積しようとすると，古い情報から順に，新しい情報に上書きされます。	1,000
トレース取得レベル	HiRDB_for_Java_TraceLevel	トレース取得レベルを指定します。指定できる範囲は 0〜5 です。 5 を指定すると，内部呼び出しを含めたすべてのトレース取得対象メソッドを取得します。 0 を指定すると，Exception トレースログを取得しません。	1
最大ファイルサイズ	HiRDB_for_Java_FileSize	ファイルの最大サイズをバイト単位で指定します。指定できる範囲は 4,096〜2,000,000,000 です。 指定サイズを超えると出力先を切り替えます。 このプロパティを指定していない場合，HiRDB_for_Java_FileOutNUM プロパティによって出力先を切り替えます。	なし

注※

次のような場合に取得する Exception トレースログでは，システムプロパティに値が指定されなかったものとみなされます。その場合はデフォルトが仮定されます。

- システムプロパティに不正な値が指定され，データベース接続時に SQLException を投入した場合
- Java 仮想マシンが，セキュリティマネジャによって JDBC ドライバに対するシステムプロパティの受け渡しを拒否した場合
- Java 仮想マシンの，最初の接続が確立する前

17.19.2　出力形式

Exception トレースログには，次の 4 種類の形式があります。

形式 1：ヘッダ部分

JDBC2.0 の場合：

```
[AA....AA] HiRDB_Type4_JDBC_Driver BB-CC
```

JDBC4.0 の場合：

```
[AA....AA] HiRDB_Type4_JDBC40_Driver BB-CC
```

JDBC4.3 の場合：

```
[AA....AA] HiRDB_Type4_JDBC43_Driver BB-CC
```

形式 2：メソッドの実行履歴（メソッドの実行開始）

```
AAAAAAAAAAAAAAAAAAAAAA BB....BB:[C][DD....DD]
                       ConnectionID(EE....EE) : SID(FF....FF)
                       GG....GG
                       II....II
```

形式 3：メソッドの実行履歴（メソッドの正常終了）

```
AAAAAAAAAAAAAAAAAAAAAA BB....BB:[C][DD....DD]
                       ConnectionID(EE....EE) : SID(FF....FF)
                       HH....HH
                       II....II
```

形式 4：Exception トレースログの出力契機

```
AAAAAAAAAAAAAAAAAAAAAA BB....BB:JJ....JJ:
KK....KK
```

形式 2 及び形式 3 は，時系列順に，メソッドを実行した分だけ繰り返して出力されます。

Exception を投入しない警告レベルの事象が発生した場合では，形式 2 及び形式 3 の情報は出力しません。

(1)　形式 1 の変数の説明

AA....AA：

出力情報の通番です。

通番は 1 回の出力（出力エラーによる失敗も含む）ごとに 1 増加します。2,147,483,647 を超えると，0 に戻ります。

BB：

JDBC ドライバのバージョンです。

CC：

JDBC ドライバのリビジョンです。

(2) 形式 2～形式 4 の変数の説明

AAAAAAAAAAAAAAAAAAAAAAA：

Exception トレースログの取得日時を，次の形式で出力します。それぞれの変数には，すべて 0～9 の値が入ります。

```
YYYY/MM/DD hh:mm:ss.sss
```

YYYY：年（西暦）

MM：月

DD：月

hh：時（24 時間形式）

mm：分

ss.sss：秒（小数点以下 3 けたを含みます）

BB....BB：

メソッドを実行したスレッド識別情報を，次の形式で出力します。

```
Thread[aa....aa]@bb....bb@cc....cc
```

aa....aa：スレッド名，優先順位，スレッドグループ名を含む，スレッド情報です。形式は Java 仮想マシンによって決まります。

bb....bb：オブジェクトのハッシュコードです。形式は Java 仮想マシンによって決まります。

cc....cc：スレッド ID です。JDBC4.0 版以降でだけ付加します。

C：

メソッドの呼び出し識別情報です。

E：メソッドが開始したときの履歴であることを示します。

R：メソッドが正常終了したときの履歴であることを示します。

DD....DD：

オブジェクト識別子及びメソッド名を，次の形式で出力します。

```
aa....aa.bb....bb
```

aa....aa：オブジェクト識別子です（最大 32 文字）。

形式は Java 仮想マシンによって決まります。

bb....bb：メソッド名です。

EE....EE：

接続 ID を次の形式で出力します。

```
aa....aa:bb....bb:cc....cc
```

aa....aa：フロントエンドサーバ名又はシングルサーバ名です（最大 32 文字）。

取得できない場合，"*"を出力します。

bb....bb：aa....aa で示すサーバの接続通番です（最大 10 文字）。

取得できない場合，"*"を出力します。

cc....cc：aa....aa で示すサーバのプロセス ID です（最大 10 文字）。

取得できない場合，"*"を出力します。

FF....FF：

セクション ID を出力します（最大 4 文字）。

GG....GG：

メソッドの引数を次の形式で出力します。引数がないメソッドでは出力しません。

```
aa....aa=bb....bb
aa....aa=bb....bb
         :
aa....aa=bb....bb
```

aa....aa：引数の名称です。

bb....bb：引数の内容です（最大 256 文字）。

参照型値の場合，形式はオブジェクトによって決まります。

なお，次のメソッドの引数 password については，bb....bb に"*"1 個を出力します。

- DataSource クラスの getConnection(String username, String password)
- ConnectionPoolDataSource クラスの getPooledConnection(String username, String password)
- XADataSource の getXAConnection(String username, String password)

また，Driver クラスの connect(String url,Properties info)の次の指定値については，"*"1 個に置き換えて出力します。

- 引数 info 中のプロパティ password
- 引数 info 中のプロパティ HiRDB_for_Java_ENV_VARIABLES
- 引数 url 中の PASSWORD

HH....HH：

メソッドの戻り値を次の形式で出力します。

```
Return=aa....aa
```

aa....aa：引数の名称です。

戻り値がないメソッドでは出力しません。戻り値が参照型値の場合，形式は Java 仮想マシンによって決まります。

II....II：

メソッドの動作に関係する引数・戻り値以外の情報を次の形式で出力します。

```
[aa....aa]=bb....bb
```

aa....aa：付加情報の種別です。

bb....bb：付加情報の内容です。

付加情報の種別と内容を次の表に示します。

種別	内容
[SQL]	SQL 文
[COUNT]	ResultSet.next メソッドの呼び出し回数

JJ....JJ：

ログの出力契機を示す次のどちらかの文字列を出力します。

Exception 投入時

```
Exception
```

Exception を投入しない警告レベルのメッセージ出力時

```
Warning
```

KK....KK：

トラブルシュート情報を，次のどちらかの形式で出力します。

Exception 投入時

```
ExceptionClass: aa....aa
UapEnvironment: bb....bb
Message: cc....cc
ErrorInfo: kk....kk
ErrorCode: dd....dd
SQLState: eeeee
UpdateCounts: ff....ff, ..<省略>.. ,ff....ff
SocketInfo: ll....ll
Etc.: gg....gg, hh....hh, iiii
URLInfo：n....n
RootAP：o....o
PropertyInfo：
p....p
ExtendedSQLErrorInfo:
mm....mm
jj....jj
```

Exception を投入しない警告レベルのメッセージ出力

```
UapEnvironment: b....b
Message: c....c
ErrorInfo: k....k
```

```
SocketInfo: L....L
Etc.: gg....gg, hh....hh, iiii
URLInfo：n....n
RootAP：o....o
PropertyInfo：
p....p
```

aa....aa：投入した例外オブジェクトの実行クラス名です。

bb....bb：例外オブジェクトの接続で使用しているクライアント環境定義を，次の形式で出力します。出力しない場合は，"*"1 個に置き換えて出力します。

```
yy....yy（zz....zz），..〈省略〉..，yy....yy（zz....zz）
```

yy....yy：先頭の"PD"を省略した，クライアント環境定義の名称です。次のクライアント環境定義が出力対象となっています。

項番	クライアント環境定義
1	PDUSER
2	PDNAMEPORT
3	PDCWAITTIME
4	PDSWAITTIME
5	PDHOST
6	PDFESHOST
7	PDSERVICEGRP
8	PDSWATCHTIME
9	PDSERVICEPORT
10	PDSRVTYPE
11	PDCLTRCVPORT
12	PDCLTRCVADDR
13	PDFESGRP

zz....zz：クライアント環境定義の内容です。なお，PDUSER のパスワード部分は出力しません。

cc....cc：例外オブジェクトが持つメッセージです。

dd....dd：SQLCODE のエラーコード（XAException の場合，XAException オブジェクトのフィールド errorCode が示すエラーコード）です（最大 11 文字）。

投入した例外オブジェクトの実行クラスが次のクラス又はサブクラスの場合に出力します。

- SQLException
- XAException

eeeee：SQLSTATE を出力します（5 文字）。

投入した例外オブジェクトの実行クラスが SQLException，又は SQLException のサブクラスの場合に出力します。

ff....ff：この例外が発生するまでに正常に実行されたバッチ更新の，各更新文の更新行数を出力します（最大 11 文字）。

例外オブジェクトの実行クラスが BatchUpdateException の場合に出力します。

更新行数が取得できない場合，"*"を出力します。

gg....gg：SQL カウンタを出力します（最大 6 文字）。

SQL トレース機能によって出力したトレース情報との対応付けに使用できます。

SQL カウンタが取得できない場合，"*"を出力します。

hh....hh：HiRDB サーバでエラーが発生している場合，HiRDB サーバの障害情報を出力します（最大 22 文字）。

障害情報は，保守員が使用します。

HiRDB サーバでエラーが発生していない場合，"*"を出力します。

iiii：HiRDB サーバでエラーが発生している場合，JDBC ドライバが HiRDB サーバに対して行った要求の種別（オペレーションコード）を出力します。

HiRDB サーバでエラーが発生していない場合，"****"を出力します。

jj....jj：例外投入メソッドを基点としたスタックトレースを出力します。

形式は Java 仮想マシンによって決まります。

kk....kk：トラブルシュート用の付加情報を，メッセージの形式で出力します。

ll....ll：例外が発生したオブジェクトの接続で使用しているソケット情報を，次の形式で出力します。

```
aa....aa(bb....bb), ....省略...., aa....aa(bb....bb)
```

ソケット情報の内容を次に示します。

ソケット情報の名称（aa....aa）	ソケット情報の内容（bb....bb）
LocalAddr	ローカルの IP アドレス
LocalPort	ローカルのポート番号
RemoteAddr	リモートの IP アドレス
RemotePort	リモートのポート番号
SendBufferSize	送信バッファサイズ
RecvBufferSize	受信バッファサイズ
SoLinger	SO_LINGER
KeepAlive	SO_KEEPALIVE
ReuseAddr	SO_REUSEADDR

注　内容が設定されていないソケット情報については，空括弧( )を出力します。また，ソケット情報が一つも設定されていない場合は，ll....ll の項目を出力しません。

mm....mm：拡張 SQL エラー情報を出力します（JDBC4.0 以降の場合だけです）。

出力内容については，「拡張 SQL エラー情報出力機能」を参照してください。拡張 SQL エラー情報出力の設定がされていない場合は，何も出力しません。

nn....nn：DriverManeger の getConnection メソッドで指定した URL 文字列を出力します。

URL で PASSWORD を指定している場合，PASSWORD 指定値を"*"1 個に置き換えて出力します。

DataSource を使用して取得した Connection の場合は何も出力しません。

oo....oo：Cosminexus を使用している場合，ルートアプリケーション情報を出力します。

ルートアプリケーション情報を取得できない（当該製品ではない，又はルートアプリケーション情報の適用範囲外から呼び出された）場合，何も出力しません。

ルートアプリケーション情報については，当該製品のマニュアルを参照してください。

pp....pp：JDBC4.0 以降の場合だけ出力します。Type4 JDBC ドライバ固有の各種設定項目（プロパティ，URL 項目，及び DataSource の setXXX メソッド）について，明示的に指定されたものを次の形式で出力します。

```
[a....a]b....b=c....c
[a....a]b....b=c....c
       :
[a....a]b....b=c....c
```

各要素の意味を次に示します。

aa....aa：次に示す設定方法

設定方法	出力文字列
システムプロパティ	SPR
ユーザプロパティ	UPR
URL	URL
DataSource 又は DriverManeger の setXXX メソッド	MTD
クライアント環境定義	ENV

b....b：設定項目名

c....c：設定値

一つの項目を複数の方法で設定した場合，それぞれについて設定情報を出力します。

17.19.3　出力例と解析方法

(1)　出力例

Exception トレースログの出力例を次に示します。

```
[1] HiRDB_Type4_JDBC40_Driver 09-50
2006/07/06 23:07:09.129 Thread[main,5,main]@1259414@1:[E][PrdbConnection@82c01f.createStatem
ent(1)]
                        ConnectionID(sds:23:20484) : SID(0)
2006/07/06 23:07:09.160 Thread[main,5,main]@1259414@1:[R][PrdbConnection@82c01f.createStatem
ent(1)]
                        ConnectionID(sds:23:20484) : SID(0)
                        Return=JP.co.Hitachi.soft.HiRDB.JDBC.PrdbStatement@1e4cbc4
2006/07/06 23:07:09.160 Thread[main,5,main]@1259414@1:[E][PrdbStatement@1e4cbc4.execute]
                        ConnectionID(sds:23:20484) : SID(0)
                        sql=DELETE FROM SEINO_TABLE
2006/07/06 23:07:14.285 Thread[main,5,main]@1259414@1:[E][PrdbConnection@82c01f.commit]
                        ConnectionID(sds:23:20484) : SID(0)
2006/07/06 23:07:14.301 Thread[main,5,main]@1259414@1:[R][PrdbConnection@82c01f.commit]
                        ConnectionID(sds:23:20484) : SID(0)
2006/07/06 23:07:14.301 Thread[main,5,main]@1259414@1:[R][PrdbStatement@1e4cbc4.execute]
                        ConnectionID(sds:23:20484) : SID(1)
                        Return=false
2006/07/06 23:07:14.301 Thread[main,5,main]@1259414@1:[E][PrdbConnection@82c01f.prepareState
ment(1)]
                        ConnectionID(sds:23:20484) : SID(0)
                        sql=INSERT INTO SEINO_TABLE VALUES(?, ?)
2006/07/06 23:07:14.348 Thread[main,5,main]@1259414@1:[R][PrdbConnection@82c01f.prepareState
ment(1)]
                        ConnectionID(sds:23:20484) : SID(0)
                        Return=JP.co.Hitachi.soft.HiRDB.JDBC.PrdbPreparedStatement@15d56d5
2006/07/06 23:07:26.567 Thread[main,5,main]@1259414@1:[E][PrdbConnection@82c01f.commit]
                        ConnectionID(sds:23:20484) : SID(0)
2006/07/06 23:07:26.567 Thread[main,5,main]@1259414@1:[R][PrdbConnection@82c01f.commit]
                        ConnectionID(sds:23:20484) : SID(0)
2006/07/06 23:07:26.567 Thread[main,5,main]@1259414@1:[E][PrdbStatement@1e4cbc4.executeQuery
]
                        ConnectionID(sds:23:20484) : SID(0)
                        sql=SELECT * FROM SEINO_TABLE
2006/07/06 23:07:26.676 Thread[main,5,main]@1259414@1:[R][PrdbStatement@1e4cbc4.executeQuery
]
                        ConnectionID(sds:23:20484) : SID(1)
                        Return=JP.co.Hitachi.soft.HiRDB.JDBC.PrdbResultSet@3eca90
2006/07/06 23:07:28.332 Thread[main,5,main]@1259414@1:[E][PrdbResultSet@3eca90.close]
                        ConnectionID(sds:23:20484) : SID(1)
2006/07/06 23:07:28.332 Thread[main,5,main]@1259414@1:[E][PrdbConnection@82c01f.commit]
                        ConnectionID(sds:23:20484) : SID(0)
2006/07/06 23:07:28.332 Thread[main,5,main]@1259414@1:[R][PrdbConnection@82c01f.commit]
                        ConnectionID(sds:23:20484) : SID(0)
2006/07/06 23:07:28.332 Thread[main,5,main]@1259414@1:[R][PrdbResultSet@3eca90.close]
                        ConnectionID(sds:23:20484) : SID(0)
2006/07/06 23:07:28.332 Thread[Thread-0,5,main]@30090737@2:[E][PrdbConnection@82c01f.prepare
Statement(1)]
                        ConnectionID(sds:23:20484) : SID(0)
                        sql=SELECT * FROM SEINO_TABLE
2006/07/06 23:07:28.332 Thread[Thread-0,5,main]@30090737@2:[R][PrdbConnection@82c01f.prepare
Statement(1)]
                        ConnectionID(sds:23:20484) : SID(0)
                        Return=JP.co.Hitachi.soft.HiRDB.JDBC.PrdbPreparedStatement@2808b3
2006/07/06 23:07:28.348 Thread[Thread-1,5,main]@5462872@3:[E][PrdbConnection@82c01f.prepareS
tatement(1)]
                        ConnectionID(sds:23:20484) : SID(0)
```

```
                        sql=DELETE FROM SEINO_TABLE WHERE I1=?
2006/07/06 23:07:28.358 Thread[Thread-1,5,main]@5462872@3:[E][PrdbConnection@82c01f.commit]
                        ConnectionID(sds:23:20484) : SID(0)
2006/07/06 23:07:29.672 Thread[Thread-1,5,main]@5462872@3:[R][PrdbConnection@82c01f.commit]
                        ConnectionID(sds:23:20484) : SID(0)
2006/07/06 23:07:30.098 Thread[Thread-1,5,main]@5462872@3:[R][PrdbConnection@82c01f.prepareS
tatement(1)]
                        ConnectionID(sds:23:20484) : SID(0)
                        Return=JP.co.Hitachi.soft.HiRDB.JDBC.PrdbPreparedStatement@922804
2006/07/06 23:07:30.332 Thread[Thread-2,5,main]@25253977@4:[E][PrdbConnection@82c01f.rollbac
k(1)]
                        ConnectionID(sds:23:20484) : SID(0)
2006/07/06 23:07:42.098 Thread[Thread-2,5,main]@25253977@4:[R][PrdbConnection@82c01f.rollbac
k(1)]
                        ConnectionID(sds:23:20484) : SID(0)
2006/07/06 23:07:42.098 Thread[Thread-2,5,main]@25253977@4:[E][PrdbConnection@82c01f.close]
                        ConnectionID(sds:23:20484) : SID(0)
2006/07/06 23:07:42.098 Thread[Thread-2,5,main]@25253977@4:[R][PrdbConnection@82c01f.close]
                        ConnectionID(sds:23:20484) : SID(0)
2006/07/06 23:07:42.535 Thread[Thread-1,5,main]@5462872:Exception:
ExceptionClass: SQLException
UapEnvironment: *
Message: KFPJ20006-E Connection closed[PrdbPreparedStatement.setInt]
ErrorCode: -1020006
SQLState: R2400
Etc.: *,*,****
URLInfo: jdbc:hitachi:hirdb://DBHOST=localhost,DBID=22200, ENCODELANG=UTF-8
RootAP : 127.0.0.1/1234/0x0000000000000001
PropertyInfo :
  [URL]ENCODELANG=UTF-8
java.sql.SQLException: KFPJ20006-E Connection closed[PrdbPreparedStatement.setInt]
at JP.co.Hitachi.soft.HiRDB.JDBC.JdbMakeException.generateSQLException(JdbMakeException.java
:31)
at JP.co.Hitachi.soft.HiRDB.JDBC.PrdbStatement.generateClosedSQLException(PrdbStatement.java
:3005)
at JP.co.Hitachi.soft.HiRDB.JDBC.PrdbPreparedStatement.setInt(PrdbPreparedStatement.java:117
0)
at Exception1.run(ExceptionTraceSample.java:57)
[2] HiRDB_Type4_JDBC_Driver 09-50
2006/07/06 23:07:25.863 Thread[Thread-5,5,main]@24431647@5:Warning:
UapEnvironment: USER(USER1), NAMEPORT(22200), CWAITTIME(0), SWAITTIME(600), HOST(localhost),
 FESHOST( ), SERVICEGRP(sds), SWATCHTIME( ), SERVICEPORT( ), SRVTYPE(WS), CLTRCVPORT( ), CLT
RCVADDR( ), FESGRP( )
Message: KFPZ02445-I Client information, Type=Reconnect,inf=AUTO_RECONNECT,number=5,interval
=5 [PrdbStatement.prepare]
Etc.: 4,sqapyac1.c(651),SET
URLInfo: jdbc:hitachi:hirdb://DBHOST=localhost,DBID=22200, ENCODELANG=UTF-8
RootAP : 127.0.0.1/1234/0x0000000000000001
PropertyInfo :
  [URL]ENCODELANG=UTF-8
```

(2) 解析方法

Exception トレースログの解析方法について説明します。Exception トレースログはテキストエディタなどで参照できます。

ここでは，「出力例」の Exception トレースログを解析する例を示します。

解析例

1. 調査対象の例外を含む通番の情報を抜き出します。
2. 情報を Thread 識別情報で分類し，スレッドごとに分割します。
3. 取得時刻によって，情報を時系列に並べます。
   次の表に示すようになります。

表 17–130　Exception トレースログを時系列に並べた例

日時	スレッド 1	スレッド 2	スレッド 3	スレッド 4
	Thread[main,5,main] @1259414@1	Thread[Thread-0,5,main] @30090737@2	Thread[Thread-1,5,main] @5462872@3	Thread[Thread-2,5,main] @25253977@4
2006/07/06 23:07:09.129	PrdbConnection @82c01f.createStatement(1)			
2006/07/06 23:07:09.160	PrdbStatement @1e4cbc4.execute			
2006/07/06 23:07:14.285	PrdbConnection @82c01f.commit			
2006/07/06 23:07:14.301	PrdbConnection @82c01f.prepareStatement(1)			
2006/07/06 23:07:26.567	PrdbConnection @82c01f.commit			
2006/07/06 23:07:26.567	PrdbStatement @1e4cbc4.executeQuery			
2006/07/06 23:07:26.567	PrdbStatement @1e4cbc4.execute			
2006/07/06 23:07:28.332	PrdbResultSet @3eca90.close	PrdbConnection @82c01f.prepareStatement(1)		
2006/07/06 23:07:28.332	PrdbConnection @82c01f.commit			
2006/07/06 23:07:28.348			PrdbConnection @82c01f.prepareStatement(1)	

日時	スレッド 1	スレッド 2	スレッド 3	スレッド 4
	Thread[main,5,main] @1259414@1	Thread[Thread-0,5,main] @30090737@2	Thread[Thread-1,5,main] @5462872@3	Thread[Thread-2,5,main] @25253977@4
2006/07/06 23:07:28.358			PrdbConnection @82c01f.commit	
2006/07/06 23:07:30.332				PrdbConnection @82c01f.rollback
2006/07/06 23:07:42.098				PrdbConnection @82c01f.close
2006/07/06 23:07:42.535			SQLException 発生 KFPJ20006-E Connection closed	

4. Exception のエラーの内容を確認します。

   2006/07/06 23:07:42.535 のスレッド 3 で SQLException が発生し，Statement オブジェクト又は Connection オブジェクトが既にクローズされていることが分かります。

5. 時系列でオブジェクトの操作を確認します。

   次のスレッドの Connection オブジェクトのオブジェクト ID が同じであることから，四つのスレッドが同一のコネクションで処理されていることが分かります。

   - 2006/07/06 23:07:09.129 のスレッド 1
   - 2006/07/06 23:07:28.332 のスレッド 2
   - 2006/07/06 23:07:28.348 のスレッド 3
   - 2006/07/06 23:07:30.332 のスレッド 4

6. エラーの原因である箇所を探します。

   四つのスレッドが同一のコネクションであることは分かったので，Statement.close メソッド又は Connection.close メソッドを実行している箇所を探すと，スレッド 4 が 2006/07/06 23:07:42.098 で Connection.close メソッドを実行していることが分かります。このことから，2006/07/06 23:07:42.535 のスレッド 3 で発生した SQLException の原因は，スレッド 4 が 2006/07/06 23:07:42.098 で Connection.close メソッドを実行したことであると分かります。

17.19.4　必要となるメモリ所要量及びファイルサイズ

(1)　メモリ所要量

Exception トレースログを取得するためのメモリ所要量は，次に示す計算式で求められます。

計算式

```
↑360×n÷1024↑　　（単位：キロバイト）
```

変数の説明

n：メモリ内取得情報数

(2) 必要となるファイルサイズ

Exception トレースログのファイルサイズの上限はクライアント環境定義 PDJDBFILESIZE，又はシステムプロパティ HiRDB_for_Java_FileSize で指定したサイズ（単位：バイト）となります。これらのクライアント環境定義，及びシステムプロパティを指定していない場合，Exception トレースログを取得するためのファイルサイズの概算は，次に示す計算式で求められます。

計算式

```
↑（180×n×m＋a×m）÷1024↑＋1　　（単位：キロバイト）
```

変数の説明

a：拡張 SQL エラー情報長

241 ＋ SQL 文長＋ b

拡張 SQL エラー情報長は次のどちらかに該当する場合に加算します。

- クライアント環境定義 PDUAPEXERLOGUSE に YES を指定した場合
- クライアント環境定義 PDUAPEXERLOGUSE を省略し，かつシステム定義 pd_uap_exerror_log_use が YES，又は AUTO の場合

b：パラメタ情報長

$$\sum_{i=1}^{d} (41 + \uparrow c \div 16 \uparrow \times 83)$$

パラメタ情報長は，クライアント環境定義 PDUAPEXERLOGPRMSZ に 0 以外，又はシステム定義 pd_uap_exerror_log_param_size に 0 以外を指定した場合に加算してください。

c：パラメタ情報の最大データ長

クライアント環境定義 PDUAPEXERLOGPRMSZ，又はシステム定義 pd_uap_exerror_log_param_size の値になります。

c の値は次のように決定してください。

- クライアント環境定義 PDUAPEXERLOGPRMSZ に 0 以外を指定した場合，PDUAPEXERLOGPRMSZ に指定した値が c の値になります。
- クライアント環境定義 PDUAPEXERLOGPRMSZ を省略し，かつシステム定義 pd_uap_exerror_log_param_size に 0 以外を指定した場合，pd_uap_exerror_log_param_size に指定した値が c の値になります。

d：入出力パラメタ数

n：メモリ内取得情報数※

m：ファイル出力情報数※

注※　詳細については，「Exception トレースログを取得するためのシステムプロパティの設定」を参照してください。

17.19.5　注意事項

(1)　システムプロパティとクライアント環境定義の設定が異なる場合

システムプロパティとクライアント環境定義の設定が異なる場合，最初の HiRDB 接続確立前に JDBC ドライバのメモリに蓄積したメソッド実行履歴を，次の表に示すように移行します。

表 17–131　JDBC ドライバのメモリに蓄積したメソッド実行履歴の移行

項目	システムプロパティとクライアント環境定義の関係	移行時の動作
メモリ内取得情報数	HiRDB_for_Java_OnMemNUM < PDJDBONMEMNUM	PDJDBONMEMNUM の指定に基づいてメソッド実行履歴の蓄積用メモリを再確保し，それまでに蓄積した実行履歴を再確保した領域にコピーします。 ただし，PDJDBTRACELEVEL の指定が 0 の場合はメモリの再確保はしません。
	HiRDB_for_Java_OnMemNUM > PDJDBONMEMNUM	PDJDBONMEMNUM の指定に基づいてメソッド実行履歴の蓄積用メモリを再確保します。それまでに蓄積した実行履歴のうち，再確保した領域に格納できない情報を破棄して，再確保領域にコピーします。 ただし，PDJDBTRACELEVEL の指定が 0 の場合はメモリの再確保はしません。
トレース取得レベル	HiRDB_for_Java_TraceLevel < PDJDBTRACELEVEL	それまでに蓄積した実行履歴をそのまま移行します。
	HiRDB_for_Java_TraceLevel > PDJDBTRACELEVEL	PDJDBTRACELEVEL の指定が 1 以上の場合，それまでに蓄積した実行履歴をそのまま移行します。PDJDBTRACELEVEL で指定したトレース取得レベルでは取得対象外になっているメソッドの実行履歴についても，そのまま移行します。 PDJDBTRACELEVEL の指定が 0 の場合は，それまでに蓄積した実行履歴を蓄積用メモリごと破棄します。

(2)　Java 仮想マシン起動後の最初の出力

Java 仮想マシン起動後，最初にファイルに Exception トレースログを出力する場合は，更新日時が古い方のファイルに出力されます。同じ場合は，pdexc1.trc に出力されます。

(3)　ファイル出力先の指定

複数のプロセスから Exception トレースログを取得する場合，同じファイル出力先を指定していると，同一ファイルに異なるプロセスのトレースが出力されます。プロセスごとにトレースを取得する場合は，プロセスごとに異なるファイル出力先を指定してください。

JDBC ドライバは，Java 仮想マシンの機能を通じて OS が提供するファイルシステム上にログファイルを作成します。そのため，次の点については，使用する Java 仮想マシン及びファイルシステムに依存します。

- 絶対パス名の接頭辞
- パスの区切り文字
- 出力先ファイル（絶対パス）の最大文字数
- 1 ファイル当たりのサイズ

(4)　エラー発生時の処理

ファイルの作成や出力に失敗しても，Exception トレースログには情報が出力されません。また，エラーメッセージが UAP に返されたり，ファイル出力をリトライすることもありません。

(5)　文字コード

Exception トレースログは，使用する Java 仮想マシンのデフォルトの変換文字セットで出力されます。

17.20　不正電文トレース

サーバから不正電文を受信した場合に，不正電文トレースを取得します。

次のどれかのエラーが発生した場合，不正電文トレースファイルを出力することがあります。ただし，XDM/RD E2 接続機能使用時にエラーが発生した場合，不正電文トレースは出力されません。

- KFPA11720-E
- KFPA11722-E（reason=INVALID DATA）
- KFPA11723-E（reason=HiRDB DATA ERROR）
- KFPA11733-E
- KFPA11735-E

17.20.1　不正電文トレースを取得するための設定

不正電文トレースのファイル出力先，ファイルへの出力数，ファイル出力電文長は，システムプロパティ又はクライアント環境定義で設定します。優先順位は次のとおりです。

1. システムプロパティ
2. クライアント環境定義

JavaVM 起動後の最初の接続が確立するまでは，クライアント環境定義の指定は有効となりません。

(1)　クライアント環境定義の設定

クライアント環境定義では，ファイル出力先として次の項目を指定します。

- PDJDBFILEDIR
- PDCLTPATH

PDJDBFILEDIR と PDCLTPATH を共に指定した場合，PDJDBFILEDIR の指定が有効となります。

接続ごとに異なる値を指定した場合，最初の接続に対する指定が有効となります。

指定値の詳細は，「クライアント環境定義の設定内容」を参照してください。

(2)　システムプロパティの設定

システムプロパティでは，次の表に示す項目を指定します。

表 17–132　不正電文トレースを取得するためのシステムプロパティの設定

項目	システムプロパティ	内容	デフォルト
ファイル出力先	HiRDB_for_Java_FileDIR	不正電文トレースを出力するディレクトリを，絶対パスで指定します。不正電文トレースは，指定したディレクトリの直下に出力されます。 指定したファイル出力先が存在しない，又は書き込み権限がない場合は，カレントディレクトリに出力されます。	カレントディレクトリ
ファイルへの出力数	HiRDB_for_Java_DataErrOutNum	1 ファイルに出力する不正電文の個数の上限を指定します。指定できる範囲は 50～2147483647 です。範囲外の値を指定した場合，デフォルトが仮定されます。 また，上限値を超えてファイルに出力する場合は，2 ファイルでラップアラウンドします。	50
ファイル出力電文長	HiRDB_for_Java_DataErrSize	出力する送信電文，受信電文の最大長(バイト)を指定します。指定できる範囲は 1024～2147483647 です。範囲外の値を指定した場合，デフォルトが仮定されます。 電文のこの指定値を超える部分については，不正電文トレースに出力しません。	1024

Java 仮想マシンが，セキュリティマネジャによって JDBC ドライバに対するシステムプロパティの受け渡しを拒否した場合，デフォルトが仮定されます。

17.20.2　ファイル名

不正電文トレースファイルのファイル名は pdjdataerr1.trc 又は pdjdataerr2.trc となります。

17.20.3　必要となるファイルサイズ

不正電文トレースファイルの１ファイル当たりの最大サイズは，次に示す計算式で求められます。

計算式

```
((↑len÷16↑×76)×2+1024)×num　　（単位：バイト）
```

変数の説明

len：システムプロパティ HiRDB_for_Java_DataErrSize

num：システムプロパティ HiRDB_for_Java_DataErrOutNum

17.21　JDBC ドライバを使用した UAP 例

JDBC ドライバを使用した UAP 例を次に示します。

指定した条件と一致する行を表示させる SQL 文を実行します。SQL 文のエラー判定をして，エラーがある場合はエラー情報を取得します。

```
import java.sql.*;
import java.io.*;
import java.util.Properties;

public class SAMPLE {

    public static void main(String[] args) {

        //接続オブジェクト変数
        String url = "jdbc:hitachi:hirdb://DBID=22200,DBHOST=host1";
        String user = "USER1";
        String passwd = "USER1";
        String driver = "JP.co.Hitachi.soft.HiRDB.JDBC.HiRDBDriver";
        try {
            Class.forName(driver);
        }catch(Exception e){e.printStackTrace();return;}

        Connection con = null;
        PreparedStatement pstmt = null;
        ResultSet rs = null;

        // *** 環境変数 ***
        java.util.Properties info = new java.util.Properties();
        info.setProperty("user",user);
        info.setProperty("password",passwd);
        info.setProperty("HiRDB_for_Java_ENV_VARIABLES",
                    "PDCLTPATH=C:¥¥tmp;PDSQLTRACE=0;PDPRMTRC=INOUT;");
        // ************

        //Driverの登録及びロード
        System.setProperty("jdbc.drivers",
                           "JP.co.Hitachi.soft.HiRDB.JDBC.HiRDBDriver");

        //環境構築
        try {
            // 接続 ..................................................1
            con = DriverManager.getConnection(url,info);

            // ********************************************
            // SAMPLE1(C1 INT,C2 INT,C3 VARCHAR(30))の検索例 .........2
            // ********************************************
            // PreparedStatement取得
            pstmt = con.prepareStatement
                           ("SELECT C2,C3 FROM SAMPLE1 WHERE C1 = ? ");
            // ?パラメタの設定(C1=200)
            pstmt.setInt(1,200);
            // ResultSet取得
            rs = pstmt.executeQuery();
```

```
            int cnt=1;
            System.out.println("**** 検索実行 ****");
            while(rs.next()){
                System.out.println("**** "+cnt+"行目検索 ***");
                // C2データの取得&表示
                int i_data = rs.getInt(1);
                System.out.println("C2="+i_data);
                // C3データの取得&表示
                String c_data = rs.getString(2);
                System.out.println("C3="+c_data);
                cnt++;
            }
            // ResultSetの解放
            rs.close();
            // PreparedStatementの解放
            pstmt.close();
            // 切断  .....................................................3
            con.close();
            }

        catch(SQLException e){  ....................................4
            // エラー情報出力
            e.printStackTrace();
            // 個別情報は以下の処理で取得
            // SQLSTATEの出力
            System.out.println("SQLSTATE=" + e.getSQLState());
            // SQLCODEの出力
            System.out.println("SQLCODE=" + e.getErrorCode());
            // SQLERRM(SQLメッセージ)の出力
            System.out.println("SQLERRM=" + e.getMessage());
            return;
        }
    }
}
```

[説明]

1. getConnection メソッドを使用して HiRDB へ接続します。
2. 指定した条件と一致する行を表示させる SQL 文を実行します。
3. close メソッドを使用して HiRDB から切断します。
4. エラーになった場合は SQLException を返し，エラー情報を出力します。

17.22　JDBC ドライバを使用した場合のメモリの容量見積もり

JDBC ドライバが使用する Java ヒープの容量見積もりについて説明します。

17.22.1　Connection オブジェクトの容量見積もり

Connection オブジェクトの容量見積もりの計算式を次に示します。

計算式（32 ビットモードの場合）

```
Connectionオブジェクトの容量＝
  70000＋（11000＋送信バッファ長＋受信バッファ長＋不正電文トレース出力電文長）×Connectio
nインスタンス数
     （単位：バイト）
```

計算式（64 ビットモードの場合）

```
Connectionオブジェクトの容量＝
  304000＋（14000＋送信バッファ長＋受信バッファ長＋不正電文トレース出力電文長）×Connecti
onインスタンス数
     （単位：バイト）
```

(1)　送信バッファ長

送信バッファ長は，次に示す計算式から求めてください。複数の SQL 文を実行する場合，SQL 文ごとに送信バッファ長を求め，最大値を使用します。

```
送信バッファ長＝
  MAX（A,
      ↑（240＋SQL文長＋ユーザ付加情報長）÷4096↑×4096,
      ↑（240＋入力パラメタ情報長＋出力パラメタ情報長
      ＋ユーザ付加情報長）÷4096↑×4096）
```

A：32768

SQL 文長：SQL 文（String）を HiRDB サーバの文字コードに変換した長さ

ユーザ付加情報長：ユーザ付加情報 1，2，3 の長さの合計

Connection.setHiRDB_Audit_Info を使用する場合に加算してください。

ユーザ付加情報 1：次に示す計算式から求めてください。

16 ＋ユーザ付加情報 1（String）を HiRDB サーバの文字コードに変換した長さ

ユーザ付加情報 2：次に示す計算式から求めてください。

16 ＋ユーザ付加情報 2（String）を HiRDB サーバの文字コードに変換した長さ

ユーザ付加情報 3：次に示す計算式から求めてください。

16 ＋ユーザ付加情報 3（String）を HiRDB サーバの文字コードに変換した長さ

入力パラメタ情報長：入力パラメタがある場合に次に示す計算式から求めて加算してください。

$$24 + \sum_{i=1}^{\text{入力パラメタ数}} (16 + \text{入力データ長}_i)$$

入力パラメタ数：次に示します。

オブジェクト	入力パラメタ数
PreparedStatement	?パラメタ数
CallableStatement	IN パラメタ及び INOUT パラメタ数

入力データ長：setXXX メソッドで設定したデータの実長

setString メソッドで設定した場合は HiRDB サーバの文字コードに変換した長さです。

出力パラメタ情報長：検索系 SQL の実行，又はストアドプロシジャを実行して OUT パラメタ又は INOUT パラメタ値を取得する場合に次に示す計算式から求めて加算してください。

24 ＋ 16×出力パラメタ数

出力パラメタ数：検索系 SQL を実行する場合は検索列数です。ストアドプロシジャを実行する場合は OUT パラメタ及び INOUT パラメタの合計数です。

(2)　受信バッファ長

受信バッファ長は，次に示す計算式から求めてください。複数の SQL 文を実行する場合，SQL 文ごとに受信バッファ長を求め，最大値を使用します。

```
受信バッファ長＝
  MAX（A,
      ↑（240＋52×（入力パラメタ数＋出力パラメタ数））÷4096↑×4096,
      （↑（240＋位置付け子機能によるアクセス時の出力データ長）÷4096↑×4096）※）
```

注※

次のどちらかの指定によって，位置付け子機能を使用してアクセスする場合に求めてください。

- DriverManager.getConnection の引数の url，又はユーザプロパティの LONGVARBINARY_ACCESS で LOCATOR を指定
- DataSource 系インタフェースの setLONGVARBINARY_Access メソッドで LOCATOR を指定

A：32768

入力パラメタ数：次に示します。

オブジェクト	入力パラメタ数
Statement	0
PreparedStatement	?パラメタ数
CallableStatement	IN パラメタ及び INOUT パラメタ数

出力パラメタ数：求め方は，送信バッファ長の出力パラメタ数と同じです。

位置付け子機能によるアクセス時の出力データ長：次のどちらかの値となります。

- HiRDB_for_Java_LONGVARBINARY_ACCESS_SIZE に 0 以外を指定している場合
  次に示す計算式から求めてください。
  HiRDB_for_Java_LONGVARBINARY_ACCESS_SIZE 指定値×1024
- HiRDB_for_Java_LONGVARBINARY_ACCESS_SIZE に 0 を指定，又は省略している場合
  BLOB 又は BINARY 型の列に格納されているデータの実長

(3)　不正電文トレース出力電文長

HiRDB_for_Java_DataErrSize 指定値

詳細については「不正電文トレースを取得するためのシステムプロパティの設定」を参照してください。

17.22.2　Statement オブジェクトの容量見積もり

Statement オブジェクトの容量見積もりの計算式を次に示します。

計算式（32 ビットモードの場合）

```
Statementオブジェクトの容量＝
  38000＋（2000＋検索結果格納領域長）×Statementインスタンス数　（単位：バイト）
```

計算式（64 ビットモードの場合）

```
Statementオブジェクトの容量＝
  38000＋（2100＋検索結果格納領域長）×Statementインスタンス数　（単位：バイト）
```

検索結果格納領域長：検索系 SQL でない場合は 0 となります。

検索系 SQL の場合は，次に示す計算式から求めてください。

$$\sum_{i=1}^{\text{検索列数}} (120 + \text{検索列のデータの定義長}_i)$$

検索列のデータの定義長：検索列が BLOB 又は BINARY の場合で，位置付け子機能を使用しているときは 4 となります。位置付け子機能を使用していないときは，次のどちらかの値となります。

- HiRDB_for_Java_MAXBINARYSIZE を指定していない場合
  8 ＋定義長
- HiRDB_for_Java_MAXBINARYSIZE を指定している場合
  MIN（8 ＋定義長，8 ＋ MAXBINARYSIZE 指定値）

17.22.3　PreparedStatement オブジェクトの容量見積もり

PreparedStatement オブジェクトの容量見積もりの計算式を次に示します。

計算式（32 ビットモードの場合）

```
PreparedStatementオブジェクトの容量＝
  39000＋（2100＋入力データ格納領域長＋検索結果格納領域長）
    ×PreparedStatementインスタンス数　（単位：バイト）
```

計算式（64 ビットモードの場合）

```
PreparedStatementオブジェクトの容量＝
  40000＋（2600＋入力データ格納領域長＋検索結果格納領域長）
    ×PreparedStatementインスタンス数　（単位：バイト）
```

入力データ格納領域長：?パラメタがある場合，次に示す計算式から求めてください。

$$\sum_{i=1}^{\text{?パラメタ数}} (120 + \text{入力データ長}_i)$$

検索結果格納領域長の求め方は，Statement オブジェクトの容量見積もりと同じです。

17.22.4　CallableStatement オブジェクトの容量見積もり

CallableStatement オブジェクトの容量見積もりの計算式を次に示します。

計算式（32 ビットモードの場合）

```
CallableStatementオブジェクトの容量＝
  39000＋（2100＋入力データ格納領域長＋出力データ格納領域長）
    ×CallableStatementインスタンス数　（単位：バイト）
```

計算式（64 ビットモードの場合）

```
CallableStatementオブジェクトの容量＝
  40000＋（2600＋入力データ格納領域長＋出力データ格納領域長）
    ×CallableStatementインスタンス数　（単位：バイト）
```

入力データ格納領域長：IN パラメタ及び INOUT パラメタがある場合，次に示す計算式から求めてください。

$$\sum_{i=1}^{\text{INパラメタ及びINOUTパラメタ数}} (120 + \text{INパラメタ及びINOUTパラメタのデータ長}_i)$$

IN パラメタ及び INOUT パラメタのデータ長の求め方は，Connection オブジェクトの容量見積もりの入力パラメタ情報長の求め方と同じです。

出力データ格納領域長：INOUT パラメタ及び OUT パラメタがある場合，次に示す計算式から求めてください。

```
INOUTパラメタ及び
OUTパラメタ数
  Σ   （120＋INOUTパラメタ及びOUTパラメタのデータの定義長i）
 i=1
```

INOUT パラメタ及び OUT パラメタのデータの定義長：

INOUT パラメタ及び OUT パラメタが BLOB 又は BINARY の場合で，位置付け子機能を使用しているときは 4 となります。位置付け子機能を使用していないときは，次のどちらかの値となります。

- HiRDB_for_Java_MAXBINARYSIZE を指定していない場合

  8 ＋定義長

- HiRDB_for_Java_MAXBINARYSIZE を指定している場合

  MIN（8 ＋定義長，8 ＋ HiRDB_for_Java_MAXBINARYSIZE 指定値）

17.22.5　ResultSet オブジェクトの容量見積もり

ResultSet オブジェクトの容量見積もりの計算式を次に示します。

計算式（32 ビットモードの場合）

```
ResultSetオブジェクトの容量＝
  16000＋（900＋ロケータアクセスオブジェクト長＋受信バッファ長
    ＋一時的作業領域長））×ResultSetインスタンス数
    ＋検索結果蓄積領域長×ResultSetインスタンス数2
      （単位：バイト）
```

計算式（64 ビットモードの場合）

```
ResultSetオブジェクトの容量＝
  17000＋（1300＋ロケータアクセスオブジェクト長＋受信バッファ長
    ＋一時的作業領域長））×ResultSetインスタンス数
    ＋検索結果蓄積領域長×ResultSetインスタンス数2
      （単位：バイト）
```

ロケータアクセスオブジェクト長：

位置付け子機能を使用する場合（DriverManager.getConnection の引数の url 又はユーザプロパティの LONGVARBINARY_ACCESS で LOCATOR を指定，又は DataSource 系インタフェースの setLONGVARBINARY_Access メソッドで LOCATOR を指定），1,600 を加算してください。

受信バッファ長：

次に示す計算式から求めてください。

```
MAX（A,
      ↑（240＋検索行データ長×ブロック転送の行数）÷4096↑×4096,
     Connectionオブジェクトの容量見積もりの受信バッファ長）
```

A：32768

検索行データ長：

次に示す計算式から求めてください。

$$8 + \sum_{i=1}^{\text{検索列数}} (\text{列データの定義長}_i)$$

列データの定義長：

BLOB 又は BINARY の場合で，ロケータを使用しているときは 4 となります。ロケータを使用していないときは，次のどちらかの値となります。

- HiRDB_for_Java_MAXBINARYSIZE を指定していない場合

  8 ＋定義長

- HiRDB_for_Java_MAXBINARYSIZE を指定している場合

  MIN（8 ＋定義長，8 ＋ HiRDB_for_Java_MAXBINARYSIZE 指定値）

ブロック転送の行数：

ブロック転送機能を使用しない場合は 1 となります。

一時的作業領域長：

次に示す計算式から求めてください。

一時的作業領域長＝検索列の「列データの実長2」の最大値
　＋検索列の「列データの実長3」の最大値

列データの実長 2：

次の表に示します。

出力データ属性	列データの実長 2
INTEGER	4
SMALLINT	2
DECIMAL（m, n）	**↓m÷2↓＋1**
FLOAT	8
SMALLFLT	4
上記以外	実長

列データの実長 3：

位置付け子機能を使用し BLOB 又は BINARY 型列を検索した場合，次のどちらかの値となります。

- HiRDB_for_Java_LONGVARBINARY_ACCESS_SIZE に 0 以外を指定している場合

  HiRDB_for_Java_LONGVARBINARY_ACCESS_SIZE 指定値×1024

- HiRDB_for_Java_LONGVARBINARY_ACCESS_SIZE に 0 を指定，又は省略している場合

  BLOB 又は BINARY 型の列に格納されているデータの実長

上記以外の場合は 0 となります。

検索結果蓄積領域長：

次に示す計算式から求めてください。

$$\sum_{i=1}^{\text{検索列数}} (100 + \text{列データの実長}_i) \times \text{検索行数}$$

列データの実長：

次に示します。

出力データ属性	列データの実長
INTEGER	4
SMALLINT	2
DECIMAL(m, n)	**↓m÷2↓＋1**
FLOAT	8
SMALLFLT	4
BLOB 又は BINARY	位置付け子機能を使用しているときは 4 となります。位置付け子機能を使用していないときは，次のどちらかの値となります。 • HiRDB_for_Java_MAXBINARYSIZE を指定していない場合 **8 ＋実長** • HiRDB_for_Java_MAXBINARYSIZE を指定している場合 **MIN（8 ＋実長，8 ＋ MAXBINARYSIZE 指定値）**
上記以外	実長

ResultSet インスタンス数 2：

結果集合の型が ResultSet.TYPE_SCROLL_INSENSITIVE，又は ResultSet.TYPE_SCROLL_SENSITIVE である ResultSet インスタンスの数

17.22.6　トレースオブジェクトの容量見積もり

トレースオブジェクトの総容量は，次に示すトラブルシュート情報を取得するためのメモリの容量の合計値です。

- インタフェースメソッドトレース
- Exception トレースログ
- SQL トレース

(1)　インタフェースメソッドトレース

インタフェースメソッドトレースを取得する場合に加算してください。

計算式

```
インタフェースメソッドトレースの容量＝300×エントリ数×Connectionインスタンス数
  （単位：バイト）
```

エントリ数：次のどちらかの値です。

- DriverManager.getConnection の引数の url，又はユーザプロパティの TRC_NO の指定値
- DataSource 系インタフェースの setTRC_NO メソッドの指定値

(2) Exception トレースログの容量見積もり

Exception トレースログのトレース取得レベルが 0 以外の場合に加算してください。

計算式

```
Exceptionトレースログの容量＝360×メモリ内取得情報数　（単位：バイト）
```

メモリ内取得情報数：次のどちらかの値です。

- システムプロパティ HiRDB_for_Java_OnMemNUM の指定値
- クライアント環境定義 PDJDBONMEMNUM の指定値

(3) SQL トレースの容量見積もり

SQL トレースを取得する場合に加算してください。

計算式

```
SQLトレースの容量＝（10000＋SQLトレース情報退避オブジェクト長）×Connectionインスタンス
数　（単位：バイト）
```

SQL トレース情報退避オブジェクト長：PDSQLTRCTYPE=ERROR 指定でない場合は 0 となります。

PDSQLTRCTYPE=ERROR 指定の場合は，次に示す計算式から求めてください。

計算式

```
1000
  ＋（200＋ルートアプリケーション情報退避領域長）×PDJDBSQLTRCMEMNUM指定値
  ＋トランザクション識別子退避領域長
  ＋自動再接続で使用したソケット情報退避領域長
  ＋パラメタトレース退避領域長
```

ルートアプリケーション情報退避領域長：

- Cosminexus から Type4 JDBC ドライバを使用している場合

  50

- Cosminexus から Type4 JDBC ドライバを使用していない場合

  0

トランザクション識別子退避領域長：

- XA インタフェースを使用している場合

  160×PDJDBSQLTRCMEMNUM 指定値内でトランザクション識別子を退避する回数

- XA インタフェースを使用していない場合

  0

トランザクション識別子は，「SQL トレース機能」の「SQL トレース情報の見方」に記載のトランザクション識別子を表示する条件の場合に退避します。

自動再接続で使用したソケット情報退避領域長：

- 自動再接続機能を使用している場合

  600×PDJDBSQLTRCMEMNUM 指定値内で自動再接続を実行する回数

- 自動再接続機能を使用していない場合

  0

パラメタトレース退避領域長：

パラメタトレースを出力する場合，PDJDBSQLTRCMEMNUM 指定値内で実行する SQL の，入力パラメタ情報長，出力パラメタ情報長及び検索データ情報長の総和を加算してください。

計算式

$$\sum_{i=1}^{n} \left(\text{入力パラメタ情報長}i + \text{出力パラメタ情報長及び検索データ情報長}i \right)$$

n：PDJDBSQLTRCMEMNUM 指定値

入力パラメタ情報長：

PDPRMTRC に IN 又は INOUT を指定の場合に次に示す計算式から求めてください。

$$140 + \sum_{i=1}^{\text{入力パラメタ数}} \left(130 + \text{入力データ長}i \right)$$

入力パラメタ数：次に示します。

オブジェクト	入力パラメタ数
PreparedStatement	?パラメタ数
CallableStatement	IN パラメタ及び INOUT パラメタ数

入力データ長：setXXX メソッドで設定したデータの実長

setString メソッドで設定した場合は HiRDB サーバの文字コードに変換した長さです。これらが PDPRMTRCSIZE 指定値を超える場合は，PDPRMTRCSIZE 指定値となります。

出力パラメタ情報長及び検索データ情報長：

PDPRMTRC に OUT 又は INOUT を指定の場合に次に示す計算式から求めてください。

データ数
140 ＋ Σ （130＋データの実長i ）
i=1

データ数：CallableStatementの場合，OUTパラメタ及びINOUTパラメタ数

検索の場合，検索列数

データの実長：

次に示す実長が PDPRMTRCSIZE 指定値を超える場合は，PDPRMTRCSIZE 指定値となります。

データ属性	データの実長
INTEGER	4
SMALLINT	2
DECIMAL(m, n)	↓m÷2↓＋1
FLOAT	8
SMALLFLT	4
BLOB 又は BINARY	• HiRDB_for_Java_MAXBINARYSIZE を指定していない場合 8＋実長 • HiRDB_for_Java_MAXBINARYSIZE を指定している場合 MIN（8＋実長，8＋MAXBINARYSIZE指定値）
上記以外	実長

18 SQLJ

この章では，SQLJ を使用して UAP を開発する方法について説明します。

SQLJ を使用する場合は，HiRDB サーバとの接続時にパスワードを指定するユーザインタフェースにおいて，29 バイト以上のパスワードを引用符で囲んで指定できません。

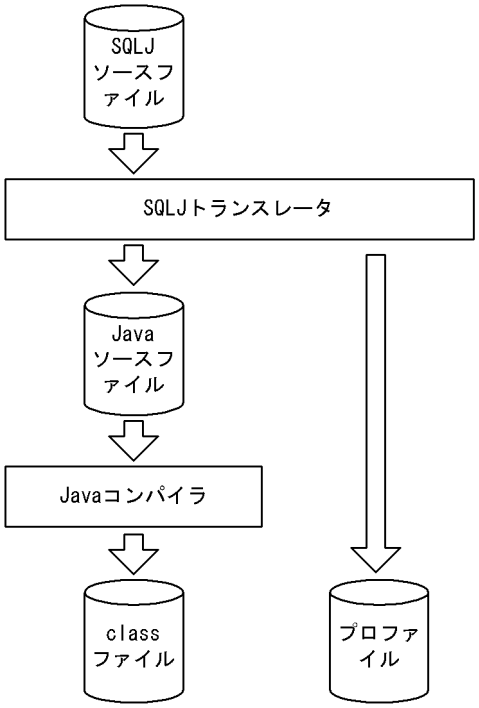
18.1 概要

18.1.1 SQLJ とは

SQLJ とは，Java で静的 SQL を埋込み型 SQL として記述，及び実行するための言語仕様です。

SQLJ を使用した UAP 開発の流れを次の図に示します。

図 18–1 SQLJ を使用した UAP 開発の流れ



SQLJ は，SQLJ トランスレータと SQLJ ランタイムライブラリから構成されます。

SQLJ トランスレータ：

SQLJ ソースプログラムを解析して，SQL 文を SQLJ ランタイムを通じてデータベースにアクセスする標準的な Java の命令に置き換えます。

SQLJ トランスレータは，Java ソースファイルと，SQL の情報を格納したプロファイルを生成します。ユーザは，Java ソースファイルを Java コンパイラでコンパイルして class ファイル（実行ファイル）を作成します。

SQLJ ランタイムライブラリ：

コンパイルされた class ファイルを実行するときに使用します。

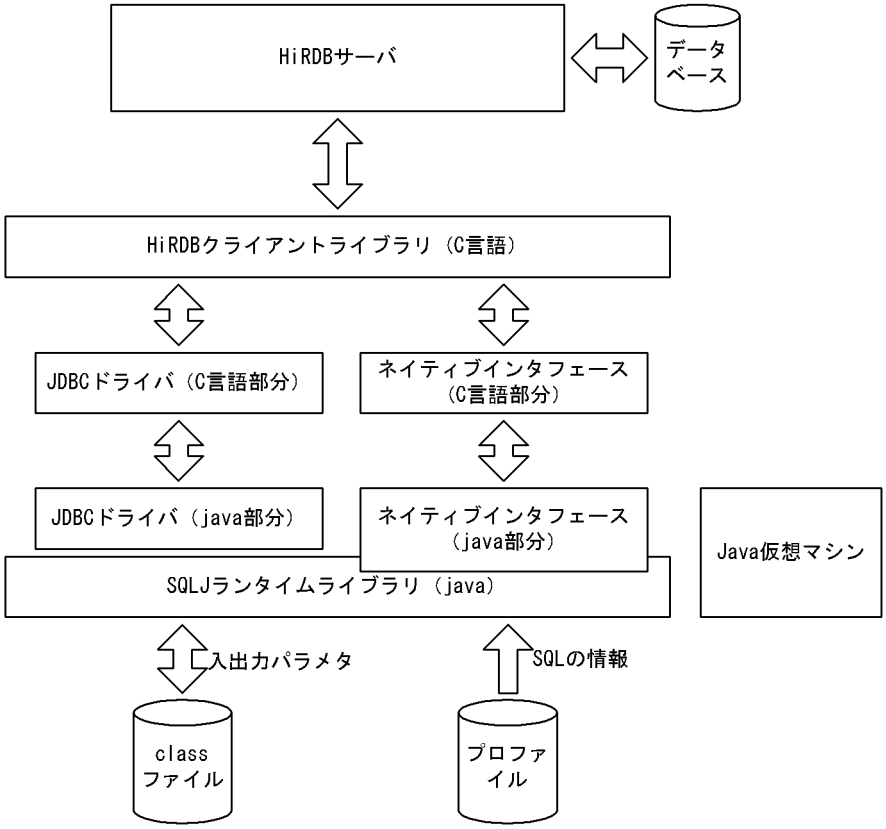
SQLJ ランタイムライブラリには，アクセスするインタフェースによって次のどちらかの使用方法があります。

- JDBC インタフェースの API を呼び出して SQL を実行する（スタンダードインタフェース版）。

- JDBC インタフェースを使用しないで，独自のインタフェースを呼び出して SQL を実行する（ネイティブインタフェース版）。

SQLJ を使用した UAP の実行を次の図に示します。

図 18–2　SQLJ を使用した UAP の実行



［説明］

- Java コンパイラでコンパイルした SQLJ ソースの class ファイルは，SQLJ ランタイムライブラリを通じてデータベースをアクセスします。
- ネイティブインタフェースを使用する場合は，SQLJ ランタイムライブラリは JDBC を呼び出さないで，HiRDB クライアントライブラリを直接呼び出します。このとき，JDBC の API を直接呼び出して，接続や結果セットを JDBC と共有するコーディングはできません。
- SQLJ ランタイムライブラリは，実行時にプロファイルを読み込みます。そのため，class ファイルとプロファイルは同じディレクトリに格納する必要があります。また，class ファイルを jar ファイルに格納する場合も，プロファイルを併せて jar ファイルに格納する必要があります。

18.1.2　環境設定

SQLJ が動作するときに必要となる，環境変数の設定を次に示します。なお，SQLJ は JDBC ドライバを使用するため，JDBC ドライバの環境設定もしておく必要があります。

(1)　UNIX 版での環境設定

実行環境の環境変数に，次の内容を設定してください。

(a)　HiRDB/Developer's Kit の場合

```
CLASSPATH=$CLASSPATH:[インストールディレクトリ]/client/lib/pdsqlj.jar
```

(b)　HiRDB/Run Time の場合

```
CLASSPATH=$CLASSPATH:[インストールディレクトリ]/client/lib/pdruntime.jar
CLASSPATH=$CLASSPATH:[インストールディレクトリ]/client/lib/pdnativert.jar
```

(2)　Windows 版での環境設定

［コントロールパネル］－［システム］－［システムのプロパティ］の「環境」に，次の内容を設定してください。

(a)　HiRDB/Developer's Kit の場合

```
CLASSPATH=%CLASSPATH%:[インストールディレクトリ]¥UTL¥pdsqlj.jar
```

(b)　HiRDB/Run Time の場合

```
CLASSPATH=%CLASSPATH%:[インストールディレクトリ]¥UTL¥pdruntime.jar
CLASSPATH=%CLASSPATH%:[インストールディレクトリ]¥UTL¥pdnativert.jar
```

18.2　SQLJ トランスレータ

SQLJ トランスレータは，SQLJ ソースプログラムを解析して，Java ソースファイルとプロファイルを生成します。

SQL 文は，JDBC の API 呼び出しを含む Java の命令に置き換えられて，Java ソースファイルに出力されます。

SQL の文字列，パラメタの個数，各パラメタのタイプとモード，及び出力する列の記述は，プロファイルに出力されます。プロファイルは，SQLJ ランタイムライブラリから参照されます。プロファイルの実体は，java.sql.runtime.Profile クラスのインスタンスです。

SQLJ トランスレータが生成，参照するファイルの一覧を次の表に示します。

表 18–1　SQLJ トランスレータが生成，参照するファイルの一覧

ファイル区分	ファイル名の形式	説明	種別
SQLJ ソースファイル	ファイル名.sqlj	SQLJ ソースファイルです。	参照
Java ソースファイル	ファイル名.java	Java ソースファイルです。	生成
プロファイル	ファイル名_SJProfile プロファイル番号.ser	SQLJ ソースファイルから抽出した各 SQL 文の情報を格納します。プロファイル番号は，コンテキストごとに付けられます。基数はどちらも 0 です。	生成

SQLJ トランスレータが内部的に生成する，クラスや変数のプリフィクスは次のようになります。

sJT：内部的に生成した変数の名称

_SJ：内部的に生成したクラスやプロファイルの名称

18.3　UAP の記述規則

SQLJ ソースファイルの記述規則について説明します。

18.3.1　名標の付け方の規則

次の名標は使用できません。

- 「_sJT_」で始まる名標
- 「_SJ」で始まる名標
- 「p_rdb」で始まる名標

これ以外の規則については，Java 言語の規則に従います。

18.3.2　SQL の記述規則

(1)　SQL 文の記述形式

SQL 文は，一つの SQL 文ごとに，SQL 先頭子（#sql）と SQL 終了子（;）で囲みます。SQL 文自体は，更に {} で囲みます。接続クラスやカーソルの宣言についても，SQL 先頭子と SQL 終了子で囲みます。

SQL 文の記述形式を次の表に示します。

表 18–2　SQL 文の記述形式

機能	形式	用途
SQL の実行	#sql [コンテキスト] { SQL 文 } ;	SQL 文を実行します。スタンダードインタフェース版とネイティブインタフェース版とでは，使用できる SQL が違います。詳細については，「SQLJ で使用できる SQL 文」を参照してください。
列名指定反復子クラスの宣言	• スタンダードインタフェース版の場合 #sql 修飾子 iterator クラス名 (データ型 列名,…) ; • ネイティブインタフェース版の場合 使用できません。	カーソルの宣言で使用するクラスを宣言します。FETCH 文には使用できません。
位置指定反復子クラスの宣言	• スタンダードインタフェース版の場合 #sql 修飾子 iterator クラス名 (データ型,…) ; • ネイティブインタフェース版の場合	カーソルの宣言で使用するクラスを宣言します。FETCH 文で使用します。

機能	形式	用途
	#sql 修飾子 iterator クラス名 [implements JP.co.Hitachi.soft.HiRDB.pdjpp.runtime. ForUpdate] [with (キーワード=値,…)] (データ型,…) ;	
接続クラスの宣言	#sql 修飾子 context クラス名 ;	接続で使用するクラスを宣言します。
カーソルの宣言	#sql 反復子オブジェクト = { SELECT 文 } ;	カーソルの定義とオープンをします。
結果セットの変換	• スタンダードインタフェース版の場合 #sql [コンテキスト] 反復子オブジェクト = {CAST :JDBC の結果セット} ; • ネイティブインタフェース版の場合 使用できません。	JDBC の結果セットを SQLJ 用に変換します。

注

修飾子：

private，public，protected，final，abstract，protected，static，native，synchronized，transient，及び volatile の組み合わせ

コンテキスト：

{接続コンテキスト｜接続コンテキスト,実行コンテキスト｜実行コンテキスト}

キーワード：

holdability，又は updateColumns

値：

true，false，又は"列名 1,列名 2,..."

データ型：

Java のデータ型

列名：

検索項目

(2) 複数接続機能を使用する場合の接続コンテキストの明示指定

複数接続機能を使用する場合，SQL 先頭子と SQL 文との間に，接続コンテキストを[ ]で囲んで記述して，使用する接続を明示します。例を次に示します。

```
#sql [connCtx] { DELETE FROM EMP WHERE SAL > 1000};
```

接続コンテキストを明示指定しない場合，デフォルトの接続コンテキストが仮定されます。

(3)　実行環境の明示指定

SQLJ では，デフォルトの実行環境のほかに，ユーザが明示的に指定することもできます。実行環境を指定する場合は，SQL 先頭子と SQL 文との間に，実行接続コンテキストを[]で囲んで記述します。

一つの接続に対して，複数のスレッドで同時に SQL を実行している場合，複数の実行環境を使い分けることで，実行結果がほかの SQL に上書きされることを防げます。例を次に示します。

```
ExecutionContext execCtx = new ExecutionContext();
try {
 #sql [execCtx] { DELETE FROM ZAIKO WHERE SCODE > 1000 };
 System.out.println
  (”removed ” + execCtx.getUpdateCount() + ”goods”);
}
catch(SQLException e){
 System.out.println(”SQLException has occurred with ”+ ” exception ” + e);
}
```

実行接続コンテキストを明示指定しない場合，デフォルトの実行環境が使用されます。

実行環境では，次の値を保持します。値は，set<名称>メソッドで設定し，get<名称>で求めます。

名　称	内　　容
MaxRows	検索で返す最大の行数です。
MaxFieldSize	列や OUTPUT 変数の値で返される，データの最大バイト数です。
QueryTimeout	SQL の実行完了までの待ち時間最大値です。HiRDB では無効となります。
UpdateCount	更新，挿入，又は削除した行数です（参照だけです）。
SQLWarnings	SQLWARN0～SQLWARNF に相当します（参照だけです）。

なお，複数接続の指定と併用する場合は，接続コンテキストと実行接続コンテキストの順に，コンマで区切って指定します。例を次に示します。

```
 #sql [connCtX, execCtx] { DELETE FROM ZAIKO WHERE SCODE > 1000 };
```

(4)　埋込み変数の指定方法

SQLJ では，埋込み変数を宣言するための BEGIN DECLARE SECTION は使用しません。

任意の変数，パラメタ，及びオブジェクトのフィールドを，埋込み変数として使用できます。SQL 文中では，先頭にコロンを付けて":変数名"と記述します。コロンと変数名の間には空白があってもかまいません。

CALL 文の IN，OUT，又は INOUT パラメタは，":｛IN ｜ OUT ｜ INOUT｝変数名"と記述します。

また，SQLJ では，埋込み変数として":（式）"を使用できます。式は必ず()で囲む必要があります。なお，これは Java の式であり，SQL の式ではありません。例を次に示します。

```
#sql { SELECT COL1, COL2 FROM TABLE1 WHERE :(x[--i]) > COL3 };
```

(5) 標識変数の指定方法

SQLJ では標識変数がないため，埋込み変数にナル値を設定する場合は，基本データ型ではなく，sql.lang パッケージで定義されている Wrapper 型を使用します。ナル値を基本データ型の Java 変数で受け取った場合，SQLNullException 例外が発生します。

(6) 例外処理

SQLJ では，埋込み型 SQL の WHENEVER での例外処理ができません。このため，WHENEVER の代わりに，Java の例外処理（try...catch）を使用します。例を次に示します。

```
try{
   #sql { DELETE FROM ZAIKO WHERE SCODE > 1000 };
}
catch(SQLException e){
   System.out.println("SQLCODE:" + e.getErrorCode() +
                      "¥nERRMSG:" + e.getMessage() );
}
```

SQL 実行時にエラーが発生した場合，JDBC の例外オブジェクト（java.sql.SQLException）が発行されます。

SQLCODE，SQLSTATE，及びエラーメッセージは例外オブジェクトに格納され，値は getErrorCode メソッド，getSQLState メソッド，及び getMessage メソッドで取得できます。

(7) 静的 SQL と動的 SQL

SQLJ では静的 SQL だけ記述できます。動的 SQL は記述できません。

動的 SQL を使用したい場合は，JDBC の API を使用してください。

(8) 動的カーソルの結果セットの読み出し

JDBC の API を使用して作成した，動的カーソルの結果セットを CAST 文で変換し，SQLJ のカーソルの結果セットとして読み出しができます。例を次に示します。

```
#sql iterator Employees(String ename, double sal);
Statement stmt=conn.createStatement();
String Query="SELECT sname, scode FROM zaiko WHERE scode > 1000";
ResultSet rs=stmt.executeQuery(query);
Employees emps;
#sql emps ={CAST :rs };
```

ネイティブインタフェース版では，CAST 文は使用できません。使用した場合，トランスレート時にエラーとなります。

(9)　HiRDB サーバとの接続，切り離し

スタンダードインタフェース版の場合，CONNECT 文及び DISCONNECT 文は使用できません。ネイティブインタフェース版の場合は使用できます。なお，スタンダードインタフェース版，及びネイティブインタフェース版共に，Java の命令を使用すると，HiRDB サーバとの接続，切り離しができます。

(10)　例外の発生条件

HiRDB の埋込み型 SQL では，次の場合に警告が発生しますが，SQLJ では例外が発生します。

- 1 行 SELECT 文で，検索項目と INTO 句に指定した変数の個数が一致していない場合
- 1 行 SELECT 文で，検索結果が 0 行の場合
- 1 行 SELECT 文で，検索結果が複数行の場合
- FETCH 文で，検索項目数と INTO 句に指定した変数の個数が一致していない場合
- 位置指定反復子で定義されている列数と，検索項目数が一致していない場合
- 列名指定反復子で定義されている列数より，検索項目数が少ない場合

(11)　注釈及び SQL 最適化指定の扱い

SQL 先頭子から SQL 終了子までの間に記述した注釈（/*～*/）は削除します。ただし，カーソルの宣言と SQL 文の実行の場合，{ }内に記述した SQL 最適化指定（/*>>～<<*/）は削除しないで，SQL 文として扱います。これ以外の SQL 最適化指定（/*>>～<<*/）については，注釈とみなされます。SQL 文中での注釈及び SQL 最適化指定については，マニュアル「HiRDB SQL リファレンス」を参照してください。

18.3.3　SQLJ で使用できる SQL 文

SQLJ で使用できる SQL 文を次の表に示します。

表 18–3　SQLJ で使用できる SQL 文

種別	SQL 文	使用可否		代替手段
		スタンダードインタフェース版	ネイティブインタフェース版	
定義系 SQL	すべて	○	○	－
操作系 SQL	ASSIGN LIST 文	△	△	JDBC を使用します。
	CALL 文	○	○	－
	CLOSE 文	△	△	反復子を使用します。

種別	SQL 文	使用可否		代替手段
		スタンダードインタフェース版	ネイティブインタフェース版	
	DECLARE CURSOR	△	△	
	DELETE 文	○	○	－
	DESCRIBE 文	▲	▲※	JDBC を使用します。
	DESCRIBE TYPE 文	▲	▲※	
	DROP LIST 文	▲	▲※	
	EXECUTE 文	▲	▲※	
	EXECUTE IMMEDIATE 文	▲	▲※	
	FETCH 文（形式 1 又は形式 3）	○	○	－
	FETCH 文（形式 2）	×	×	－
	INSERT 文	○	○	－
	OPEN 文（形式 1）	△	△	反復子を使用します。
	OPEN 文（形式 2）	△	△	JDBC を使用します。
	PREPARE 文	△	△	
	PURGE TABLE 文	○	○	－
	1 行 SELECT 文	○	○	－
	動的 SELECT 文	▲	▲※	JDBC を使用します。
	UPDATE 文	○	○	－
制御系 SQL	COMMIT 文	○	○	－
	COMMIT 文（RELEASE 指定）	△	△	COMMIT と DISCONNECT に分けます。
	CONNECT 文	×	○	－
	DISCONNECT 文	×	○	－
	LOCK 文	○	○	－
	ROLLBACK 文	○	○	－
	ROLLBACK 文（RELEASE 指定）	△	△	ROLLBACK と DISCONNECT に分けます。
	SET SESSION AUTHORIZATION 文	×	×	－
埋込み言語文法	BEGIN DECLARE SECTION	×	×	－

種別	SQL 文	使用可否		代替手段
		スタンダードインタフェース版	ネイティブインタフェース版	
	END DECLARE SECTION	×	×	−
	ALLOCATE CONNECTION HANDLE	△	△	接続コンテキストを使用します。
	DECLARE CONNECTION HANDLE	△	△	
	FREE CONNECTION HANDLE	△	△	
	GET CONNECTION HANDLE	×	×	−
	COPY	×	×	−
	GET DIAGNOSTICS	×	×	−
	WHENEVER	△	△	try…catch で実装します。

（凡例）

○：SQLJ で使用できます。

△：SQLJ では使用できませんが，SQLJ，又は JAVA が提供する機能で同等の機能を使用できます。

▲：SQLJ では使用できませんが，JDBC を使用することで同等の機能を使用できます。

×：SQLJ では使用できません。

−：代替手段はありません。

注

SQLJ では，使用する JDBC ドライバが提供していない HiRDB の機能は使用できません。使用できない機能を次に示します。

- 反復子を使用した UPDATE 文及び DELETE 文
- 反復子の宣言時に WITH 句のキーワードを指定
- 配列を使用した INSERT，UPDATE 及び DELETE 機能は使用できません。
- ?パラメタに NULL 述語を指定した SQL（? IS〔NOT〕 NULL）は使用できません。

注※

JDBC の接続オブジェクトを使用して接続コンテキストを作成した場合，ネイティブインタフェースでも代替手段が使用できます。JDBC の接続オブジェクトを使用しないで接続コンテキストを作成した場合は，代替手段は使用できません。

18.3.4　HiRDB のデータ型と SQLJ のデータ型の対応

HiRDB のデータ型と SQLJ のデータ型の対応を次の表に示します。SQLJ で埋込み変数を使用する場合は，この表に従って変数を宣言してください。

表 18–4　HiRDB のデータ型と SQLJ のデータ型の対応

HiRDB のデータ型	SQLJ のデータ型（Java のデータ型）	
	ナル値を含む場合	ナル値を含まない場合
CHAR※1	java.lang.String	–
	JP.co.Hitachi.soft.HiRDB.pdjpp.runtime.HiRDBCHAR※4	–
VARCHAR	java.lang.String	–
	JP.co.Hitachi.soft.HiRDB.pdjpp.runtime.HiRDBVARCHAR※4	–
NCHAR※1	java.lang.String	–
	JP.co.Hitachi.soft.HiRDB.pdjpp.runtime.HiRDBNCHAR※4	–
NVARCHAR※1	java.lang.String	–
	JP.co.Hitachi.soft.HiRDB.pdjpp.runtime.HiRDBNVARCHAR※4	–
MCHAR※1	java.lang.String	–
	JP.co.Hitachi.soft.HiRDB.pdjpp.runtime.HiRDBMCHAR※4	–
MVARCHAR※1	java.lang.String	–
	JP.co.Hitachi.soft.HiRDB.pdjpp.runtime.HiRDBMVARCHAR※4	–
DECIMAL※2	java.math.BigDecimal	–
	JP.co.Hitachi.soft.HiRDB.pdjpp.runtime.HiRDBDECIMAL※4	–
SMALLINT	java.lang.Short	short
INTEGER	java.lang.Integer	int
REAL，SMALLFLT	java.lang.Float	float
FLOAT，DOUBLE PRECISION	java.lang.Double	double
DATE	java.sql.Date	–
TIME	java.sql.Time	–
TIMESTAMP	java.sql.Timestamp	–
INTERVAL HOUR TO SECOND	–	–
INTERVAL YEAR TO DAY	–	–

HiRDB のデータ型	SQLJ のデータ型（Java のデータ型）	
	ナル値を含む場合	ナル値を含まない場合
BLOB※3	JP.co.Hitachi.soft.HiRDB.pdjpp.runtime.HiRDBBLOB※4	byte[]
BINARY	JP.co.Hitachi.soft.HiRDB.pdjpp.runtime.HiRDBBINARY※4	byte[]

（凡例）

－：使用できない，又は該当しないことを意味します。

注

繰返し列は使用できません。

注※1

ネイティブインタフェース版の場合，java.lang.String を指定したときは，VARCHAR 型としてサーバに要求します。また，出力変数に指定した場合は，データの受け取り領域の長さとして 32,000 バイトが仮定されます。

注※2

ネイティブインタフェース版の場合，java.math.BigDecimal を出力変数として使用するときは，精度 15，位取り 0 として設定されます。

注※3

ネイティブインタフェース版の場合，byte[]で指定したときは，BINARY 型としてサーバ側に要求します。なお，HiRDB サーバがバージョン 06-02 以前の場合に BLOB 型を使用するときは，HiRDB のデータ型である JP.co.Hitachi.soft.HiRDB.pdjpp.runtime.HiRDBBLOB を使用してください。byte[]を指定するとエラーとなります。

注※4

ネイティブインタフェース版の場合に指定できます。

18.3.5　出力変数の設定（ネイティブインタフェース版限定）

1 行検索，及び CALL 文の OUT パラメタで使用する出力変数は，出力変数の初期値によって，サーバへ実行要求をするときの SQL 記述領域に設定するデータの長さが変わります。各データ型の初期値と SQL 記述領域に設定されるデータの長さを次の表に示します。

表 18–5　各データ型の初期値と SQL 記述領域に設定されるデータの長さ

データ型	初期値	SQL 記述領域に設定されるデータの長さ
HiRDBCHAR	変数 = null;	30,000 バイト
	変数 = new HiRDBCHAR(int n);	n バイト（1≦n≦30,000）
	変数 = new HiRDBCHAR(String t);	t の長さ（t.getBytes()で得られる byte 配列の長さ）

データ型	初期値	SQL 記述領域に設定されるデータの長さ
HiRDBVARCHAR	変数 = null;	32,000 バイト
	変数 = new HiRDBVARCHAR(int n);	n バイト（1≦n≦32,000）
	変数 = new HiRDBVARCHAR(String t)	t の長さ（t.getBytes()で得られる byte 配列の長さ）
HiRDBNCHAR	変数 = null;	30,000 バイト（全角 15,000 文字）
	変数 = new HiRDBNCHAR(int n);	(n*2)バイト（全角 n 文字）（1≦n≦15,000）
	変数 = new HiRDBNCHAR(String t)	t の長さ（t.getBytes()で得られる(byte 配列/2)の長さ）
HiRDBNVARCHAR	変数 = null;	32,000 バイト（全角 16,000 文字）
	変数 = new HiRDBNVARCHAR(int n);	(n*2)バイト（全角 n 文字）（1≦n≦16,000）
	変数 = new HiRDBNVARCHAR(String t)	t の長さ（t.getBytes()で得られる(byte 配列/2)の長さ）
HiRDBMCHAR	変数 = null;	30,000 バイト
	変数 = new HiRDBMCHAR(int n);	n バイト（1≦n≦30,000）
	変数 = new HiRDBMCHAR(String t)	t の長さ（t.getBytes()で得られる byte 配列の長さ）
HiRDBMVARCHAR	変数 = null;	32,000 バイト
	変数 = new HiRDBMVARCHAR(int n);	n バイト（1≦n≦32,000）
	変数 = new HiRDBMVARCHAR(String t)	t の長さ（t.getBytes()で得られる byte 配列の長さ）
HiRDBDECIMAL	変数 = null;	精度 15，位取り 0
	変数 = new HiRDBDECIMAL(int p,int s);	精度 p，位取り s（1≦p≦38，0≦s≦p）
	変数 = new HiRDBDECIMAL(String t)	精度は，t から符号及びピリオドを除いた文字列長です。位取りは，ピリオド以降の文字列長（ピリオドは含まない）。
	変数 = newHiRDBDECIMAL(java.math.BigDecimal t)	精度は，t から toString()で取り出した文字列の符号及びピリオドを除いた文字列長です。位取りは，BigDecimal オブジェクトの scale()メソッドで取り出した値です。
HiRDBBLOB	変数 = null;	1 メガバイト
	変数 = new HiRDBBLOB(int n);	n バイト（1≦n≦2,147,483,647）
	変数 = new HiRDBBLOB(byte[] t)	t の長さ
HiRDBBINARY	変数 = null;	1 メガバイト
	変数 = new HiRDBBINARY(int n);	n バイト（1≦n≦2,147,483,647）
	変数 = new HiRDBBINARY(byte[] t)	t の長さ
Java.math.BigDecimal	変数 = null;	精度 15，位取り 0

データ型	初期値	SQL 記述領域に設定されるデータの長さ
	変数 = new java.math.BigDecimal;	精度には，BigDecimal オブジェクトから toString()で取り出した文字列の符号及びピリオドを除いた文字列長を設定します。位取りには，scale()メソッドで取り出した値を設定します。
byte[]	変数 = null;	1 メガバイト
	変数 = new byte[int n]	n バイト（1≦n≦2,147,483,647）

18.3.6　カーソル宣言時のデータ型の使用（ネイティブインタフェース版限定）

カーソル宣言時に HiRDB のデータ型を使用する場合は，次の表のように記述してください。

表 18–6　カーソル宣言時に HiRDB のデータ型を使用する場合の記述と受け取り領域の設定

データ型	カーソル宣言時の記述	受け取り領域の設定
HiRDBCHAR	#sql iterator カーソル名(HiRDBCHAR(int n));	n バイト（1≦n≦30,000）
HiRDBVARCHAR	#sql iterator カーソル名(HiRDBVARCHAR(int n));	n バイト（1≦n≦32,000）
HiRDBNCHAR	#sql iterator カーソル名(HiRDBNCHAR(int n));	(n*2)バイト（全角 n 文字） （1≦n≦15,000）
HiRDBNVARCHAR	#sql iterator カーソル名(HiRDBNVARCHAR(int n));	(n*2)バイト（全角 n 文字） （1≦n≦16,000）
HiRDBMCHAR	#sql iterator カーソル名(HiRDBMCHAR(int n));	n バイト（1≦n≦30,000）
HiRDBMVARCHAR	#sql iterator カーソル名(HiRDBMVARCHAR(int n));	n バイト（1≦n≦32,000）
HiRDBDECIMAL	#sql iterator カーソル名(HiRDBMVARCHAR(int p,int s));	精度 p，位取り s （1≦p≦38，0≦s≦p）
HiRDBBLOB	#sql iterator カーソル名(HiRDBBLOB(int n));	n バイト （1≦n≦2,147,483,647）
HiRDBBINARY	#sql iterator カーソル名(HiRDBBINARY(int n));	n バイト （1≦n≦2,147,483,647）

18.3.7　HiRDB サーバとの接続，切り離しの記述

SQLJ には CONNECT 文及び DISCONNECT 文がないため，HiRDB サーバとの接続，切り離しは Java の命令で記述します。

(1)　HiRDB サーバとの接続

HiRDB サーバと接続する場合，接続コンテキストを使用して次のように記述します。

(a)　接続コンテキストクラスの定義

接続コンテキストのクラスを定義します。クラス名は Java の識別子です。定義したクラスは，sqlj.runtime.ConnectionContext を継承しています。

```
#sql 修飾子 context　クラス名 ;
```

(b)　接続コンテキストの宣言

宣言したクラスを使用して，接続コンテキストを宣言します（Java の変数宣言として宣言します)。接続コンテキストは Java の識別子です。

```
修飾子 クラス名　接続コンテキスト ;
```

(c)　HiRDB サーバへの接続

new 演算子で接続コンテキストのオブジェクトを生成します。このとき，HiRDB サーバへ接続されます。接続パラメタには，接続先の HiRDB サーバ，ポート番号，認可識別子，及びパスワードを，JDBC と同じ形式で記述します。

```
接続コンテキスト = new クラス名(接続パラメタ）;
```

(d)　ネイティブインタフェース版を使用した場合の HiRDB サーバへの接続

ネイティブインタフェース版を使用した場合，次の 3 種類の接続方法があります。

- Java の命令として記述
- CONNECT 文を使用
- JDBC の接続オブジェクト（Connection）を使用

各接続方法の説明を次に示します。

1.Java の命令として記述

new 演算子で接続コンテキストのオブジェクトを生成します。ただし，JDBC を使用しないため，接続パラメタには認可識別子，パスワード，サーバ名，及びポート番号を指定します。

```
接続コンテキスト = new クラス名(接続パラメタ);
```

接続パラメタに何も指定しない場合，クライアント環境定義を参照します。

```
接続コンテキスト = new クラス名();
```

接続コンテキストの作成例を次に示します。

```
#sql context Ctx;
String Userid=new String( “user1” );
String Passwd=new String( “puser1” );
String Host=new String( “HiRDB_SV” );
```

```
short port=22000;

Ctx con = new Ctx(:Userid,:Passwd,:Host,:port);
```

2.CONNECT 文を使用

接続パラメタには，認可識別子，及びパスワードを指定します。

ポート番号，及びサーバ名は，クライアント環境定義を参照します。

```
#sql [接続コンテキスト]{CONNECT USER :埋込み変数 USING :埋込み変数};
又は
#sql [接続コンテキスト]{CONNECT :埋込み変数 IDENTIFIED BY :埋込み変数};
```

接続パラメタを指定しない場合，クライアント環境定義を参照します。

```
#sql [接続コンテキスト]{CONNECT};
```

CONNECT 文の例を次に示します。

```
#sql context Ctx;
String Userid=new String( “user1” );
String Passwd=new String( “puser1” );
Ctx con;

#sql [con] {CONNECT USER :Userid USING :Passwd };
```

3.JDBC の接続オブジェクト（Connection）を使用

new 演算子で接続コンテキストのオブジェクトを生成します。接続パラメタには，JDBC の接続オブジェクト（java.sql.Connection）を指定します。

```
接続コンテキスト = new クラス名(接続オブジェクト);
```

接続コンテキストの作成例を次に示します。

```
#sql context Ctx;
java sql.Connection con =

java.sql.DriverManager.getConnection(”jdbc:hitachi:PrdbDrive://DBID=22200,
DBHOST=HiRDB_SV”,” user1” ,” user1” );

Ctx ctx = new Ctx(con);
```

(2)　HiRDB サーバとの切り離し

HiRDB サーバとの切り離しをする場合は，接続コンテキストの close メソッドを呼び出します。なお，再接続用のメソッドはありません。再接続する場合は，オブジェクトを新たに作成します。

```
接続コンテキスト.close() ;
```

接続コンテキストの close メソッドを呼び出す例を次に示します。

```
#sql context DeptContext;
          :
```

```
{
    DeptContext deptCtx = new DeptContext(deptURL,true);
    #sql [deptCtx] { DELETE FROM TAB };
    deptCtx.close();
}
```

ネイティブインタフェース版を使用した場合，接続テキストの close メソッドを呼び出す方法以外に，DISCONNECT 文を使用することもできます。

```
#sql[接続コンテキスト]{DISCONNECT};
```

DISCONNECT 文の例を次に示します。

```
#sql context Ctx;
Ctx con;
#sql[con]{CONNECT};

#sql[con]{DISCONNECT};
```

(3)　デフォルト接続

(a)　スタンダードインタフェース版の場合

スタンダードインタフェース版の場合，SQL 文に接続コンテキストを明示しないときは，デフォルトの接続コンテキストが指定されたものとみなされます。

デフォルト接続コンテキストを使用する場合は，UAP があらかじめ接続コンテキストを作成し，その接続コンテキストをデフォルト接続コンテキストとして設定しておく必要があります。一度設定したデフォルト接続コンテキストは，デフォルト接続コンテキストの close()メソッドを発行するか，又は新たな接続コンテキストをデフォルトの接続コンテキストとして再設定しないかぎり有効です。

デフォルト接続コンテキストは，デフォルト接続コンテキストクラス(JP.co.Hitachi.soft.HiRDB.sqj.runtime.PrdbContext)内の変数に保持されています。

デフォルト接続コンテキストは，次のような異なる引数を持つ，複数のコンストラクタを持っています。

- JDBC の connection オブジェクトを引数に持つコンストラクタ
- 接続先の URL，認可識別子，パスワード，及びオートコミットの指定を引数に持つコンストラクタ
- 接続先の URL，Properties オブジェクト，及びオートコミットの指定を引数に持つコンストラクタ
- 接続コンテキストを引数に持つコンストラクタ

接続先の URL，認可識別子，及びパスワードの指定は，HiRDB の JDBC ドライバと同じ形式で記述します。

SQLJ では，接続コンテキスト作成時に接続 URL を含むコンストラクタを使用する場合，オートコミットの指定が必要であり，有効にするときは TRUE，無効にするときは FALSE と指定します。

JDBC の接続コンテキストから生成する場合は，オートコミットの設定は JDBC の接続コンテキストの設定を引き継ぎます。

- **デフォルト接続コンテキストの作成と設定**

  デフォルト接続コンテキストの作成と設定の例を次に示します。

```
import JP.co.Hitachi.soft.HiRDB.sqj.runtime.PrdbContext;
                          :
PrdbContext pctx = new PrdbContext(url,user,passwd,autoCommit);
PrdbContext.setDefaultContext(pctx);
```

- **デフォルト接続コンテキストの解放と再設定**

  デフォルト接続コンテキストの解放と再設定の例を次に示します。

```
import JP.co.Hitachi.soft.HiRDB.sqj.runtime.PrdbContext;
                            :
PrdbContext pctx = new PrdbContext(url,user,passwd,autoCommit);
PrdbContext.setDefaultContext(pctx);
                 :
pctx.close();
PrdbContext new_pctx = new PrdbContext(url.use,passwd,autoCommit);

PrdbContext.setDefaultContext(new_pctx);
```

- **デフォルト接続コンテキストの取得**

  次のメソッドを呼び出すと，デフォルトの接続コンテキストを取得できます。

```
JP.co.Hitachi.soft.HiRDB.sqj.runtime.PrdbContext.getDefaultContext();
```

  デフォルトコンテキストを明示的に指定した場合の例を次に示します。

```
void print_address(String name) throws SQLException;
{
  String telno;
  sqlj.runtime.ConnectionContext ctx;
  ctx = JP.co.Hitachi.soft.HiRDB.sqj.runtime.PrdbContext.getDefaultContext();
  #sql [ctx] { SELECT TELNO INTO :telno
               FROM PERSON
               WHERE :name = NAME } ;
}
```

(b)　ネイティブインタフェース版の場合

ネイティブインタフェース版の場合，デフォルト接続コンテキストクラスは JP.co.Hitachi.soft.HiRDB.pdjpp.runtime.PrdbContext 内の変数に保持されています。

デフォルト接続コンテキストクラスは，次の複数のコンストラクタを持ちます。

- JDBC の connection オブジェクトを引数に持つコンストラクタ
- 接続先の認可識別子，パスワード，サーバ名，及びポート番号を引数に持つコンストラクタ
- 接続先の認可識別子，及びパスワード指定を引数に持つコンストラクタ

- 接続コンテキストを引数に持つコンストラクタ
- 引数なしのコンストラクタ

- **デフォルト接続コンテキストの作成と設定**

  デフォルト接続コンテキストの作成と設定の例を次に示します。

```
import JP.co.Hitachi.soft.HiRDB.pdjpp.runtime.PrdbContext;
                          :
PrdbContext pctx = new PrdbContext();
PrdbContext.setDefaultContext(pctx);
```

- **デフォルト接続コンテキストの解放と再設定**

  デフォルト接続コンテキストの解放と再設定の例を次に示します。

```
import JP.co.Hitachi.soft.HiRDB.pdjpp.runtime.PrdbContext;
                          :
PrdbContext pctx = new PrdbContext(user,passwd,host,port);
PrdbContext.setDefaultContext(pctx);
                          :
pctx.close();
PrdbContext new_pctx = new PrdbContext(user,passwd,host,port);

PrdbContext.setDefaultContext(new_pctx);
```

- **デフォルト接続コンテキストの取得**

  次のメソッドを呼び出すと，デフォルトの接続コンテキストが得られます。

```
JP.co.Hitachi.soft.HiRDB.pdjpp.runtime.PrdbContext.getDefaultContext () ;
```

  デフォルトコンテキストを明示的に指定した場合のコーディング例を次に示します。

```
void print_address(String name) throws SQLException;
{
  String telno;
  JP.co.Hitachi.soft.HiRDB.pdjpp.runtime.ConnectionContext ctx;
  ctx = JP.co.Hitachi.soft.HiRDB.sqj.runtime.PrdbContext.getDefaultContext();
  #sql [ctx] { SELECT TELNO INTO :telno FROM PERSON WHERE :name = NAME } ;
}
```

18.3.8　カーソルによる検索の記述

SQLJ には DECLARE CURSOR，OPEN 文，及び CLOSE 文がないため，カーソルの宣言，オープン，及びクローズは，Java の命令として記述します。このとき，カーソル名の代わりに反復子オブジェクトを使用します。反復子オブジェクトは，オブジェクトへの参照変数として宣言されるため，名称規則と有効範囲については Java の規則に従います。

検索結果の取得は，使用する反復子オブジェクトの型によって，FETCH 文で取得する方法と，FETCH 文を使用しない方法とがあります。FETCH 文では，位置指定反復子型のオブジェクトを使用し，列名指定反復子のオブジェクトは使用できません。

(1)　FETCH 文を使用した検索

FETCH 文を使用して検索をする場合の，記述方法について説明します。

(a)　位置指定反復子クラスの定義と反復子オブジェクトの宣言

- **スタンダードインタフェース版の場合**

  スタンダードインタフェース版の場合，位置指定反復子のクラスを定義して，反復子オブジェクトを宣言します。クラス名は Java の識別子です。データ型 N は，FETCH 文で N 番目の検索項目を格納する Java の変数のデータ型です。

```
#sql 修飾子 iterator クラス名
            （データ型１,データ型2,…）;
修飾子 クラス名 反復子オブジェクト ;
```

- **ネイティブインタフェース版の場合**

  ネイティブインタフェース版の場合，次のようになります。

```
#sql 修飾子 iterator クラス名
         [ implements JP.co.Hitachi.soft.HiRDB.pdjpp.runtime.ForUpdate ]
         [ with キーワード=設定値,…]
            （データ型１,データ型2,…）;
修飾子 クラス名 反復子オブジェクト ;
```

  UPDATE 文，DELETE 文で反復子を使用する場合，
  JP.co.Hitachi.soft.HiRDB.pdjpp.runtime.ForUpdate インタフェースを継承します。

  WITH 句のキーワードは，反復子の機能を示します。設定値は定数だけです。WITH 句のキーワードと値の組み合わせを次の表に示します。

表 18–7　WITH 句のキーワードと値の組み合わせ

WITH 句のキーワード	機能	設定値
holdability	ホールダブルカーソルであることを示します。	TRUE
updateColumns	更新する列を示します。	"列名,列名,..."

(b)　カーソルの定義とオープン

宣言した反復子オブジェクトに，SELECT 文の結果セットを代入します。

```
#sql［コンテキスト］反復子オブジェクト = { SELECT文 } ;
```

(c)　検索結果の取り出し

カーソル名の代わりに，反復子オブジェクトを指定して FETCH 文を実行します。反復子オブジェクトは，先頭にコロンを付けます。

```
#sql［コンテキスト］{
   FETCH :反復子オブジェクト INTO :変数1,:変数2,…} ;
```

(d)　NOT FOUND の判定

反復子オブジェクトの endFetch メソッドを呼び出して，NOT FOUND かどうかを判定します。検索する行がない場合は true が返ります。次の行がある場合は false が返ります。カーソルを閉じた後に呼んだ場合は true が返ります。

```
while(! 反復子オブジェクト.endFetch()) {
     取り出した行に対する処理
}
```

(e)　カーソルのクローズ

カーソルをクローズする場合は，close メソッドを呼び出します。

```
反復子オブジェクト.close() ;
```

FETCH 文を使用した検索例を次に示します。

```
#sql public iterator ByPos(String, int);
                  :
{
  ByPos positer;
  String name = null;
  int code = 0;

  #sql positer = { SELECT SNAME,SCODE FROM ZAIKO };
  #sql { FETCH :positer INTO :name,:code };
  while( !positer.endFetch() ){
    System.out.println(name + ":" + code);
    #sql { FETCH :positer INTO :name,:scode };
  }
  positer.close();
}
```

(2)　FETCH 文を使用しない検索

列名指定反復子のフィールドを使用して，検索結果の各列を読み出します。

(a)　列名指定反復子クラスの定義

検索項目と同じ名称（大文字と小文字は区別しない）を，クラスのフィールドとして定義します。データ型は，検索結果を受け取る Java の変数のデータ型を指定します。なお，ネイティブインタフェースの場合は使用できません。

検索項目が，値式，Java で使用できない文字を含む列名などの場合，AS 句で検索項目に別名を定義しておいて，その別名を使用します。

```
#sql 修飾子 iterator クラス名
          (データ型1 列名1,
```

```
            データ型2 列名2,…) ;
修飾子 クラス名 反復子オブジェクト;
```

(b) カーソルの定義と OPEN

宣言した反復子オブジェクトに，SELECT 文の結果セットを代入します。

```
#sql［コンテキスト］反復子オブジェクト = { SELECT文 } ;
```

(c) 次の行の取り出しと NOT FOUND の判定

反復子オブジェクトの next メソッドを呼び出して，NOT FOUND かどうかを判定します。NOT FOUND の場合は TRUE が返ります。行がある場合は FALSE が返ります。カーソルがオープンした後，最初の next メソッドが実行されるまで，カーソルは検索結果の最初の行に位置づけられません。

```
while(反復子オブジェクト.next()){
    取り出した行に対する処理
}
```

(d) 検索結果の取得

反復子オブジェクトの各フィールドから，データを読み出します。NOT FOUND の場合や，カーソルがクローズされた後に読み出すと，結果は不定となります。フィールドのデータ型が Java の基本データ型で，検索結果がナル値の場合に読み出すと，SQLNullException オブジェクトが発生します。

フィールドに対して代入しても，データベースには反映されません。

```
変数1 = 反復子オブジェクト.列名1 ;
変数2 = 反復子オブジェクト.列名2 ;
            :
```

(e) カーソルの CLOSE

反復子オブジェクトの close メソッドを呼び出します。

```
反復子オブジェクト.close() ;
```

FETCH 文を使用しない検索例を次に示します。

(例)

```
#sql public iterator ByName(String sname,
                            int scode);
                :
{
   ByName nameiter;
   String s;
   int i;
```

```
    #sql nameiter = { SELECT SNAME, SCODE FROM ZAIKO };
    while( nameiter.next() ){
      s = nameiter.sname();
      i = nameiter.scode();
      System.out.println(s + ":" + i);
    }
    nameiter.close();
}
```

(3)　カーソルを使った更新

ネイティブインタフェース版の場合，カーソルを使った更新ができます。

UPDATE 文，DELETE 文で，カーソルが位置づけられている行を操作する場合，カーソル名の代わりに反復子を指定します。なお，反復子のクラス定義のときに，必ず ForUpdate インタフェースを継承しておく必要があります。

```
#sql [コンテキスト] { DELETE文 WHERE CURRENT OF :反復子オブジェクト } ;

#sql [コンテキスト] { UPDATE文 WHERE CURRENT OF :反復子オブジェクト } ;
```

反復子を使用した更新例を次に示します。

```
#sql public iterator ByPos
            implements JP.co.Hitachi.soft.HiRDB.pdjpp.runtime.ForUpdate
            (String, int);
             :

{
  ByPos positer;
  String name = null;
  int year = 0;
  int newyear;

  #sql positer = { SELECT FULLNAME, BIRTHYEAR FROM PEOPLE };
  #sql { FETCH :positer INTO :name,:year };
  while( !positer.endFetch() ){
    newyear=year+10;
    #sql { UPDATE PEOPLE SET YEAR=:newyear WHERE CURRENT OF :positer; };
  }
  positer.close();
}
```

18.3.9　動的結果セットの受け取り

動的結果セットを返すプロシジャを呼び出して，動的結果セットを受け取る場合，実行コンテキストの getNextResultSet()メソッドを使用します。なお，ネイティブインタフェース版の場合，JDBC の結果セットを使用できないため，動的結果セットを返すプロシジャは使用できません。

getNextResultSet メソッドは，返り値として動的結果セット（ResultSet オブジェクト）を返します。呼び出されるごとに，次の結果セットを返します。最後の結果セットを返した後は，ナル値を返します。

動的結果セットを返さないプロシジャや SQL の場合は，ナル値を返します。SQL の実行が正常終了しなかった場合もナル値を返します。

getNextResultSet メソッド実行中にエラーが発生した場合は，SQLException が発生します。

例を次に示します。

```
#sql  [execCtx] { CALL MULTI_RESULTS() };
ResultSet rs;
while((rs == execCtx.getNextResultSet() ) != null){
  検索結果の処理；
  rs.close();
}
```

18.3.10　JDBC との相互運用

JDBC と SQLJ を併用するときの運用について説明します。

(1)　SQLJ の反復子からの，JDBC の結果セットの取得

SQLJ の反復子を，JDBC の結果セット（ResultSet オブジェクト）に変換して，JDBC の API を利用して検索結果を取得できます。なお，ネイティブインタフェース版の場合，JDBC の結果セットを取得できません。

JDBC の結果セットを取得するには，反復子クラス(ResultSetIterator)の getResultSet メソッドを使用します。返り値として，getResultSet メソッドは JDBC の結果セットを返します。反復子で next メソッドを実行した後は，getResultSet メソッドを呼び出さないでください。

getResultSet メソッドで JDBC の結果セットに変換した後は，元の反復子を使用して検索結果を受け取ってはいけません。

例を次に示します。

```
public void showEmployeeName() throws SQLException
{
sqlj.runtime.ResultSetIterator iter;
#sql iter = { SELECT ename FROM rmp } ;
ResultSet rs = iter.getResultSet();
while(rs.next()){
System.out.println(”employee name: ” + rs.getString(1));
}
iter.close();
}
```

(2) JDBC の結果セットを，SQLJ の反復子の結果セットとして読み出す方法（スタンダードインタフェース版限定）

JDBC の API を使用して作成した，JDBC の結果セット（ResultSet）を CAST 文で変換し，SQLJ のカーソルの結果セットとして読み出せます。

コーディング例を次に示します。

```
#sql iterator Employees(String ename, double sal);
Statement stmt=conn.createStatement();
String query=”SELECT sname, scode FROM zaiko WHERE scode > 1000”;
ResultSet rs=stmt.executeQuery(query);
Employees emps;
#sql emps = {CAST :rs } ;
```

(3) JDBC の接続の，SQLJ の接続コンテキストへの変換

SQLJ の接続コンテキストには，JDBC の接続からオブジェクトを生成するコンストラクタが定義されています。このコンストラクタを使用することで，JDBC の接続から SQLJ の接続コンテキストに変換できます。なお，コンストラクタの引数として JDBC の接続を渡します。また，両方の接続は併用できます。

例を次に示します。

```
java.sql.Connection jdbcConCtx =java.sql.DriverManager.getConnection(…);
 #sql context Inventory;
Inventory sljConCtx = new Inventory(jdbcConCtx);
```

(4) SQLJ の接続の，JDBC の接続への変換

SQLJ の接続コンテキストは，getConnection メソッドで JDBC の接続を取得できます。両方の接続は併用できます。

ネイティブインタフェース版の場合，SQLJ の接続を JDBC に変換することはできません。JDBC と同一の接続を使用したい場合は，あらかじめ JDBC 側で接続を作成し，SQLJ の接続コンテキストに変換する必要があります。

例を次に示します。

```
#sql context Inventory;
Inventory sljConCtx = new Inventory(url);
java.sql.Connection jdbcConCtx = sqljConCtx.getConnection() ;
```

(5) 動的 SQL

SQLJ では静的 SQL だけ記述できるため，動的 SQL を実行する場合は JDBC の API を使用する必要があります。

(a) 動的 SQL の実行

動的 SQL は，JDBC の PreparedStatement オブジェクトを使用して実行します。

接続コンテキストの prepareStatement メソッドを，SQL を引数にして実行すると，返り値として PreparedStatement オブジェクトが返ります。

動的 SQL へのパラメタの設定には，PreparedStatement の set メソッドを使用します。また，SQL の実行には，PreparedStatement オブジェクトの execute メソッドを使用します。

動的 SQL の実行例を次に示します。

```
java.sql.PreparedStatement pstmt = con.prepareStatement(
          ”INSERT INTO FOO_TABLE VALUES(?, ?)”);
pstmt.setInt(1, 100);
pstmt.setString(2, ”テスト”);
pstmt.execute();
```

(b) 動的カーソルの検索

SQLJ では静的カーソルだけ使用できるため，動的カーソルを実行するには JDBC の API を使用する必要があります。

接続コンテキストの prepareStatement メソッドを，SELECT 文を示す文字列で実行すると，返り値として PreparedStatement オブジェクトが返ります。

パラメタの設定には，PreparedStatement の set メソッドを使用します。また，SQL の実行には，PreparedStatement オブジェクトの executeQuery メソッドを使用します。この executeQuery メソッドは，JDBC の結果セットを返します。

検索結果の受け取りには，結果セットの get メソッドを使用します。

動的カーソルの検索例を次に示します。

```
java.sql.PreparedStatement pstmt = con.prepareStatement(
          ”SELECT NAME, POINT FROM FOO_TABLE WHERE BAR=100”);
ResultSet rs = pstmt.executeQuery();
String name;
Integer point;
rs.next();
name = pstmt.getString(1);
point = pstmt.getInteger(2);
```

(c) DESCRIBE 文の実行

動的カーソルの各検索項目の列名及びデータ型を求めるためには，ResultSetMetaData オブジェクトを使用します。

ResultSetMetaData オブジェクトは，結果セットの getMetaData オブジェクトから取得できます。また，ResultSetMetaData オブジェクトの getColumnClassName メソッドで，各検索項目のデータ型を示す文字列を取得できます。

列名は getColumnName メソッドで取得できます。

対象とする検索項目は，番号（1〜）で指定します。また，列数は getColumnCount メソッドで取得できます。

DESCRIBE 文の実行例を次に示します。

```
java.sql.PreparedStatement pstmt = con.prepareStatement(
     ”SELECT * FROM FOO_TABLE”);
java.sql.ResultSetMetaData aMeta = pstmt.getMetaData();
int columCount = aMeta.getColumnCount();
Vector nameList = new Vector();
Vector classList = new Vector();
for(int i = 1; i <= columnCount; i++){
     nameList.addElement(aMeta.getColumnName(i));
     classList.addElement(a.Meta.getColumnClassName(i));
}
Vector dataList = new Vector();
rs.next();
for(int i = 1; i <= columnCount; i++){
     dataList.addElement(rs.getObject(i));
}
```

18.3.11　UAP の作成と実行

(1)　SQLJ トランスレータの実行方法

1．環境変数を設定する

HiRDB クライアントが IPF 版の UNIX 版の場合：

環境変数に次の内容を設定してください。なお，下線部はデフォルトのインストールディレクトリです。

- HiRDB/Developer's Kit のとき
  CLASSPATH=$CLASSPATH:/HiRDB/client/lib/pdsqlj.jar
- HiRDB/Run Time のとき
  CLASSPATH=$CLASSPATH:/HiRDB/client/lib/pdruntime.jar
  CLASSPATH=$CLASSPATH:/HiRDB/client/lib/pdnativert.jar

HiRDB クライアントが Windows 版の場合：

「コントロールパネル」−「システム」−「詳細」-「環境変数」に，次の内容を設定してください。なお，下線部はデフォルトのインストールディレクトリです。

- HiRDB/Developer's Kit のとき
  CLASSPATH=%CLASSPATH%:¥HiRDB¥UTL¥pdsqlj.jar
- HiRDB/Run Time のとき
  CLASSPATH=%CLASSPATH%¥HiRDB¥UTL¥pdruntime.jar
  CLASSPATH=%CLASSPATH%¥HiRDB¥UTL¥pdnativert.jar

2．SQLJ トランスレータを実行する

SQLJ トランスレータは，Java 仮想マシン上で動作します。

形式：

```
pdjava［オプション］ファイル名1.sqlj［ファイル名2.java］
```

説明：

オプション：

SQLJ トランスレータのオプションを次の表に示します。

ファイル名 1：

SQLJ を記述した UAP ソースファイルです。

ファイル名 2：

ポストソースファイルです。

ファイル名 1，ファイル名 2 にはパスを含んでもかまいません。ファイル名 2.java を指定しない場合は，ファイル名 1.java を指定したとみなされます。

表 18–8　SQLJ トランスレータのオプション

オプション	記述形式	説明
-dir	-dir=ディレクトリ名	ポストソースファイルを生成するディレクトリを指定します。
-d	-d=ディレクトリ名	
-status	-status	プリプロセスするときの内部状態を表示します。デバッグ用のオプションです。
-J	-J-オプション	SQLJ トランスレータ実行時の，Java 仮想マシンのオプションを指定します。
-version	-version	SQLJ トランスレータのバージョンを表示します。トランスレートは行いません。
-help	-help	オプションの説明を表示する場合に指定します。トランスレートは行いません。
-native	-native	ネイティブインタフェース用のポストソースを生成します。オプションを複数指定する場合は，必ず先頭に指定してください。

注 1

オプションを複数指定する場合は，スペースを入れて指定してください。

スタンダードインタフェース版の場合は二つまで，ネイティブインタフェース版の場合は三つ（-native 含む）まで指定できます。それ以上指定するとエラーとなります。

注 2

ネイティブインタフェース版を使用するための-native オプションは，必ず先頭に指定してください。2 番目以降に指定するとエラーとなります。

注 3

-help 又は-version オプションを指定した場合，ほかのオプションは無視されます。ただし，-help と-version を同時に指定した場合は，どちらも有効となります。

実行例：

実行例を次に示します。

- スタンダードインタフェース版の場合

  (例 1)pdjava ファイル名.sqlj

  (例 2)pdjava -dir=d:¥sqljsrc ファイル名.sqlj

- ネイティブインタフェース版の場合

  (例 1)pdjava -native ファイル名.sqlj

  (例 2)pdjava -native -dir=d:¥sqljsrc ファイル名.sqlj

(2)　UAP のコンパイルと実行

1．環境変数を設定する

「SQLJ トランスレータの実行方法」の 1.を参照してください。

2．ポストソースをコンパイルする

SQLJ トランスレータで生成したポストソースを，java コンパイラでコンパイルします。コンパイル時の形式を次に示します。

```
javac ファイル名2.java
```

3．CLASSPATH に JDBC のドライバのパスを設定する

JDBC のドライバのパスの設定方法については，「インストールと環境設定」を参照してください。

4．DriverManager を使用した DB 接続

DriverManager を使用した DB 接続については，「Driver クラス」を参照してください。

5．Java 仮想マシンで CLASS ファイルを実行する

Java 仮想マシンで Class ファイルを実行します。実行時の形式を次に示します。

```
java ファイル名2
```

18.3.12 スタンダードインタフェース版からネイティブインタフェース版への移行

スタンダードインタフェース版からネイティブインタフェース版へ SQLJ ソースを移行する場合，修正が必要になる部分があります。ネイティブインタフェース版に移行した場合の修正要否を次の表に示します。

表 18–9 スタンダードインタフェース版からネイティブインタフェース版へ移行した場合の修正要否

コマンド名	スタンダードインタフェース版	ネイティブインタフェース版	修正の要否
UAP(入力)ソース	ファイル名.sqlj	同じです。	−
UAP(出力)ソース	JAVA ソースファイル名.java プロファイル名.ser	JAVA ソースファイル.java	−
オプション	出力ファイル名指定など	同じです。	−
SQL 先頭子	#sql	同じです。	−
SQL 終了子	;	同じです。	−
SQL 宣言節	不要です。	同じです。	−
埋込み変数	:変数名	同じです。	−
宣言文	#sql context クラス名 #sql iterator クラス名	同じです。※1	−※2
接続コンテキストの作成	パラメタに JDBC 接続オブジェクトを指定できます。	同じです。	−
	パラメタに JDBC 接続オブジェクト以外を指定できます。	同じパラメタを取得するものはありません。	○※3
デフォルト接続コンテキストの使用	JP.co.Hitachi.soft.HiRDB.sqj.runtime. PrdbContext	JP.co.Hitachi.soft.HiRDB.pdjpp.runtime. PrdbContext	○※4
実行コンテキストの明示指定	sqlj.runtime.ExecutionContext	JP.co.Hitachi.soft.HiRDB.pdjpp.runtime. ExecutionContext	○※5
CAST 文の使用(JDBC 結果セットの受け取り)	実行できます。	実行できません。	○※6
動的結果セットの受け取り	実行できます。	実行できません。	○※7
データ型	byte[] java.math.BigDecimal java.lang.String	JP.co.Hitachi.soft.HiRDB.pdjpp.runtime. HiRDBBLOB JP.co.Hitachi.soft.HiRDB.pdjpp.runtime. HiRDBDECIMAL JP.co.Hitachi.soft.HiRDB.pdjpp.runtime. HiRDBCHAR など	○※8

コマンド名	スタンダードインタフェース版	ネイティブインタフェース版	修正の要否
同一反復子オブジェクト名を使用した異なる SELECT 文の実行	実行できます。	実行できません。	○※9

（凡例）

○：修正する必要があります。

－：修正する必要はありません。

注※1

名前反復子は使用できません。また，位置反復子を使用した場合は，インナークラスには使用できません。

注※2

名前反復子を使用する場合，及びインナークラスを使用する場合は，修正が必要となります。

注※3

接続処理を変更してください。詳細については，「ネイティブインタフェース版を使用した場合の HiRDB サーバへの接続」を参照してください。

注※4

パッケージ名の JP.co.Hitachi.soft.HiRDB.sqj.runtime.PrdbContext を JP.co.Hitachi.soft.HiRDB.pdjpp.runtime.PrdbContext に変更してください。

注※5

sqlj.runtime.ExecutionContext を JP.co.Hitachi.soft.HiRDB.pdjpp.runtime.ExecutionContext に変更してください。

注※6

CAST 文はトランスレート時にエラーとなります。そのため，CAST 文は削除してください。また，JDBC の結果セットは，SQLJ の反復子を使用しないで直接操作するように UAP を変更してください。

注※7

動的結果セットを受け取るためのメソッドがないため，java コンパイル時にエラーとなります。そのため，ExecutionContext.getNextResultSet()を発行している部分を削除してください。なお，動的結果セットを取得したい場合は，JDBC を直接使用するように UAP を変更してください。

注※8

byte[]は，BINARY 型として HiRDB サーバに要求します。HIRDB サーバのバージョンが 06-02 以前の場合に変更が必要です。

BigDecimal 型を受け取り変数に指定している場合は，精度 15，位取り 0 が設定されるため，それ以外の精度，位取りのときは変更が必要です。

String を入力変数に使用した場合，VARCHAR 型として HiRDB サーバに要求します。データ型を HiRDB サーバのデータ型に対応させたい場合は，変更が必要です。

注※9

同一の反復子オブジェクト名を使用して，異なる SELECT 文を実行することはできません。この場合，SELECT 文ごとに，反復子オブジェクト名を分ける必要があります。例を次に示します。

```
#sql iterator pos(HiRDBCHAR(10));
   :
pos positer = null
pos positer2 = null;
HiRDBCHAR out = null;
   :
#sql positer = {SELECT * FROM T1};
#sql {FETCH :positer INTO :out}
positer.close();

#sql positer2 = {SELECT * FROM T2};
#sql {FETCH :positer2 INTO :out}
positer2.close();
```

18.3.13　UAP 開発時の注意事項

マルチスレッドを使用した UAP を開発する場合，接続コンテキストにデフォルト接続コンテキストを使用しないでください。同一の接続コンテキストを複数のスレッドで使用すると，エラーとなります。

マルチスレッドを使用する場合は，必ず接続コンテキストを明示的に指定してください。接続コンテキストを明示的に指定した例を次に示します。

```
#sql context Ctx;

public class sample{
 public void main(String args[]){
   Ctx con = null;
   #sql [con] {CONNECT};                       //接続コンテキストの明示指定
   ..
   int data = 100;
   #sql [con] {INSERT INTO T1 VALUES(:data)}; //接続コンテキストの明示指定

   #sql [con] {DISCONNECT};                    //接続コンテキストの明示指定
 }
}
```

SQLJ ネイティブインタフェース版を使用する場合，SELECT 文の検索項目数と使用する反復子オブジェクトの列数が一致するようにしてください。一致しないと，不正にエラーとなることがあります。

18.4　ネイティブランタイム

ネイティブインタフェースで使用する SQLJ ランタイムライブラリを，**ネイティブランタイム**といいます。

ネイティブランタイムでは，次の機能を提供します。

- -native オプション指定時のコンパイルで使用するクラス，及びインタフェース
- HiRDB へのアクセス

18.4.1　パッケージの構成

ネイティブランタイムのパッケージ構成を次の表に示します。

表 18–10　ネイティブランタイムのパッケージ構成

パッケージ名称	収録内容
JP.co.Hitachi.soft.HiRDB.pdjpp.runtime	クラス，及びインタフェース
JP.co.Hitachi.soft.HiRDB.pdjpp.runtime.error	エラークラス

18.4.2　ネイティブランタイムの公開クラス一覧

ネイティブランタイムの公開クラス一覧を次の表に示します。

表 18–11　ネイティブランタイムの公開クラス一覧

パッケージ	クラス名，又はインタフェース名	機能
–	接続コンテキスト	SQLJ トランスレータの「#sql context クラス名;」で生成されるクラスです。SQLJ の接続コンテキストに該当します。
JP.co.Hitachi.soft.HiRDB.pdjpp.runtime	PrdbContext	デフォルト接続コンテキストです。SQLJ のデフォルト接続コンテキストに該当します。
JP.co.Hitachi.soft.HiRDB.pdjpp.runtime	ExecutionContext	実行コンテキストです。SQLJ の実行コンテキストに該当し，SQL の実行を管理するクラスです。
–	反復子（イテレータ）	SQLJ トランスレータの「#sql iterator クラス名;」で生成されるクラスです。SQLJ の反復子に該当します。
JP.co.Hitachi.soft.HiRDB.pdjpp.runtime	RTResultSet	結果セットオブジェクトです。JDBC の結果セットに該当し，結果を管理するクラスです。

パッケージ	クラス名，又はインタフェース名	機能
JP.co.Hitachi.soft.HiRDB.pdjpp.runtime	ForUpdate	反復子を使用したカーソル更新を使用する場合に，反復子宣言で implement するインタフェースです。
JP.co.Hitachi.soft.HiRDB.pdjpp.runtime	HiRDBCHAR	HiRDB の CHAR 型を表します。
JP.co.Hitachi.soft.HiRDB.pdjpp.runtime	HiRDBVARCHAR	HiRDB の VARCHAR 型を表します。
JP.co.Hitachi.soft.HiRDB.pdjpp.runtime	HiRDBNCHAR	HiRDB の NCHAR 型を表します。
JP.co.Hitachi.soft.HiRDB.pdjpp.runtime	HiRDBNVARCHAR	HiRDB の NVARCHAR 型を表します。
JP.co.Hitachi.soft.HiRDB.pdjpp.runtime	HiRDBMCHAR	HiRDB の MCHAR 型を表します。
JP.co.Hitachi.soft.HiRDB.pdjpp.runtime	HiRDBMVARCHAR	HiRDB の MVARCHAR 型を表します。
JP.co.Hitachi.soft.HiRDB.pdjpp.runtime	HiRDBDECIMAL	HiRDB の DECIMAL 型を表します。
JP.co.Hitachi.soft.HiRDB.pdjpp.runtime	HiRDBBLOB	HiRDB の BLOB 型を表します。
JP.co.Hitachi.soft.HiRDB.pdjpp.runtime	HiRDBBINARY	HiRDB の BINARY 型を表します。

（凡例）－：パッケージはありません。

18.4.3　クラス仕様

各クラスのメソッドと，フィールドの値について説明します。

(1)　JP.co.Hitachi.soft.HiRDB.pdjpp.runtime.HiRDBCHAR クラス

説明：

HiRDB の CHAR 型と対応します。

コンストラクタ：

戻り値	メソッド	機能説明
HiRDBCHAR	HiRDBCHAR(String s) throws SQLException	新しい HiRDBCHAR クラスを生成します。 指定した文字列の長さが 30,001 バイト以上の場合，SQLException が投入されます。
HiRDBCHAR	HiRDBCHAR(int len) throws SQLException	長さ len を持つ HiRDBCHAR クラスを返却します。 1 行検索，及び CALL 文の OUT パラメタに指定する場合に使用します。なお，入力変数に指定した場合，(半角スペース*len)を指定したとみなされます。指定した len が 1～30,000 の範囲でない場合，SQLException が投入されます。

メソッド：

戻り値	メソッド	機能説明
String	getString()	String オブジェクトを返却します。
int	length()	文字列の長さを返却します。

(2)　JP.co.Hitachi.soft.HiRDB.pdjpp.runtime.HiRDBVARCHAR クラス

説明：

HiRDB の VARCHAR 型と対応します。

コンストラクタ：

戻り値	メソッド	機能説明
HiRDBVARCHAR	HiRDBVARCHAR(String s) throws SQLException	新しい HiRDBVARCHAR クラスを生成します。 指定した文字列の長さが 32,001 バイト以上の場合，SQLException が投入されます。
HiRDBVARCHAR	HiRDBVARCHAR(int len) throws SQLException	長さ len を持つ HiRDBVARCHAR クラスを返却します。 1 行検索，及び CALL 文の OUT パラメタに指定する場合に使用します。なお，入力変数に指定した場合，(半角スペース*len)を指定したとみなされます。指定した len が 1～32,000 の範囲でない場合，SQLException が投入されます。

メソッド：

戻り値	メソッド	機能説明
String	getString()	String オブジェクトを返却します。
int	length()	文字列の長さを返却します。

(3)　JP.co.Hitachi.soft.HiRDB.pdjpp.runtime.HiRDBNCHAR クラス

説明：

HiRDB の NCHAR 型と対応します。

コンストラクタ：

戻り値	メソッド	機能説明
HiRDBNCHAR	HiRDBNCHAR(String s) throws SQLException	新しい HiRDBNCHAR クラスを生成します。 指定した文字列の長さが 15,001 文字以上の場合，SQLException が投入されます。
HiRDBNCHAR	HiRDBNCHAR(int len) throws SQLException	長さ len（len は全角の文字数）を持つ HiRDBNCHAR クラスを返却します。 1 行検索，及び CALL 文の OUT パラメタに指定する場合に使用します。なお，入力変数に指定すると，(全角スペー

戻り値	メソッド	機能説明
		ス*len)を指定したとみなされます。指定した len が 1～15,000 の範囲でない場合，SQLException が投入されます。

メソッド：

戻り値	メソッド	機能説明
String	getString()	String オブジェクトを返却します。
int	length()	文字列の長さを返却します。

(4)　JP.co.Hitachi.soft.HiRDB.pdjpp.runtime.HiRDBNVARCHAR クラス

説明：

HiRDB の NVARCHAR 型と対応します。

コンストラクタ：

戻り値	メソッド	機能説明
HiRDBNVARCHAR	HiRDBNVARCHAR(String s) throws SQLException	新しい HiRDBNVARCHAR クラスを生成します。 指定した文字列の長さが 16,001 文字以上の場合，SQLException が投入されます。
HiRDBNVARCHAR	HiRDBNVARCHAR(int len) throws SQLException	長さ len（len は全角の文字数）を持つ HiRDBNVARCHAR クラスを返却します。 1 行検索，及び CALL 文の OUT パラメタに指定する場合に使用します。なお，入力変数に指定すると，(全角スペース*len)を指定したとみなされます。指定した len が 1～16,000 の範囲でない場合，SQLException が投入されます。

メソッド：

戻り値	メソッド	機能説明
String	getString()	String オブジェクトを返却します。
int	length()	文字列の長さを返却します。

(5)　JP.co.Hitachi.soft.HiRDB.pdjpp.runtime.HiRDBMCHAR クラス

説明：

HiRDB の MCHAR 型と対応します。

コンストラクタ：

戻り値	メソッド	機能説明
HiRDBMCHAR	HiRDBMCHAR(String s) throws SQLException	新しい HiRDBMCHAR クラスを生成します。 指定した文字列の長さが 30,001 バイト以上の場合，SQLException が投入されます。

戻り値	メソッド	機能説明
HiRDBMCHAR	HiRDBMCHAR(int len) throws SQLException	長さ len を持つ HiRDBMCHAR クラスを返却します。 1 行検索，及び CALL 文の OUT パラメタに指定する場合に使用します。なお，入力変数に指定すると，(半角スペース*len)を指定したとみなされます。指定した len が 1～30,000 の範囲でない場合，SQLException が投入されます。

メソッド：

戻り値	メソッド	機能説明
String	getString()	String オブジェクトを返却します。
int	length()	文字列の長さを返却します。

(6)　JP.co.Hitachi.soft.HiRDB.pdjpp.runtime.HiRDBMVARCHAR クラス

説明：

HiRDB の MVARCHAR 型と対応します。

コンストラクタ：

戻り値	メソッド	機能説明
HiRDBMVARCHAR	HiRDBMVARCHAR(String s) throws SQLException	新しい HiRDBMVARCHAR クラスを生成します。 指定した文字列の長さが 32,001 バイト以上の場合，SQLException が投入されます。
HiRDBMVARCHAR	HiRDBMVARCHAR(int len) throws SQLException	長さ len を持つ HiRDBMVARCHAR クラスを返却します。 1 行検索，及び CALL 文の OUT パラメタに指定する場合に使用します。なお，入力変数に指定すると，(半角スペース*len)を指定したとみなされます。指定した len が 1～32,000 の範囲でない場合，SQLException が投入されます。

メソッド：

戻り値	メソッド	機能説明
String	getString()	String オブジェクトを返却します。
int	length()	文字列の長さを返却します。

(7)　JP.co.Hitachi.soft.HiRDB.pdjpp.runtime.HiRDBBLOB クラス

説明：

HiRDB の BLOB 型と対応します。

コンストラクタ：

戻り値	メソッド	機能説明
HiRDBLOB	HiRDBBLOB(byte[] b)	新しい HiRDBBLOB クラスを生成します。

戻り値	メソッド	機能説明
HiRDBBLOB	HiRDBBLOB(int len) throws SQLException	長さ len を持つ HiRDBBLOB クラスを返却します。 1 行検索，及び CALL 文の OUT パラメタに指定する場合に使用します。なお，入力変数に指定すると，(数字の 0(0x30)*len)を指定したとみなされます。指定した len が 0 以下の場合，SQLException が投入されます。

メソッド：

戻り値	メソッド	機能説明
byte[]	getBytes[]	byte[]を返却します。
int	length()	byte[]の長さを返却します。

(8)　JP.co.Hitachi.soft.HiRDB.pdjpp.runtime.HiRDBBINARY クラス

説明：

HiRDB の BINARY 型と対応します。

コンストラクタ：

戻り値	メソッド	機能説明
HiRDBBINARY	HiRDBBINARY(byte[] b)	新しい HiRDBBINARY クラスを生成します。
HiRDBBINARY	HiRDBBINARY(int len) throws SQLException	長さ len を持つ HiRDBBINARY クラスを返却します。 1 行検索，及び CALL 文の OUT パラメタに指定する場合に使用します。なお，入力変数に指定すると，(数字の 0(0x30)*len)を指定したとみなされます。指定した len が 0 以下の場合，SQLException が投入されます。

メソッド：

戻り値	メソッド	機能説明
byte[]	getBytes()	byte[]を返却します。
int	length()	byte[]長さを返却します。

(9)　JP.co.Hitachi.soft.HiRDB.pdjpp.runtime.HiRDBDECIMAL クラス

説明：

HiRDB の DECIMAL 型と対応します。

コンストラクタ：

戻り値	メソッド	機能説明
HiRDBDECIMAL	HiRDBDECIMAL(String s) throws SQLException	新しい HiRDBDECIMAL クラスを生成します。

戻り値	メソッド	機能説明
		引数の文字列に数値，ピリオド，及び符号以外の文字列を含む場合，又は文字列から求められる精度，位取りが 39 以上の場合は，SQLException が発生します。
HiRDBDECIMAL	HiRDBDECIMAL (java.math.BigDecimal) throws SQLException	新しい HiRDBDECIMAL クラスを生成します。 引数の精度及び位取りが 39 以上の場合，SQLException が発生します。
HiRDBDECIMAL	HiRDBDECIMAL(int x,int y) throws SQLException	精度 x，位取り y の HiRDBDECIMAL クラスを返却します。 1 行検索，及び CALL 文の OUT パラメタに指定する場合に使用します。なお，入力変数に指定すると，0 を指定したものとみなされます。x が 1～38 の範囲外，y が 0～38 の範囲外で，かつ x ＜ y の場合は，SQLException が発生します。

メソッド：

戻り値	メソッド	機能説明
String	getString()	String オブジェクトを返却します。
java.math.BigDecimal	getBigDecimal()	java.math.BigDecimal オブジェクトを返却します。
int	precision()	精度を返却します。
int	scale()	位取りを返却します。

18.4.4　ネイティブインタフェースを使用したコーディング例

(1)　データの挿入と検索

データの挿入と検索をするコーディング例（sample1.sqlj）を次に示します。

```
import java.sql.*;
import JP.co.Hitachi.soft.HiRDB.pdjpp.runtime.*;
//反復子(カーソル)宣言
#sql iterator Pos(int,HiRDBCHAR(10),HiRDBNCHAR(5),HiRDBDECIMAL(10,5));

public class sample1{
  public static void main(String args[]){

    //接続及びテーブル作成
    try{
      #sql{CONNECT};    //クライアント環境変数を参照し接続します
      #sql{CREATE TABLE SAMPLE1(c1 int,c2 char(10),c3 nchar(5),c4 decimal(10,5))};
    }catch(SQLException e){System.out.println(e.getMessage());};

    //データのインサート
    try{
      int InInt = 100;
```

```
      HiRDBCHAR InChar = new HiRDBCHAR("CHAR");
      HiRDBNCHAR InNchar = new HiRDBNCHAR("ＮＣＨＡＲ");
      HiRDBDECIMAL InDecimal = new HiRDBDECIMAL("12345.678");

      #sql{INSERT INTO SAMPLE1 VALUES(:InInt,:InChar,:InNchar,:InDecimal)};
      #sql{COMMIT};
    }catch(SQLException e){System.out.println(e.getMessage());};

    //データの検索(FETCH)
    try{
      Pos sampleCur = null;
      int OutInt = 0;
      HiRDBCHAR OutChar = null;
      HiRDBNCHAR OutNchar = null;
      HiRDBDECIMAL OutDecimal = null;

      #sql sampleCur  = {SELECT * FROM SAMPLE1};
      while(true){
        #sql {FETCH :sampleCur INTO :OutInt ,:OutChar ,:OutNchar ,:OutDecimal };
        if(sampleCur.endFetch()) break;
          System.out.println("c1="+ OutInt +" c2="+ OutChar.getString() +
             " c3="+ OutNchar.getString() + " c4="+ OutDecimal.getString());
      }
    }catch(SQLException e){System.out.println(e.getMessage());};
    try{#sql{DISCONNECT};}catch(SQLException e){System.out.println(e.getMessage());}
  }
}
```

(2)　データの挿入と 1 行検索

データの挿入と 1 行検索をするコーディング例（sample2.sqlj）を次に示します。

```
import java.sql.*;
import JP.co.Hitachi.soft.HiRDB.pdjpp.runtime.*;

public class sample2{
  public static void main(String args[]){
    String userid = "user1";
    String passwd = "user1";

    //接続及びテーブル作成
    try{
    //指定された認可識別子，パスワードで接続します
      #sql{CONNECT USER :userid USING :passwd};
      #sql{CREATE TABLE SAMPLE1(c1 int,c2 char(10),c3 nchar(5),c4 decimal(10,5))};
    }catch(SQLException e){System.out.println(e.getMessage());};

    //データのインサート
    try{
      int InInt = 100;
      HiRDBCHAR InChar = new HiRDBCHAR("CHAR");
      HiRDBNCHAR InNchar = new HiRDBNCHAR("ＮＣＨＡＲ");
      HiRDBDECIMAL InDecimal = new HiRDBDECIMAL("12345.678");

      #sql{INSERT INTO SAMPLE1 VALUES(:InInt,:InChar,:InNchar,:InDecimal)};
      #sql{COMMIT};
```

```
    }catch(SQLException e){System.out.println(e.getMessage());};

    //データの検索(1行検索)
    try{
      //出力変数の宣言
      int OutInt = 0;
      HiRDBCHAR OutChar = new HiRDBCHAR(10);
      HiRDBNCHAR OutNchar = new HiRDBNCHAR(5);
      HiRDBDECIMAL OutDecimal = new HiRDBDECIMAL(10,5);

      #sql {SELECT * INTO :OutInt,:OutChar,:OutNchar,:OutDecimal FROM SAMPLE1};
      System.out.println("c1="+ OutInt +" c2="+ OutChar.getString() +
        " c3="+ OutNchar.getString() + " c4="+ OutDecimal.getString());
    }catch(SQLException e){System.out.println(e.getMessage());};
    try{#sql{DISCONNECT};}catch(SQLException e){System.out.println(e.getMessage());}
  }
}
```

(3) CALL 文の実行

CALL 文を実行するコーディング例（sample3.sqlj）を次に示します。

```
import java.sql.*;
import JP.co.Hitachi.soft.HiRDB.pdjpp.runtime.*;

public class sample3{
  public static void main(String args[]){

    Integer PInteger1 = new Integer(99);
    Integer PInteger2 = new Integer(100);
    Integer PInteger3 = new Integer(101);try{
      #sql {CONNECT};
    }catch(SQLException e){System.out.println(e.getMessage());}

    try{
      #sql {DROP PROCEDURE PROCSQLJ};
      #sql {DROP TABLE PROCTABLE};
    }catch(SQLException e1){}

    try{
      #sql {CREATE TABLE PROCTABLE(c1 int, c2 int)};
      #sql {CREATE PROCEDURE PROC1(in p1 int,out p2 int,inout p3 int)
        begin
        insert into PROCTABLE values(p1,p3);
        select * into p2,p3 from PROCTABLE;
        end};
      #sql {COMMIT};
    }catch(SQLException e){System.out.println(e.getMessage());}

    try{
      #sql {CALL PROC1(in :PInteger1 ,out :PInteger2 ,inout :PInteger3 )};
    }catch(SQLException e){System.out.println(e.getMessage());}

    System.out.println("INパラメタPInteger1 = " + PInteger1 );
    System.out.println("OUTパラメタPInteger2 = " + PInteger2 );
    System.out.println("INOUTパラメタPInteger3 = " + PInteger3 );
```

```
    try{#sql {DISCONNECT};}catch(SQLException e){System.out.println(e.getMessage());}
  }
}
```

(4) カーソルを使用した更新

カーソルを使用した更新をするコーディング例（sample4.sqlj）を次に示します。

```
import java.sql.*;
import JP.co.Hitachi.soft.HiRDB.pdjpp.runtime.*;
#sql iterator iterP implements JP.co.Hitachi.soft.HiRDB.pdjpp.runtime.ForUpdate(short);

public class sample4{
  public static void main(String args[]){
    iterP positer = null;
    iterP positer2 = null;
    short indata;
    short indata2 = 0;
    short indata3 = 999;
    try{
      #sql {CONNECT};
      #sql {DROP TABLE CURTABLE};
    }catch(SQLException e){System.out.println(e.getMessage());}
    //テーブル作成
    try{#sql {CREATE TABLE CURTABLE(c1 smallint)};
    }catch(SQLException e){System.out.println(e.getMessage());}
    //データのインサート
    for(short i = 0;i < 5;i++){
      indata = i;
      try{#sql{INSERT INTO CURTABLE VALUES(:indata)};}catch(SQLException e){}
    }
    //SELECTの実行とカーソルを使用した更新
    try{
      #sql positer = {SELECT * FROM CURTABLE};
    }catch(SQLException e){}
    try{
      while(true){
        #sql {FETCH :positer INTO :indata2};
        if(positer.endFetch()) break;
        System.out.println(indata2);
        #sql { UPDATE CURTABLE SET C1=:indata3 WHERE CURRENT OF :positer };
      }
    }catch(SQLException e){e.getMessage();}
    //更新結果の確認
    try{#sql positer2 = {SELECT * FROM CURTABLE};}catch(SQLException e){}
    try{
      while(true){
        #sql {FETCH :positer2 INTO :indata2};
        if(positer2.endFetch()) break;
        System.out.println(indata2);
      }
    }catch(SQLException e){System.out.println(e.getMessage());}
    try{#sql{DISCONNECT};}catch(SQLException e){}
```

```
    }
}
```

付録

付録 A　SQL 連絡領域

SQL を実行すると，HiRDB は SQL が正常に実行されたかどうかを示すリターンコードと関連する情報を UAP に返します。これらの情報を受け取るための領域を SQL 連絡領域といいます。ここでは，SQL 連絡領域の構成と内容，及び SQL 連絡領域の展開について説明します。

なお，SQL 連絡領域の使用方法については，「SQL のエラーの判定と処置」を参照してください。

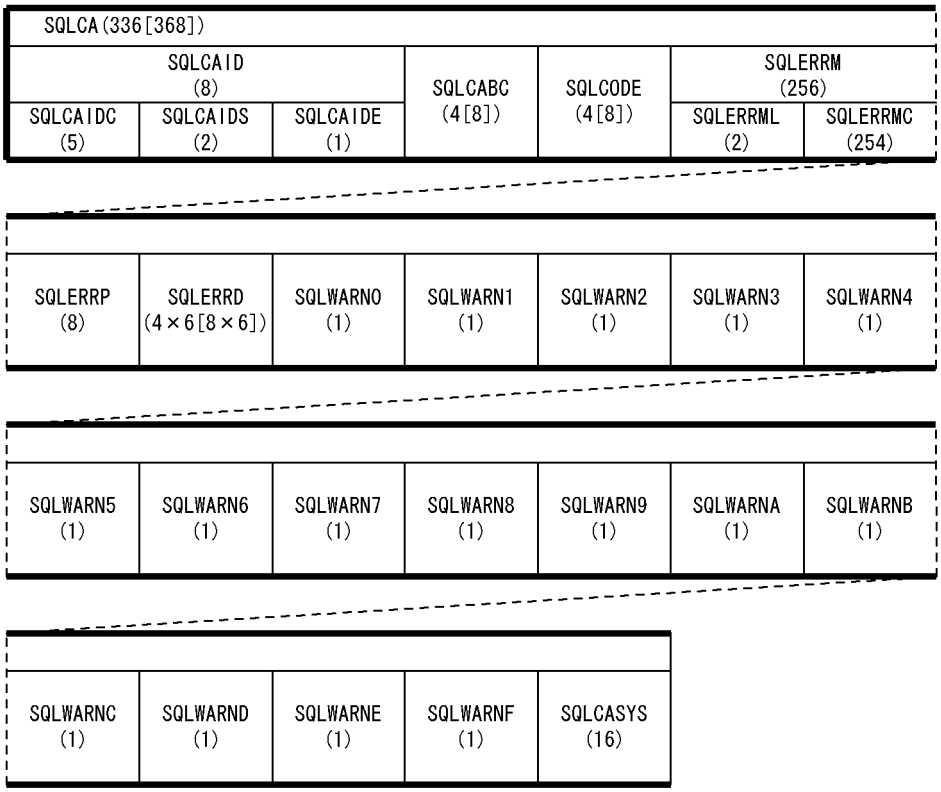
付録 A.1　SQL 連絡領域の構成と内容

SQL 実行時の情報を受け取る領域の構成，及び内容について説明します。

(1)　SQL 連絡領域の構成

SQL 連絡領域の構成を次の図に示します。

図 A–1　SQL 連絡領域の構成



注 1

( )内は領域の長さ(単位：バイト)を示します。

注 2

( )内の[ ]の値は，64 ビットモードの場合の長さを示します。なお，64 ビットモードの Windows の場合，SQLCA は 336 バイトとなります。

注 3

64 ビットモードでの SQLCABC，SQLCODE，及び SQLERRD の長さは，プラットフォームごとの long 型のサイズになります。

(2)　SQL 連絡領域の内容

SQL 連絡領域の内容を次の表に示します。

表 A–1　SQL 連絡領域の内容

レベル番号※1	連絡領域名	データ型	長さ(バイト)	内　　容
1	SQLCA	−	336 [368]	SQL 連絡領域全体の名前を表します。
2	SQLCAID	−	8	SQLCAIDC 領域，SQLCAIDS 領域，及び SQLCAIDE 領域の内容を値として持ちます。
3	SQLCAIDC	char	5	SQL 連絡領域である文字列('SQLCA')が設定されます。
3	SQLCAIDS	char	2	HiRDB が使用します。
3	SQLCAIDE	char	1	HiRDB が使用します。
2	SQLCABC	long	4 [8]※3	SQL 連絡領域の長さ(336 [368]バイト)が設定されています。
2	SQLCODE	long	4 [8]※3	SQL 実行後に HiRDB から返されるリターンコードを受け取る領域です。リターンコードには次に示す意味があります。 負：正常に終了していません ０：正常に終了しました 正：正常に終了したが，メッセージ情報があります。 リターンコードに対応するメッセージについては，マニュアル「HiRDB メッセージ」を参照してください。なお，リターンコードに対応するメッセージは，次のように検索します。 リターンコード→対応するメッセージ ID -yyy → KFPA11yyy -1yyy → KFPA19yyy -3yyy → KFPA18yyy yyy → KFPA12yyy 3yyy → KFPA13yyy (例) <リターンコード>→<メッセージ ID> -125 → KFPA11125 -1200 → KFPA19200 -3200 → KFPA18200 100 → KFPA12100 3010 → KFPA13010

レベル番号※1	連絡領域名	データ型	長さ(バイト)	内　　容
2	SQLERRM	−	256	SQLERRML 領域，及び SQLERRMC 領域の内容を値として持ちます。なお，SQLCODE 領域に返されるリターンコードの正負によって次に示すように異なります。 • リターンコードが負の場合，誤りの箇所や原因を示す文字列を返すときがある。 • リターンコードが正の場合，メッセージ情報を示す文字列を返すときがある。
3	SQLERRML	short	2	SQLERRMC 領域に返されるメッセージの長さを示します。
3	SQLERRMC	char	254	SQLCODE 領域に返されるリターンコードに対応するメッセージが格納される領域です。設定内容については，マニュアル「HiRDB メッセージ」を参照してください。
2	SQLERRP	char	8	HiRDB が使用します。
2	SQLERRD	long	4※3×6	HiRDB の内部状態を示す領域で，データ型が long の 6 個の配列です。 • SQLERRD[0]:未使用 • SQLERRD[1]:未使用 • SQLERRD[2]: **SQLCODE の値が 0，又は正の場合：** 次のどれかの値が返されます。 ・SELECT 文で取り出した行数 ・UPDATE 文で更新した行数 ・DELETE 文で削除した行数 ・INSERT 文で挿入した行数 ・FETCH 文で取り出した行数 ・ASSIGN LIST 文で作成した行数 ・ASSIGN LIST 文で名称変更したリストの行数※4 上記以外の SQL を実行する場合は，この領域は使用しないでください。 **SQLCODE の値が負の場合：** 配列を使った更新時にエラーとなった配列要素を示す値が返されます。 • SQLERRD[3]:システムが使用します。 • SQLERRD[4]:システムが使用します。 • SQLERRD[5]:未使用
2	SQLWARN0	char	1	SQLWARN1～F の領域のどれかに警告フラグ('W')が設定された場合に'W'が設定されます。
2	SQLWARN1	char	1	文字データの検索で，データを受け取る埋込み変数の長さが短いために，切り捨てられた値を受け取った場合，'W'が設定されます。

レベル番号※1	連絡領域名	データ型	長さ(バイト)	内　　容
				又は，繰返し列の検索で，データを受け取る埋込み変数の要素数が足りなかったために，要素が切り捨てられた値を受け取った場合，'W'が設定されます。 それ以外は空白が設定されます。
2	SQLWARN2	char	1	集合関数の処理でナル値を無視した場合に'W'が設定されます。それ以外は空白が設定されます。 ただし，次のどちらかの場合は，集合関数の処理でナル値を無視しても空白が設定されることがあります。 • ナル値を除外値とするインデクスを定義した表を検索する場合 • グループ分け高速化機能を使用する場合
2	SQLWARN3	char	1	検索結果の列の数と，その検索結果を受け取る埋込み変数の数が一致しない場合，'W'が設定されます。それ以外は空白が設定されます。
2	SQLWARN4	char	1	WHERE 句がない UPDATE 文，又は WHERE 句がない DELETE 文が実行された場合，'W'が設定されます。それ以外は空白が設定されます。
2	SQLWARN5	char	1	予備
2	SQLWARN6	char	1	暗黙的にトランザクションが取り消された場合に'W'が設定されます。それ以外は空白が設定されます。
2	SQLWARN7	char	1	UPDATE 文で，SET 句又は DELETE 句に添字付きの繰返し列を指定した場合，更新する行に要素がなくてその更新が無視されたときに，'W'が設定されます。 それ以外は空白が設定されます。
2	SQLWARN8	char	1	予備
2	SQLWARN9	char	1	予備
2	SQLWARNA※2	char	1	日付演算の結果，存在しない日付が現れると，その月の最終日に修正した場合　'W'　が設定されます。それ以外は空白が設定されます。
2	SQLWARNB※2	char	1	SQL 実行時の演算途中でオーバフロー，又は 0 除算が発生し，その演算結果をナル値とした場合，'W'が設定されます。それ以外は空白が設定されます。
2	SQLWARNC※2	char	1	日付演算の結果の日間隔中の日の部分が，00～99 の範囲内にない場合，'W'が設定されます。それ以外の場合，空白'△'になります。
2	SQLWARND	char	1	HiRDB が使用します。
2	SQLWARNE	char	1	予備
2	SQLWARNF	char	1	予備

レベル番号※1	連絡領域名	データ型	長さ(バイト)	内　　容
2	SQLCASYS	char	16	HiRDB が使用します。

(凡例)　−：該当しません。

注

長さの[ ]内の値は，64 ビットモードの場合の長さです。なお，64 ビットモードの Windows の場合，SQLCA は 336 バイトとなります。

注※1

表のレベル番号は，SQL 連絡領域の包含関係を示します。例えば，レベル番号 1 の連絡領域はレベル番号 2 の連絡領域で構成されることを示します。

注※2

ソート処理を含む SQL 文，又は EXISTS 述語を使用した SQL 文を実行した場合，最初の FETCH 文で'W'を返します。

また，HiRDB/パラレルサーバ環境のとき，WHERE 句で警告を出す要因が発生した場合，'W'を返す行が不定となります。

注※3

64 ビットモードの場合，プラットフォームごとの long 型のサイズになります。

注※4

システム共通定義で pd_list_rowcount_in_rename オペランドに Y を指定した場合だけ参照してください。

付録 A.2　SQL 連絡領域の展開

SQL 連絡領域は，SQL プリプロセサが高級言語のソースプログラム中に展開するため，UAP 内に記述する必要はありません。

ここでは，SQL プリプロセサがソースプログラム中に展開した SQL 連絡領域の形を示します。

(1)　C 言語の場合

C 言語の場合の SQL 連絡領域の展開形を次に示します。

```
#define  SQLCAIDE sqlca.sqlcaide
#define  SQLCODE  sqlca.sqlcode
#define  SQLERRML sqlca.sqlerrml
#define  SQLERRMC sqlca.sqlerrmc
#define  SQLERRMD sqlca.sqlerrmd
#define  SQLERRD0 sqlca.sqlerrd[0]
#define  SQLERRD1 sqlca.sqlerrd[1]
#define  SQLERRD2 sqlca.sqlerrd[2]
```

```
#define  SQLERRD3 sqlca.sqlerrd[3]
#define  SQLERRD4 sqlca.sqlerrd[4]
#define  SQLERRD5 sqlca.sqlerrd[5]
#define  SQLWARN0 sqlca.sqlwarn0
#define  SQLWARN1 sqlca.sqlwarn1
#define  SQLWARN2 sqlca.sqlwarn2
#define  SQLWARN3 sqlca.sqlwarn3
#define  SQLWARN4 sqlca.sqlwarn4
#define  SQLWARN5 sqlca.sqlwarn5
#define  SQLWARN6 sqlca.sqlwarn6
#define  SQLWARN7 sqlca.sqlwarn7
#define  SQLWARN8 sqlca.sqlwarn8
#define  SQLWARN9 sqlca.sqlwarn9
#define  SQLWARNA sqlca.sqlwarna
#define  SQLWARNB sqlca.sqlwarnb
#define  SQLWARNC sqlca.sqlwarnc
#define  SQLWARND sqlca.sqlwarnd
#define  SQLWARNE sqlca.sqlwarne
#define  SQLWARNF sqlca.sqlwarnf
typedef struct sqlca {
 char       sqlcaidc[5];    /* 表ＩＤ                */
 char       sqlcaids[2];    /* ＨｉＲＤＢで使用      */
 char       sqlcaide;       /* ＨｉＲＤＢで使用      */
 long       sqlcabc;        /* ＳＱＬＣＡの領域長    */
 long       sqlcode;        /* ＳＱＬＣＯＤＥ        */
 short      sqlerrml;       /* メッセージの有効長    */
 char       sqlerrmc[254];  /* メッセージテキスト    */
 char       sqlerrp[8];     /* ＨｉＲＤＢで使用      */
 long       sqlerrd[6];     /* ＨｉＲＤＢ内部状態    */
 char       sqlwarn0;       /* 警告情報有無フラグ    */
 char       sqlwarn1;       /* 警告情報１            */
 char       sqlwarn2;       /* 警告情報２            */
 char       sqlwarn3;       /* 警告情報３            */
 char       sqlwarn4;       /* 警告情報４            */
 char       sqlwarn5;       /* 警告情報５            */
 char       sqlwarn6;       /* 警告情報６            */
 char       sqlwarn7;       /* 警告情報７            */
 char       sqlwarn8;       /* 警告情報８            */
 char       sqlwarn9;       /* 警告情報９            */
 char       sqlwarna;       /* 警告情報１０          */
 char       sqlwarnb;       /* 警告情報１１          */
 char       sqlwarnc;       /* 警告情報１２          */
 char       sqlwarnd;       /* 警告情報１３（予備）  */
 char       sqlwarne;       /* 警告情報１４（予備）  */
 char       sqlwarnf;       /* 警告情報１５（予備）  */
 char       sqlcasys1[16];  /* 予備                  */
} SQLCA;
extern SQLCA sqlca;
```

(2)　COBOL 言語の場合

COBOL 言語の場合の SQL 連絡領域の展開形を次に示します。

```
01  SQLCA IS EXTERNAL.
  02  SQLCAID    PIC X(8).
  02  FILLER    REDEFINES SQLCAID.
```

```
      03  SQLCAIDC  PIC X(5).
      03  SQLCAIDS  PIC X(2).
      03  SQLCAIDE  PIC X(1).
    02  SQLCABC     PIC S9(9) COMP.
    02  SQLCODE     PIC S9(9) COMP.
    02  SQLERRM.
      03  SQLERRML  PIC S9(4) COMP.
      03  SQLERRMC  PIC X(254).
    02  SQLERRP     PIC X(8).
    02  SQLERRD     PIC S9(9) COMP OCCURS 6 TIMES.
    02  SQLWARN.
      03  SQLWARN0  PIC X.
      03  SQLWARN1  PIC X.
      03  SQLWARN2  PIC X.
      03  SQLWARN3  PIC X.
      03  SQLWARN4  PIC X.
      03  SQLWARN5  PIC X.
      03  SQLWARN6  PIC X.
      03  SQLWARN7  PIC X.
    02  SQLEXT.
      03  SQLWARN8  PIC X.
      03  SQLWARN9  PIC X.
      03  SQLWARNA  PIC X.
      03  SQLWARNB  PIC X.
      03  SQLWARNC  PIC X.
      03  SQLWARND  PIC X.
      03  SQLWARNE  PIC X.
      03  SQLWARNF  PIC X.
    02  SQLCASYS1   PIC X(16).
```

付録 B　SQL 記述領域

UAP 実行時に動的に SQL を組み立てて実行する場合，その SQL の実行に必要な入出力変数(データの受け渡し領域)の個数や属性なども UAP 実行時にしか決まらないことがあります。そのために，入出力変数を UAP の実行時に動的に決定して，その情報(個数，属性，番地など)を HiRDB に通知する領域が必要です。

SQL 記述領域は，UAP 実行時に動的に決定した入出力変数の情報を記述して，OPEN 文，FETCH 文，又は EXECUTE 文でシステムに通知するための領域です。また，動的に実行する場合，前処理した SQL の検索項目，又は?パラメタの情報を DESCRIBE 文で指定して受け取るために SQL 記述領域を使用することもできます。

SQL 記述領域を使用できる UAP の記述言語については，「UAP の記述」を参照してください。

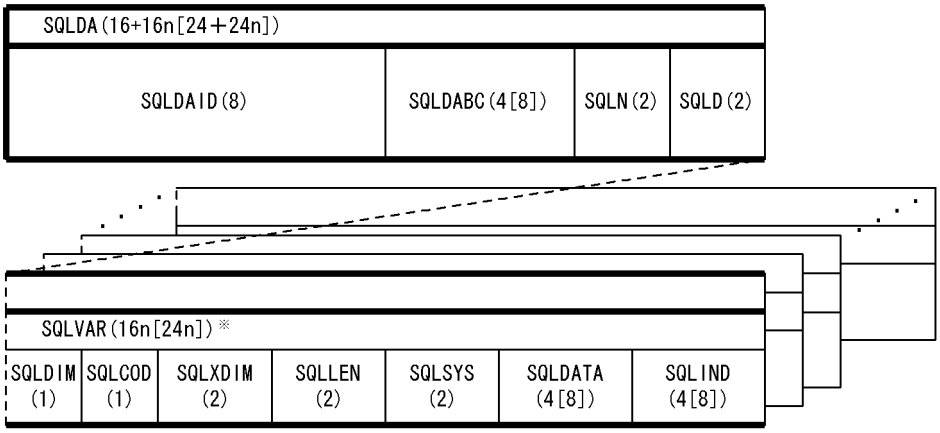
付録 B.1　SQL 記述領域の構成と内容

UAP 実行時に動的に決定した入出力変数の情報を記述する領域の構成，及び内容について説明します。

(1)　SQL 記述領域の構成

SQL 記述領域の構成を次の図に示します。

図 B–1　SQL 記述領域の構成



注 1

( )内は，領域の長さ(単位：バイト)を示します。

注 2

n は SQLN に指定した SQLVAR の数を示します。

注 3

( )の[ ]内の値は，64 ビットモードの場合の長さです。64 ビットモードの Windows の場合，SQLDA は 16 ＋ 24n バイトとなります。

注 4

64 ビットモードでの SQLDABC の長さは，プラットフォームごとの long 型のサイズになります。

注※

BLOB 型，及び BINARY 型のデータを使用する場合，領域名は SQLVAR_LOB となり，SQLDIM(1)，SQLCOD(1)，SQLXDIM(2)，SQLLOBLEN(4)，SQLDATA(4 [8])，SQLLOBIND(4 [8])から構成されます。

SQLVAR_LOB 領域は，SQLVAR 領域で定義し，BLOB 型のデータ入出力時に SQLVAR 領域に上書きする形で使用してください。SQLVAR_LOB の内容については，表「SQLVAR_LOB の内容」を参照してください。

(2)　SQL 記述領域の内容

SQL 記述領域の内容を次の表に示します。

表 B–1　SQL 記述領域の内容

レベル番号※1	記述領域名	データ型	長さ(バイト)	値の設定元	内　　容
1	SQLDA	−	16+16n [24 ＋ 24n]	−	SQL 記述領域全体の名前です。
2	SQLDAID	char	8	HiRDB	SQLDA を示す ID 'SQLDA△△△'です。DESCRIBE 文，又は DESCRIBE TYPE 文を発行したとき格納されます。
2	SQLDABC	long	4 [8]※6	HiRDB	SQLDA の長さです。DESCRIBE 文，又は DESCRIBE TYPE 文を発行したとき格納されます。
2	SQLN※2	short	2	UAP	SQLDA の領域を確保したとき，又は SQLDA を使用するとき，確保した SQLDA 領域の SQLVAR の個数(1～30000)を指定します。
				HiRDB	DESCRIBE 文，又は DESCRIBE TYPE 文を発行したとき SQLDA の領域が不足(SQLN<SQLD)する場合，2 進数の 0 が格納されます。
2	SQLD	short	2	UAP	OPEN 文，又は EXECUTE 文を発行するとき，USING 句に指定する SQL 記述領域の SQLD には，入力?パラメタ数を指定します。 EXECUTE 文を発行するとき，INTO 句に指定する SQL 記述領域の SQLDA には，出力?パラメタ数を指定します。FETCH 文を発行するときは，検索項目数(1～4000)を指定します。
				HiRDB	DESCRIBE[OUTPUT]文を発行したとき，2 進数の 0，検索項目数，又は出力?パラメタ数が設定されます。 • 0：前処理した SQL が SELECT 文以外で，かつ出力?パラメタがある CALL 文以外

レベル番号※1	記述領域名	データ型	長さ(バイト)	値の設定元	内　　容
					• 検索項目数：前処理した SQL が SELECT 文 • 出力?パラメタ数：前処理した SQL が CALL 文 DESCRIBE INPUT 文を発行したとき，入力?パラメタ数が設定されます。 DESCRIBE TYPE 文を発行したとき，受け取ろうとしたユーザ定義型と継承している，上位のユーザ定義型の全構成要素数が設定されます。ただし，構成要素数が 30000 を超えた場合，30001 が設定されます。
2	SQLVAR	−	16n [24n]	−	SQLDIM，SQLCOD，SQLXDIM，SQLLEN，SQLSYS，SQLDATA，及び SQLIND から構成される領域。この領域は，SQLN で指定した個数，又はそれ以上の領域として繰り返し定義する必要があります。
3	SQLDIM	unsigned char	1	−	未使用
3	SQLCOD	unsigned char	1	UAP	EXECUTE 文，OPEN 文,又は FETCH 文を発行するときデータコード※3 を指定します。
				HiRDB	DESCRIBE 文，又は DESCRIBE TYPE 文を発行後，データコード※3 が格納されます。
3	SQLXDIM	short	2	UAP	EXECUTE 文，OPEN 文，又は FETCH 文を発行するとき，SQLDA で指定する変数の領域の構造によって，次の値を指定します。 単純構造の場合： 1 繰返し構造の場合： 2～30000（領域の最大要素数を示す整数） データ領域の構造については，「SQL のデータ型とデータ記述」を参照してください。
				HiRDB	DESCRIBE 文，又は DESCRIBE TYPE 文を発行したとき，検索項目又は？パラメタの構造によって，次の値が設定されます。 単純構造の場合： 1 繰返し構造の場合： 2～30000（領域の最大要素数を示す整数）
3	SQLLEN※3※4	short	2	UAP	EXECUTE 文，OPEN 文，又は FETCH 文を発行するときの埋込み変数の長さ※3 を指定します。

レベル番号※1	記述領域名	データ型	長さ(バイト)	値の設定元	内　　容
				HiRDB	DESCRIBE 文，又は DESCRIBE TYPE 文を発行したとき，データの定義長※3 が格納されます。
3	SQLSYS	short	2	UAP	EXECUTE 文，OPEN 文，又は FETCH 文を発行するとき，次の値を指定します。 • 繰返し構造，又は配列構造の可変長の文字列型（VARCHAR，NVARCHAR，MVARCHAR）を指定した場合，ギャップを含む 1 要素の領域の長さ • 上記以外の場合は 0
				HiRDB	DESCRIBE 文，又は DESCRIBE TYPE 文を発行したとき，0 が設定されます。
3	SQLDATA※5	unsigned char *	4 [8]	UAP	EXECUTE 文，又は OPEN 文を発行するとき，?パラメタ値※5 が格納されているデータ領域のアドレスを指定します。FETCH 文を発行するとき，データを受け取るためのデータ領域のアドレスを指定します。
3	SQLIND※5	short *	4 [8]	UAP	EXECUTE 文，OPEN 文，又は FETCH 文を発行するとき，標識変数有のデータコードを SQLCODE に設定したときだけ，標識変数の値を受け取るための領域のアドレスを指定します。標識変数の値を受け取るための領域は 2 バイトです。標識変数の有無の指定については，表「SQL 記述領域に設定するデータコードとデータの長さ」を参照してください。

(凡例)

△：空白を示します。

－：該当しません。

注

長さの[ ]内の値は，64 ビットモードの場合の長さです。なお，64 ビットモードの Windows の場合，SQLCA は 16 ＋ 24n バイトとなります。

注※1

表のレベル番号は，SQL 記述領域の包含関係を示しています。例えば，レベル番号 2 の記述領域はレベル番号 3 の記述領域から構成されることを示します。

注※2

UAP が SQLN に設定する SQLVAR の数は，SQLD に設定する?パラメタ数，又は検索項目数以上にしてください。?パラメタ数，又は検索項目数より SQLVAR の数が小さい場合，それを知らせるために，HiRDB は SQLN に 2 進数の 0(ゼロ)を返します。

注※3

データの長さ，及びデータコードの詳細については，表「SQL 記述領域に設定するデータコードとデータの長さ」を参照してください。

注※4

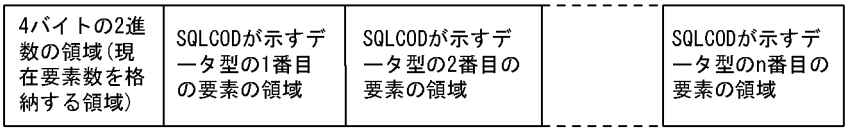
パック形式 10 進数(DECIMAL，INTERVAL YEAR TO DAY，又は INTERVAL HOUR TO SECOND)の場合，SQLLEN の領域は次の内容で構成されます。

記述領域名	データ型	長さ(バイト)	内　容
SQLPRCSN	B	1	精度(p)
SQLSCALE	B	1	位取り(s)

注※5

SQLDATA，及び SQLIND は，DESCRIBE 文を実行した時にクリアされるので，DESCRIBE 文を使用する場合，その実行後に値を再設定してください。繰返し列の場合に，値を設定するときは次のような構造になります（例として，SQLDATA について説明します）。

繰返し列をSQLDATAに設定する場合の変数の構造



（凡例）nは，変数の最大要素数を示します。

注※6

64 ビットモードの場合，プラットフォームごとの long 型のサイズになります。また，COBOL 言語の場合，64 ビットモードのときは，long 型に対応する宣言は，S9(18) COMP となります。

SQL のデータの詳細を次の表に示します。

表 B–2　SQL 記述領域に設定するデータコードとデータの長さ

データコード 10 進数	データコード 16 進数	標識変数	データ型	データの長さ	単位
0	00	−	HiRDB にはないデータ型	0	バイト
1	01	−	任意のデータ型※1	0	
48	30	なし	C VARCHAR(n)※4	1≦n≦32000※2	
49	31	あり			
68	44	なし	ROW	操作対象表の行長 L 1≦L≦30000	
69	45	あり			
100	64	なし	INTERVAL YEAR TO DAY	精度 8，位取り 0	けた
101	65	あり			

データコード 10 進数	データコード 16 進数	標識変数	データ型	データの長さ	単位
110	6E	なし	INTERVAL HOUR TO SECOND	精度 6，位取り 0	
111	6F	あり			
112	70	なし	DATE	4	バイト
113	71	あり			
120	78	なし	TIME	3	
121	79	あり			
124	7C	なし	TIMESTAMP[(p)]	7 ＋↑p÷2↑ p=0,2,4,又は 6	
125	7D	あり			
131	83	−	抽象データ型※3	−	−
144	90	なし	BINARY(n)	1≦n≦2147483647※2	バイト
145	91	あり			
146	92	なし	BLOB[(n)]	1≦n≦2147483647	
147	93	あり			
154	9A	なし	BINARY 位置付け子	4	
155	9B	あり			
158	9E	なし	BLOB 位置付け子	4	
159	9F	あり			
160	A0	なし	MVARCHAR(n)	1≦n≦32000※2	
161	A1	あり			
164	A4	なし	MCHAR[(n)]	1≦n≦30000	
165	A5	あり			
176	B0	なし	NVARCHAR(n)	1≦n≦16000※2	文字
177	B1	あり			
180	B4	なし	NCHAR[(n)]，又は NATIONAL CHAR[ACTER][(n)]	1≦n≦15000	
181	B5	あり			
192	C0	なし	VARCHAR(n)	1≦n≦32000※2	バイト
193	C1	あり			
196	C4	なし	CHAR[ACTER](n)	1≦n≦30000	
197	C5	あり			

データコード 10 進数	データコード 16 進数	標識変数	データ型	データの長さ	単位
224	E0	なし	FLOAT，又は DOUBLE PRECISION	8	
225	E1	あり			
226	E2	なし	SMALLFLT，又は REAL	4	
227	E3	あり			
228	E4	なし	[LARGE]DEC[IMAL][(p[，s])]※5	精度 p，位取り因数 s 1≦p≦38，0≦s≦p	けた
229	E5	あり			
234	EA	なし	[LARGE]DEC[IMAL][(p[，s])]※5	精度 p，位取り因数 s 1≦p≦38，0≦s≦p	
235	EB	あり			
236	EC	なし	[LARGE]DEC[IMAL][(p[，s])]※5	精度 p，位取り因数 s 1≦p≦38，0≦s≦p	
237	ED	あり			
240	F0	なし	INT[EGER]	4	バイト
241	F1	あり			
244	F4	なし	SMALLINT	2	
245	F5	あり			

（凡例）

－：該当しません。

注※1

?パラメタに NULL 述語を指定した SQL（? IS NULL）に対して，DESCRIBE INPUT 文，又は INPUT を指定した PREPARE 文を実行した場合にだけ，HiRDB がこのデータコードを設定します。それ以外の用途ではこのデータコードを使用できません。?パラメタに値を指定するために，DESCRIBE INPUT 文，又は INPUT を指定した PREPARE 文で受け取った SQL 記述領域を使用する場合は，データコードとデータ長を設定し直してください。

注※2

UAP で長さ 0 の可変長文字列を設定する場合，SQLLEN 領域に 1 を設定してください。

注※3

DESCRIBE 文を実行した場合に，サーバからデータ型が返されます。UAP ではデータ型の参照ができます。データ型の設定，及びデータの長さの参照，設定はできません。

注※4

C 言語の場合に設定できます。

注※5

DECIMAL 型を使用する場合の COBOL 言語のデータ記述との対応を次に示します。COBOL 言語のデータ記述の詳細は，「SQL のデータ型と COBOL 言語のデータ記述」を参照してください。

HiRDB のデータ型	データコード			COBOL 言語のデータ記述
	10 進数	16 進数	標識変数	
[LARGE] DEC[IMAL] [(p[，s])]	228	E4	なし	L1 基本項目名 PICTURE S9(p-s)[V9(s)] COMPUTATIONAL-3.
	229	E5	あり	
	234	EA	なし	L1 基本項目名 PICTURE S9(p-s)[V9(s)] DISPLAY SIGN LEADING SEPARATE.
	235	EB	あり	
	236	EC	なし	L1 基本項目名 PICTURE S9(p-s)[V9(s)] DISPLAY SIGN TRAILING.
	237	ED	あり	

（凡例）

L1：レベル番号 01～49，又は 77

p：精度（全体のけた数）

s：位取り（小数点以下のけた数）

表 B–3　SQLVAR_LOB の内容

レベル番号※1	記述領域名	データ型	長さ(バイト)	値の設定元	内　　容
2	SQLVAR_LOB	−	16n [24n]	−	SQLDIM，SQLCOD，SQLXDIM，SQLLOBLEN，SQLDATA， SQLDATA，及び SQLLOBIND から構成される領域です。この領域は，SQLVAR で定義し，BLOB 型，及び BINARY 型のデータを入出力するときに SQLVAR 領域に上書きする形で使用してください。
3	SQLDIM	unsigned char	1	−	未使用
3	SQLCOD	unsigned char	1	UAP	EXECUTE 文，OPEN 文，又は FETCH 文を発行するときデータコード※2 を指定します。
				HiRDB	DESCRIBE 文，又は DESCRIBE TYPE 文を発行後，データコード※2 が格納されます。
3	SQLXDIM	short	2	UAP	EXECUTE 文，OPEN 文，又は FETCH 文を発行するとき，1 を指定します。データ領域の構造については，「SQL のデータ型とデータ記述」を参照してください。
				HiRDB	DESCRIBE 文，又は DESCRIBE TYPE 文を発行したとき，1 が格納されます。

レベル番号※1	記述領域名	データ型	長さ(バイト)	値の設定元	内　　容
3	SQLLOBLEN※2	long [int]	4	UAP	EXECUTE 文，OPEN 文，又は FETCH 文を発行するときの埋込み変数の長さ※2 を指定します。
				HiRDB	DESCRIBE 文，又は DESCRIBE TYPE 文を発行したとき，データの定義長※2 が格納されます。
3	SQLDATA※3	unsigned char *	4 [8]	UAP	EXECUTE 文，又は OPEN 文を発行するとき，?パラメタ値が格納されているデータ領域のアドレスを指定します。FETCH 文を発行するとき，データを受け取るためのデータ領域のアドレスを指定します。
3	SQLLOBIND※3	long * [int *]	4 [8]	UAP	EXECUTE 文，OPEN 文，又は FETCH 文を発行するとき，標識変数有のデータコードを SQLCODE に設定したときだけ，標識変数の値を受け取るための領域のアドレスを指定します。標識変数の値を受け取るための領域は 4 バイトです。標識変数の有無の指定については，表「SQL 記述領域に設定するデータコードとデータの長さ」を参照してください。

(凡例)

－：該当しません。

注

データ型，及び長さの[ ]内は，64 ビットモードの場合のデータ型，及び長さです。

注※1

表のレベル番号は，SQL 記述領域の包含関係を示しています。例えば，レベル番号 2 の記述領域はレベル番号 3 の記述領域から構成されることを示します。

注※2

データの長さ，及びデータコードの詳細については，表「SQL 記述領域に設定するデータコードとデータの長さ」を参照してください。

注※3

SQLDATA，及び SQLLOBIND は，DESCRIBE 文を実行した時にクリアされるので，DESCRIBE 文を使用する場合，その実行後に値を再設定してください。繰返し列の場合に，値を設定するときの構造については，表「SQL 記述領域の内容」の注※5 を参照してください。

付録 B.2　SQL 記述領域の展開

SQL 記述領域は，UAP 内で宣言することで確保します。

ここでは，ソースプログラム中に展開する SQL 記述領域の形，及び使用例を示します。

(1)　SQL 記述領域の展開形

(a)　C 言語の場合

C 言語の場合の SQL 記述領域の展開形を次に示します。

```
struct {
  char   sqldaid[8];              /* 表ＩＤ                          */
  long   sqldabc;                 /* 表の長さ                        */
  short   sqln;                   /* ＳＱＬＶＡＲの配列の要素数 */
  short   sqld;                   /* ？パラメタ数，検索項目数   */
  struct  sqlvar{                 /* データ情報エリア                */
    unsigned char  sqldim;        /* 未使用                          */
    unsigned char  sqlcod;        /* データコード                    */
    short      sqlxdim;           /* 最大要素数                      */
    union {
      short      sqllen;          /* データ長                        */
      struct {
        unsigned char  sqlprcsn;  /* 精度                            */
        unsigned char  sqlscale;  /* 位取り                          */
      } s_sqllen;
    } sqllen;
    short      sqlsys;            /* 未使用                          */
    unsigned char  *sqldata;      /* データ領域アドレス              */
    short      *sqlind;           /* 標識変数アドレス                */
  } SQLVAR[n];※1
} sqlda;※2
```

注※1　n は，必要な個数(1～30000)を指定します。

注※2　構造体名称('sqlda' の部分)は，任意の文字列を指定してください。なお，構造体名称に'SQL'で始まる文字列は指定できません。

(b)　COBOL 言語の場合

COBOL 言語の場合の SQL 記述領域の展開形を次に示します。

- Windows 版，及び 32 ビットモードの UNIX 版の場合

```
 01  USQLDA.※1
   02  USQLDAID          PIC  X(8)  VALUE  'SQLDA'.
   02  USQLDABC          PIC  S9(9)  COMP.
   02  USQLN             PIC  S9(4)  COMP.
   02  USQLD             PIC  S9(4)  COMP.
   02  USQLVAR   OCCURS  n  TIMES.※2
     03  USQLTYPE        PIC  S9(4)  COMP.
     03  FILLER  REDEFINES  USQLTYPE.
       04  USQLDIM       PIC  X(1).
       04  USQLCOD       PIC  X(1).
     03  USQLXDIM        PIC  S9(4)  COMP  VALUE  IS  1.
     03  USQLATTR.
       04  USQLLEN       PIC  S9(4)  COMP.
       04  FILLER  REDEFINES  USQLLEN.
         05  USQLPRCSN  PIC  X(1).
```

```
          05  USQLSCALE  PIC  X(1).
        04  USQLSYS      PIC  S9(4)  COMP.
      03  FILLER  REDEFINES  USQLATTR.
        04  USQLLOBLEN   PIC  S9(9)  COMP.
      03  USQLDATA  USAGE  IS  ADDRESS.
      03  USQLIND   USAGE  IS  ADDRESS.
```

注※1　集合項目の名称（'USQLDA'の部分）は，任意の名称を指定してください。なお，データ項目にSQL で始まる文字列は指定できません。

注※2　n は必要な個数（１～30000）を指定します。

- 64 ビットモードの UNIX 版の場合

```
01  USQLDA.※1
  02  USQLDAID         PIC  X(8)  VALUE  'SQLDA'.
  02  USQLDABC         PIC  S9(18)  COMP. ※3
  02  USQLN            PIC  S9(4)  COMP.
  02  USQLD            PIC  S9(4)  COMP.
  02  FILLER           PIC  X(4). ※3
  02  USQLVAR   OCCURS  n  TIMES.※2
    03  USQLTYPE       PIC  S9(4)  COMP.
    03  FILLER  REDEFINES  USQLTYPE.
      04  USQLDIM      PIC  X(1).
      04  USQLCOD      PIC  X(1).
    03  USQLXDIM       PIC  S9(4)  COMP  VALUE  IS  1.
    03  USQLATTR.
      04  USQLLEN      PIC  S9(4)  COMP.
      04  FILLER  REDEFINES  USQLLEN.
        05  USQLPRCSN  PIC  X(1).
        05  USQLSCALE  PIC  X(1).
      04  USQLSYS      PIC  S9(4)  COMP.
    03  FILLER  REDEFINES  USQLATTR.
      04  USQLLOBLEN   PIC  S9(9)  COMP.
    03  USQLDATA  USAGE  IS  ADDRESS.
    03  USQLIND   USAGE  IS  ADDRESS.
```

注※1　集合項目の名称（'USQLDA'の部分）は，任意の名称を指定してください。なお，データ項目にSQL で始まる文字列は指定できません。

注※2　n は必要な個数（１～30000）を指定します。

注※3　Windows 版，及び 32 ビットモードの UNIX 版との違いを次に示します。

・USQLDABC の PICTURE 句の内容

・USQLD と USQLVAR の間の FILLER 項目の有無

(2)　SQL 記述領域の使用例

(a)　SQL 記述領域を使用するための宣言，及び領域の確保

SQL 記述領域は，UAP 内で宣言し，確保します。

(b)　検索項目情報の取得

検索項目情報の取得例を次に示します。

```
EXEC SQL BEGIN DECLARE SECTION;   ..................... 1
struct{   ..............................................  1
long※  cmd_len;   ..................................... 1
char  cmd_data[1000];   ................................ 1
}XCMND;   ..............................................  1
EXEC SQL END DECLARE SECTION;   ....................... 1
XCMND.cmd_len=(long※)sprintf(XCMND.cmd_data,
                ”SELECT*FROM ZAIKO WHERE GNO=1”)   .. 2
EXEC SQL WHENEVER SQLERROR GO TO :RERROR;   .......... 3
EXEC SQL PREPARE ST1 FROM :XCMND;   ................... 4
EXEC SQL DESCRIBE ST1 INTO :DAREA;   .................. 5
```

注 1　DESCRIBE 文を実行すると，SQLD 領域に 2 進数の 0，又は検索項目が設定されます。

- **0**：前処理した SQL が SELECT 文以外の場合
- **検索項目**：前処理した SQL が SELECT 文の場合

注 2　各検索項目のデータコードが SQLCOD に，データの定義長が SQLLEN に，最大要素数が SQLXDIM にそれぞれ設定されます。

注※　64 ビットモードの場合は，int となります。

＜説明＞

1. SQL 格納用の埋込み変数(XCMND)を宣言します。
2. SQL 文を変数(XCMND)に設定します。
3. SQL 実行後のエラーに対する処置を指定します。
4. 変数 XCMND として指定した SQL を前処理して，SQL 文識別子(ST1)を付けます。
5. SQL(ST1)の検索項目情報を SQL 記述領域(DAREA)に取得します。

(c)　受取り領域を動的に決定した検索結果の取出し

DESCRIBE 文で得た情報を基に割り当てた領域に検索結果を取り出す場合の例を次に示します。

```
for(n=0;n<DAREA.sqld;n++){   ..................................... 1
    DAREA.SQLVAR[n].sqldata=(unsigned char *)&(X_INT_DATA[n]);   .. 1
    DAREA.SQLVAR[n].sqlind=&(X_IND[n]);   ......................... 1
}   ..............................................................1
EXEC SQL DECLARE CR1 CURSOR FROM ST1;   ........................ 2
EXEC SQL OPEN CR1   ............................................ 3
EXEC SQL WHENEVER NOT FOUND GO TO:FEND;   ...................... 4
for(;;){   ..................................................... 5
    EXEC SQL FETCH CR1 USING DESCRIPTOR:DAREA   ................ 5
                  :   ..................................... 5,6
}   ..................................................... 5
```

```
    EXEC SQL WHENEVER NOT FOUND CONTINUE;   ..................... 7
FEND:EXEC SQL CLOSE CR1;   ..................................... 8
```

注　FETCH 文を実行するまでに，次に示す情報を DAREA に設定してください。

- SQLVAR 配列の大きさ(SQLN)
- 検索結果を受け取る領域の個数(SQLD)：DESCRIBE 文を実行すると設定される。
- 受取り領域のデータ型(SQLCOD)：DESCRIBE 文を実行すると設定される。
- 受取り領域のデータ長(SQLLEN)：DESCRIBE 文を実行すると設定される。

＜説明＞

1. 割り当てた領域の番地を SQL 記述領域(DAREA)に設定する。
2. SQL 文識別子(ST1)に対してカーソル(CR1)を宣言する。
3. カーソル(CR1)を開く。
4. 検索終了時の処理(FEND への分岐)を指定する。
5. カーソル(CR1)を次の行に位置付け，その行を SQL 記述領域(DAREA)で指定した領域に取り出す。
6. 編集・出力など検索結果に対する処理を指定する。
7. 検索終了時の処置を無効にする。
8. カーソル(CR1)を閉じる。

(d)　?パラメタに対する値を指定するためのデータ領域の動的決定

動的に指定された表にデータを挿入する場合の例を次に示します。

```
char TNAME[30];   ............................................... 1
scanf("%S",TNAME);   ............................................ 2
XCMND.cmd_len=(long※)sprint(XCMND.cmd_data,
                    "SELECT * FROM %S",TNAME);   ................ 3

EXEC SQL PREPARE ST1 FROM:XCMND;   .............................. 3

EXEC SQL DESCRIBE ST1 INTO:DAREA;   ............................. 3
         :   ..................................................... 4
for(n=0;n<DAREA.sqld;n++){   .................................... 5
DAREA.SQLVAR[n].sqldata=(unsigned char *)&(X_INT_DATA[n]);   .... 5
DAREA.SQLVAR[n].sqlind=&(X_IND[n]);   ........................... 5
}   ............................................................. 3
XCMND.cmd_len=(long※)sprit(XCMND.cmd_data,
              "INSERT INTO %S VALUES(?,…,?)",TNAME);   ......... 6
 EXEC SQL PREPARE ST2 FROM:XCMND;   ............................. 7
for(;;){   ...................................................... 8
    〔挿入データの入力（データがない場合，ＩＥＮＤに分岐）〕;   .. 8
    〔データ領域，標識変数領域にデータを挿入〕;   ................ 8
  EXEC SQL EXECUTE ST2 USING DESCRIPTOR:DAREA;   ................ 8
}   ............................................................. 8
IEND:
```

注※　64 ビットモードの場合は，int となります。

＜説明＞

1. 表名を格納する変数(TNAME)を宣言します。
2. 入力データから変数(TNAME)に表名を読み込みます。
3. ?パラメタの個数，及び各?パラメタに対するデータ領域のデータ型，データ長，最大要素数として，2 で指定された表の列数，各列のデータ型，データの定義長，最大要素数を DESCRIBE 文を利用して，SQL 記述領域(DAREA)に設定します。
4. 各?パラメタに対するデータ領域を確保して割り当てます。
5. 割り当てた領域の番地を SQL 記述領域(DAREA)に設定します。
6. 指定された表にデータを挿入するための INSERT 文を生成します。
7. XCMND 中の INSERT 文を前処理して，SQL 文識別子(ST2)を付けます。
8. 挿入するデータがない間，行単位に挿入データの入力，データ領域への設定，EXECUTE 文による実行を繰り返します。

(e)　FETCH 文で DECIMAL 型のデータを検索する場合の例

FETCH 文で DECIMAL 型のデータを検索する例を次に示します。

1. データ領域，及び標識変数の宣言

```
EXEC  SQL  BEGIN  DECLARE  SECTION ;

   SQL TYPE IS DECIMAL(20,0) xdec1 ;     /* データ領域       */
   short                     xdec1_i ;   /* 標識変数         */

EXEC  SQL  END  DECLARE  SECTION ;
```

2. SQL 記述領域の設定

```
PDSQLCOD(usrsqlda, 2)=PDSQL_DECIMAL_I ; /* データコード設定  */

  PDSQLPRCSN(usrsqlda, 2)=20          ; /* 精度の設定        */
  PDSQLSCALE(usrsqlda, 2)= 0          ; /* 位取りの設定      */
  PDSQLDATA(usrsqlda, 2)=(void*)xdec1 ; /* 埋込み変数アドレス*/
                                        /* 設定              */
  PDSQLXDIM(usrsqlda, 2)=1;             /* 繰返し列ではない  */

PDSQLIND(usrsqlda, 2)=(void*)&xdec1_i ; /* 標識変数アドレス  */
                                        /* 設定              */
```

(3)　SQL 記述領域の展開方法

SQL 記述領域の展開方法を次の表に示します。

表 B–4　SQL 記述領域の展開方法

言語	インクルードファイルを利用する	ユーザが直接記述する
C	#include <pdbsqlda.h> PDUSRSQLDA(n) usrsqlda;	SQL 記述領域の展開形を直接コーディングします。
COBOL	COPY SQLDA※ [REPLACING 256 BY n].	SQL 記述領域の展開形を直接コーディングします。必ず 01 レベルから記述してください。

注※　64 ビットモードの場合は SQLDA を SQLDA64 に変更してください。

COBOL 言語の場合の SQL 記述領域でパラメタを指定した場合の記述例を次に示します。

```
EXEC  SQL
    BEGIN  DECLARE  SECTION
END   EXEC
01  IN-CHR1  PIC  X(15).
01  IN-IND1  PIC  S9(4)  COMP.
EXEC  SQL
    END  DECLARE  SECTION
END-EXEC
COPY  SQLDA.
        :
COMPUTE  USQLDABC=32
COMPUTE  USQLN=1
COMPUTE  USQLD=1
COMPUTE  USQLDATA(1)=FUNCTION  ADDR(IN-CHR1)
MOVE  SQLCNST0    TO USQLDIM(1)
MOVE  SQLDCOD197  TO USQLCOD(1)
COMPUTE  USQLXDIM(1)=1
COMPUTE  USQLLEN(1)=15
COMPUTE  USQLIND(1)=FUNCTION  ADDR(IN-INT1)
EXEC  SQL
    EXECUTE  ST1  USING  DESCRIPTOR  :USQLDA
END-EXEC
```

(4)　SQL 記述領域操作用マクロ

C 言語では，SQLDA の宣言，及び値の設定・参照用に各種のマクロが定義されています。これらのマクロは専用のヘッダファイル（pdbsqlda.h）を UAP にインクルードすることで使用できます。SQL 記述領域操作用マクロを次の表に示します。

表 B–5　SQL 記述領域操作用マクロ

マクロ名	機能
PDUSRSQLDA(m)	ユーザ用 SQLDA を宣言します。
PDSETSIZE(usrsqlda,m)	SQLDA のサイズを設定します。
PDSQLN(usrsqlda)	?パラメタ数を設定します。
PDSQLD(usrsqlda)	?パラメタ，検索項目数を設定・参照します。

マクロ名	機能
PDSQLCOD(usrsqlda,n)	データコードを設定・参照します。
PDSQLLEN(usrsqlda,n)	?パラメタの実長を設定・参照します（BLOB と 10 進数以外）。
PDSQLPRCSN(usrsqlda,n)	精度を設定・参照します（10 進数だけ）。
PDSQLSCALE(usrsqlda,n)	位取りを設定・参照します（10 進数だけ）。
PDSQLDATA(usrsqlda,n)	データ領域のアドレスを設定します。
PDSQLIND(usrsqlda,n)	標識変数アドレスを設定します。
PDSQLLOBLEN(usrsqlda,n)	BLOB のデータ長を設定・参照します。
PDSQLDIM(usrsqldata,n)	未使用領域の値を設定・参照します。
PDSQLXDIM(usrsqldata,n)	繰り返し構造の最大要素数を設定・参照します。
PDSQLSYS(usrsqldata,n)	繰り返し構造又は配列構造の可変長の文字列型のギャップを含む 1 要素の長さを設定します。

（凡例）

usrsqlda：ユーザ定義の SQL 記述領域名。任意の値を指定してください。

m：？パラメタの個数（1～30000）

n：設定，又は参照する？パラメタの番号（0～29999）

データ型指定用マクロを次の表に示します。

表 B–6　データ型指定用マクロ

マクロ名	標識変数	対応するデータ型
PDSQL_FLOAT	なし	FLOAT
PDSQL_FLOAT_I	あり	
PDSQL_SMALLFLT	なし	SMALLFLT
PDSQL_SMALLFLT_I	あり	
PDSQL_DECIMAL	なし	DECIMAL
PDSQL_DECIMAL_I	あり	
PDSQL_INTEGER	なし	INTEGER
PDSQL_INTEGER_I	あり	
PDSQL_SMALLINT	なし	SMALLINT
PDSQL_SMALLINT_I	あり	
PDSQL_VARCHAR	なし	VARCHAR
PDSQL_VARCHAR_I	あり	

マクロ名	標識変数	対応するデータ型
PDSQL_CHAR	なし	CHAR
PDSQL_CHAR_I	あり	
PDSQL_NVARCHAR	なし	NVARCHAR
PDSQL_NVARCHAR_I	あり	
PDSQL_NCHAR	なし	NCHAR
PDSQL_NCHAR_I	あり	
PDSQL_MVARCHAR	なし	MVARCHAR
PDSQL_MVARCHAR_I	あり	
PDSQL_MCHAR	なし	MCHAR
PDSQL_MCHAR_I	あり	
PDSQL_DATE	なし	DATE
PDSQL_DATE_I	あり	
PDSQL_TIME	なし	TIME
PDSQL_TIME_I	あり	
PDSQL_YEARTODAY	なし	INTERVAL YEAR TO DAY
PDSQL_YEARTODAY_I	あり	
PDSQL_HOURTOSEC	なし	INTERVAL HOUR TO SECOND
PDSQL_HOURTOSEC_I	あり	
PDSQL_ROW	なし	ROW
PDSQL_ROW_I	あり	
PDSQL_BLOB	なし	BLOB
PDSQL_BLOB_I	あり	
PDSQL_TIMESTAMP	なし	TIMESTAMP
PDSQL_TIMESTAMP_I	あり	
PDSQL_BINARY	なし	BINARY
PDSQL_BINARY_I	あり	
PDSQL_BLOB_LOC	なし	BLOB 位置付け子
PDSQL_BLOB_LOC_I	あり	
PDSQL_BINARY_LOC	なし	BINARY 位置付け子
PDSQL_BINARY_LOC_I	あり	
PDSQL_CVARCHAR	なし	C 言語用の VARCHAR

マクロ名	標識変数	対応するデータ型
PDSQL_CVARCHAR_I	あり	

C 言語の場合の SQL 記述領域でパラメタを指定した場合の記述例を次に示します。

```
#include <pdbsqlda.h>              /* ヘッダファイルのインクルード */

EXEC  SQL  BEGIN  DECLARE  SECTION ;
short  xint1 ;
char   xchr1[16] ;
EXEC  SQL  END  DECLARE  SECTION ;
PDUSRSQLDA(2)  usrsqlda ;               /* ＳＱＬ記述領域の宣言    */
        :
ClearSqlda(2) ;                         /* SQL記述領域のクリア    */
PDSQLCOD(usrsqlda, 0)=PDSQL_SMALLINT ;  /* データコード設定        */
PDSQLLEN(usrsqlda, 0)=sizeof(short) ;   /* データ長設定            */
PDSQLDATA(usrsqlda, 0)=(void*)&xint ;   /* 埋込み変数アドレス設定 */
PDSQLIND(usrsqlda, 0)=NULL ;            /* 標識変数アドレス設定    */
PDSQLCOD(usrsqlda, 1)=PDSQL_CHAR ;      /* データコード設定        */
PDSQLLEN(usrsqlda, 1)=sizeof(xchar)-1 ; /* データ長設定            */
PDSQLDATA(usrsqlda, 1)=(void*)xchr ;    /* 埋込み変数アドレス設定 */
PDSQLIND(usrsqlda, 1)=NULL ;            /* 標識変数アドレス設定    */

EXEC  SQL
    EXECUTE  ST1  USING  DESCRIPTOR  :usrsqlda ;
        :
/*********************************************************************/
/* SQL記述領域初期設定                                               */
/*********************************************************************/
void ClearSqlda(
short num)
{
  PDSETSIZE(usrsqlda, 2) ;            /* SQL記述領域のサイズ設定 */
  PDSQLN(usrsqlda) = num ;            /* 配列サイズの上限設定    */
  PDSQLD(usrsqlda) = num ;            /* ？パラメタ数の設定      */
  while(num-->0) {
    PDSQLDATA(usrsqlda, num) = NULL ; /* 埋込み変数のクリア        */
    PDSQLIND(usrsqlda, num) = NULL ;  /* 標識変数のクリア          */
    PDSQLDIM(usrsqlda, num) = 0 ;     /* 未使用領域のクリア        */
    PDSQLXDIM(usrsqlda, num) = 1 ; /* 繰り返し構造の要素数クリア    */
    PDSQLSYS(usrsqlda, num) = 0 ;  /* 配列，繰り返し構造の要素サイズ */
    PDSQLCOD(usrsqlda, num) = 0 ;     /* データコードクリア        */
    PDSQLLEN(usrsqlda, num) = 0 ;     /* データ長クリア            */
  }
  return ;
}
```

(5)　繰返し列の展開形式

繰返し列の埋込み変数は，コンパイル時にマクロ定義によって表 B-7 に示す構造体が展開されます。ここでは，C 言語の場合について説明します。

繰返し列操作用のマクロは，展開される構造体のメンバを使用して繰返し列の要素を参照します。

ユーザが領域を確保して，SQL 記述領域にアドレスを直接設定する場合は，領域を語境界に割り当てる必要があります。FLOAT ARRAY の場合は，語境界調整のために空領域が明示的に含まれていますが，SQL 記述領域に設定するアドレスは，空領域分を考慮して設定する必要があります。

なお，これらの展開形式は，上記の領域確保での境界調整，及び構造体のサイズを取得する場合にだけ指定してください。埋込み変数として繰返し列を指定する場合は，展開形式を指定しないで，「SQL のデータ型とデータ記述」のマクロを使用してください。

表 B–7　繰返し列の展開形式

SQL のデータ型	マクロ名	展開形式

SMALL INT ARRAY[m]	PD_MV_SINT(m)	struct {
		long mcnt;
		short data[m];
		}

INTTGER ARRAY[m]	PD_MV_INT(m)	struct{
		long mcnt
		long data[m];
		}

SMALL FLT ARRAY[m]	PD_MV_SFLT(m)	struct{
		long mcnt;
		float data[m];
		}

FLOAT ARRAY[m]	PD_MV_FLT(m)	struct{
		union {
		double resv1;
		struct {
		long resv2;
		long mcnt;
		}mcnt_dmy2;
		} mcnt_dmy1;
		double data[m];
		}

CHAR(n) ARRAY[m]	PD_MV_CHAR(m, n)	struct {
		long mcnt;
		char data[m][(n)+1];
		}

NCHAR(n) ARRAY[m]	PD_MV_NCHAR(m, n)	struct {
		long mcnt;
		char data[m][2*(n)+1];
		}

VARCHAR(n) ARRAY[m]	PD_MV_VCHAR(m, n)	struct {

SQL のデータ型	マクロ名	展開形式

		long mcnt;
		struct {
		short len;
		char str[n];
		} data[m];
		}

	PD_MV_CVCHAR(m, n)	struct {
		long mcnt;
		char data[m][(n)+1];
		}

NVARCHAR(n) ARRAY[m]	PD_MV_NVCHAR(m, n)	struct {
		long mcnt;
		struct {
		short len;
		char str[2*(n)+1];
		} data[m];
		}

DECIMAL
〔(p〔,s〕)〕ARRAY[m]
	PD_MV_DEC(m, p, s)	struct {
		long mcnt;
		unsigned char data[m][(p)/2+1];
		}

付録 C　列名記述領域

SQL 記述領域を使用して次の情報を受け取る場合，同時に列名記述領域を指定することで，列名情報，及びルーチンのパラメタ情報も受け取れます。

- 検索項目情報（検索項目数，及び各検索項目のデータ型，データ長，最大要素数）
- CALL 文の入力，出力？パラメタ情報（？パラメタ数，及び？パラメタのデータ型，データ長）

ここでは，列名記述領域の構成と内容，及び列名記述領域の展開について説明します。

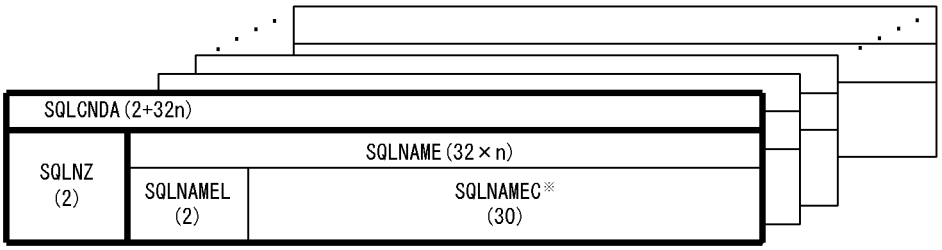
付録 C.1　列名記述領域の構成と内容

列名情報を受け取る列名記述領域の構成と内容について説明します。

(1)　列名記述領域の構成

列名記述領域の構成を次の図に示します。

図 C–1　列名記述領域の構成



注

( )内は領域の長さ(単位：バイト)を示します。

注※

SQLNAMEC は，最大長 30 バイトの可変長文字列の配列です。配列の大きさは，SQL 記述領域の SQLVAR 配列の大きさと同じです。SQLVAR 配列の大きさについては，「SQL 記述領域の構成と内容」を参照してください。

(2)　列名記述領域の内容

列名記述領域の内容を次の表に示します。

表 C–1　列名記述領域の内容

取得情報項目	検索項目の種類	内　　容
検索項目	列（添字指定なし）	列名
	列（添字指定あり）	列名［添字］

取得情報項目	検索項目の種類	内　　容
	集合関数	△△COUNT(*) △△COUNT_FLOAT(*) {△{■｜△}関数名(列名)｜ △{■｜△}関数名(DISTINCT 列名)｜ △△関数名(△EXP)}
	ウィンドウ関数	△△EXP(整数)
	値式(定数を含む)	△△EXP(整数)
	WRITE 指定	△△EXP(整数)
	ROW	△△ROW
CALL 文の入力？パラメタ	？パラメタ	ルーチンのパラメタ名
CALL 文の出力？パラメタ		
ユーザ定義型の構成要素	属性	属性名
CALL COMMAND 文の入力？パラメタ	？パラメタ	コマンド名称：'COMMAND_NAME' WITH 句： （1 番目の場合）'WITH' （2 番目以降の場合）'WITH（引数の番号）' INPUT 句：'INPUT' ENVIRONMENT 句：'ENVIRONMENT' SERVER 句：'SERVER'
CALL COMMAND 文の出力？パラメタ	？パラメタ	OUTPUT TO 句：'OUTPUT' ERROR TO 句：'ERROR' RETRUN CODE TO 句：'RETURN_CODE'

(凡例)

■：X'FF'を示します。

△：空白を示します。

注 1

列名記述領域の i 番目の要素に検索項目の i 番目の列名情報が設定されます。

注 2

検索項目が列の場合は先頭から列名が設定され，添字を含めた長さが 30 バイトを超えるときは，30 バイト以下になるように列名の後ろを切り捨てます。

検索項目が列以外の場合は，1〜2 バイト目に空白が設定されます。なお，30 バイトを超える場合は，2 バイト目に X'FF'が設定されます。

注 3

整数は，項目が何番目に検索されるかを表します。

注 4

UNION[ALL]，又は EXCEPT[ALL]の場合，最初に指定した問合せの検索項目の内容が設定されます。

注 5

AS 列名を指定している場合は，AS 列名で指定した列名の内容が設定されます。

注 6

ルーチンのパラメタ名は，CALL 文の引数に単独で？パラメタを指定したときだけ設定されます。？パラメタを含む値式を指定した場合は，SQLNAMEL が 0 になります。

注 7

検索項目が導出表の列の場合，導出表の後の導出列リストを省略し，かつ問合せの選択式に列名がない導出列のときは，△NONAME が設定されます。

付録 C.2　列名記述領域の展開

列名記述領域は，UAP 内で宣言することで静的なエリアとして確保します。

(1)　C 言語の場合

C 言語の場合のソースプログラム中に展開する列名記述領域の形式を次に示します。

```
struct {
    short       sqlnz;              /* 配列の有効数    */
    struct {
      short     sqlnamel;           /* 列名の有効長    */
      char      sqlnamec[30];       /* 列名格納エリア */
    } SQLNAME[n];※1
}×××××;※2
```

注※1　n は，SQL 記述領域の SQLVAR の配列の大きさと同じ個数(1～30000)を指定します。

注※2　構造体名称('×××××'の部分)は，任意の文字列を指定してください。なお，構造体名称に'SQL'で始まる文字列は指定できません。また，DESCRIBE 文で列名記述領域を指定する場合，確保した領域の名称を指定します。

(2)　COBOL 言語の場合

COBOL 言語の場合のソースプログラム中に展開する列名記述領域の形式を次に示します。

```
01  SQLCNDA.※1
  02  SQLNZ  PIC  S9(4) COMP.
  02  SQLNAME  OCCURS  1  TIMES  n.※2
```

```
    03  SQLNAMEL  PIC  S9(4)  COMP.
    03  SQLNAMEC  PIC  X(30).
```

注※1　集合項目の名称（'SQLCNDA'の部分）は，任意の名称を指定してください。なお，データ項目に'SQL'で始まる文字列は指定できません。また，集合項目のレベルは必ず 01 レベルにしてください。

注※2　n は必要な個数（1～30000）を指定します。

付録 D　型名記述領域

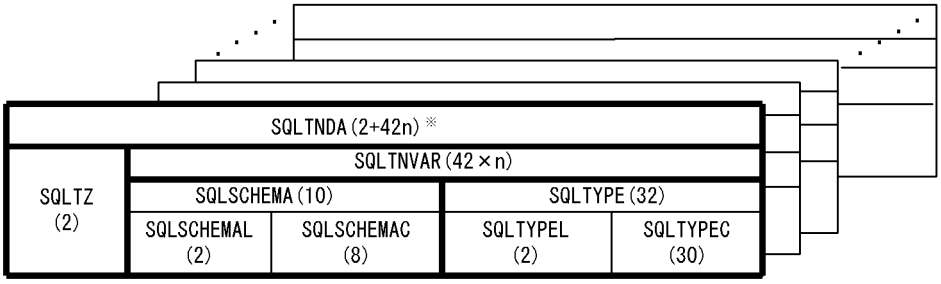
SQL 記述領域を使用して，検索項目情報，及びユーザ定義型の定義情報を受け取る場合，型名記述領域を指定すればユーザ定義型のデータ型名も受け取れます。

ここでは，型名記述領域の構成と内容，及び型名記述領域の展開について説明します。

付録 D.1　型名記述領域の構成

型名記述領域の構成を次の図に示します。

図 D–1　型名記述領域の構成



注

( )内は領域の長さ(バイト)を示します。

注※

SQLTNVAR は，最大長 10 バイトの可変長文字列 SQLSCHEMA，及び最大長 32 バイトの可変長文字列 SQLTYPE から構成される構造体の配列です。配列の大きさは，SQL 記述領域の SQLVAR 配列の大きさと同じです。SQLVAR 配列の大きさについては，「SQL 記述領域の構成と内容」を参照してください。

付録 D.2　型名記述領域の内容

型名記述領域の内容を次の表に示します。

表 D–1　型名記述領域の内容

レベル番号※	型名領域名	データ型	長さ(バイト)	内　容
1	SQLTNDA	−	2 ＋ 42×n	型名記述領域全体の名前を表します。
2	SQLTZ	short	2	検索項目数を指定します。
2	SQLTNVAR	−	42×n	認可識別子，データ型識別子から構成される領域です。
3	SQLSCHEMA	−	10	ユーザ定義型の認可識別子の情報を持つ領域です。

レベル番号※	型名領域名	データ型	長さ(バイト)	内　容
4	SQLSCHEMAL	short	2	認可識別子の長さが設定されます。 対応する検索項目のデータ型がユーザ定義型でない場合，0 が設定されます。
4	SQLSCHEMAC	char	8	認可識別子が設定されます。
3	SQLTYPE	−	32	ユーザ定義型のデータ型識別子の情報を持つ領域です。
4	SQLTYPEL	short	2	ユーザ定義型の長さが設定されます。 対応する検索項目のデータ型がユーザ定義型でない場合，0 が設定されます。
4	SQLTYPEC	char	30	ユーザ定義型のデータ型識別子が設定されます。

（凡例）−：該当しません。

注※

表のレベル番号は，型名記述領域の包含関係を示します。例えば，レベル番号 2 の記述領域はレベル番号 3 の記述領域から構成されることを示します。

付録 D.3　型名記述領域の展開

型名記述領域は，UAP 内で宣言することで静的なエリアとして確保します。

(1)　C 言語の場合

C 言語の場合のソースプログラム中に展開する型名記述領域の形式を次に示します。

```
struct {
    short         sqltz;                /*  配列の有効数                */
    struct {
        struct {
            short     sqlschemal;       /*  認可識別子の有効長          */
            char      sqlschemac[8 ];   /*  認可識別子格納エリア        */
        } sqlschema;
        struct {
            short     sqltypel;         /*  ユーザ定義型名の有効長      */
            char      sqltypec[30];     /*  ユーザ定義型名格納エリア    */
        } sqltype;
    } sqltnvar[n];※1
} Usrsqltnda;※2
```

注※1

n は，SQL 記述領域の SQLVAR の配列の大きさと同じ個数（1〜30000）を指定します。

注※2

構造体名称（'Usrsqltnda'の部分）は，任意の文字列を指定してください。なお，構造体名称に'SQL'で始まる文字列は指定できません。また，DESCRIBE 文で型名記述領域を指定する場合，確保した領域の名称を指定します。

(2) COBOL 言語の場合

COBOL 言語の場合のソースプログラム中に展開する型名記述領域の形式を次に示します。

```
01  USQLTNDA. ※1
  02  USQLTZ  PIC  S9(4) COMP.
  02  USQLTNVAR  OCCURS  1  TIMES  n. ※2
    03  USQLSCHEMA.
      04  USQLSCHEMAL  PIC  S9(4)  COMP.
      04  USQLSCHEMAC  PIC  X(8).
    03  USQLTYPE.
      04  USQLTYPEL  PIC  S9(4)  COMP.
      04  USQLTYPEC  PIC  X(30).
```

注※1

集合項目の名称（'USQLTNDA'の部分）は，任意の名称を指定してください。なお，データ項目に'SQL'で始まる文字列は指定できません。

注※2

n は SQL 記述領域の SQLVAR の配列の大きさと同じ個数（1～30000）を指定します。

付録 E　文字集合名記述領域

文字集合名記述領域は，UAP 実行時に動的に決定した入出力変数の文字集合名を記述して，OPEN 文，FETCH 文，又は EXECUTE 文でシステムに通知するための領域です。また，動的に実行する場合，前処理した SQL の検索項目，又は？パラメタの文字集合名の受け取りのために文字集合名記述領域を使用することもできます。

文字集合名記述領域を使用できる UAP の記述言語については，「UAP の記述」を参照してください。

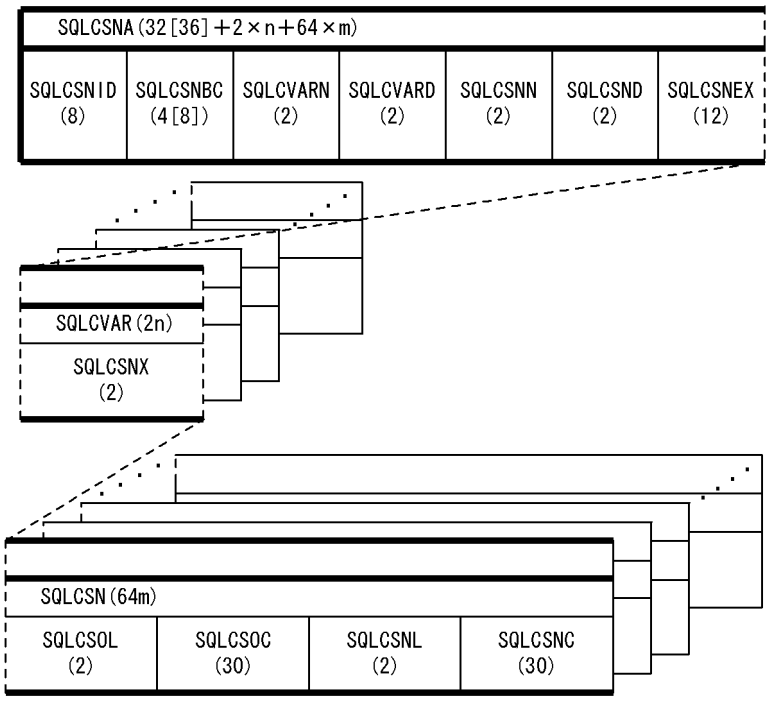
付録 E.1　文字集合名記述領域の構成と内容

UAP 実行時に動的に決定した入出力変数の文字集合名を記述する領域の構成，及び内容について説明します。

(1)　文字集合名記述領域の構成

文字集合名記述領域の構成を次の図に示します。

図 E–1　文字集合名記述領域の構成



注 1

( )内は領域の長さ(バイト)を示します。

注 2

n は SQLCVARN に指定した SQLCVAR の数を示します。

m は SQLCSNN に指定した SQLCSN の数を示します。

注 3

( )の[ ]内の値は，64 ビットモードの場合の長さです。なお，64 ビットモードの Windows の場合は 32 ビットモードと同じになります。

注 4

64 ビットモードでの SQLCSNBC の長さは，プラットフォームごとの long 型のサイズになります。

(2)　文字集合名記述領域の内容

文字集合名記述領域の内容を次の表に示します。

表 E–1　文字集合名記述領域の内容

レベル番号※1	記述領域名	データ型	長さ(バイト)	値の設定元	内　　容
1	SQLCSNA	−	32[36]※2 +2×n +64×m	−	文字集合名記述領域全体の名前です。
2	SQLCSNID	char	8	HiRDB	SQLCSNA を示す ID 'SQLCSNA△'です。DESCRIBE 文，DESCRIBE CURSOR 文，又は PREPARE 文を発行したとき格納されます。
2	SQLCSNBC	long	4 [8]※2	HiRDB	SQLCSNA の長さです。DESCRIBE 文，DESCRIBE CURSOR 文，又は PREPARE 文を発行したとき格納されます。
2	SQLCVARN	short	2	UAP	SQLCSNA の領域を確保したとき，又は SQLCSNA を使用するとき，確保した SQLCSNA 領域の SQLCVAR の個数(1～30000)を指定します。
				HiRDB	DESCRIBE 文，DESCRIBE CURSOR 文，又は PREPARE 文を発行したとき SQLCSNA の領域が不足(SQLCVARN<SQLCVARD)する場合，2 進数の 0 が格納されます。
2	SQLCVARD	short	2	UAP	OPEN 文，EXECUTE 文，EXECUTE IMMEDIATE 文，又は FETCH 文を発行するとき，入力？パラメタに対する文字集合情報を指定した SQLCVAR の要素番号の最大値を指定します。
				HiRDB	DESCRIBE[OUTPUT]，DESCRIBE CURSOR 文，又は PREPARE 文を発行したとき，2 進数の 0，検索項目数，又は出力？パラメタに対応する文字集合情報が設定された SQLCVAR の要素番号の最大値が設定されます。 0： 次のどれかの場合に設定されます。

レベル番号※1	記述領域名	データ型	長さ(バイト)	値の設定元	内　　容
					• SQL が SELECT 文以外で，かつ出力？パラメタがある CALL 文以外である • 検索項目，又は出力？パラメタが既定文字集合以外の文字集合を持たない 最大要素番号： 次のすべての条件を満たす場合に設定されます。 • SQL が SELECT 文，又は出力？パラメタがある CALL 文である • 検索項目，出力？パラメタが既定文字集合以外の文字集合を持つ DESCRIBE INPUT 文，又は PREPARE 文を発行したとき，2 進数の 0，又は入力？パラメタに対応する文字集合情報が設定された SQLCVAR の要素番号の最大値が設定されます。 0： 次のどれかの場合に設定されます。 • SQL 文中に入力？パラメタを含まない • 入力？パラメタが既定文字集合以外の文字集合を持たない 最大要素番号： 次のすべての条件を満たす場合に設定されます。 • SQL 文中に入力？パラメタを含む • 入力？パラメタが既定文字集合以外の文字集合を持つ
2	SQLCSNN	short	2	UAP	SQLCSNA の領域を確保したとき，確保した SQLCSN 領域の個数（1）を指定します。
				HiRDB	DESCRIBE 文，DESCRIBE CURSOR 文，又は PREPARE 文を発行したとき，SQLCSN の領域が不足（SQLCSNN ＜ SQLCSND）する場合，2 進数の 0 が格納されます。
2	SQLCSND	short	2	UAP	OPEN 文，EXECUTE 文，EXECUTE IMMEDIATE 文，又は FETCH 文を発行するとき，入力？パラメタで使用する文字集合数（1）を指定します。
				HiRDB	DESCRIBE[OUTPUT]，DESCRIBE CURSOR 文，又は PREPARE 文を発行した場合，SQLCSN の領域が不足（SQLCSNN ＜ SQLCSND）するとき，2 進数の 0，検索項目，又は出力？パラメタで使用する文字集合数が格納されます。

レベル番号※1	記述領域名	データ型	長さ(バイト)	値の設定元	内　　容
					0： 次のどれかの場合に設定されます。 • SQL が SELECT 文以外で，かつ出力？パラメタがある CALL 文以外である • 検索項目，又は出力？パラメタが既定文字集合以外の文字集合を持たない 最大要素番号： 次のすべての条件を満たす場合に設定されます。 • SQL が SELECT 文，又は出力？パラメタがある CALL 文である • 検索項目，出力？パラメタが既定文字集合以外の文字集合を持つ DESCRIBE INPUT 文，又は PREPARE 文を発行したとき，2 進数の 0，又は入力？パラメタで使用している文字集合数が設定されます。 0： 次のどれかの場合に設定されます。 • SQL 文中に入力？パラメタを含まない • 入力？パラメタが既定文字集合以外の文字集合を持たない 最大要素番号： 次のすべての条件を満たす場合に設定されます。 • SQL 文中に入力？パラメタを含む • 入力？パラメタが既定文字集合以外の文字集合を持つ
2	SQLCSNEX	−	12	−	未使用
2	SQLCVAR	−	2×n	−	SQLCSNX から構成される領域。この領域は，SQLDA の SQLVAR に対応する領域であり，SQLCVARN で指定した個数分を繰り返し定義する必要があります。
3	SQLCSNX	short	2	UAP	OPEN 文，EXECUTE 文，EXECUTE IMMEDIATE 文，又は FETCH 文を発行するとき，入力？パラメタに対する文字集合名を持つ SQLCSN の要素番号を指定します。
				HiRDB	DESCRIBE 文，DESCRIBE CURSOR 文，又は PREPARE 文を発行したとき，検索項目又は？パラメタで使用している文字集合名を持つ SQLCSN の要素番号が格納されます。検索項目，又は出力？パラメタ中に文字集合指定がない場合は，2 進数の 0 が格納されます。

レベル番号※1	記述領域名	データ型	長さ(バイト)	値の設定元	内　　容
					DESCRIBE INPUT 文，PREPARE 文を発行したとき，入力？パラメタで使用している文字集合名を持つ SQLCSN の要素番号が格納されます。入力？パラメタ中に文字集合指定がない場合は，2 進数の 0 が格納されます。
2	SQLCSN	−	64×m	−	SQLCSOL，SQLCSOC，SQLCSNL，及び SQLCSNC から構成される領域。この領域は文字集合情報ごとの領域であり，SQLCSNN で指定した個数の領域を繰り返し定義する必要があります。
3	SQLCSOL	short	2	UAP	OPEN 文，EXECUTE 文，EXECUTE IMMEDIATE 文，又は FETCH 文を発行するとき，認可識別子の長さを指定します。
				HiRDB	DESCRIBE 文，DESCRIBE CURSOR 文，又は PREPARE 文を発行したとき，認可識別子の長さが格納されます。
3	SQLCSOC	char	30	UAP	OPEN 文，EXECUTE 文，EXECUTE IMMEDIATE 文，又は FETCH 文を発行するとき，認可識別子を指定します。
				HiRDB	DESCRIBE 文，DESCRIBE CURSOR 文，又は PREPARE 文を発行したとき，認可識別子が格納されます。
3	SQLCSNL	short	2	UAP	OPEN 文，EXECUTE 文，EXECUTE IMMEDIATE 文，又は FETCH 文を発行するとき，文字集合名の長さを指定します。
				HiRDB	DESCRIBE 文，DESCRIBE CURSOR 文，又は PREPARE 文を発行したとき，文字集合名の長さが格納されます。
3	SQLCSNC	char	30	UAP	OPEN 文，EXECUTE 文，EXECUTE IMMEDIATE 文，又は FETCH 文を発行するとき，文字集合名を指定します。
				HiRDB	DESCRIBE 文，DESCRIBE CURSOR 文，又は PREPARE 文を発行したとき，文字集合名が格納されます。

(凡例)

△：空白を示します。

−：該当しません。

注※1

表のレベル番号は，文字集合名記述領域の包含関係を示しています。例えば，レベル番号 2 の記述領域はレベル番号 3 の記述領域から構成されることを示します。

注※2

長さの[ ]内の値は，64 ビットモードの場合の長さです。なお，64 ビットモードの Windows の場合は 32 ビットモードと同じになります。

(3)　文字集合名記述領域の SQLCSN に設定できる文字集合情報

UAP が SQLCSN に設定できる文字集合情報の内容を次の表に示します。

表 E–2　UAP が SQLCSN に設定できる文字集合情報の内容

SQLCSOL	SQLCSOC	SQLCSNL	SQLCSNC
6	MASTER	6	EBCDIK※1
6	MASTER	8	UTF-16LE※2
6	MASTER	8	UTF-16BE※2
6	MASTER	5	UTF16※2

注※1

pdntenv コマンド（UNIX 版の場合は pdsetup コマンド）で，文字コード種別に sjis を指定した場合にだけ使用できます。

注※2

pdntenv コマンド（UNIX 版の場合は pdsetup コマンド）で，文字コード種別に utf-8，又は utf-8_ivs を指定した場合にだけ使用できます。

C 言語の UTF-16 のデータ記述を使用する場合，又は COBOL2002 の Unicode 機能を使用して，COBOL 言語の日本語項目（PICTURE N）を含むデータ記述を使用する場合，文字集合 UTF16，UTF-16LE 及び UTF-16BE を設定します。

C 言語の UTF-16 のデータ記述については，「SQL のデータ型と C 言語のデータ記述」を参照してください。COBOL 言語の日本語項目（PICTURE N）を含むデータ記述については，「SQL のデータ型と COBOL 言語のデータ記述」を参照してください。

文字集合 UTF16，UTF-16LE 及び UTF-16BE の属性を次の表に示します。

表 E–3　文字集合 UTF16，UTF-16LE 及び UTF-16BE の属性

文字集合名	使用形式	文字レパートリ
UTF-16LE	UTF-16 リトルエンディアン	UTF-16 でコード化できる Unicode の文字
UTF-16BE，UTF16	UTF-16 ビッグエンディアン	

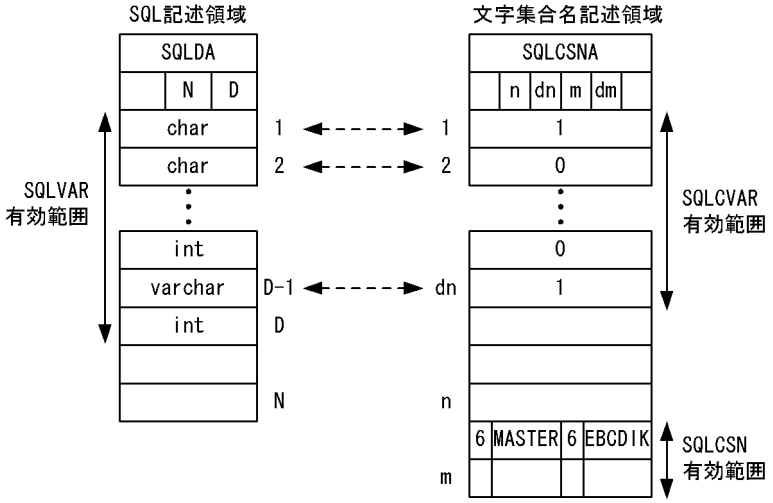
結果の文字集合が UTF16 となる SQL の検索項目，又は？パラメタに対して DESCRIBE 文，DESCRIBE TYPE 文，DESCRIBE CURSOR 文，PREPARE 文を発行した場合，HiRDB は文字集合名記述領域に文字集合名として UTF16 を設定します。

OPEN 文，EXECUTE 文，EXECUTE IMMEDIATE 文，FETCH 文で結果の文字集合が UTF16 となる SQL の検索項目，又は？パラメタのデータを取り出す場合，又は UTF-16 で表現されたデータを？パラメタを用いて HiRDB にデータを渡す場合，文字集合名に UTF16，UTF-16LE，又は UTF-16BE を指定してください。

(4)　SQL 記述領域との関係

SQL 記述領域と文字集合名記述領域の関係を次の図に示します。

図 E–2　SQL 記述領域と文字集合名記述領域の関係



（凡例）

N：SQLN 設定値（SQLVAR の最大数）

D：SQLD 設定値（SQLVAR の有効数）

n：SQLCVARN 設定値（SQLCVAR の最大数）

dn：SQLCVARD 設定値（SQLCVAR の有効数）

m：SQLCSNN 設定値（SQLCSN の最大値）

dm：SQLCSND 設定値（SQLCSN の有効値）

付録 E.2　文字集合名記述領域の展開

文字集合名記述領域は，UAP 内で宣言することで確保します。

ここでは，ソースプログラム中に展開する文字集合名記述領域の形，及び使用例を示します。

(1)　文字集合名記述領域の展開形

(a)　C 言語の場合

C 言語の場合の文字集合名記述領域の展開形を次に示します。

```
struct {
    char    sqlcsnid[8];          /* SQLCSNAを示す名称        */
    long    sqlcsnbc;             /* SQLCSNAの長さ            */
    short   sqlcvarn;             /* SQLCVAR配列の要素数      */
    short   sqlcvard;             /* SQLCVARの有効数          */
    short   sqlcsnn;              /* SQLCSN配列の要素数       */
    short   sqlcsnd;              /* SQLCSNの有効数           */
    char    sqlcsnex[12];         /* 予備                      */
    struct sqlcvar {
        short   sqlcsnx;          /* 対応する文字集合名を持つ  */
                                  /* SQLCSNの番号             */
    } SQLCVAR[m]; ※1
    struct sqlcsn {
       short   sqlcsol;           /* 認可識別子の有効長        */
       char    sqlcsoc[30];       /* 認可識別子格納エリア      */
       short   sqlcsnl;           /* 文字集合名の有効長        */
       char    sqlcsnc[30];       /* 文字集合名格納エリア      */
    } SQLCSN[n]; ※2
} sqlcsna; ※3
```

注※1　m は，必要な個数(1～30000)を指定します。

注※2　n は，必要な個数(1)を指定します。

注※3　構造体名称('sqlcsna' の部分)は，任意の文字列を指定してください。なお，構造体名称に'SQL'で始まる文字列は指定できません。

(b)　COBOL 言語の場合

COBOL 言語の場合の文字集合名記述領域の展開形を次に示します。

```
01  USQLCSNA.※1
    02  USQLCSNID        PIC  X(8)  VALUE  'SQLCSNA'.
    02  USQLCSNBC        PIC  S9(9)  COMP.※4
    02  USQLCVARN        PIC  S9(4)  COMP.
    02  USQLCVARD        PIC  S9(4)  COMP.
    02  USQLCSNN         PIC  S9(4)  COMP.
    02  USQLCSND         PIC  S9(4)  COMP.
    02  USQLCSNEX        PIC  X(12).
    02  USQLCVAR  OCCURS  m  TIMES. ※2
      03  USQLCSNX       PIC  S9(4)  COMP.
    02  USQLCSN   OCCURS  n  TIMES. ※3
      03  USQLCSOL       PIC  S9(4)  COMP.
      03  USQLCSOC       PIC  X(30).
      03  USQLCSNL       PIC  S9(4)  COMP.
      03  USQLCSNC       PIC  X(30).
```

注※1　集合項目の名称（'USQLCSNA'の部分）は，任意の名称を指定してください。なお，データ項目にSQL で始まる文字列は指定できません。

注※2　m は必要な個数（１～30000）を指定します。

注※3　n は必要な個数（１）を指定します。

注※4　64 ビットモードの UNIX の場合は，USQLCSNBC の PICTURE 句を S9(18)に変更してください。

(2)　文字集合名記述領域の展開方法

文字集合名記述領域の展開方法を次の表に示します。

表 E–4　文字集合名記述領域の展開方法

言語	インクルードファイルを利用する	ユーザが直接記述する
C	#include <pdbsqlcsna.h> PDUSRSQLCSNA(m,n) usrsqlcsna m，n，usrsqlcsna については，「文字集合名記述領域操作用マクロ」を参照してください。	文字集合名記述領域の展開形を直接コーディングします。
COBOL	COPY SQLCSNA※ [REPLACING 256 BY m ==OCCURS 1 == BY ==OCCURS n ==].	文字集合名記述領域の展開形を直接コーディングします。必ず 01 レベルから記述してください。

注※　64 ビットモードの場合は，SQLCSNA を SQLCSNA64 に変更してください。

(3)　文字集合名記述領域操作用マクロ

C 言語では，SQLCSNA の宣言，及び値の設定・参照用に各種のマクロが定義されています。これらのマクロは専用のヘッダファイル（pdbsqlcsna.h）を UAP にインクルードすることで使用できます。文字集合名記述領域操作用マクロを次の表に示します。

表 E–5　文字集合名記述領域操作用マクロ

マクロ名	機能
PDUSRSQLCSNA(m,n)	ユーザ用 SQLCSNA を宣言します。
PDSETCSNASIZE(usrsqlcsna,m,n)	SQLCSNA のサイズを設定します。
PDSQLCVARN(usrsqlcsna)	SQLCVAR 配列の要素数を設定します。
PDSQLCVARD(usrsqlcsna)	SQLCVAR の有効数を設定・参照します。
PDSQLCSNN(usrsqlcsna)	SQLCSN 配列の要素数を設定します。
PDSQLCSND(usrsqlcsna)	SQLCSN の有効数を設定・参照します。
PDSQLCSNX(usrsqlcsna,o)	対応する文字集合名を持つ SQLCSN の番号を設定・参照します。
PDSQLCSOL(usrsqlcsna,p)	文字集合の認可識別子の長さを設定・参照します。
PDSQLCSOC(usrsqlcsna,p)	文字集合の認可識別子を設定・参照します。
PDSQLCSNL(usrsqlcsna,p)	文字集合の文字集合名の長さを設定・参照します。
PDSQLCSNC(usrsqlcsnaa,p)	文字集合の文字集合名を設定・参照します。

（凡例）

usrsqlcsna：ユーザ定義の文字集合名記述領域名。任意の値を指定してください。

m：？パラメタの個数（1～30000）

n：文字集合名の個数（0～1）

o：設定，若しくは参照する？パラメタ，又は検索項目の番号（0～29999）

p：設定，又は参照する文字集合名の番号（0）

文字集合名指定用マクロを次の表に示します。

表 E–6　文字集合名指定用マクロ

マクロ名	機能
PDSQL_CS_MASTER_L	認可識別子 MASTER の長さ
PDSQL_CS_MASTER_C	認可識別子 MASTER の文字列
PDSQL_CS_EBCDIK_L	文字集合名 EBCDIK の長さ
PDSQL_CS_EBCDIK_C	文字集合名 EBCDIK の文字列
PDSQL_CS_UTF16_L	文字集合名 UTF16 の長さ
PDSQL_CS_UTF16_C	文字集合名 UTF16 の文字列
PDSQL_CS_UTF_16LE_L	文字集合名 UTF-16LE の長さ
PDSQL_CS_UTF_16LE_C	文字集合名 UTF-16LE の文字列
PDSQL_CS_UTF_16BE_L	文字集合名 UTF-16BE の長さ
PDSQL_CS_UTF_16BE_C	文字集合名 UTF-16BE の文字列

付録 F　SQL のデータ型とデータ記述

付録 F.1　SQL のデータ型と C 言語のデータ記述

SQL のデータ型と C 言語のデータ記述の対応を示します。

なお，データの受け渡しには，対応するデータ型以外に，変換，又は代入できるデータ型も使用できます。

(1)　SQL のデータ型と C 言語のデータ記述

SQL のデータ型と C 言語のデータ記述を次の表に示します。

表 F–1　SQL のデータ型と C 言語のデータ記述

SQL のデータ型	C 言語のデータ記述	備　　考
SMALLINT	short 変数名;	–
INTEGER	long 変数名;	–
DECIMAL [(p[，s])]	SQL TYPE IS DECIMAL(p,s) 変数名;※5	1≦p≦38，0≦s≦p
SMALLFLT，REAL	float 変数名;	–
FLOAT(DOUBLE PRECISION)	double 変数名;	–
CHAR[(n)] [CHARACTER SET 文字集合指定]	char [CHARACTER SET[IS] 文字集合指定] 変数名[n+1];※1	1≦n≦30000 文字集合指定：[MASTER.]EBCDIK
CHAR[(2n)] CHARACTER SET 文字集合指定	char CHARACTER SET[IS] 文字集合指定 変数名[2n+2];	1≦n≦15000 文字集合指定：[MASTER.]UTF16
	SQL TYPE IS CHAR(2n) CHARACTER SET[IS] 文字集合指定 変数名;※10	
VARCHAR(n) [CHARACTER SET 文字集合指定]	struct{ short 変数名 1; char 変数名 2[n]; }[CHARACTER SET[IS] 文字集合指定] 構造体名;	1≦n≦32000 変数名 1：文字列長（バイト数） 変数名 2：文字列 文字集合指定：[MASTER.]EBCDIK

SQL のデータ型	C 言語のデータ記述	備　　考
	SQL TYPE IS VARCHAR(n) [CHARACTER SET[IS] 文字集合指定] 変数名;※6	
	VARCHAR [CHARACTER SET[IS] 文字集合指定] 変数名[n+1];※9	
VARCHAR(2n) CHARACTER SET 文字集合指定	struct{ short 変数名 1; char 変数名 2[2n]; }CHARACTER SET[IS] 文字集合指定 構造体名;	1≦n≦16000 変数名 1：文字列長 変数名 2：文字列 文字列長： バイト数で，2 の倍数 文字集合指定：[MASTER.]UTF16
	SQL TYPE IS VARCHAR(2n) CHARACTER SET[IS] 文字集合指定 変数名;※11	
	VARCHAR CHARACTER SET[IS] 文字集合指定 変数名[2n+2];※10	
NCHAR[(n)]	char 変数名[2n+1];※1	1≦n≦15000
NVARCHAR(n)	struct{ short 変数名 1; char 変数名 2[2n]; }構造体名;	1≦n≦16000 変数名 1：文字列長（文字数） 変数名 2：文字列
	SQL TYPE IS NVARCHAR(n) 変数名;※6	
MCHAR[(n)]	char 変数名[n+1];※1	1≦n≦30000
MVARCHAR(n)	struct{ short 変数名 1; char 変数名 2[n]; }構造体名;	1≦n≦32000 変数名 1：文字列長（バイト数） 変数名 2：文字列
	SQL TYPE IS	

SQL のデータ型		C 言語のデータ記述	備　　考
		MVARCHAR(n) 変数名;※6	
DATE		char 変数名[11];※2	−
TIME		char 変数名[9];※2	−
INTERVAL YEAR TO DAY		SQL TYPE IS DECIMAL(8,0) 変数名;※5	−
INTERVAL HOUR TO SECOND		SQL TYPE IS DECIMAL(6,0) 変数名;※5	−
TIMESTAMP[(p)]		char 変数名[n+1];※2	p=0 の場合は n=19 p=2 の場合は n=21,22 p=4 の場合は n=23,24 p=6 の場合は n=25,26
ROW※3		char 変数名[n+1];	1≦全体長≦30000
BLOB		SQL TYPE IS BLOB(n{K ｜ M ｜ G}) 変数名;※4	単位省略時： 1≦n≦2147483647 単位が K： 1≦n≦2097152 単位が M： 1≦n≦2048 単位が G： 1≦n≦2
BINARY(n)		struct{ long 変数名 1; char 変数名 2[n]; }構造体名;	1≦n≦2147483647 変数名 1：文字列長（バイト数） 変数名 2：文字列
		SQL TYPE IS BINARY(n) 変数名;※7	
BLOB 位置付け子		SQL TYPE IS BLOB AS LOCATOR 変数名;※8	−
BINARY 位置付け子		SQL TYPE IS BINARY AS LOCATOR 変数名;※8	−
標識変数	BLOB，BINARY， BLOB 位置付け子， BINARY 位置付け子 以外	short 変数名;	−

SQL のデータ型		C 言語のデータ記述	備　　考
	BLOB，BINARY， BLOB 位置付け子， BINARY 位置付け子	long 変数名;	
SQL 文		struct{ long 変数名 1; char 変数名 2[n]; }構造体名;	1≦n≦2000000 変数名 1：文字列長（バイト数） 変数名 2：文字列

（凡例）

n：長さ（バイト）

p：精度（全体のけた数）

s：位取り（小数点以下のけた数）

−：なし

注

64 ビットモードの場合，long の代わりに int を使用してください。

注※1

SQL のデータ型(CHAR(n)，NCHAR(n)，MCHAR(n))と，C 言語のデータ型(char[n+1]，char[2n+1]，char[n+1])との間での変換規則を示します。

・入力時

- char[n+1]から CHAR(n)への変換
- char[2n+1]から NCHAR(n)への変換
- char[n+1]から MCHAR(n)への変換

C 言語の文字列から HiRDB が受け取る固定長文字列の長さは，先頭からナル文字の一つ前までの長さとします。なお，n+1 個の配列要素の中にナル文字がない場合，長さを n とします。

・出力時

- CHAR(n)から char[n+1]への変換
- NCHAR(n)から char[2n+1]への変換
- MCHAR(n)から char[n+1]への変換

文字列の終端にナル文字を付け加え，UAP での文字列長は SQL の文字列長に 1 を加えます。

注※2

動的 SQL で日付データ(DATE)を検索する場合，DESCRIBE 文で取得した検索項目情報のデータコードを文字データ型に，データ長を 10 バイト以上に設定する必要があります。また，動的 SQL で時刻データを検索する場合，DESCRIBE 文で取得した検索項目情報のデータコードを文字データ型に，データ長を 8 バイト以上に設定する必要があります。

動的 SQL で時刻印データ（TIMESTAMP）を検索する場合，次のように設定してください。

- DESCRIBE 文で取得した検索項目情報のデータコードは，文字データ型にしてください。
- p が 0 の場合は，データ長を 19 バイト以上にしてください。p が 2，4，又は 6 の場合は，20+p バイト以上にしてください。

注※3

HiRDB サーバと HiRDB クライアントのエンディアンが同じ場合にだけ，ROW 型を使用できます。

注※4

BLOB の UAP の記述は，内部的に次のように展開されます。

```
struct{
      long           変数名_reserved;          … 1
      unsigned long  変数名_length;            … 2
      char           変数名_data[m];           … 3
}変数名
```

[説明]

1. 変数名_reserved は，使用しません。64 ビットモードの場合は，int 変数名_reserved;となります。
2. 変数名_length は，BLOB の実際の長さになります。64 ビットモードの場合は，unsigned int 変数名_length;となります。
3. 変数名_data[m]は，BLOB のデータ格納領域(m は実際のデータ長)になります。

注※5

DECIMAL の UAP の記述は，内部的に次のように展開されます。

```
unsigned char 変数名[↓p/2↓+1];
```

DECIMAL のデータは 1 バイトで 2 けたの数字を表現します。符号は，右端のバイトの右 4 ビットで表します。そのため，偶数けたの DECIMAL 型の場合，左端の左 4 ビットに 0 を補う必要があります。0 以外の値は設定しないでください。

標準的な符号表現を次に示します。HiRDB での DECIMAL 型の符号については，マニュアル「HiRDB SQL リファレンス」を参照してください。

符号の 16 進数表現	意　味
X'C'	正の符号とみなします。正数には 0 を含みます。
X'D'	負の符号とみなします。

（記述例）

123.4567 の場合（奇数けた）

```
unsigned char ex1[4]={0x12,0x34,0x56,0x7c} ;
```

-123.456 の場合（偶数けた）

```
unsigned char ex2[4]={0x01,0x23,0x45,0x6d} ;
```

0 の場合（奇数けた）

```
unsigned char ex3[1]={0x0c} ;
```

注※6

内部的に次のように展開されます。

```
struct{
         short   len;
         char    str[n];
}変数名
```

NVARCHAR の場合は，str[2n]になります。

注※7

内部的に次のように展開されます。

```
struct{
         long  len;
         char  str[n];
}変数名
```

なお，64 ビットモードの場合，「long len;」は「int len;」になります。

注※8

内部的に次のように展開されます。

```
unsigned long 変数名;
```

なお，64 ビットモードの場合，「unsigned long 変数名;」は「unsigned int 変数名;」になります。

注※9

内部的に次のように展開されます。

```
char 変数名[n+1];
```

文字列長は，先頭から NULL 文字の一つ前までの長さになります。なお，C 言語の文字列を受け取る場合，n+1 個の配列要素中に NULL 文字がないときは，エラーとなります。

注※10

内部的に次のように展開されます。

```
＜プリプロセサの-XU16オプションで型指定子を指定しなかった場合＞

char 変数名[2n+2];

＜プリプロセサの-XU16オプションで型指定子を指定した場合＞

型指定子 変数名[n+1];
```

-XU16 オプションについては，「UNIX 環境でのプリプロセス」，又は「Windows 環境でのプリプロセス」を参照してください。

注※11

内部的に次のように展開されます。

```
＜プリプロセサの-XU16オプションで型指定子を指定しなかった場合＞
```

```
struct{
         short   len;
         char    str[2n];
} 変数名;
```

＜プリプロセサの-XU16オプションで型指定子を指定した場合＞

```
struct{
         short      len;
         型指定子   str[n];
} 変数名;
```

-XU16 オプションについては，「UNIX 環境でのプリプロセス」，又は「Windows 環境でのプリプロセス」を参照してください。

(2)　配列を使用した場合の SQL のデータ型と C 言語のデータ記述

配列を使用した場合の SQL のデータ型と C 言語のデータ記述を次の表に示します。

表 F–2　配列を使用した場合の SQL のデータ型と C 言語のデータ記述

SQL のデータ型	C 言語のデータ記述	備　　考
SMALLINT	short 変数名[m];	–
INTEGER	long 変数名[m];	–
DECIMAL[(p[,s])]	SQL TYPE IS DECIMAL(p,s) 変数名[m];	1≦p≦38，0≦s≦p
SMALLFLT， REAL	float 変数名[m];	–
FLOAT (DOUBLE PRECISION)	double 変数名[m];	–
CHAR[(n)] [CHARACTER SET 文字集合指定]	char [CHARACTER SET[IS] 文字集合指定] 変数名[m][n+1];	1≦n≦30000 文字集合指定：[MASTER.]EBCDIK
CHAR[(2n)] CHARACTER SET 文字集合指定	char CHARACTER SET[IS] 文字集合指定 変数名[m][2n+2];	1≦n≦15000 文字集合指定：[MASTER.]UTF16
	SQL TYPE IS CHAR(2n) CHARACTER SET[IS] 文字集合指定 変数名[m];	
VARCHAR(n) [CHARACTER SET 文字集合指定]	struct{ short 変数名 1;	1≦n≦32000 文字集合指定：[MASTER.]EBCDIK

SQL のデータ型	C 言語のデータ記述	備　　考
	char 変数名 2[n]; }[CHARACTER SET[IS] 文字集合指定] 構造体名[m];	
	SQL TYPE IS VARCHAR(n) [CHARACTER SET[IS] 文字集合指定] 変数名[m];	
	VARCHAR [CHARACTER SET[IS] 文字集合指定] 変数名[m][n+1];	
VARCHAR(2n) CHARACTER SET 文字集合指定	struct{ short 変数名 1; char 変数名 2[2n]; }CHARACTER SET[IS] 文字集合指定 構造体名[m];	1≦n≦16000 変数名 1：文字列長 変数名 2：文字列 文字列長： バイト数で，2 の倍数 文字集合指定： [MASTER.]UTF16
	SQL TYPE IS VARCHAR(n) CHARACTER SET[IS] 文字集合指定 変数名[m];	
	VARCHAR CHARACTER SET[IS] 文字集合指定 変数名[m][2n+2];	
NCHAR[(n)]	char 変数名[m][2n+1];	1≦n≦15000
NVARCHAR(n)	struct{ short 変数名 1; char 変数名 2[2n]; }構造体名[m];	1≦n≦16000
	SQL TYPE IS NVARCHAR(n) 変数名[m];	
MCHAR[(n)]	char 変数名[m][n+1];	1≦n≦30000
MVARCHAR(n)	struct{ short 変数名 1; char 変数名 2[n]; }構造体名[m];	1≦n≦32000

SQL のデータ型		C 言語のデータ記述	備　　考
		SQL TYPE IS MVARCHAR(n) 変数名[m];	
DATE		char 変数名[m][11];	−
TIME		char 変数名[m][9];	−
TIMESTAMP[(p)]		char 変数名[m][n+1];	p=0 の場合は n=19 p=2 の場合は n=21,22 p=4 の場合は n=23,24 p=6 の場合は n=25,26
INTERVAL YEAR TO DAY		SQL TYPE IS DECIMAL(8,0) 変数名[m];	−
INTERVAL HOUR TO SECOND		SQL TYPE IS DECIMAL(6,0) 変数名[m];	−
ROW		char 変数名[m][n+1];	1≦n≦30000
BLOB		記述できない	−
BINARY		struct{ long 変数名 1; char 変数名 2[n]; }構造体名[m];	配列を使用した FETCH の場合 4≦n≦2147483644（n は 4 の倍数であること） 配列を使用した FETCH 以外の場合 4≦n≦32000（n は 4 の倍数であること）
		SQL TYPE IS BINARY(n) 変数名[m];	
BLOB 位置付け子		記述できない	−
BINARY 位置付け子		SQL TYPE IS BINARY AS LOCATOR 変数名[m];	−
標識変数	BINARY，BINARY 位置付け子以外	short 変数名[m];	−
	BINARY，BINARY 位置付け子	long 変数名[m];	−
SQL 文		記述できない	−

(凡例)

m：配列の要素数（1〜4,096）

n：長さ（バイト）

p：精度（全体のけた数）

s：位取り（小数点以下のけた数）

−：なし

注

64 ビットモードの場合，long の代わりに int を使用してください。

(3)　繰返し列を使用した場合の SQL のデータ型と C 言語のデータ記述

繰返し列を使用した場合の SQL のデータ型と C 言語のデータ記述を次の表に示します。

表 F–3　繰返し列を使用した場合の SQL のデータ型と C 言語のデータ記述

SQL のデータ型	C 言語のデータ記述	備　　考
SMALLINT	PD_MV_SINT(m)　変数名;	−
INTEGER	PD_MV_INT(m)　変数名;	−
DECIMAL	PD_MV_DEC(m,p,s)　変数名;	1≦p≦38，0≦s≦p
SMALLFLT， REAL	PD_MV_SFLT(m)　変数名;	−
FLOAT (DOUBLE PRECISION)	PD_MV_FLT(m)　変数名;	−
CHAR[(n)]	PD_MV_CHAR(m,n)　変数名;	1≦n≦30000
VARCHAR(n)	PD_MV_VCHAR(m,n)　変数名;	1≦n≦32000
	PD_MV_CVCHAR(m,n)　変数名;	
NCHAR[(n)]	PD_MV_NCHAR(m,n)　変数名;	1≦n≦15000
NVARCHAR(n)	PD_MV_NVCHAR(m,n)　変数名;	1≦n≦16000
MCHAR[(n)]	PD_MV_CHAR(m,n)　変数名;	1≦n≦30000
MVARCHAR(n)	PD_MV_VCHAR(m,n)　変数名;	1≦n≦32000
DATE	PD_MV_CHAR(m,10)　変数名;	−
TIME	PD_MV_CHAR(m,8)　変数名;	−
TIMESTAMP[(p)]	PD_MV_CHAR(m,n)　変数名;	p=0 の場合は n=19 p=2 の場合は n=21,22 p=4 の場合は n=23,24 p=6 の場合は n=25,26
INTERVAL YEAR TO DAY	PD_MV_DEC(m,8,0)　変数名;	−
INTERVAL HOUR TO SECOND	PD_MV_DEC(m,6,0)　変数名;	−
ROW	記述できない	−
BLOB	記述できない	−
BINARY	記述できない	−

SQL のデータ型	C 言語のデータ記述	備　　考
標識変数（BLOB，BINARY，BLOB 位置付け子，BINARY 位置付け子以外）	PD_MV_SINT(m)　変数名;	−
SQL 文	記述できない	−

(凡例)

m：繰返し列の最大要素数（2〜30,000）

n：長さ（バイト）

p：精度（全体のけた数）

s：位取り（小数点以下のけた数）

−：なし

(4)　埋込み変数の参照・設定に使用するマクロ

繰返し列を使用した場合の SQL のデータ型と C 言語のデータ記述では，各データ型の埋込み変数の参照・設定に，専用のマクロを使用します。埋込み変数の参照・設定に使用するマクロを次の表に示します。

表 F−4　埋込み変数の参照・設定に使用するマクロ

SQL のデータ型	マクロ名	参照・設定するデータ	データ型
SMALLINT	PD_MV_SINT_CNT(変数名)	繰返しデータの現在要素数	long※
	PD_MV_SINT_DATA(変数名，m)	繰返しの各要素	short
INTEGER	PD_MV_INT_CNT(変数名)	繰返しデータの現在要素数	long※
	PD_MV_INT_DATA(変数名，m)	繰返しの各要素	long※
DECIMAL[(p[,s])]	PD_MV_DEC_CNT(変数名)	繰返しデータの現在要素数	long※
	PD_MV_DEC_DATA(変数名，m)	繰返しの各要素の 10 進数の先頭アドレス	unsigned
SMALLFLT，REAL	PD_MV_SFLT_CNT(変数名)	繰返しデータの現在要素数	long※
	PD_MV_SFLT_DATA(変数名，m)	繰返しの各要素	float
FLOAT (DOUBLE PRECISION)	PD_MV_FLT_CNT(変数名)	繰返しデータの現在要素数	long※
	PD_MV_FLT_DATA(変数名，m)	繰返しの各要素	double
CHAR[(n)]	PD_MV_CHAR_CNT(変数名)	繰返しデータの現在要素数	long※
	PD_MV_CHAR_DATA(変数名，m)	繰返しの各要素の，文字列の先頭アドレス	char[ ]
VARCHAR(n)	PD_MV_VCHAR_CNT(変数名)	繰返しデータの現在要素数	long※
	PD_MV_VCHAR_LEN(変数名，m)	繰返しの各要素の，文字列の実長	short

SQL のデータ型	マクロ名	参照・設定するデータ	データ型
	PD_MV_VCHAR_STR(変数名，m)	繰返しの各要素の，文字列のアドレス	char[ ]
	PD_MV_CVCHAR_CNT(変数名)	繰返しデータの現在要素数	long※
	PD_MV_CVCHAR_DATA(変数名，m)	繰返しの各要素の，文字列のアドレス	char[ ]
NCHAR[(n)]	PD_MV_NCHAR_CNT(変数名)	繰返しデータの現在要素数	long※
	PD_MV_NCHAR_DATA(変数名，m)	繰返しの各要素の，文字列の先頭アドレス	char[ ]
NVARCHAR(n)	PD_MV_NVCHAR_CNT(変数名)	繰返しデータの現在要素数	long※
	PD_MV_NVCHAR_LEN(変数名，m)	繰返しの各要素の，文字列の実長	short
	PD_MV_NVCHAR_STR(変数名，m)	繰返しの各要素の，文字列の先頭アドレス	char[ ]
MCHAR[(n)]	PD_MV_CHAR_CNT(変数名)	繰返しデータの現在要素数	long※
	PD_MV_CHAR_DATA(変数名，m)	繰返しの各要素の，文字列の先頭アドレス	char[ ]
MVARCHAR(n)	PD_MV_VCHAR_CNT(変数名)	繰返しデータの現在要素数	long※
	PD_MV_VCHAR_LEN(変数名，m)	繰返しの各要素の，文字列の実長	short
	PD_MV_VCHAR_STR(変数名，m)	繰返しの各要素の，文字列のアドレス	char[ ]
DATE	CHAR(10)と同じ	−	−
TIME	CHAR(8)と同じ	−	−
TIMESTAMP[(p)]	CHAR(n)と同じ p=0 の場合は n=19 p=2 の場合は n=21,22 p=4 の場合は n=23,24 p=6 の場合は n=25,26	−	−
INTERVAL YEAR TO DAY	DECIMAL(8,0)と同じ	−	−
INTERVAL HOUR TO SECOND	DECIMAL(6,0)と同じ	−	−
標識変数	PD_MV_SINT_CNT(変数名)	繰返し列全体の標識	long※
	PD_MV_SINT_DATA(変数名，m)	繰返し列の各要素の標識	short

(凡例)

−：該当しません。

m：繰返し列の各要素を示す番号（0〜m-1）

n：長さ（バイト）

p：精度（全体のけた数）

s：位取り（小数点以下のけた数）

注※

64 ビットモードの場合，int となります。

繰返し列用の埋込み変数を参照・設定するマクロの使用例を次に示します。

```
EXEC SQL BEGIN DECLARE SECTION;
char xname[5];
PD_MV_SINT(4) xmten;
PD_MV_CHAR(4,5) xmkamoku;
EXEC SQL END DECLARE SECTION;
         :
strcpy(xname,"山田")
PD_MV_SINT_DATA(xmten,0)=90;
PD_MV_SINT_DATA(xmten,1)=65;
PD_MV_SINT_DATA(xmten,2)=85;
PD_MV_SINT_DATA(xmten,3)=55;
PD_MV_SINT_CNT(xmten)=4;
strcpy(PD_MV_CHAR_DATA(xmkamoku,0),"数学");
strcpy(PD_MV_CHAR_DATA(xmkamoku,1),"国語");
strcpy(PD_MV_CHAR_DATA(xmkamoku,2),"理科");
strcpy(PD_MV_CHAR_DATA(xmkamoku,3),"社会");
PD_MV_CHAR_CNT(xmkamoku)=4;
EXEC SQL
  INSERT INTO 成績表(氏名,科目,成績) VALUES(:xname,:xmkamoku,:xmten);
```

(5)　ポインタ変数と C 言語のデータ記述

ポインタ変数と C 言語のデータ記述を次の表に示します。

表 F–5　ポインタ変数と C 言語のデータ記述

SQL のデータ型	C 言語のデータ記述	備考
SMALLINT	short *変数名;	−
INTEGER	long *変数名;	−
DECIMAL[(p[,s])]	SQL TYPE IS DECIMAL(p,s) *変数名;	1≦p≦38，0≦s≦p
SMALLFLT，REAL	float *変数名;	−
FLOAT (DOUBLE PRECISION)	double *変数名;	−
CHAR[(n)] [CHARACTER SET 文字集合指定]	char [CHARACTER SET[IS] 文字集合指定]*変数名;	1≦n≦30000※ 文字集合指定： [MASTER.]EBCDIK
CHAR[(2n)] CHARACTER SET 文字集合指定	SQL TYPE IS CHAR(2n) CHARACTER SET[IS] 文字集合指定 *変数名;	1≦n≦15000 文字集合指定： [MASTER.]UTF16

SQL のデータ型	C 言語のデータ記述	備考
VARCHAR(n) [CHARACTER SET 文字集合指定]	struct { short 変数名 1; char 変数名 2[n]; } [CHARACTER SET[IS] 文字集合指定]*構造体名;	1≦n≦32000 変数名 1：文字列長（バイト数） 変数名 2：文字列 文字集合指定： [MASTER.]EBCDIK
	SQL TYPE IS VARCHAR(n) [CHARACTER SET[IS] 文字集合指定] *変数名;	
	VARCHAR [CHARACTER SET[IS] 文字集合指定] *変数名;※	
VARCHAR(2n) CHARACTER SET 文字集合指定	struct { short 変数名 1; char 変数名 2[2n]; } CHARACTER SET[IS] 文字集合指定 *構造体名;	1≦n≦16000 変数名 1：文字列長 変数名 2：文字列 文字列長： バイト数で，2 の倍数 文字集合指定： [MASTER.]UTF16
	SQL TYPE IS VARCHAR(2n) CHARACTER SET[IS] 文字集合指定 *変数名;	
NCHAR[(n)]	char *変数名;	1≦n≦15000※
NVARCHAR(n)	struct { short 変数名 1; char 変数名 2[2n]; } *構造体名;	1≦n≦16000 変数名 1：文字列長（文字数） 変数名 2：文字列
	SQL TYPE IS NVARCHAR(n) *変数名;	
MCHAR[(n)]	char *変数名;	1≦n≦30000※
MVARCHAR(n)	struct { short 変数名 1; char 変数名 2[n]; } *構造体名;	1≦n≦32000 変数名 1：文字列長（バイト数） 変数名 2：文字列
	SQL TYPE IS MVARCHAR(n) *変数名;	
DATE※	char *変数名;	−
TIME※	char *変数名;	−
TIMESTAMP※	char *変数名;	−
INTERVAL YEAR TO DAY	SQL TYPE IS DECIMAL(8,0)	−

SQL のデータ型		C 言語のデータ記述	備考
		*変数名;	
INTERVAL HOUR TO SECOND		SQL TYPE IS DECIMAL(6,0) *変数名;	−
ROW		char *変数名;	1≦全体長≦30000※
BLOB		SQL TYPE IS BLOB(n[{K ｜ M ｜ G}]) *変数名;	単位を省略した場合： 1≦n≦2147483647 単位が K の場合： 1≦n≦2097152 単位が M の場合： 1≦n≦2048 単位が G の場合： 1≦n≦2
BINARY(n)		struct { long 変数名 1; char 変数名 2[n]; } *構造体名;	1≦n≦2147483647
		SQL TYPE IS BINARY(n) *変数名;	
BLOB 位置付け子		SQL TYPE IS BLOB AS LOCATOR *変数名;	−
BINARY 位置付け子		SQL TYPE IS BINARY AS LOCATOR *変数名;	−
標識変数	BLOB，BINARY， BLOB 位置付け子， BINARY 位置付け子 以外	short *変数名;	−
	BLOB，BINARY， BLOB 位置付け子， BINARY 位置付け子	long *変数名;	
SQL 文		struct { long 変数名 1; char 変数名 2[n]; } *構造体名;	1≦n≦2000000
SMALLINT ARRAY m		PD_MV_SINT(m) *変数名;	−
INTEGER ARRAY m		PD_MV_INT(m)	−

SQL のデータ型	C 言語のデータ記述	備考
	*変数名;	
DECIMAL[(p[,s])] ARRAY m	PD_MV_DEC(m,p,s) *変数名;	1≦p≦38，0≦s≦p
SMALLFLT ARRAY m (REAL)	PD_MV_SFLT(m) *変数名;	−
FLOAT ARRAY m (DOUBLE PRECISION)	PD_MV_FLT(m) *変数名;	−
CHAR[(n)] ARRAY m，及び MCHAR[(n)] ARRAY m	PD_MV_CHAR(m,n) *変数名;	1≦n≦30000
VARCHAR[(n)] ARRAY m，及び MVARCHAR[(n)] ARRAY m	PD_MV_VCHAR(m,n) *変数名;	1≦n≦32000
	PD_MV_CVCHAR(m,n) *変数名;	
NCHAR[(n)] ARRAY m	PD_MV_NCHAR(m,n) *変数名;	1≦n≦15000
NVARCHAR[(n)] ARRAY m	PD_MV_NVCHAR(m,n) *変数名;	1≦n≦16000
DATE ARRAY m	PD_MV_CHAR(m,10) *変数名;	−
TIME ARRAY m	PD_MV_CHAR(m,8) *変数名;	−
TIMESTAMP ARRAY m	PD_MV_CHAR(m,n) *変数名;	p=0 の場合は n=19 p=2 の場合は n=21,22 p=4 の場合は n=23,24 p=6 の場合は n=25,26
INTERVAL YEAR TO DAY ARRAY m	PD_MV_DEC(m,8,0) *変数名;	−
INTERVAL HOUR TO SECOND ARRAY m	PD_MV_DEC(m,6,0) *変数名;	−
繰返し列用標識変数	PD_MV_SINT(m) *変数名;	−

(凡例)

m：繰返し列の各要素を示す番号（0～m-1）

n：長さ（バイト）

p：精度（全体のけた数）

s：位取り（小数点以下のけた数）

–：なし

注

64 ビットモードの場合，long の代わりに int を使用してください。

注※

プリプロセス時に領域の定義長が決定できないため，実行時にポインタが指す領域に格納されている文字列の長さを，strlen(変数名)で求めて，領域長の代わりに使用します。検索結果を受け取る場合は，ポインタが指す領域を NULL 文字以外でクリアして，最後に NULL 文字を入れてください。

(6)　ポインタ型の繰返し列用のマクロ

ポインタ型の繰返し列用の変数を参照，及び設定するには，専用のマクロを使用します。ポインタ型の繰返し列用のマクロを次の表に示します。

表 F–6　ポインタ型の繰返し列用のマクロ

SQL のデータ型	マクロ名	参照・設定するデータ	データ型
SMALLINT ARRAY m	PD_MV_SINTP_CNT(変数名)	繰返しデータの現在要素数	long※
	PD_MV_SINTP_DATA(変数名，m)	繰返しの各要素	short
INTEGER ARRAY m	PD_MV_INTP_CNT(変数名)	繰返しデータの現在要素数	long※
	PD_MV_INTP_DATA(変数名，m)	繰返しの各要素	long※
DECIMAL[(p[,s])] ARRAY m	PD_MV_DECP_CNT(変数名)	繰返しデータの現在要素数	long※
	PD_MV_DECP_DATA(変数名，m)	繰返しの各要素の 10 進数の先頭アドレス	char[ ]
SMALLFLT ARRAY m (REAL)	PD_MV_SFLTP_CNT(変数名)	繰返しデータの現在要素数	long※
	PD_MV_SFLTP_DATA(変数名，m)	繰返しの各要素	float
FLOAT ARRAY m (DOUBLE PRECISION)	PD_MV_FLTP_CNT(変数名)	繰返しデータの現在要素数	long※
	PD_MV_FLTP_DATA(変数名，m)	繰返しの各要素	double
CHAR[(n)] ARRAY m，又は MCHAR[(n)] ARRAY m	PD_MV_CHARP_CNT(変数名)	繰返しデータの現在要素数	long※
	PD_MV_CHARP_DATA(変数名，m)	繰返しの各要素の，文字列の先頭アドレス	char[ ]
VARCHAR(n) ARRAY m，又は MVARCHAR(n) ARRAY m	PD_MV_VCHARP_CNT(変数名)	繰返しデータの現在要素数	long※
	PD_MV_VCHARP_LEN(変数名，m)	繰返しの各要素の，文字列の実長	short
	PD_MV_VCHARP_STR(変数名，m)	繰返しの各要素の，文字列のアドレス	char[ ]
	PD_MV_CVCHARP_CNT(変数名)	繰返しデータの現在要素数	long※

SQL のデータ型	マクロ名	参照・設定するデータ	データ型
	PD_MV_CVCHARP_DATA(変数名，m)	繰返しの各要素の，文字列のアドレス	char[ ]
NCHAR[(n)] ARRAY m	PD_MV_NCHARP_CNT(変数名)	繰返しデータの現在要素数	long※
	PD_MV_NCHARP_DATA(変数名，m)	繰返しの各要素の，文字列の先頭アドレス	char[ ]
NVARCHAR(n) ARRAY m	PD_MV_NVCHARP_CNT(変数名)	繰返しデータの現在要素数	long※
	PD_MV_NVCHARP_LEN(変数名，m)	繰返しの各要素の，文字列の実長	short
	PD_MV_NVCHARP_STR(変数名，m)	繰返しの各要素の，文字列の先頭アドレス	char[ ]
DATE ARRAY m	CHAR(10)と同じ	−	−
TIME ARRAY m	CHAR(8)と同じ	−	−
TIMESTAMP[(p)] ARRAY m	CHAR(n)と同じ p=0 の場合は n=19 p=2 の場合は n=21,22 p=4 の場合は n=23,24 p=6 の場合は n=25,26	−	−
INTERVAL YEAR TO DAY	DECIMAL(8,0)と同じ	−	−
INTERVAL HOUR TO SECOND	DECIMAL(6,0)と同じ	−	−
標識変数	PD_MV_SINTP_CNT(変数名)	繰返し列全体の標識	long※
	PD_MV_SINTP_DATA(変数名，m)	繰返し列の各要素の標識	short

(凡例)

−：該当しません。

m：繰返し列の各要素を示す番号（0〜m-1）

n：長さ（バイト）

p：精度（全体のけた数）

s：位取り（小数点以下のけた数）

注※

64 ビットモードの場合，int となります。

(7) 一括指定用の構造体

一括指定用の構造体を次の表に示します。

表 F–7　一括指定用の構造体

SQL のデータ型	C 言語のデータ記述	項目の記述	備考
複数の項目	表 F-1～表 F-3 のデータ型をメンバとして含む構造体	複数の埋込み変数を一括指定します。	ポインタ宣言ができます。
複数の項目用の標識変数	表 F-1～表 F-3 の標識変数をメンバとして含む構造体	複数の標識変数を一括指定します。	ポインタ宣言ができます。

付録 F.2　SQL のデータ型と COBOL 言語のデータ記述

SQL のデータ型と COBOL 言語のデータ記述の対応を示します。

なお，データの受け渡しには，対応するデータ型以外に，変換，又は代入できるデータ型も使用できます。

(1)　SQL のデータ型と COBOL 言語のデータ記述

SQL のデータ型と COBOL 言語のデータ記述を次の表に示します。なお，表中のデータ記述は，次のように表記することもできます。

PICTURE：
　PIC

COMPUTATIONAL：
　COMP

COMPUTATIONAL-n：
　COMP-n

9(n)：
　99…9

X(n)：
　XX…X

OCCURS n TIMES：
　OCCURS 1 TO n TIMES
　OCCURS 1 TO n
　OCCURS n

表 F–8　SQL のデータ型と COBOL 言語のデータ記述

SQL のデータ型	COBOL 言語のデータ記述	項目の記述	備　　考
SMALLINT	L1 基本項目名 PICTURE S9(4) COMPUTATIONAL.	基本項目，又は独立項目	－

SQL のデータ型	COBOL 言語のデータ記述	項目の記述	備　　考
INTEGER	L1 基本項目名 PICTURE S9(9) COMPUTATIONAL.	基本項目，又は独立項目	−
DECIMAL[(p[，s])]	L1 基本項目名 PICTURE S9(p-s)[V9(s)] COMPUTATIONAL-3.	基本項目，又は独立項目	1≦p≦38※１０， 0≦s≦p p=s の場合，SV9(s)とします。 s=0 の場合，[V9(s)]を省略します。
	L1 基本項目名 PICTURE S9(p-s)[V9(s)] DISPLAY SIGN LEADING SEPARATE.※９※１２		
	L1 基本項目名 PICTURE S9(p-s)[V9(s)] DISPLAY SIGN TRAILING.※１１※１２※１３		
SMALLFLT (REAL)	L1 基本項目名 COMPUTATIONAL-1.	基本項目，又は独立項目	−
FLOAT (DOUBLE PRECISION)	L1 基本項目名 COMPUTATIONAL-2.	基本項目，又は独立項目	−
CHAR[(n)] [CHARACTER SET [MASTER.]EBCDIK]	L1 基本項目名 [CHARACTER SET[IS] [MASTER.]EBCDIK] PICTURE X(n).※５	基本項目，又は独立項目	1≦n≦30000
CHAR[(2n)] CHARACTER SET [MASTER.]UTF16	**HiRDB サーバの既定文字集合が UTF-8，又は IVS 対応 UTF-8 以外の場合：**※15 CHAR 型は使用できません。 **HiRDB サーバの既定文字集合が UTF-8，又は IVS 対応 UTF-8 の場合：**※15 L1 基本項目名 PICTURE N(n).	基本項目，又は独立項目	1≦n≦15000
VARCHAR(n) [CHARACTER SET [MASTER.]EBCDIK]	L2 集団項目名 [CHARACTER SET[IS] [MASTER.]EBCDIK]. L3 基本項目名 1 PICTURE S9(4) COMPUTATIONAL. L3 基本項目名 2	二つの基本項目から構成される集団項目 基本項目名 1：文字列長 基本項目名 2：文字列 文字列長はバイト数	1≦n≦32000

SQL のデータ型	COBOL 言語のデータ記述	項目の記述	備　　考
	PICTURE X(n).※5		
VARCHAR(2n) CHARACTER SET [MASTER.]UTF16	**HiRDB サーバの既定文字集合が UTF-8，又は IVS 対応 UTF-8 以外の場合：**※15 VARCHAR 型は使用できません。 **HiRDB サーバの既定文字集合が UTF-8，又は IVS 対応 UTF-8 の場合：**※15 L2 集団項目名. L3 基本項目名 1 PICTURE S9(4) COMPUTATIONAL. L3 基本項目名 2 PICTURE N(n).	二つの基本項目から構成される集団項目 基本項目名 1：文字列長 基本項目名 2：文字列 文字列長はバイト数	1≦n≦16000
NCHAR[(n)]	**HiRDB サーバの既定文字集合が UTF-8，又は IVS 対応 UTF-8 以外の場合：**※15 L1 基本項目名 PICTURE N(n). **HiRDB サーバの既定文字集合が UTF-8，又は IVS 対応 UTF-8 の場合：**※15 NCHAR 型は使用できません。	基本項目，又は独立項目	1≦n≦15000
NVARCHAR(n)	**HiRDB サーバの既定文字集合が UTF-8，又は IVS 対応 UTF-8 以外の場合：**※15 L2 集団項目名. L3 基本項目名 1 PICTURE S9(4) COMPUTATIONAL. L3 基本項目名 2 PICTURE N(n). **HiRDB サーバの既定文字集合が UTF-8，又は IVS 対応 UTF-8 の場合：**※15 NVARCHAR 型は使用できません。	二つの基本項目から構成される集団項目 基本項目名 1：文字列長基本項目名 2：文字列	1≦n≦16000
MCHAR[(n)]	L1 基本項目名 PICTURE X(n).※6 HiRDB サーバの既定文字集合が UTF-8，又は IVS 対応 UTF-8 の場合は※15，次のように記述することもできます。	基本項目，又は独立項目	1≦n≦30000

SQL のデータ型	COBOL 言語のデータ記述	項目の記述	備　　考
	L1 基本項目名 PICTURE N(n_2).※14		
MVARCHAR(n)	L2 集団項目名. L3 基本項目名 1 PICTURE S9(4) COMPUTATIONAL. L3 基本項目名 2 PICTURE X(n).※6 HiRDB サーバの既定文字集合が UTF-8，又は IVS 対応 UTF-8 の場合は※15，次のように記述することもできます。 L2 集団項目名. L3 基本項目名 1 PICTURE S9(4) COMPUTATIONAL. L3 基本項目名 2 PICTURE N(n_2).※14	二つの基本項目から構成される集団項目 基本項目名 1：文字列長 基本項目名 2：文字列	1≦n≦32000
DATE	L1 基本項目名 PICTURE X(10).※6	基本項目，又は独立項目	−
TIME	L1 基本項目名 PICTURE X(8).※6	基本項目，又は独立項目	−
TIMESTAMP[(p)]	L1 基本項目名 PICTURE X(n).※6	基本項目，又は独立項目	p=0 の場合は n=19 p=2 の場合は n=21,22 p=4 の場合は n=23,24 p=6 の場合は n=25,26
INTERVAL YEARTO DAY	L1 基本項目名 PICTURE S9(8) COMPUTATIONAL-3.	基本項目，又は独立項目	−
INTERVAL HOUR TO SECOND	L1 基本項目名 PICTURE S9(6) COMPUTATIONAL-3.	基本項目，又は独立項目	−
ROW※3	この表中のデータ項目と集団項目の組み合わせ※1	複数の基本項目から構成される集団項目	1≦全体長≦30000
BLOB	L2 集団項目名※2 [USAGE [IS]] SQL TYPE IS BLOB(n{K ｜ M ｜ G}).※4※7	基本項目	単位省略時： 1≦n≦2147483647 単位が K： 1≦n≦2097152 単位が M： 1≦n≦2048

SQL のデータ型		COBOL 言語のデータ記述	項目の記述	備　　考
				単位が G： 1≦n≦2
BINARY(n)		L2 集団項目名. L3 基本項目名 1 PICTURE S9(9) COMPUTATIONAL. L3 基本項目名 2 PICTURE X(n).※5※7	二つの基本項目から構成される集団項目 基本項目名 1：文字列長 基本項目名 2：文字列 文字列長はバイト数	1≦n≦2147483647
BLOB 位置付け子		L1 基本項目名 SQL TYPE IS BLOB AS LOCATOR.※8	基本項目，又は独立項目	−
BINARY 位置付け子		L1 基本項目名 SQL TYPE IS BINARY AS LOCATOR.※8	基本項目，又は独立項目	−
標識変数	BLOB，BINARY，BLOB 位置付け子，BINARY 位置付け子以外	L1 基本項目名 PICTURE S9(4) COMPUTATIONAL.	基本項目，又は独立項目	−
	BLOB，BINARY，BLOB 位置付け子，BINARY 位置付け子	L1 基本項目名 PICTURE S9(9) COMPUTATIONAL.		
SQL 文		L2 集団項目名. L3 基本項目名 1 PICTURE S9(9) COMPUTATIONAL. L3 基本項目名 2 PICTURE X(n).	二つの基本項目から構成される集団項目 基本項目名 1：文字列長 基本項目名 2：文字列	1≦n≦2000000

(凡例)

L1：レベル番号 01～49，又は 77

L2：レベル番号 01～48

L3：レベル番号 02～49(L2<L3)

n：長さ（バイト）

p：精度（全体のけた数）

s：位取り（小数点以下のけた数）

−：なし

注※1

次に示す句を使用できます。

- REDEFINES 句
- OCCURS 句
- ADDRESSED BY 句

FIX 属性の表を行単位に操作する場合，1 行分のデータを格納するデータ項目を集団項目として記述します。操作対象の表の各列と集団項目に従属する各データ項目とが対応するように記述します。各列に対応するデータ項目はその列のデータ型に対応するデータ記述を使用して記述します。ただし，データ形式の変換が必要になるデータ記述は使用できません。

データ形式の変換が必要になるデータ記述と，その代わりに使用できるデータ記述を次の表に示します。

列のデータ型	データ形式の変換が必要になるため使用できないデータ記述	代わりに使用できるデータ記述
DECIMAL	DISPLAY （外部 10 進形式）	COMP-3 （内部 10 進形式）
MCHAR	PICTURE N（日本語項目）	PICTURE X（英数字項目）
DATE	PICTURE X(10) 10=文字列表現の文字数	PICTURE X(4) 4=X'YYYYMMDD'形式のバイト数
TIME	PICTURE X(8) 8=文字列表現の文字数	PICTURE X(3) 3=X'hhmmss'形式のバイト数
TIMESTAMP(p)	PICTURE X(n) n=文字列表現の文字数	PICTURE X(n) n=7 ＋ p/2 n=X'YYYYMMDDhhmmss〔nn…n〕'形式のバイト数

操作対象の表の列に対応するデータ項目を，REDEFINES 句を使用して再定義できます。この場合，再定義する項目は再定義される項目に対応する列のデータ型と関係なく記述できます。ただし，表「SQL のデータ型と COBOL 言語のデータ記述」の INTEGER，SMALLINT，DECIMAL，FLOAT，SMALLFLT，CHAR，及び NCHAR 型に対応する基本項目とこれらを組み合わせた集団項目だけが使用できます。これらの基本項目と集団項目に，OCCURS 句を指定できます。

プリプロセサの-Xb オプションを指定する場合は，ROW 型のデータ項目に含まれる 2 進項目のデータ記述の USAGE 句に COMP の代わりに COMP-5 を指定する必要があります。-Xb オプションを指定しない場合は，ROW 型のデータ項目に含まれる 2 進項目のデータ記述の USAGE 句に COMP と COMP-5 のどちらでも指定できます。

注※2

集団項目名は 21 文字以下にしてください。ただし，COBOL2002 の場合は，22 文字以下にしてください。

注※3

HiRDB サーバと HiRDB クライアントのエンディアンが同じ場合にだけ，ROW 型を使用できます。

注※4

BLOB の UAP の記述は，内部的に次のように展開されます。

```
L2 集団項目名.
   49 集団項目名-RESERVED PIC S9(9) USAGE IS BINARY.  … 1
   49 集団項目名-LENGTH   PIC S9(9) USAGE IS BINARY.  … 2
   49 集団項目名-DATA     PIC X(m).                   … 3
```

1. 変数名-RESERVED は，使用しません。
2. 変数名-LENGTH は，BLOB の実際の長さになります。
3. 変数名-DATA は，BLOB のデータ格納領域(m は実際のデータ長)になります。

注※5

X の代わりに 9 を使用して定義できます。9 を使用して定義した場合，数字以外の文字を含む文字列を代入したり，検索結果として受け取ったときの動作は COBOL 言語のコンパイラの実装に依存します。

注※6

X の代わりに 9 を使用して定義した場合，プリプロセス時にエラーとなりませんが，使用しないでください。

注※7

宣言できる上限値は，COBOL コンパイラの実装に依存します。詳細については，使用する COBOL コンパイラのマニュアルを参照してください。

注※8

内部的には，次のように展開されます。

```
L1 基本項目名 PICTURE S9(9) COMPUTATIONAL.
```

注※9

HiRDB サーバのデータ型は DECIMAL 型ですが，先頭 1 バイトに演算符号が付く外部 10 進項目として表します。

注※10

COBOL コンパイラの仕様に依存します。例えば，COBOL85 の場合は 1≦p≦18 となります。

注※11

HiRDB サーバのデータ型は DECIMAL 型ですが，右端 1 バイトの上位 4 ビットに演算符号が付く外部 10 進項目（ゾーン形式）として表します。

注※12

演算符号が付く数字項目にデータ項目の表現形式が指定されていない場合は，DISPLAY（外部 10 進項目）とみなします。演算符号が付く外部 10 進項目に SIGN 句が指定されていない場合は，演算符号の位置と表現形式は DISPLAY SIGN TRAILING 形式と同じであるとみなします。

注※13

DISPLAY SIGN TRAILING 形式は，COBOL85 及び COBOL2002 でだけサポートしています。

注※14

COBOL 言語の日本語項目（PICTURE N）を含むこのデータ記述は，COBOL2002 の Unicode 機能を使用する場合にだけ，SQL の混在文字データ型（MCHAR 又は MVARCHAR）に対応するデータ記述として使用できます。

COBOL2002 の Unicode 機能を使用した UAP の実行については「COBOL2002 の Unicode 機能を使用した UAP の実行」を参照してください。

このデータ記述は，埋込み変数の宣言，及び？パラメタの値を格納するデータ領域の宣言に使用できます。

埋込み変数の場合

-XU16 オプションの指定に従って，文字集合名 UTF-16LE 又は UTF-16BE を指定した文字データ型（CHAR 又は VARCHAR）のデータとして扱います。詳細を次の表に示します。

データ記述の形式	COBOL 言語のデータ記述	埋込み変数の属性	
		データ型	文字集合名
固定長	L1 基本項目名 PICTURE N(n_2).	CHAR(m) $m = 2 \times n_2$	UTF-16LE 又は UTF-16BE
可変長	L2 集団項目名. L3 基本項目名 1 PICTURE S9(4) COMPUTATIONAL. L3 基本項目名 2 PICTURE N(n_2).	VARCHAR(m) $m = 2 \times n_2$	UTF-16LE 又は UTF-16BE

注　可変長形式の基本項目名 1 には，基本項目名 2 に格納する日本語文字列の長さをバイト単位で指定します。日本語項目に格納するデータの文字コードは UTF-16（UCS-2 の範囲内）になるため，日本語文字列のバイト数は文字数の 2 倍になります。なお，このバイト数は $2 \times n_2$ 以下になります。

？パラメタの値を格納するデータ領域の場合

？パラメタの値を格納するデータ領域の宣言にこのデータ記述を使用する場合は，SQL 記述領域と文字集合名記述領域を次の表に示すとおりに設定してください。

データ記述の形式	COBOL 言語のデータ記述	SQL 記述領域の設定値		文字集合名記述領域の設定値
		データコード	データ長	
固定長	L1 基本項目名 PICTURE N(n_2).	CHAR 型のデータコード	$2 \times n_2$	UTF-16LE 又は UTF-16BE
可変長	L2 集団項目名. L3 基本項目名 1 PICTURE S9(4) COMPUTATIONAL. L3 基本項目名 2 PICTURE N(n_2).	VARCHAR 型のデータコード	$2 \times n_2$	UTF-16LE 又は UTF-16BE

注　可変長形式の基本項目名 1 には，基本項目名 2 に格納する日本語文字列の長さをバイト単位で指定します。日本語項目に格納するデータの文字コードは UTF-16（UCS-2 の範囲内）になるため，日本語文字列のバイト数は文字数の 2 倍になります。なお，このバイト数は $2 \times n_2$ 以下になります。

なお，文字集合名の詳細については，「文字集合名記述領域の SQLCSN に設定できる文字集合情報」を参照してください。

注※15

pdntenv コマンド（UNIX 版の場合は pdsetup コマンド）で，文字コード種別に utf-8 を指定した場合に，HiRDB サーバの既定文字集合が UTF-8 になります。

また，文字コード種別に utf-8_ivs を指定した場合，HiRDB サーバの既定文字集合が IVS 対応 UTF-8 になります。

(2)　配列を使用した場合の SQL のデータ型と COBOL 言語のデータ記述

配列を使用した場合の SQL のデータ型と COBOL 言語のデータ記述を次の表に示します。

表 F–9　配列を使用した場合の SQL のデータ型と COBOL 言語のデータ記述

SQL のデータ型	COBOL 言語のデータ記述	項目の記述	備　　考
SMALLINT	L2 基本項目名 PICTURE S9(4) COMPUTATIONAL OCCURS m TIMES.	OCCURS の指定によって同じデータ構造を繰り返す反復データ項目から構成される集団項目	−
INTEGER	L2 基本項目名 PICTURE S9(9) COMPUTATIONAL OCCURS m TIMES.		−
DECIMAL [(p[,s])]	L2 基本項目名 PICTURE S9(p-s)[V9(s)] COMPUTATIONAL-3 OCCURS m TIMES.		1≦p≦38※3， 0≦s≦p p=s の場合，SV9(s)とします。 s=0 の場合，[V9(s)]を省略します。
	L2 基本項目名 PICTURE S9(p-s)[V9(s)] DISPLAY SIGN LEADING SEPARATE OCCURS m TIMES.		
	L2 基本項目名 PICTURE S9(p-s)[V9(s)] DISPLAY SIGN TRAILING OCCURS m TIMES.		
SMALLFLT (REAL)	L2 基本項目名 COMPUTATIONAL-1		−

SQL のデータ型	COBOL 言語のデータ記述	項目の記述	備　　考
	OCCURS m TIMES.		
FLOAT (DOUBLE PRECISION)	L2 基本項目名 COMPUTATIONAL-2 OCCURS m TIMES.		−
CHAR[(n)] [CHARACTER SET [MASTER.]EBCDIK]	L2 基本項目名 [CHARACTER SET[IS] [MASTER.]EBCDIK] PICTURE X(n) OCCURS m TIMES.※1		1≦n≦30000
CHAR[(2n)] CHARACTER SET [MASTER.]UTF16	**HiRDB サーバの既定文字集合が UTF-8，又は IVS 対応 UTF-8 以外の場合：**※5 CHAR 型は使用できません。 **HiRDB サーバの既定文字集合が UTF-8，又は IVS 対応 UTF-8 以外の場合：**※5 L2 基本項目名 PICTURE N(n) OCCURS m TIMES.		1≦n≦15000
VARCHAR(n) [CHARACTER SET [MASTER.]EBCDIK]	L2 集団項目名 2 [CHARACTER SET[IS] [MASTER.]EBCDIK] OCCURS m TIMES. L3 基本項目名 1 PICTURE S9(4) COMPUTATIONAL. L3 基本項目名 2 PICTURE X(n).※1		1≦n≦32000
VARCHAR(2n) CHARACTER SET [MASTER.]UTF16	**HiRDB サーバの既定文字集合が UTF-8，又は IVS 対応 UTF-8 以外の場合：**※5 VARCHAR 型は使用できません。 **HiRDB サーバの既定文字集合が UTF-8，又は IVS 対応 UTF-8 以外の場合：**※5 L2 集団項目名 2 OCCURS m TIMES. L3 基本項目名 1 PICTURE S9(4) COMPUTATIONAL. L3 基本項目名 2		1≦n≦16000

SQL のデータ型	COBOL 言語のデータ記述	項目の記述	備　　考
	PICTURE N(n).		
NCHAR[(n)]	**HiRDB サーバの既定文字集合が UTF-8，又は IVS 対応 UTF-8 以外の場合：**※5 L2 基本項目名 PICTURE N(n) OCCURS m TIMES. **HiRDB サーバの既定文字集合が UTF-8，又は IVS 対応 UTF-8 以外の場合：**※5 NCHAR 型は使用できません。		1≦n≦15000
NVARCHAR(n)	**HiRDB サーバの既定文字集合が UTF-8，又は IVS 対応 UTF-8 以外の場合：**※5 L2 集団項目名 2 OCCURS m TIMES. L3 基本項目名 1 PICTURE S9(4) COMPUTATIONAL. L3 基本項目名 2 PICTURE N(n). **HiRDB サーバの既定文字集合が UTF-8，又は IVS 対応 UTF-8 以外の場合：**※5 NVARCHAR 型は使用できません。		1≦n≦16000
MCHAR[(n)]	L2 基本項目名 PICTURE X(n) OCCURS m TIMES.※2 HiRDB サーバの既定文字集合が UTF-8，又は IVS 対応 UTF-8 の場合は※5，次のように記述することもできます。 L2 基本項目名 PICTURE N(n_2) OCCURS m TIMES.※4		1≦n≦30000
MVARCHAR(n)	L2 集団項目名 2 OCCURS m TIMES. L3 基本項目名 1 PICTURE S9(4) COMPUTATIONAL. L3 基本項目名 2 PICTURE X(n).※2		1≦n≦32000

SQL のデータ型	COBOL 言語のデータ記述	項目の記述	備　　考
	HiRDB サーバの既定文字集合が UTF-8，又は IVS 対応 UTF-8 の場合は※5，次のように記述することもできます。 L2 集団項目名 2 OCCURS m TIMES. L3 基本項目名 1 PICTURE S9(4) COMPUTATIONAL. L3 基本項目名 2 PICTURE N(n_2).※4		
DATE	L2 基本項目名 PICTURE X(10) OCCURS m TIMES.※2		−
TIME	L2 基本項目名 PICTURE X(8) OCCURS m TIMES.※2		−
TIMESTAMP(p)	L2 基本項目名 PICTURE X(n) OCCURS m TIMES.※2		p=0 の場合，n=19 p=2 の場合，n=21,22 p=4 の場合，n=23,24 p=6 の場合，n=25,26
INTERVAL YEARTO DAY	L2 基本項目名 PICTURE S9(8) COMPUTATIONAL-3 OCCURS m TIMES.		−
INTERVAL HOUR TO SECOND	L2 基本項目名 PICTURE S9(6) COMPUTATIONAL-3 OCCURS m TIMES.		−
ROW	L2 集団項目名 2 OCCURS m TIMES. この表の中のデータ項目と集団項目の組み合わせ※6		−
BLOB	記述できない	記述できない	−
BINARY	L2 集団項目名 2 OCCURS m TIMES. L3 基本項目名 1 PICTURE S9(9) COMPUTATIONAL.	OCCURS の指定によって同じデータ構造を繰り返す反復データ項目から構成される集団項目	• 配列を使用した FETCH の場合，4≦n≦2147483644（n は 4 の倍数であること）

SQL のデータ型		COBOL 言語のデータ記述	項目の記述	備　　考
		L3 基本項目名 2 PICTURE X(n).※1		• 配列を使用した FETCH 以外の場合， 4≦n≦32000（n は 4 の倍数であること）
BLOB 位置付け子		記述できない	記述できない	−
BINARY 位置付け子		L2 基本項目名 SQL TYPE IS BINARY AS LOCATOR OCCURS m TIMES.	OCCURS の指定によって同じデータ構造を繰り返す反復データ項目から構成される集団項目	−
標識変数	BINARY，BINARY 位置付け子以外	L2 基本項目名 PICTURE S9(4) COMPUTATIONAL OCCURS m TIMES.		−
	BINARY，BINARY 位置付け子	L2 基本項目名 PICTURE S9(9) COMPUTATIONAL OCCURS m TIMES.		−
SQL 文		記述できない	記述できない	−

（凡例）

L2：レベル番号 02～49（L2<L3）

L2 にはレベル番号 01，66，77，又は 88 を指定できません。詳細は COBOL 言語のマニュアルの OCCURS 句の構文規則を参照してください。

L3：レベル番号 03～49

m：配列の要素数（1～4,096）

n：長さ（バイト）

p：精度（全体のけた数）

s：位取り（小数点以下のけた数）

−：なし

注※1

X の代わりに 9 を使用して定義できます。9 を使用して定義した場合，数字以外の文字を含む文字列を代入したり，検索結果として受け取ったときの動作は COBOL 言語のコンパイラの実装に依存します。

注※2

X の代わりに 9 を使用して定義した場合，プリプロセス時にエラーとなりませんが，使用しないでください。

注※3

COBOL コンパイラの仕様に依存します。例えば，COBOL85 の場合は 1≦p≦18 となります。

注※4

COBOL 言語の日本語項目（PICTURE N）を含むこのデータ記述は，COBOL2002 の Unicode 機能を使用する場合にだけ，SQL の混在文字データ型（MCHAR 又は MVARCHAR）に対応するデータ記述として使用できます。詳細については，表「SQL のデータ型と COBOL 言語のデータ記述」の MCHAR[(n)]及び MVARCHAR(n)の注を参照してください。

注※5

pdntenv コマンド（UNIX 版の場合は pdsetup コマンド）で，文字コード種別に utf-8 を指定した場合に，HiRDB サーバの既定文字集合が UTF-8 になります。

また，文字コード種別に utf-8_ivs を指定した場合，HiRDB サーバの既定文字集合が IVS 対応 UTF-8 になります。

注※6

組み合わせの詳細については，表「SQL のデータ型と COBOL 言語のデータ記述」を参照してください。

(3)　繰返し列を使用した場合の SQL のデータ型と COBOL 言語のデータ記述

繰返し列を使用した場合の SQL のデータ型と COBOL 言語のデータ記述を次の表に示します。

表 F–10　繰返し列を使用した場合の SQL のデータ型と COBOL 言語のデータ記述

SQL のデータ型	COBOL 言語のデータ記述	項目の記述	備考
SMALLINT	L2 集団項目名. L3 基本項目名 1 PICTURE S9(9) COMPUTATIONAL. L3 基本項目名 2 PICTURE S9(4) COMPUTATIONAL OCCURS m TIMES.	二つの基本項目から構成される集団項目	－
INTEGER	L2 集団項目名. L3 基本項目名 1 PICTURE S9(9) COMPUTATIONAL. L3 基本項目名 2 PICTURE S9(9) COMPUTATIONAL OCCURS m TIMES.		－
DECIMAL [(p[,s])]	L2 集団項目名. L3 基本項目名 1 PICTURE S9(9) COMPUTATIONAL. L3 基本項目名 2 PICTURE S9(p-s)[V9(s)] COMPUTATIONAL-3 OCCURS m TIMES.		1≦p≦38※3， 0≦s≦p p=s の場合，SV9(s)とします。 s=0 の場合，[V9(s)]を省略します。

SQL のデータ型	COBOL 言語のデータ記述	項目の記述	備考
	L2 集団項目名. L3 基本項目名 1 PICTURE S9(9) COMPUTATIONAL. L3 基本項目名 2 PICTURE S9(p-s)[V9(s)] DISPLAY SIGN LEADING SEPARATE OCCURS m TIMES.		
	L2 集団項目名. L3 基本項目名 1 PICTURE S9(9) COMPUTATIONAL. L3 基本項目名 2 PICTURE S9(p-s)[V9(s)] DISPLAY SIGN TRAILING OCCURS m TIMES.		
SMALLFLT (REAL)	L2 集団項目名. L3 基本項目名 1 PICTURE S9(9) COMPUTATIONAL. L3 基本項目名 2 COMPUTATIONAL-1 OCCURS m TIMES.		−
FLOAT (DOUBLE PRECISION)	L2 集団項目名. L3 基本項目名 1 PICTURE S9(9) COMPUTATIONAL. L3 基本項目名 2 COMPUTATIONAL-2 OCCURS m TIMES.		−
CHAR[(n)]	L2 集団項目名. L3 基本項目名 1 PICTURE S9(9) COMPUTATIONAL. L3 基本項目名 2 PICTURE X(n) OCCURS m TIMES.※1		1≦n≦30000
VARCHAR(n)	L2 集団項目名. L3 基本項目名 1 PICTURE S9(9) COMPUTATIONAL. L3 集団項目名 2 OCCURS m TIMES. L4 基本項目名 3 PICTURE S9(4) COMPUTATIONAL. L4 基本項目名 4 PICTURE X(n).※1	二つの基本項目から構成される集団項目と，一つの基本項目から成る集団項目	1≦n≦32000
NCHAR[(n)]	**HiRDB サーバの既定文字集合が UTF-8，又は IVS 対応 UTF-8 以外の場合：**※5	二つの基本項目から構成される集団項目	1≦n≦15000

SQL のデータ型	COBOL 言語のデータ記述	項目の記述	備考
	L2 集団項目名. L3 基本項目名 1 PICTURE S9(9) COMPUTATIONAL. L3 基本項目名 2 PICTURE N(n) OCCURS m TIMES. **HiRDB サーバの既定文字集合が UTF-8，又は IVS 対応 UTF-8 の場合：**※5 NCHAR 型は使用できません。		
NVARCHAR(n)	**HiRDB サーバの既定文字集合が UTF-8，又は IVS 対応 UTF-8 以外の場合：**※5 L2 集団項目名. L3 基本項目名 1 PICTURE S9(9) COMPUTATIONAL. L3 集団項目名 2 OCCURS m TIMES. L4 基本項目名 3 PICTURE S9(4) COMPUTATIONAL. L4 基本項目名 4 PICTURE N(n). **HiRDB サーバの既定文字集合が UTF-8，又は IVS 対応 UTF-8 の場合：**※5 NVARCHAR 型は使用できません。	二つの基本項目から構成される集団項目と，一つの基本項目から成る集団項目	1≦n≦16000
MCHAR[(n)]	L2 集団項目名. L3 基本項目名 1 PICTURE S9(9) COMPUTATIONAL. L3 基本項目名 2 PICTURE X(n) OCCURS m TIMES.※1 HiRDB サーバの既定文字集合が UTF-8，又は IVS 対応 UTF-8 の場合は※5，次のように記述することもできます。 L2 集団項目名. L3 基本項目名 1 PICTURE S9(9) COMPUTATIONAL. L3 基本項目名 2 PICTURE N(n_2) OCCURS m TIMES.※4	二つの基本項目から構成される集団項目	1≦n≦30000
MVARCHAR(n)	L2 集団項目名. L3 基本項目名 1 PICTURE S9(9) COMPUTATIONAL. L3 集団項目名 2 OCCURS m TIMES. L4 基本項目名 3 PICTURE S9(4) COMPUTATIONAL.	二つの基本項目から構成される集団項目と，一つの基本項目から成る集団項目	1≦n≦32000

SQL のデータ型	COBOL 言語のデータ記述	項目の記述	備考
	L4 基本項目名 4 PICTURE X(n).※1 HiRDB サーバの既定文字集合が UTF-8，又は IVS 対応 UTF-8 の場合は※5，次のように記述することもできます。 L2 集団項目名. L3 基本項目名 1 PICTURE S9(9) COMPUTATIONAL. L3 集団項目名 2 OCCURS m TIMES. L4 基本項目名 3 PICTURE S9(4) COMPUTATIONAL. L4 基本項目名 4 PICTURE N(n_2).※4		
DATE	L2 集団項目名. L3 基本項目名 1 PICTURE S9(9) COMPUTATIONAL. L3 基本項目名 2 PICTURE X(10) OCCURS m TIMES.※2	二つの基本項目から構成される集団項目	−
TIME	L2 集団項目名. L3 基本項目名 1 PICTURE S9(9) COMPUTATIONAL. L3 基本項目名 2 PICTURE X(8) OCCURS m TIMES.※2		−
TIMESTAMP[(n)]	L2 集団項目名. L3 基本項目名 1 PICTURE S9(9) COMPUTATIONAL. L3 基本項目名 2 PICTURE X(n) OCCURS m TIMES.※2		p=0 の場合は n=19 p=2 の場合は n=21,22 p=4 の場合は n=23,24 p=6 の場合は n=25,26
INTERVAL YEARTO DAY	L2 集団項目名. L3 基本項目名 1 PICTURE S9(8) COMPUTATIONAL. L3 基本項目名 2 PICTURE S9(8) COMPUTATIONAL-3 OCCURS m TIMES.		−
INTERVAL HOUR TO SECOND	L2 集団項目名. L3 基本項目名 1 PICTURE S9(6) COMPUTATIONAL. L3 基本項目名 2		−

SQL のデータ型	COBOL 言語のデータ記述	項目の記述	備考
	PICTURE S9(6) COMPUTATIONAL-3 OCCURS m TIMES.		
ROW	記述できない	記述できない	−
BLOB	記述できない	記述できない	−
BINARY	記述できない	記述できない	−
BLOB 位置付け子	記述できない	記述できない	−
BINARY 位置付け子	記述できない	記述できない	−
標識変数（BLOB，BINARY，BLOB 位置付け子，BINARY 位置付け子以外）	L2 集団項目名. L3 基本項目名 1 PICTURE S9(9) COMPUTATIONAL. L3 基本項目名 2 PICTURE S9(4) COMPUTATIONAL OCCURS m TIMES.	二つの基本項目から構成される集団項目	−
SQL 文	記述できない	記述できない	−

（凡例）

L2：レベル番号 02～49

L3，L4：レベル番号 03～49

m：繰返し列の最大要素数（2～30,000）

n：長さ（バイト）

p：精度（全体のけた数）

s：位取り（小数点以下のけた数）

−：なし

注 1

基本項目名 1 の値は，現在の要素数としてください。

注 2

基本項目名 2 又は集団項目名 2 の値は，繰返しの各要素の値としてください。

注 3

標識変数の基本項目名 1 は，繰返し列全体の標識としてください。

注 4

標識変数の基本項目名 2 は，繰返し列の各要素の標識としてください。

注※1

X の代わりに 9 を使用して定義できます。9 を使用して定義した場合，数字以外の文字を含む文字列を代入したり，検索結果として受け取ったときの動作は COBOL 言語のコンパイラの実装に依存します。

注※2

X の代わりに 9 を使用して定義した場合，プリプロセス時にエラーとなりませんが，使用しないでください。

注※3

COBOL コンパイラの仕様に依存します。例えば，COBOL85 の場合は 1≦p≦18 となります。

注※4

COBOL 言語の日本語項目（PICTURE N）を含むこのデータ記述は，COBOL2002 の Unicode 機能を使用する場合にだけ，SQL の混在文字データ型（MCHAR 又は MVARCHAR）に対応するデータ記述として使用できます。詳細については，表「SQL のデータ型と COBOL 言語のデータ記述」の MCHAR[(n)] 及び MVARCHAR(n)の注を参照してください。

注※5

pdntenv コマンド（UNIX 版の場合は pdsetup コマンド）で，文字コード種別に utf-8 を指定した場合に，HiRDB サーバの既定文字集合が UTF-8 になります。

また，文字コード種別に utf-8_ivs を指定した場合，HiRDB サーバの既定文字集合が IVS 対応 UTF-8 になります。

付録 G　データディクショナリ表の検索

ここでは，データディクショナリ表の検索及び参照について説明します。

HiRDB のデータディクショナリ表は，次の二つの方法で参照できます。

- GUI 版 HiRDB SQL Executer
- 操作系 SQL

付録 G.1　GUI 版 HiRDB SQL Executer によるデータディクショナリ表の参照

GUI 版 HiRDB SQL Executer のディクショナリビューでは，接続ユーザが参照可能なデータディクショナリ情報を，ツリー形式で表示します。

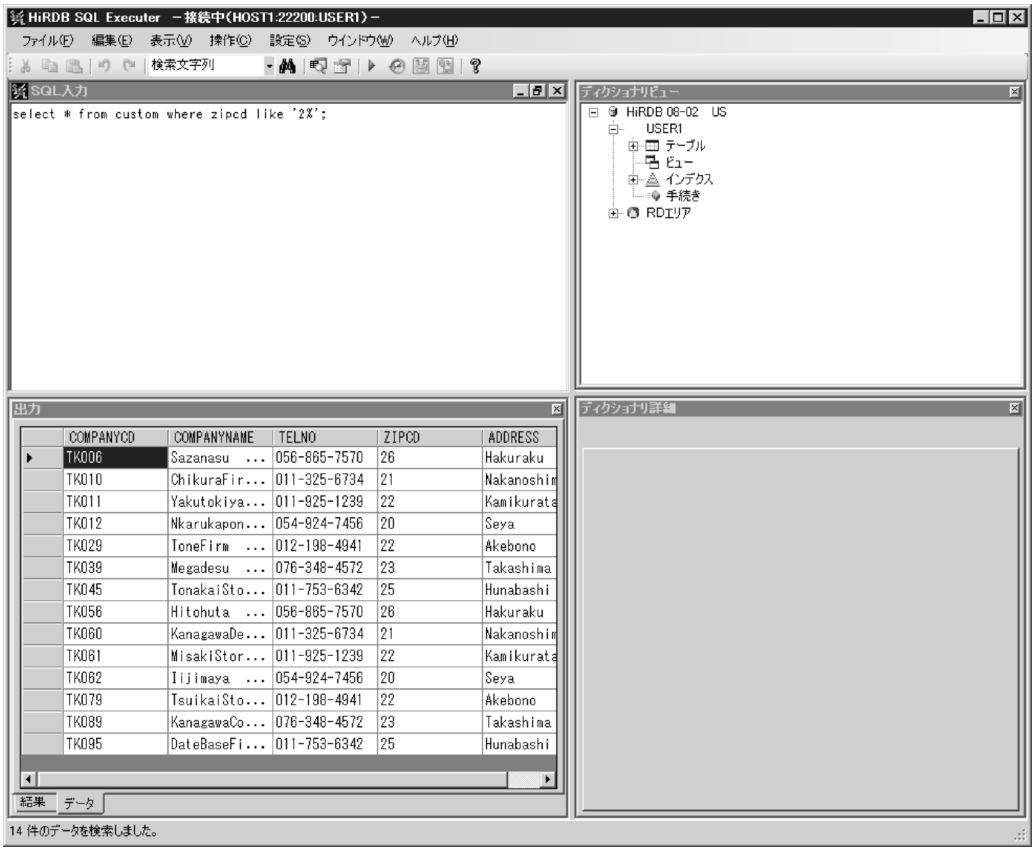
ここでは，GUI 版 HiRDB SQL Executer で表定義情報を参照する場合について説明します。

(1)　GUI 版 HiRDB SQL Executer の起動

GUI 版 HiRDB SQL Executer は，次の手順で起動します。

［スタート］−［プログラム］−［HiRDB SQL Executer］−［GUI 版 HiRDB SQL Executer］を選択します。

GUI 版 HiRDB SQL Executer が起動します。

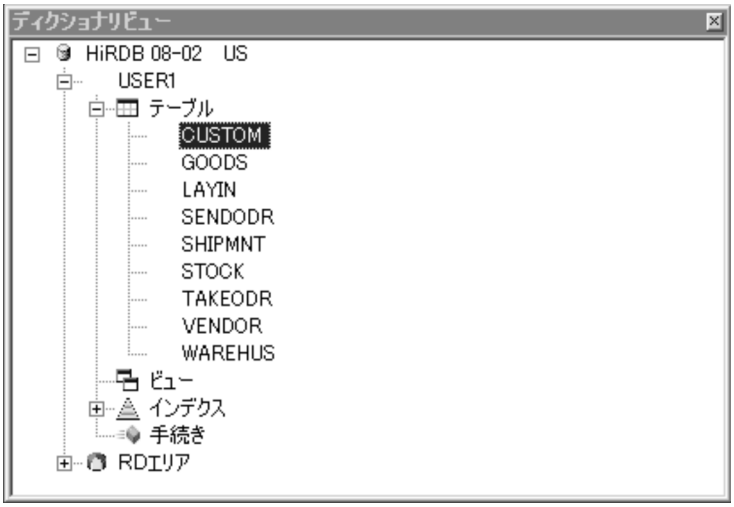


HiRDB サーバに接続後，［ディクショナリビュー］画面の接続データベース名（ここでは HiRDB 08-02 US）をクリックすると，接続ユーザ名（ここでは USER1）と RD エリアの階層が表示されます。接続ユーザ名をクリックすると，テーブル，ビュー，インデクス，手続きの階層が表示されます。

(2)　表定義情報の参照

表定義情報の参照について説明します。

［ディクショナリビュー］画面で［テーブル］をクリックすると，そのユーザが所有する表の一覧が表示されます。表名を選択すると，［ディクショナリ詳細］画面に表定義情報が表示されます。



［プロパティ］タブでは，列数，定義長，格納先 RD エリア名などが確認できます。



［構成列］タブでは，構成列の列名や属性などが確認できます。



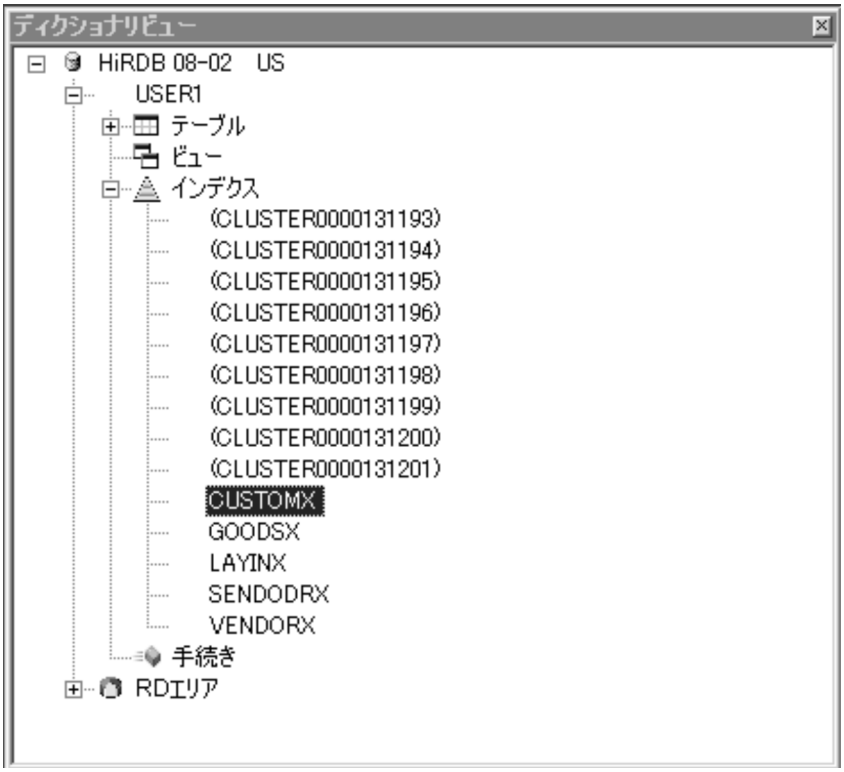
［インデクス］タブでは，表に定義されているインデクス名などが確認できます。



(3)　インデクス定義情報の参照

インデクス定義情報の参照について説明します。

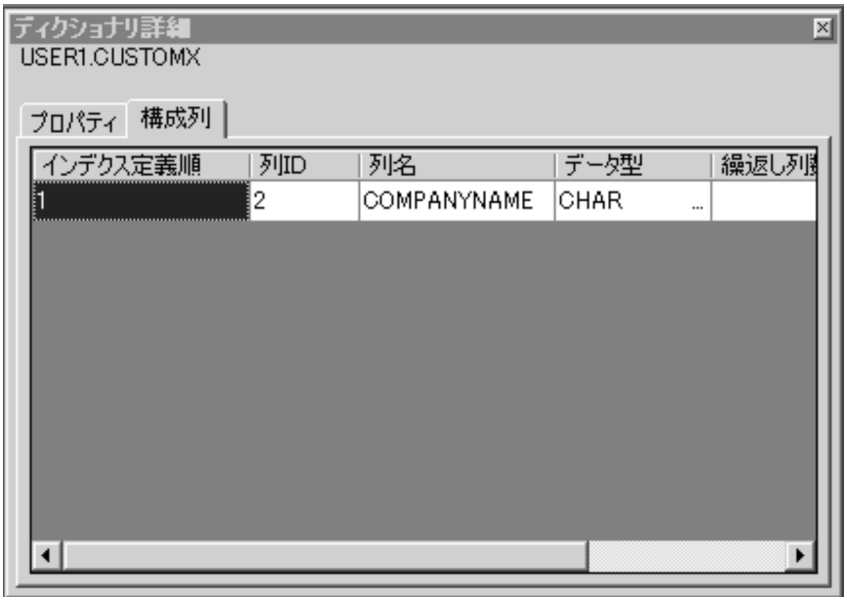
［ディクショナリビュー］画面で［インデクス］をクリックすると，そのユーザが所有するインデクスの一覧が表示されます。インデクス名を選択すると，［ディクショナリ詳細］画面にインデクス定義情報が表示されます。



［プロパティ］タブでは，インデクスの格納先 RD エリア名などが確認できます。



［構成列］タブでは，インデクス構成列名などが確認できます。



［ディクショナリビュー］画面の［ビュー］及び［手続き］の階層も，同様に各資源の定義情報が参照できます。詳細は，GUI 版 HiRDB SQL Executer のヘルプを参照してください。

付録 G.2　操作系 SQL によるデータディクショナリ表の参照

HiRDB のデータディクショナリ表は，一般の HiRDB のデータベースと同様に，操作系の SQL で参照できます。なお，ディクショナリ表の認可識別子は MASTER となります。

ここでは，検索時の SQL の記述例と，参照するために必要な定義情報について説明します。

参照するデータディクショナリ表の一覧を次の表に示します。

表 G–1　データディクショナリ一覧

表　　名	内　　容	情　報　量 (1 行当たり)
SQL_PHYSICAL_FILES	HiRDB ファイルの情報(HiRDB ファイルシステム名，RD エリア名との対応関係)	1HiRDB ファイル分
SQL_RDAREAS	RD エリア名称，定義情報，RD エリア種別，格納表数，インデクス数などの情報	1RD エリア分
SQL_TABLES	データベース中の各表(ディクショナリ表を含む)の所有者名，表名	1 表分
SQL_COLUMNS	列に関する列名，データ型などの定義情報	1 列分
SQL_INDEXES	データベース中の各インデクス(ディクショナリ表を含む)の所有者名，インデクス名	1 インデクス分
SQL_USERS	ユーザの実行権限，及びデータベースに対するアクセスを許可したユーザの認可識別子	1 ユーザ分

表　　名	内　　容	情　報　量 (1 行当たり)
SQL_RDAREA_PRIVILEGES	RD エリア利用権限の許可状況	1 認可識別子の 1RD エリア分
SQL_TABLE_PRIVILEGES	表に対するアクセス権限の付与状況	1 認可識別子の 1 表分
SQL_VIEW_TABLE_USAGE	ビュー表の基の実表名	1 ビュー表分
SQL_VIEWS	ビュー定義情報	1 ビュー表分
SQL_DIV_TABLE	表の分割情報(CREATE TABLE 時に指定した分割条件，及び格納 RD エリア名)	n 行で 1 表分
SQL_INDEX_COLINF	インデクスが定義された列名	n 行で 1 インデクス分
SQL_DIV_INDEX	インデクスの分割情報(格納 RD エリア名)	n 行で 1 インデクス分
SQL_DIV_COLUMN	BLOB 型列の分割情報(CREATE TABLE 時に指定した格納 RD エリア名)	n 行で 1 列分
SQL_ROUTINES	ルーチン定義情報	1 行で 1 ルーチン分
SQL_ROUTINE_RESOURCES	ルーチン中の使用リソース情報	n 行で 1 ルーチン分
SQL_ROUTINE_PARAMS	ルーチン中のパラメタ定義情報	n 行で 1 ルーチン分
SQL_TABLE_STATISTICS	表の統計情報	1 表分
SQL_COLUMN_STATISTICS	列の統計情報	1 列分
SQL_INDEX_STATISTICS	インデクスの統計情報	1 インデクス分
SQL_DATATYPES	ユーザ定義型の情報	1 ユーザ定義型分
SQL_DATATYPE_DESCRIPTORS	ユーザ定義型の構成属性の情報	1 属性分
SQL_TABLE_RESOURCES	表で使用するリソース情報	1 リソース分
SQL_PLUGINS	プラグイン情報	1 プラグイン分
SQL_PLUGIN_ROUTINES	プラグインのルーチン情報	1 プラグインのルーチン分
SQL_PLUGIN_ROUTINE_PARAMS	プラグインのルーチンのパラメタ情報	1 パラメタ情報
SQL_INDEX_TYPES	インデクス型の情報	1 インデクス型分
SQL_INDEX_RESOURCES	インデクスで使用するリソース情報	1 リソース情報分
SQL_INDEX_DATATYPE	インデクスの対象項目情報	1 対象項目情報分（1 段分）
SQL_INDEX_FUNCTION	インデクスで利用する抽象データ型関数の情報	一つの抽象データ型関数の情報分
SQL_TYPE_RESOURCES	ユーザ定義型で使用するリソース情報	1 リソース情報分
SQL_INDEX_TYPE_FUNCTION	インデクス型を定義したインデクスで利用できる抽象データ型関数の情報	n 行で 1 インデクス型分

表　　名	内　　容	情　報　量 (1 行当たり)
SQL_EXCEPT	インデクスの除外キー値の情報	n 行で 1 インデクスの除外キー群
SQL_IOS_GENERATIONS	**UNIX 版の場合：** インナレプリカ機能使用時の HiRDB ファイルシステム領域の世代情報 **Windows 版の場合：** システムが使用する情報（内容は空となります）	**UNIX 版の場合：** 1 行で 1HiRDB ファイルシステム領域分 **Windows 版の場合：** なし
SQL_TRIGGERS	スキーマ内にあるトリガの情報	1 行で 1 トリガ分
SQL_TRIGGER_COLUMNS	UPDATE トリガの契機列リスト情報	1 行で 1 契機列情報
SQL_TRIGGER_DEF_SOURCE	トリガ定義ソース情報	n 行で 1 トリガ定義ソース情報
SQL_TRIGGER_USAGE	トリガ動作条件中で参照している資源情報	1 行で，トリガ動作条件中で参照している資源名称一つ
SQL_PARTKEY	マトリクス分割表の分割キーの情報	1 行で 1 分割キー情報
SQL_PARTKEY_DIVISION	マトリクス分割表の分割条件値の情報	1 行で 1 分割条件値情報
SQL_AUDITS	監査対象の情報	1 行で 1 オブジェクト又は 1 ユーザに対する 1 イベント分の情報
SQL_REFERENTIAL_CONSTRAINTS	参照制約の対応状況	1 行で 1 制約分の情報
SQL_KEYCOLUMN_USAGE	外部キーを構成する列情報	1 行で 1 列分の情報
SQL_TABLE_CONSTRAINTS	スキーマ内にある整合性制約の情報	1 行で 1 整合性制約分の情報
SQL_CHECKS	検査制約の情報	1 行で 1 検査制約分の情報
SQL_CHECK_COLUMNS	検査制約で使用している列の情報	1 行で一つの検査制約で使用している 1 列分の情報
SQL_DIV_TYPE	キーレンジ分割とハッシュ分割を組み合わせたマトリクス分割表の分割キーの情報	1 行で 1 分割キー数分の情報
SQL_SYSPARAMS	連続認証失敗回数制限，及びパスワードの文字列制限の情報	1 行で 1 設定項目数分，n 行で一つの連続認証失敗回数制限分，又は一つのパスワードの文字列制限分の情報

表　　名	内　　容	情　報　量 (1 行当たり)
SQL_INDEX_XMLINF	部分構造インデクスの構成部分構造パス情報	1 行で 1 インデクスの情報
SQL_SEQUENCES	順序数生成子の情報	1 行で 1 順序数生成子分の情報
SQL_ACCESS_SECURITY	IP アドレスによる接続制限の情報	1 行で 1 接続制約分の情報
SQL_ROLES	ロールの情報	1 行で 1 ロールの情報
SQL_ROLE_PRIVILEGES	ロール利用権限の付与関係の情報	1 行で 1 認可識別子（ロール名含む）に付与された 1 ロール利用権限分の情報

検索時の SQL の記述例

データディクショナリ表を検索する SQL 文の例を次に示します。SQL の詳細については，マニュアル「HiRDB SQL リファレンス」を参照してください。

ディクショナリ表の参照権限の設定によっては，検索できる情報が限られます。データディクショナリ表の参照権限の設定については，マニュアル「HiRDB システム運用ガイド」を参照してください。

なお，ディクショナリ表を検索した後は，すぐに COMMIT 文を発行するか，又は検索例のように WITHOUT LOCK NOWAIT を指定するかしてください。

＜例 1 ＞

HiRDB のシステムにない RD エリアのサーバ名，HiRDB ファイル名，及び所属する RD エリア名を知りたい場合

```
SELECT X.SERVER_NAME, PHYSICAL_FILE_NAME, X.RDAREA_NAME
   FROM MASTER.SQL_PHYSICAL_FILES X, MASTER.SQL_RDAREAS Y
   WHERE X.RDAREA_NAME=Y.RDAREA_NAME
   ORDER BY X.SERVER_NAME
   WITHOUT LOCK NOWAIT
```

＜例 2 ＞

自分の所有する表の列の定義情報の中から，列を含む表の名称，列名，データ型，及び列データ長を知りたい場合

```
SELECT TABLE_NAME, COLUMN_NAME, DATA_TYPE, DATA_LENGTH
   FROM MASTER.SQL_COLUMNS
   WHERE TABLE_SCHEMA=USER※
   ORDER BY TABLE_NAME
   WITHOUT LOCK NOWAIT
```

＜例 3 ＞

自分の所有する表のインデクス定義情報の中から，インデクスを含む表の名称，インデクスの名称，及びページ内未使用領域の比率を知りたい場合

```
SELECT TABLE_NAME, INDEX_NAME, FREE_AREA
   FROM MASTER.SQL_INDEXES
   WHERE TABLE_SCHEMA=USER※
   ORDER BY TABLE_NAME
   WITHOUT LOCK NOWAIT
```

＜例 4 ＞

自分がアクセスできる表と，その表に対するアクセス権限の種類(SELECT 権限，INSERT 権限，DELETE 権限，及び UPDATE 権限)の有無について知りたい場合

```
SELECT TABLE_NAME, SELECT_PRIVILEGE, INSERT_PRIVILEGE,
   DELETE_PRIVILEGE, UPDATE_PRIVILEGE
   FROM MASTER.SQL_TABLE_PRIVILEGES
   WHERE GRANTEE=USER※ OR GRANTEE='PUBLIC'
   WITHOUT LOCK NOWAIT
```

＜例 5 ＞

コマンドをグループ指定するとき，対象となる RD エリア(先行文字列が RD1 の RD エリア)の数を知りたい場合

```
SELECT COUNT(*) FROM MASTER.SQL_RDAREAS
   WHERE RDAREA_TYPE='U' AND
   RDAREA_NAME LIKE 'RD1%'
   WITHOUT LOCK NOWAIT
```

＜例 6 ＞

コマンドをグループ指定するとき，対象となる RD エリア(先行文字列が RD1 の RD エリア)の名称を知りたい場合

```
SELECT RDAREA_NAME FROM MASTER.SQL_RDAREAS
   WHERE RDAREA_TYPE='U' AND
   RDAREA_NAME LIKE 'RD1%' ORDER BY RDAREA_NAME
   WITHOUT LOCK NOWAIT
```

＜例 7 ＞

自分の所有する非分割表(名称が T1 の表)が格納されている RD エリアの RD エリア名を知りたい場合

```
SELECT X.RDAREA_NAME
   FROM MASTER.SQL_RDAREAS X, MASTER.SQL_TABLES Y
   WHERE Y.TABLE_SCHEMA=USER※
     AND Y.TABLE_NAME='T1'
     AND X.RDAREA_NAME=Y.RDAREA_NAME
   WITHOUT LOCK NOWAIT
```

注※　USER は，実行ユーザの認可識別子を値に持つ変数です。認可識別子の詳細については，マニュアル「HiRDB SQL リファレンス」を参照してください。

＜例 8 ＞

データディクショナリ LOB 用 RD エリアを再初期化するとき，実行時に使用するストアドプロシジャ及びストアドファンクションのオブジェクトを格納している RD エリアの名称(オブジェクト格納用 RD エリア名)を知りたい場合

```
SELECT RDAREA_NAME FROM MASTER.SQL_DIV_COLUMN
   WHERE TABLE_SCHEMA='HiRDB'
     AND TABLE_NAME='SQL_ROUTINES'
     AND COLUMN_NAME='ROUTINE_OBJECT'
   WITHOUT LOCK NOWAIT
```

注　データディクショナリ LOB 用 RD エリアを再初期化した場合，実行後にすべての SQL オブジェクトを再作成する必要があります。

＜例 9 ＞

無効となった SQL オブジェクト，又はインデクス無効状態のストアドプロシジャ及びストアドファンクションの名称を知りたい場合

```
SELECT ROUTINE_SCHEMA,ROUTINE_NAME
   FROM MASTER.SQL_ROUTINES
   WHERE ROUTINE_VALID='N'
     OR INDEX_VALID='N'
   WITHOUT LOCK NOWAIT
```

＜例 10 ＞

ユーザ定義関数 FUNC1 の引数に埋込み変数を利用する場合に，実際に使用するユーザ定義関数の引数のデータ型を知りたい場合

```
SELECT PARAMETER_NAME,DATA_TYPE,UDT_OWNER,UDT_NAME,PARAMETER_NO
   FROM MASTER.SQL_ROUTINE_PARAMS
   WHERE ROUTINE_SCHEMA=USER AND ROUTINE_NAME='FUNC1'
     ORDER BY PARAMETER_NO
   WITHOUT LOCK NOWAIT
```

＜例 11 ＞

ユーザ（USERA）が所有するすべての表を再編成する場合に，その表が格納されている RD エリアを知りたいとき（閉塞する必要がある RD エリアを知りたいとき）

```
非分割表の場合：
   SELECT DISTINCT(RDAREA_NAME) FROM MASTER.SQL_TABLES
      WHERE TABLE_SCHEMA=USERA AND RDAREA_NAME IS NOT NULL
      WITHOUT LOCK NOWAIT

分割表の場合：
   SELECT DISTINCT(RDAREA_NAME) FROM MASTER.SQL_DIV_TABLE
      WHERE TABLE_SCHEMA=USERA
      WITHOUT LOCK NOWAIT
```

求められた RD エリア名を重複排除した結果が，閉塞する必要がある RD エリア名となります。

付録 G.3　ディクショナリ表の詳細

参照するために必要なデータディクショナリ表の定義情報を表ごとに示します。

なお，各ディクショナリ表に VARCHAR 又は MVARCHAR のデータ型の列がありますが，これはデータベース初期設定ユティリティ，又はデータベース構成変更ユティリティの dictionary datatype オペランドで，データ型をどちらにするか設定してください。

(1)　SQL_PHYSICAL_FILES 表の内容

この表では，HiRDB ファイルの情報(HiRDB ファイルと RD エリアとの関係)を管理します（1 行で 1HiRDB ファイル分）。

SQL_PHYSICAL_FILES 表の内容を次の表に示します。

表 G–2　SQL_PHYSICAL_FILES 表の内容

項番	列　　名	データ型	内　　容
1	SERVER_NAME	CHAR(8)	サーバ名称(バックエンドサーバ名又はディクショナリサーバ名)。
2	PHYSICAL_FILE_NAME	VARCHAR(167)	HiRDB ファイル名。
3	RDAREA_NAME	VARCHAR(30)，又は MVARCHAR(30)	HiRDB ファイルを割り当てた RD エリア名。
4	INITIAL_SIZE	INTEGER	HiRDB ファイルのセグメント数。
5	PHYSICAL_FILE_ID	INTEGER	物理ファイル ID。

(2)　SQL_RDAREAS 表の内容

この表では，RD エリアの定義情報を管理します（1 行で 1RD エリア分）。

SQL_RDAREAS 表の内容を次の表に示します。

表 G–3　SQL_RDAREAS 表の内容

項番	列　　名	データ型	内　　容
1	RDAREA_NAME	VARCHAR(30)，又は MVARCHAR(30)	RD エリアの名称。
2	SERVER_NAME	CHAR(8)	サーバ名称(バックエンドサーバ名又はディクショナリサーバ名)。
3	RDAREA_TYPE	CHAR(1)	RD エリアの種類。 M：マスタディレクトリ用 RD エリア D：データディレクトリ用 RD エリア S：データディクショナリ用 RD エリア W：ワーク用のユーザ用 RD エリア U：ユーザ用 RD エリア P：データディクショナリ LOB 用 RD エリア L：ユーザ LOB 用 RD エリア

項番	列　　名	データ型	内　　容
			R：レジストリ用 RD エリア K：レジストリ LOB 用 RD エリア A：リスト用 RD エリア
4	PAGE_SIZE	INTEGER	ページ長(単位：バイト)。
5	SEGMENT_SIZE	INTEGER	セグメントサイズ(単位：ページ)。
6	FILE_COUNT	INTEGER	HiRDB ファイル数。
7	N_TABLE	INTEGER	格納表数(定義数)。 初期値は 0 となります。表と順序数生成子を定義している場合は，表と順序数生成子を合わせて 500 が上限値となります。
8	N_INDEX	INTEGER	格納インデクス数(定義数)。 初期値は 0 となります。
9	RDAREA_ID	INTEGER	RD エリア ID。
10	REBALANCE_TABLE	CHAR(1)	リバランス表の有無。 Y：リバランス表あり ナル値：リバランス表なし
11	MAX_ENTRIES	INTEGER	最大リスト登録数。 リスト用 RD エリア以外の場合，又は max entries を指定していない場合はナル値となります。
12	EXTENSION	CHAR(1)	RD エリアの増分指定有無。 U：指定あり N：指定なし
13	EXTENSION_SEGMENT_SIZE	INTEGER	増分セグメント数。 RD エリアの増分指定がない場合はナル値となります。
14	ORIGINAL_RDAREA_NAME	VARCHAR(30)，又は MVARCHAR(30)	**UNIX 版の場合：** オリジナル RD エリアの名称。 レプリカ RD エリアでない場合はナル値となります。 **Windows 版の場合：** システムが使用する情報（内容は空となります）。
15	ORIGINAL_RDAREA_ID	INTEGER	**UNIX 版の場合：** オリジナル RD エリアの ID。 レプリカ RD エリアでない場合はナル値となります。 **Windows 版の場合：** システムが使用する情報（内容は空となります）。

項番	列　　名	データ型	内　　容
16	GENERATION_NUMBER	SMALLINT	**UNIX 版の場合：** 世代番号。 オリジナル RD エリア及びレプリカ RD エリア以外の RD エリアの場合はナル値となります。 **Windows 版の場合：** システムが使用する情報（内容は空となります）。
17	REPLICA_COUNT	SMALLINT	**UNIX 版の場合：** レプリカカウンタ。 オリジナル RD エリア以外の場合，及びレプリカ RD エリアがなくなった RD エリアの場合はナル値となります。 **Windows 版の場合：** システムが使用する情報（内容は空となります）。
18	REPLICA_STATUS	CHAR(1)	**UNIX 版の場合：** レプリカステータス。 C：カレント RD エリア S：サブ RD エリア オリジナル RD エリア及びレプリカ RD エリア以外の RD エリアの場合はナル値となります。 **Windows 版の場合：** システムが使用する情報（内容は空となります）。
19	SHARED	CHAR(1)	共用 RD エリアの指定。 S：共用 RD エリア ナル値：非共用 RD エリア
20	N_SEQUENCE	INT	格納順序数生成子数。 順序数生成子数が 0 の場合はナル値となります。表と順序数生成子を定義している場合は，表と順序数生成子を合わせて 500 が上限値となります。
21	DATA_MODEL	CHAR(1)	システムが使用する情報。
22	TEMPORARY_ATTR	CHAR(1)	一時表用 RD エリアの属性。 S：SQL セッション間共有属性の一時表用 RD エリア C：特定 SQL セッション占有属性の一時表用 RD エリア ユーザ用 RD エリア及び一時表用 RD エリアでない場合はナル値となります。

(3)　SQL_TABLES 表の内容

この表では，スキーマ内にある表の情報を管理します（1 行で 1 表分）。

なお，SQL_TABLES 表の行は，表を定義するときに作成され，表を削除するときに削除されます。

SQL_TABLES 表の内容を次の表に示します。

表 G–4　SQL_TABLES 表の内容

項番	列　　名	データ型	内　　容
1	TABLE_SCHEMA	VARCHAR(30)，又は MVARCHAR(30)	表の所有者。パブリックビュー表の場合は PUBLIC。
2	TABLE_NAME	VARCHAR(30)，又は MVARCHAR(30)	表の名称。
3	TABLE_TYPE	CHAR(16)	表の種別。 BASE TABLE：永続実表 VIEW：ビュー表 READ ONLY VIEW：読み込み専用ビュー表 GLOBAL TEMPORARY：一時表
4	TABLE_ID	INTEGER	表 ID。 システム内のユニークな内部 ID を示します。
5	N_COLS	SMALLINT	構成列数。
6	N_INDEX	SMALLINT	インデクス定義数。 次のインデクスの定義数の合計です。 • B-tree インデクス（主キー，クラスタキー，及び部分構造インデクスを含む） • プラグインインデクス 初期値は 0 です。
7	DCOLUMN_NAME	VARCHAR(30)，又は MVARCHAR(30)	分割列名（複数列の分割，及びマトリクス分割の場合は先頭の分割キーの列名）。 非分割表，及びビュー表の場合はナル値となります。
8	VDEFLEN	INTEGER	ビュー解析情報長。 実表の場合はナル値となります。
9	FREE_AREA	SMALLINT	ページ内未使用領域の比率(%)。 ビュー表の場合は 0 となります。
10	FREE_PAGE	SMALLINT	セグメント内空きページ(未使用ページ)比率(%)。 ビュー表の場合は 0 となります。
11	TABLE_COMMENT	VARCHAR(255)，又は MVARCHAR(255)	コメント。 初期値はナル値となります。
12	CREATE_TIME	CHAR(14)	表作成時刻(YYYYMMDDHHMMSS)。

項番	列　　名	データ型	内　　容
13	ENQ_RESOURCE_SIZE	CHAR(1)	排他資源単位。 P：ページ単位 行単位の排他の場合，ビュー表，及び一時表の場合はナル値となります。
14	DEFAULT_COLUMN	SMALLINT	既定値（DEFAULT 句又は WITH DEFAULT）の指定列数。 ビュー表及びディクショナリ表の場合はナル値となります。
15	RDAREA_NAME	VARCHAR(30)，又は MVARCHAR(30)	非分割格納先 RD エリア名称。 分割表，ビュー表，及び一時表の場合はナル値となります。
16	DEFINITION_CACHE_SIZE	INTEGER	表定義キャッシュサイズ(単位： バイト)。 ディクショナリの場合はナル値となります。
17	STATISTICS_CACHE_SIZE	INTEGER	統計情報キャッシュサイズ(単位：バイト)。 初期値はナル値となります。
18	N_RDAREA	INTEGER	格納先 RD エリア数(1～4096)。 ビュー表，及び一時表の場合は 0 となります。
19	FIX_TABLE	CHAR(1)	FIX 指定。 F：指定あり N：指定なし
20	VIEW_LEVEL	INTEGER	ビュー定義のネスト数。 実表の場合はナル値となります。
21	N_BASETABLE	INTEGER	ビュー表の基となる表の数。 実表の場合はナル値となります。
22	ROW_LENGTH	INTEGER	FIX 表の行長。 FIX 表でない表，及びビュー表の場合はナル値となります。
23	N_NOTNULL	INTEGER	非ナル値列数。 ビュー表，及びディクショナリ表の場合はナル値となります。
24	COMPRESS_TYPE	VARCHAR(8)	データ圧縮情報。 • 圧縮種別(先頭 1 バイト) S：データ抑制(SUPPRESS) • 2 バイト目以降 抑制データ型 D：DECIMAL

項番	列　　名	データ型	内　　容
			SUPPRESS を指定していない表，ビュー表，及びディクショナリ表の場合はナル値となります。
25	DIV_TYPE	CHAR(1)	分割種別。 P：境界値分割，及びマトリクス分割 H：フレキシブルハッシュ分割 F：FIX ハッシュ分割 M：ハッシュ混在マトリクス分割 非分割表，キーレンジ分割表，及びビュー表の場合はナル値となります。
26	HASH_NAME	VARCHAR(8)，又は MVARCHAR(8)	ハッシュ関数名。 "HASH1" "HASH2" "HASH3" "HASH4" "HASH5" "HASH6" "HASH0" "HASHA" "HASHB" "HASHC" "HASHD" "HASHE" "HASHF" "HASHZ" HASH を指定していない表，マトリクス分割表，ビュー表，及びディクショナリ表の場合はナル値となります。
27	N_LOB_COLUMN	SMALLINT	データ型が BLOB の列数。 ビュー表，及び BLOB の列がない表の場合はナル値となります。
28	N_LOB_RDAREA	INTEGER	表が持つユーザ LOB 用 RD エリア数。 ビュー表，BLOB の列がない表，及び BLOB 属性を含む抽象データ型がない表の場合はナル値となります。
29	CHANGE_TIME	CHAR(14)	表定義の変更時刻(YYYYMMDDHHMMSS)。 表の初期作成時はナル値となります。
30	N_DIV_COLUMN	SMALLINT	分割キー列数(2～16)。 非分割表，単一列分割キーを指定した表，及びビュー表の場合はナル値となります。
31	COLUMN_SUP_INF	CHAR(1)	列ごとのデータ抑制指定有無。

項番	列　　名	データ型	内　　容
			Y：指定あり ナル値：指定なし 列ごとのデータ抑制を指定していない表，及びビュー表の場合はナル値となります。
32	N_ADT_COLUMN	SMALLINT	データ型が抽象データ型の列数。 抽象データ型が定義されていない表の場合はナル値となります。
33	WITHOUT_ROLLBACK	CHAR(1)	WITHOUT ROLLBACK 指定有無。 Y：指定あり ナル値：指定なし WITHOUT ROLLBACK を指定していない表，及びビュー表の場合はナル値となります。
34	N_EXCEPT_VALUES	INTEGER	インデクス除外キー値の数。 除外値指定がないインデクス，及びビュー表の場合はナル値となります。
35	EXCEPT_VALUES_LEN	INTEGER	インデクス除外キー値の合計長（バイト）。 除外値指定がないインデクス，及びビュー表の場合はナル値となります。
36	REBALANCE	CHAR(1)	リバランス機能の使用有無。 Y：使用します。 リバランス機能を使用しない表，及びビュー表の場合はナル値となります。
37	INDEXLOCK_OPT	CHAR(1)	システムが使用する情報。
38	N_PK_COLUMNS	SMALLINT	主キーの列数。 主キーを定義していない場合はナル値となります。
39	FOREIGN_SERVER_NAME	VARCHAR(30)，又は MVARCHAR(30)	システムで使用する情報。
40	FOREIGN_SERVER_ID	INTEGER	システムで使用する情報。
41	BASE_FOREIGN_TABLE_SCHEMA	VARCHAR(30)，又は MVARCHAR(30)	システムで使用する情報。
42	BASE_FOREIGN_TABLE_NAME	VARCHAR(30)，又は MVARCHAR(30)	システムで使用する情報。
43	N_RDAREA_BEFORE_REBALANCE	INTEGER	ALTER TABLE ADD RDAREA 実行前の分割情報数（SQL_DIV_TABLE 表の行数）※1 リバランスを開始した場合，及びリバランス表以外の表，ビュー表の場合はナル値となります。

項番	列　　名	データ型	内　　容
44	ON_REBALANCE	CHAR(1)	リバランス実行状態。 Y：実行中 ナル値：実行中でない リバランス開始後に Y となり，リバランスが正常終了するとナル値になります。
45	SEGMENT_REUSE	CHAR(1)	SEGMENT REUSE の指定有無。 Y：指定あり ナル値：指定なし SEGMENT REUSE に NO を指定している場合（省略している場合も含む），及びビュー表の場合はナル値となります。
46	N_REUSE_SEGMENT	INTEGER	空き領域の再利用を開始するセグメント数。※2 SEGMENT REUSE に NO を指定している場合（SEGMENT REUSE を省略している場合も含む），及びビュー表の場合はナル値となります。
47	REUSE_SEGMENT_SIZE	CHAR(10)	空き領域の再利用を開始するセグメント数の指定値。※3 SEGMENT REUSE にセグメント数以外を指定している場合，及びビュー表の場合はナル値となります。
48	REUSE_SEGMENT_SIZE_TYPE	CHAR(1)	空き領域の再利用を開始するセグメント数の単位。 K：K 指定時 M：M 指定時 ナル値：省略時 SEGMENT REUSE にセグメント数以外を指定している場合，及びビュー表の場合はナル値となります。
49	INSERT_ONLY	CHAR(1)	改竄防止機能の指定有無。 Y：指定あり ナル値：指定なし 改竄防止機能を使用していない場合，及びビュー表の場合はナル値となります。
50	DELETE_PROHIBIT_TERM_TYPE	CHAR(1)	行削除禁止期間の種別。 I：日間隔データ Y：ラベル付き間隔（YEAR） M：ラベル付き間隔（MONTH） D：ラベル付き間隔（DAY） ナル値：指定なし

項番	列　　名	データ型	内　　容
			改竄防止機能を使用していない場合，行削除禁止期間を指定していない場合，及びビュー表の場合はナル値となります。
51	DELETE_PROHIBIT_TERM	CHAR(10)	行削除禁止期間の指定値※4。 改竄防止機能を使用していない場合，行削除禁止期間を指定していない場合，及びビュー表の場合はナル値となります。
52	SYSGEN_COLUMN_NAME	VARCHAR(30)，又は MVARCHAR(30)	挿入履歴保持列の名称。 改竄防止機能を使用していない場合，行削除禁止期間を指定していない場合，及びビュー表の場合はナル値となります。
53	N_TRIGGER	INTEGER	トリガ定義数。 トリガ未定義，ビュー表，及びディクショナリ表の場合はナル値となります。
54	N_DIV_DIMENSION	SMALLINT	分割次元数。 マトリクス分割表以外の場合はナル値となります。
55	AUDIT_TABLE_OPTION	CHAR(1)	この表が監査証跡表であるかどうかの値。 Y：監査証跡表 V：監査証跡表を基にしたビュー表 監査証跡表，及び監査証跡表を基にしたビュー表以外の場合はナル値となります。
56	N_PARENTS	SMALLINT	外部キーの数。 参照制約を定義していない表，及びビュー表の場合はナル値となります。
57	N_CHILDREN	SMALLINT	この表の主キーを参照する外部キーの数。 被参照表以外の表，及びビュー表の場合はナル値となります。
58	N_FK_COLUMNS	SMALLINT	外部キーの列数の合計。 参照制約を定義していない表，及びビュー表の場合はナル値となります。
59	CHECK_PEND	CHAR(1)	参照制約の検査保留状態の種別。 C：保留状態 ナル値：非保留状態 ビュー表の場合はナル値となります。
60	N_CHECK	INTEGER	定義した検査制約の数。 検査制約を定義していない表，及びビュー表の場合はナル値となります。
61	N_CHECK_LIMIT	INTEGER	検査制約制限値※5。

項番	列　　名	データ型	内　　容
			検査制約を定義していない表，及びビュー表の場合はナル値となります。
62	CHECK_PEND2	CHAR(1)	検査制約の検査保留状態の種別。 C：保留状態 ナル値：非保留状態 ビュー表の場合はナル値となります。
63	CHK_SOURCE_LEN	INTEGER	検査制約の探索条件の長さの合計。 検査制約を定義していない表，及びビュー表の場合はナル値となります。
64	SHARED	CHAR(1)	共用表の指定。 S：共用表 ナル値：非共用表
65	CHANGE_TIME_INSERT_ONLY	CHAR(14)	改竄防止表への変更時間（YYYYMMDDHHMMSS）。 表定義時，及びビュー表の場合はナル値となります。
66	N_UPDATE_COLUMN	SMALLINT	更新可能列属性の指定列数。 更新可能列属性を指定していない表，及びビュー表の場合はナル値となります。
67	TABLE_CREATOR	VARCHAR(30)，又は MVARCHAR(30)	パブリックビュー表の定義者。 パブリックビュー表以外の場合，ナル値となります。
68	N_CONSTRUCTOR_COLUMN	SMALLINT	システムが使用する情報。常にナル値となります。
69	CONSTRUCTOR_TYPE	CHAR(1)	システムが使用する情報。常にナル値となります。
70	NONE_DFLTCST_CLMCOUNT	SMALLINT	既定文字集合以外の文字集合の指定列数。 文字集合を指定していない場合はナル値となります。
71	CHARSET_SPECCOUNT	SMALLINT	文字集合を指定した列数。 文字集合を指定していない場合はナル値となります。
72	N_PARTIAL_STRUCTURE_INDEXES	SMALLINT	部分構造インデクス定義数。 部分構造インデクスを定義していない場合，及びビュー表の場合はナル値となります。
73	MEMORY_TABLE	CHAR(1)	システムが使用する情報。常にナル値となります。

項番	列　　名	データ型	内　　容
74	DBAREA_NAME	VARCHAR(30)，又は MVARCHAR(30)	システムが使用する情報。常にナル値となります。
75	XDS_NAME	CHAR(8)	システムが使用する情報。常にナル値となります。
76	N_COMPRESS	SMALLINT	圧縮列の数。 圧縮列がない場合，及びビュー表の場合はナル値となります。
77	TMP_COMMIT_ROWS	CHAR(1)	一時表のデータ有効期間。 D：DELETE（トランザクション固有一時表） P：PRESERVE（SQL セッション固有一時表） 一時表以外の場合はナル値となります。
78	REUSE_OPTION	INTEGER	空き領域の再利用の再利用オプション値。 SEGMENT_REUSE 列の値がナル値の場合，再利用オプション値が 0 の場合，及び再利用オプションを指定していない場合はナル値となります。

注※1

リバランス表に対して，ALTER TABLE ADD RDAREA で RD エリアを追加した場合，RD エリア追加前の分割情報数（SQL_DIV_TABLE 表の行数）が格納されます。ただし，一度設定した分割情報数は，リバランスユティリティ（pdrbal）を実行してリバランスが終了するまでは，ALTER TABLE ADD RDAREA で RD エリアを追加しても更新されません。リバランスが終了するとナル値となります。

注※2

セグメント数の単位を指定した場合，値は次のように格納されます。

'K'指定時：指定値×1024

'M'指定時：指定値×1024^2

注※3

値は，右詰めの文字形式で格納されます。なお，セグメント数の単位（'K'，及び'M'）は含まれません。

注※4

行削除禁止期間の種別によって，次のように格納されます。

'I'の場合："＋ YYYYMMDD."の文字形式

'Y'，'M'，及び'D'の場合：右詰めの文字形式

注※5

検査制約制限値は，検査制約の探索条件中で指定した論理演算子の数（CASE 式の WHEN 探索条件中の AND，OR を除く AND の数，及び OR の数）の合計と，検査制約数の合計となります。

例：

次のように表定義をした場合，検査制約制限値は 4 となります（論理演算子（AND と OR）の合計が 2，検査制約数の合計が 2)。

```
CREATE TABLE ”ZAIKO”
  (”GNO” CHAR(5),”GNAME” CHAR(8),”TANKA” INTEGER,
    ”SURYO” INTEGER,”NYUKOBI” DATE)
  CHECK(”SURYO”≧100 AND ”SURYO”≦1000)
  CONSTRAINT ”SURYOKISOKU”
  CHECK(”NYUKOBI”=DATE('1992-08-21')
    OR ”NYUKOBI”=DATE('1992-09-21'))
  CONSTRAINT ”NYUKOBIKISOKU”
```

(4)　SQL_COLUMNS 表の内容

この表では，列の定義情報を管理します（1 行で 1 列分)。

なお，SQL_COLUMNS 表の行は，表を定義するときに作成され，表を削除(スキーマの削除も含む)するときに削除されます。

SQL_COLUMNS 表の内容を次の表に示します。

表 G–5　SQL_COLUMNS 表の内容

項番	列　　名	データ型	内　　容
1	TABLE_SCHEMA	VARCHAR(30)，又は MVARCHAR(30)	表の所有者。パブリックビュー表の場合は PUBLIC。
2	TABLE_NAME	VARCHAR(30)，又は MVARCHAR(30)	列を含む表の名称。
3	COLUMN_NAME	VARCHAR(30)，又は MVARCHAR(30)	列名。
4	TABLE_ID	INTEGER	表 ID。
5	COLUMN_ID	SMALLINT	列 ID。 1 から始まる整数です。0 以下はありません。
6	DATA_TYPE	CHAR(24)	データ型。※1
7	DATA_LENGTH	CHAR(7)	列データ長。 右詰めで文字形式にして格納します(上位の 0 は空白にします)。
8	IS_NULLABLE	CHAR(3)	列ナル情報。 YES：ナル値を許容する NO：ナル値を許容しない
9	DIVIDED_KEY	CHAR(1)	分割キー。 Y：分割キー

項番	列　　名	データ型	内　　容
			空白：分割キー以外
10	CLUSTER_KEY	CHAR(1)	クラスタキー。 Y：クラスタキー構成列 空白：クラスタキー構成列以外
11	COLUMN_COMMENT	VARCHAR(255)，又は MVARCHAR(255)	コメント。 初期値はナル値となります。
12	BASE_TYPE	CHAR(1)	基の列の種別。※7 C：列 F：集合関数 E：そのほか 実表の場合はナル値となります。
13	BASE_OWNER	VARCHAR(30)，又は MVARCHAR(30)	基の列を含む実表の所有者。 実表の場合はナル値となります。
14	BASE_TABLE	VARCHAR(30)，又は MVARCHAR(30)	基の列を含む実表の名称。 実表の場合はナル値となります。
15	BASE_COLUMN	VARCHAR(30)，又は MVARCHAR(30)	基の列名。 実表の場合はナル値となります。
16	DEFAULT_COLUMN	CHAR(1)	WITH DEFAULT の指定。 Y：指定あり N：指定なし ビュー表の場合はナル値となります。
17	COLUMN_OFFSET	SMALLINT	列オフセット。 FIX 表でない表，及びビュー表の場合はナル値となります。
18	HASH_KEY	CHAR(1)	ハッシュキー。 Y：ハッシュキー 空白：ハッシュキー以外
19	RECOVERY_TYPE	CHAR(1)	RECOVERY の指定。 A：ALL P：PARTIAL N：NO データ型が BLOB 以外の場合はナル値となります。※8
20	LOB_LENGTH	CHAR(20)	列長の指定値。 右詰めで文字形式にして格納します（上位の 0 は空白にします）。 データ型が BLOB 及び BINARY 以外の場合はナル値となります。※8

項番	列　　名	データ型	内　　容
21	LOB_LENGTH_TYPE	CHAR(1)	列長のタイプ(列長の単位) 。 K：K 指定時 M：M 指定時 G：G 指定時 空白：省略時 データ型が BLOB 以外の場合はナル値となります。※8
22	DATA_TYPE_CODE	SMALLINT	データ型コード。※2
23	DATA_LENGTH_CODE	SMALLINT	列データ長コード。※3
24	LOB_LENGTH_CODE	CHAR(8)	BLOB 列データ長コード。※4※5 データ型が BLOB 及び BINARY 以外の場合はナル値となります。
25	DIVCOL_ORDER	SMALLINT	分割キー指定順序(0～16)。 該当する表内で 1 から始まるユニークな値です。分割キーの指定順に+1 した値となります。分割キーでない列は 0 を設定します。 非分割表，単一列分割キーを指定した表，及びビュー表の場合はナル値となります。
26	SUPPRESS_INF	CHAR(1)	データ抑制指定有無。 Y：指定あり ナル値：指定なし データ抑制を指定していない表，及びビュー表の場合はナル値となります。
27	PLUGIN_DESCRIPTION	VARCHAR(255)	プラグインオプションの内容。 PLUGIN 句の指定がない場合はナル値となります。
28	UDT_OWNER	VARCHAR(30)	ユーザ定義型の所有者。 ユーザ定義型でない場合はナル値となります。
29	UDT_NAME	VARCHAR(30)	ユーザ定義型の型名称。 ユーザ定義型でない場合はナル値となります。
30	UDT_TYPE_ID	INTEGER	ユーザ定義型の型 ID。 ユーザ定義型でない場合はナル値となります。
31	MAX_ELM	SMALLINT	繰返し列の最大要素数。 繰返し列でない列の場合はナル値となります。
32	NO_SPLIT	CHAR(1)	NO SPLIT の指定有無。

項番	列　　名	データ型	内　　容
			Y：指定あり ナル値：指定なし ビュー表の場合，及び ALTER TABLE CHANGE SPLIT 実行時の場合はナル値となります。
33	PRIMARY_KEY	CHAR(1)	主キー種別。 Y：主キーです。 空白：主キーではありません。
34	COLLATING_SEQUENCE	CHAR(1)	システムで使用する情報。
35	TRAILING_SPACE	CHAR(1)	システムで使用する情報。
36	SYSTEM_GENERATED	CHAR(1)	SYSTEM GENERATED の指定有無。 Y：指定あり ナル値：指定なし SYSTEM GENERATED を指定していない場合，及びビュー表の場合はナル値となります。
37	DEFAULT_CLAUSE	CHAR(1)	DEFAULT 句の指定有無。 Y：指定あり ナル値：指定なし DEFAULT 句を指定していない場合，及びビュー表の場合はナル値となります。
38	DEFAULT_VALUE	VARCHAR(32000)，又は MVARCHAR(32000)※6	DEFAULT 句に指定した既定値（文字形式）。※9 DEFAULT 句を指定していない場合，及びビュー表の場合はナル値となります。
39	DEFAULT_VALUE2	VARCHAR(32000)，又は MVARCHAR(32000)※6	DEFAULT 句に指定した既定値（定数指定時に，32,001～64,000 バイト目までの値を文字形式で格納）。※9 定数指定でない場合，DEFAULT 句を指定していない場合，及びビュー表の場合はナル値となります。
40	DEFAULT_VALUE3	VARCHAR(3)，又は MVARCHAR(3)	DEFAULT 句に指定した既定値（定数指定時に，64,001 バイト目以降の値を文字形式で格納）。※9 定数指定でない場合，DEFAULT 句を指定していない場合，及びビュー表の場合はナル値となります。
41	CHECK_COLUMN	CHAR(1)	検査制約の指定。 Y：あり 検査制約を定義していない場合，及びビュー表の場合はナル値となります。

項番	列　　名	データ型	内　　容
42	FOREIGN_KEY	CHAR(1)	外部キーの種別。 Y：外部キー構成列 ナル値：非外部キー構成列
43	UPDATABLE	CHAR(1)	更新可能列属性。 U：更新できます（UPDATE） N：ナル値から非ナル値へ一度だけ更新できます（UPDATE ONLY FROM NULL） 更新可能列属性を指定していない表，及びビュー表の場合はナル値となります。
44	CONSTRUCTOR_TYPE	CHAR(1)	システムが使用する情報。常にナル値となります。
45	CHARSET_SCHEMA	VARCHAR(30)，又は MVARCHAR(30)	文字集合所有者（常に"MASTER"）。 文字集合を指定していない場合はナル値となります。
46	CHARSET_NAME	VARCHAR(30)，又は MVARCHAR(30)	文字集合名。 "EBCDIK"：文字集合に EBCDIK を指定した場合 "UTF16"：文字集合に UTF16 を指定した場合 文字集合を指定していない場合はナル値となります。
47	CHARSET_ID	INTEGER	文字集合 ID。 文字集合を指定していない場合はナル値となります。 文字集合 ID については，マニュアル「HiRDB システム導入・設計ガイド」を参照してください。
48	COMPRESS_COLUMN	CHAR(1)	圧縮指定の有無。 Y：圧縮指定あり。 圧縮指定がない場合，及びビュー表の場合はナル値となります。
49	COMPRESS_PARTITION_SIZE_CODE	CHAR(8)	圧縮分割サイズ。※4※5 圧縮指定がない場合，及びビュー表の場合はナル値となります。
50	COMPRESS_PARTITION_SIZE	CHAR(20)	圧縮分割サイズ。 右詰めで文字形式にして格納します（上位の 0 は空白にします）。 圧縮指定がない場合，及びビュー表の場合はナル値となります。
51	COMPRESS_PARTITION_SIZE_TYPE	CHAR(1)	圧縮分割サイズの単位。 K：K 指定時

項番	列　　名	データ型	内　　容
			M：M 指定時 G：G 指定時 空白：省略時 圧縮指定がない場合，及びビュー表の場合はナル値となります。
52	RESERVED_COLUMN	CHAR(1)	予備列状態。 R：予備列 予備列でない列の場合，及びビュー表の場合はナル値となります。
53	DEFAULT_ROW_EXISTS	CHAR(1)	DEFAULT 句の ON ROW EXISTS 指定の有無。 Y：指定あり ON ROW EXISTS 指定でない場合，DEFAULT 句を指定していない場合，及びビュー表の場合は NULL 値となります。また，CREATE TABLE 実行時は常にナル値となります。

注※1

データ型によって格納する値は次のように異なります。

データ型	格納する値
INTEGER	INTEGER
SMALLINT	SMALLINT
DECIMAL	DECIMAL
FLOAT	FLOAT
DOUBLE PRECISION	
SMALLFLT	SMALLFLT
REAL	
CHAR	CHAR
VARCHAR	VARCHAR
NCHAR	NCHAR
NVARCHAR	NVARCHAR
MCHAR	MCHAR
MVARCHAR	MVARCHAR
DATE	DATE
TIME	TIME

データ型	格納する値
TIMESTAMP	TIMESTAMP
INTERVAL YEAR TO DAY	INTERVAL YEAR TO DAY
INTERVAL HOUR TO SECOND	INTERVAL HOUR TO SECOND
BINARY	BINARY
BLOB	BLOB
抽象データ型	ADT
BOOLEAN	BOOLEAN

注※2

データ型によって格納する値は次のように異なります。

データ型	格納する値	
	ナル値の指定ができる場合	ナル値の指定ができない場合
INTEGER	F1	F0
SMALLINT	F5	F4
DECIMAL	E5	E4
FLOAT	E1	E0
DOUBLE PRECISION		
SMALLFLT	E3	E2
REAL		
CHAR	C5	C4
VARCHAR	C1	C0
NCHAR	B5	B4
NVARCHAR	B1	B0
MCHAR	A5	A4
MVARCHAR	A1	A0
DATE	71	70
TIME	79	78
TIMESTAMP	7D	7C
INTERVAL YEAR TO DAY	65	64
INTERVAL HOUR TO SECOND	6F	6E
BINARY	91	90
BLOB	93	92

データ型	格納する値	
	ナル値の指定ができる場合	ナル値の指定ができない場合
抽象データ型	83	82
BOOLEAN	21	20

注※3

DECIMAL 型，INTERVAL YEAR TO DAY 型，INTERVAL HOUR TO SECOND 型の場合は，精度，位取りをそれぞれ 1 バイトに格納し，それ以外の場合は，2 バイトの 2 進形式で長さ（NCHAR 型，NVARCHAR 型の場合は文字数）を格納します。ただし，BLOB 型，BINARY 型，及び抽象データ型の場合は 0 になります。

注※4

列長，又は圧縮分割サイズの指定値を，4 バイトごとに区切られた 8 バイトの 2 進形式で格納します。

注※5

エンディアンが異なる接続形態でも，SQL の結果はエンディアン変換されません。したがって，アクセスする場合は，アプリケーション側でエンディアンを考慮して変換する必要があります。

注※6

NO SPLIT を指定しています。

注※7

選択式が次のどれかの場合に，E（そのほか）が設定されます。

- スカラ演算（四則演算，日付演算，時刻演算，連結演算，CASE 式，及びスカラ関数）
- 定数
- CAST 指定
- 関数呼出し（プラグイン関数を除く）
- USER 値関数
- CURRENT_DATE 値関数
- CURRENT_TIME 値関数
- CURRENT_TIMESTAMP 値関数

注※8

ビュー表の列の場合，結果のデータ型が BLOB となる関数呼出しを指定していると，次の値を設定します。これによって，関数定義時に指定した形式とは異なる形式となることがあります。

- RECOVERY_TYPE 列には NULL を設定する
- LOB_LENGTH 列には割り切れる最大の単位（K，M，又は G）で割った値を設定する
- LOB_LENGTH_TYPE 列には，割り切れる最大の単位を設定する

注※9

DEFAULT 句を指定した場合に格納される値を次の表に示します。

表 G–6　DEFAULT 句を指定した場合に格納される値

既定値			データ型※1	DEFAULT_VALUE 列，DEFAULT_VALUE2 列，又は DEFAULT_VALUE3 列に格納される値※2	
				データ長(文字形式での長さ)	既定値(文字形式)
省略			すべて	ナル値	ナル値
NULL			すべて	4	'NULL'
USER			CHAR，及び MCHAR	4	'USER'
			VARCHAR，及び MVARCHAR		
CURRENT_USER			CHAR，及び MCHAR	12	'CURRENT_USER'
			VARCHAR，及び MVARCHAR		
CURRENT DATE			DATE，又は CHAR(10)	12	'CURRENT△DATE'※3
CURRENT_DATE				12	'CURRENT_DATE'
CURRENT TIME			TIME，又は CHAR(8)	12	'CURRENT△TIME'※3
CURRENT_TIME				12	'CURRENT_TIME'
CURRENT TIMESTAMP(p) (p:小数秒精度)			TIMESTAMP，CHAR(19)，CHAR(22)，CHAR(24)，又は CHAR(26)	20	'CURRENT△TIMESTAMP(p)'※3※7
CURRENT_TIMESTAMP(p) (p:小数秒精度)				20	'CURRENT_TIMESTAMP(p)'※7
定数	文字列定数	文字列定数 例 1：'HiRDB' 例 2：'2002-10-24△10:50:23.1234'	CHAR，又は MCHAR	指定した既定値長＋2※4	指定した既定値※4 例：''HiRDB''
			VARCHAR，又は MVARCHAR		
			DATE，TIME，又は TIMESTAMP	指定した既定値長＋2※4	指定した既定値※4 例：''2002-10-24△10:50:23.1234''
		混在文字列定数 例：M'100 年'	CHAR，又は MCHAR	指定した既定値長＋3※4	指定した既定値※4 例：'M'100 年''
			VARCHAR，又は MVARCHAR		
		各国文字列定数 例：N'ソフト'	NCHAR，又は NVARCHAR	指定した既定値長＋3※4	指定した既定値※4 例：'N'ソフト''

既定値			データ型※1	DEFAULT_VALUE 列，DEFAULT_VALUE2 列，又は DEFAULT_VALUE3 列に格納される値※2	
				データ長(文字形式での長さ)	既定値(文字形式)
		16 進文字列定数 例 1：X'48692D43' 例 2：X'2002102410502312'	CHAR，VARCHAR，MCHAR，MVARCHAR，又は BINARY	指定した既定値長＋ 3※4	例：'X'48692D43''※4※6
			DATE，TIME，又は TIMESTAMP(p)		例：'X'2002102410502312''※4※6
	数定数	整数定数 例：10	INTEGER，SMALLINT，DECIMAL，FLOAT，又は SMALLFLT	指定した既定値長※5	指定した既定値※5 例：'10'
		浮動小数点定数 例：15e+3	INTEGER，SMALLINT，DECIMAL，FLOAT，又は SMALLFLT	22 又は 23	指定した既定値※5 例：'+1.500000000000000E+04' (左から，±1 バイト，仮数部 17 バイト(10 進数定数)，'E'1 バイト，±1 バイト，指数部 2～3 バイト(10 のべき乗))
		10 進数定数 例 1：15.5 例 2：-010101. 例 3：00011399.	INTEGER，SMALLINT，DECIMAL，FLOAT，SMALLINT，INTERVAL YEAR TO DAY，又は INTERVAL HOUR TO SECOND	指定した既定値長※5	指定した既定値※5 例 1：'△15.5' 例 2：'-010101.' 例 3： INTERVAL YEAR TO DAY の場合，'+00020199.' INTEGER の場合，'△00011399.' (INTERVAL YEAR TO DAY，及び INTERVAL HOUR TO SECOND の場合，値を補正し，先頭に正負を表す符号が付きます(それ以外の場合，正の値のときは空白になります))

（凡例）△：1 バイトの空白

注※1

BLOB，抽象データ型，及び 32,001 バイト以上の BINARY を除きます。

注※2

データ長が 32,001 バイト未満の場合，DEFAULT_VALUE2 列及び DEFAULT_VALUE3 列はナル値となります。データ長が 32,001～64,000 バイトの場合，DEFAULT_VALUE3 列がナル値となります。

注※3

CURRENT と，DATE，TIME，及び TIMESTAMP の間の空白は一つに編集します。

注※4

指定した既定値を文字形式の定数表現で格納します。データ長及び既定値に，定数表現の M，N，X，及びアポストロフィを含みます。したがって，データ長の範囲は，文字列定数の場合' 'を含めて 2～32,002 バイト，混在文字列定数及び各国文字列定数の場合 M' '，N' 'を含めて 3～32,003 バイト，16 進文字列定数の場合 X' 'を含めて 3～64,003 バイトとなります。

指定した定数の先頭 1～32,000 バイト目までを DEFAULT_VALUE 列に，32,001～64,000 バイト目までを DEFAULT_VALUE2 列に，64,000 バイト以降を DEFAULT_VALUE3 列に格納します。

例：

16 進文字列定数で 32,000 バイト分の既定値を指定した場合（X とアポストロフィを含めて合計 64,003 バイト）

VARCHAR(32000) DEFAULT X'C1C1C1…C1C1C1'

DEFAULT_VALUE 列には，先頭から 32,000 バイト「X'C1C1C1…」が格納されます。

DEFAULT_VALUE2 列には，続きの 32,000 バイト「C1C1C1…」が格納されます。

DEFAULT_VALUE3 列には，残りの 3 バイト「C1'」が格納されます。

注※5

指定した既定値が，文字形式の定数表現で格納されます。データ長には，文字形式表現での長さを格納します。

例：

数定数で既定値を指定した場合

INTEGER DEFAULT 100

DEFAULT_VALUE 列には，先頭から 3 バイト「100」が格納されます。

DEFAULT_VALUE2 列及び DEFAULT_VALUE3 列には，ナル値が格納されます。

注※6

値はすべて大文字です（指定した値が小文字でも大文字で格納されます）。

注※7

既定値に指定する CURRENT_TIMESTAMP 値の小数秒精度（p）を省略した場合，p=0 が仮定されます。

(5) SQL_INDEXES 表の内容

この表では，次に示すインデクスに関する情報を管理します（1 行で 1 インデクス分）。

- B-tree インデクス（主キー，クラスタキー，及び部分構造インデクスを含む）
- プラグインインデクス

SQL_INDEXES 表の内容を次の表に示します。

表 G–7　SQL_INDEXES 表の内容

項番	列　　名	データ型	内　　容
1	TABLE_SCHEMA	VARCHAR(30)，又は MVARCHAR(30)	表の所有者。
2	TABLE_NAME	VARCHAR(30)，又は MVARCHAR(30)	インデクスを含む表の名称。
3	INDEX_NAME	VARCHAR(30)，又は MVARCHAR(30)	インデクスの名称。
4	INDEX_ID	INTEGER	インデクス ID。
5	TABLE_ID	INTEGER	表 ID。
6	UNIQUE_TYPE	CHAR(1)	ユニークタイプ。 U：ユニーク N：非ユニーク
7	COLUMN_COUNT	SMALLINT	インデクス構成列数。
8	CREATE_TIME	CHAR(14)	インデクス作成時刻 (YYYYMMDDHHMMSS) 。
9	RDAREA_NAME	VARCHAR(30)，又は MVARCHAR(30)	非分割格納先 RD エリア名称。 インデクスが分割されて格納されている場合，及び一時インデクスの場合はナル値となります。
10	CLUSTER_KEY	CHAR(1)	インデクス種別。 Y：クラスタキーのインデクス N：クラスタキーのインデクス以外
11	DIV_INDEX	CHAR(1)	インデクス構成列の先頭列の種別。 Y：分割キー（複数列分割キーの場合は，CREATE TABLE で指定した分割キーと先頭から同じ並び順），又はプラグインインデクス N：分割キー以外
12	FREE_AREA	SMALLINT	ページ内未使用領域の比率(%)。
13	COLUMN_ID_LIST	VARCHAR(128)	インデクスを構成する列 ID リスト。※ +-で昇順降順を示します。単一列インデクス（クラスタキーのインデクス以外）の降順指定時には+を設定します。プラグインインデクスの場合は常に+を設定します。
14	SPLIT_OPT	CHAR(1)	ページスプリットオプション。 U：アンバランススプリット アンバランススプリットを指定していないインデクスの場合はナル値となります。

項番	列　　名	データ型	内　　容
15	ATTR_COUNT	SMALLINT	インデクスを構成する抽象データ型の属性の数。 CREATE INDEX(形式 1)の場合はナル値となります。
16	INDEX_TYPE_OWNER	VARCHAR(30)，又は MVARCHAR(30)	インデクス型の所有者。 CREATE INDEX(形式 1)の場合はナル値となります。
17	INDEX_TYPE_NAME	VARCHAR(30)，又は MVARCHAR(30)	インデクス型の名称。 CREATE INDEX(形式 1)の場合はナル値となります。
18	INDEX_TYPE_ID	INTEGER	インデクス型の ID。 CREATE INDEX(形式 1)の場合はナル値となります。
19	PLUGIN_DESCRIPTION	VARCHAR(255)	プラグインオプションの内容。 PLUGIN 指定がない場合はナル値となります。
20	N_FUNCTION	INTEGER	適用関数の数。 CREATE INDEX(形式 1)の場合はナル値となります。
21	EXCEPT_VALUES	CHAR(1)	除外値指定の有無。 Ｙ：あり Ｎ：なし
22	N_EXCEPT_VALUES	SMALLINT	インデクスの除外キー値の数。 除外値指定のないインデクスの場合はナル値となります。
23	ARRAY_TYPE	CHAR(1)	インデクス構成列の種別。 M：インデクス構成列に繰返し列を含みます。 ナル値：インデクス構成列に繰返し列を含みません。
24	LOCK_OPT	CHAR(1)	システムが使用する情報。
25	PRIMARY_KEY	CHAR(1)	インデクスの種別。 Y：主キーインデクスです。 ナル値：主キーインデクスではありません。
26	DIV_IN_SRV	CHAR(1)	非分割キーインデクスのサーバ内横分割の有無。 Y：サーバ内分割しています。 ナル値：サーバ内分割していません。 分割キーインデクスの場合もナル値となります。

項番	列　　名	データ型	内　　容
27	SHARED	CHAR(1)	共用インデクスの指定。 S：共用インデクス ナル値：非共用インデクス
28	N_PARTIAL_STRUCTURE_PATHS	SMALLINT	部分構造インデクスの構成部分構造パス数。 部分構造インデクスを定義していない場合はナル値となります。
29	USING_UNIQUE_TAG	CHAR(1)	部分構造パスのユニーク種別 'Y'：部分構造パスがユニークである NULL：上記以外 部分構造インデクスを定義していない場合，及び USING UNIQUE TAG を指定していない場合はナル値となります。
30	DBAREA_NAME	VARCHAR(30)，又は MVARCHAR(30)	システムが使用する情報。常にナル値となります。
31	TMP_INDEX_TYPE	CHAR(1)	一時インデクスの種別。 G：一時インデクス 一時インデクス以外の場合はナル値となります。

注※

エンディアンが異なる接続形態でも，SQL の結果はエンディアン変換されません。したがって，アクセスする場合は，アプリケーション側でエンディアンを考慮して変換する必要があります。

(6)　SQL_USERS 表の内容

この表では，ユーザの実行権限，及び DBA 権限の情報を管理します（1 行で 1 ユーザ分）。

なお，この表を参照できるのは，DBA 権限所有者，及び監査人だけです。

SQL_USERS 表の内容を次の表に示します。

表 G–8　SQL_USERS 表の内容

項番	列　　名	データ型	内　　容
1	USER_ID	VARCHAR(30)，又は MVARCHAR(30)	権限を持つユーザの名称。
2	DBA_PRIVILEGE	CHAR(1)	DBA 権限。 Y：DBA 権限がある N：DBA 権限がない 初期値は N となります。
3	SCHEMA_PRIVILEGE	CHAR(1)	スキーマ定義権限。

項番	列　　名	データ型	内　　容
			Y：スキーマ定義権限がある S：スキーマを所有している N：スキーマ定義権限がない 初期値は N となります。
4	CREATE_TIME	CHAR(14)	スキーマ生成時の時刻。(YYYYMMDDHHMMSS)。 初期値，及び DROP SCHEMA 実行時はナル値となります。
5	AUDIT_PRIVILEGE	CHAR(1)	監査権限の有無。 Y：監査権限あり ナル値：監査権限なし 監査人以外のユーザの場合はナル値となります。
6	AUTH_ERR_COUNT	SMALLINT	連続認証失敗回数。 連続認証失敗回数を指定していない場合，ユーザ認証の連続失敗が 0 回の場合，及び連続認証失敗回数をクリアした場合，ナル値となります。
7	CON_LOCK_TIME	TIMESTAMP(0)	連続認証失敗アカウントロック日時。 連続認証失敗回数を指定していない場合，及び連続認証失敗アカウントロック状態でない場合，ナル値となります。※
8	PWD_LOCK_TIME	TIMESTAMP(0)	パスワード無効アカウントロック日時。 パスワード文字列制限を指定していない場合，及びパスワード無効アカウントロック状態でない場合，ナル値となります。
9	PASSWORD_TEST	CHAR(1)	パスワード制限の違反種別コード。 L：最小許容バイト数 U：認可識別子の指定禁止 S：単一文字種の指定禁止 P：英大文字の指定必須 W：英小文字の指定必須 N：数字の指定必須 パスワード無効アカウントロック状態になるユーザを事前調査していない場合，及び事前調査後に違反がなかった場合，ナル値となります。
10	OS_AUTH	CHAR(1)	Y：簡易認証ユーザ ナル値：非簡易認証ユーザ
11	PWD_INTERVAL_DAY	SMALLINT	パスワードの有効期限間隔の日数。 パスワードの有効期限定義が設定されていない場合，ナル値となります。
12	PWD_LAST_CHANGE	TIMESTAMP(0)	パスワードの最終更新日時，又はパスワードの有効期限定義を設定した日時。

項番	列　　名	データ型	内　　容
			パスワードの有効期限定義が設定されていない場合，ナル値となります。
13	DEPUTY_AUDITOR	CHAR(1)	Y：副監査人 ナル値：副監査人以外
14	SCHEMA_OPERATION_PRIVILEGE	VARCHAR(30)，又は MVARCHAR(30)	スキーマ操作権限対象のスキーマ名

注※

連続認証失敗アカウントロック状態になって，設定したアカウントロック期間経過後に CONNECT をしていない場合は，連続認証失敗アカウントロック状態でないときでもナル値にはなりません。

(7)　SQL_RDAREA_PRIVILEGES 表の内容

この表では，RD エリア利用権限の付与状況を管理します（1 行で 1RD エリアの 1 ユーザ分）。

SQL_RDAREA_PRIVILEGES 表の内容を次の表に示します。

表 G–9　SQL_RDAREA_PRIVILEGES 表の内容

項番	列　　名	データ型	内　　容
1	GRANTEE	VARCHAR(30)，又は MVARCHAR(30)	RD エリア利用権限を持つユーザの名称，又は'PUBLIC'。
2	RDAREA_NAME	VARCHAR(30)，又は MVARCHAR(30)	RD エリア名。
3	GRANT_TIME	CHAR(14)	該当する権限を受け取った時刻 (YYYYMMDDHHMMSS)。

(8)　SQL_TABLE_PRIVILEGES 表の内容

この表では，表のアクセス権限の付与状況を管理します（1 行で 1 ユーザ分）。

なお，SQL_TABLE_PRIVILEGES 表の行は，GRANT アクセス権限で該当するユーザが表に対してアクセス権限を与えられたときに作成されます。また，REVOKE アクセス権限で該当するユーザが表に対してすべての権限を失効したときに行が削除されます。

SQL_TABLE_PRIVILEGES 表の内容を次の表に示します。

表 G–10　SQL_TABLE_PRIVILEGES 表の内容

項番	列　　名	データ型	内　　容
1	GRANTOR	VARCHAR(30)，又は MVARCHAR(30)	表のアクセス権限を許可するユーザの名称，又はパブリックビュー表の定義者。

項番	列　　名	データ型	内　　容
2	GRANTEE	VARCHAR(30)，又は MVARCHAR(30)	表のアクセス権限を受けるユーザの名称，又は'PUBLIC'。
3	TABLE_SCHEMA	VARCHAR(30)，又は MVARCHAR(30)	アクセス権限を許可する表の所有者。パブリックビュー表の場合は PUBLIC。
4	TABLE_NAME	VARCHAR(30)，又は MVARCHAR(30)	アクセス権限を許可する表の名称。
5	SELECT_PRIVILEGE	CHAR(1)	SELECT 権限の有無。 G：許可される(表の所有者の場合)※ Y：許可される N：許可されない 初期値は N となります。
6	INSERT_PRIVILEGE	CHAR(1)	INSERT 権限の有無。 G：許可される(表の所有者の場合)※ Y：許可される N：許可されない 初期値は N となります。
7	DELETE_PRIVILEGE	CHAR(1)	DELETE 権限の有無。 G：許可される(表の所有者の場合)※ Y：許可される N：許可されない 初期値は N となります。
8	UPDATE_PRIVILEGE	CHAR(1)	UPDATE 権限の有無。 G：許可される(表の所有者の場合)※ Y：許可される N：許可されない 初期値は N となります。
9	GRANT_TIME	CHAR(14)	該当する権限を受け取った時刻 (YYYYMMDDHHMMSS)。
10	GRANTEE_TYPE	CHAR(1)	テーブルアクセス権限を受ける相手の種別 'R'：ロール ナル値：ユーザ

注※

ビュー表の所有者とビュー表の基表の所有者が異なる場合や読み込み専用ビュー表の場合は，ビュー表の所有者であっても G 以外が格納されます。各列に格納される値を次に示します。

ビュー表の所有者とビュー表の基表の所有者	ビュー表の種別	格納される値			
		SELECT_PRIVILEGE 列	INSERT_PRIVILEGE 列	DELETE_PRIVILEGE 列	UPDATE_PRIVILEGE 列
すべて一致している	ビュー表	G	G	G	G
	読み込み専用ビュー表	G	N	N	N
不一致である（部分的不一致を含む）	ビュー表	Y	ビュー表の所有者がビュー表定義時に基表への INSERT 権限を持っている場合：Y 持っていない場合：N	ビュー表の所有者がビュー表定義時に基表への DELETE 権限を持っている場合：Y 持っていない場合：N	ビュー表の所有者がビュー表定義時に基表への UPDATE 権限を持っている場合：Y 持っていない場合：N
	読み込み専用ビュー表	Y	N	N	N

(9)　SQL_VIEW_TABLE_USAGE 表の内容

この表では，ビュー表の基になる実表の情報を管理します（1 行で 1 ビュー表分）。

SQL_VIEW_TABLE_USAGE 表の内容を次の表に示します。

表 G–11　SQL_VIEW_TABLE_USAGE 表の内容

項番	列　　名	データ型	内　　容
1	VIEW_SCHEMA	VARCHAR(30)，又は MVARCHAR(30)	ビュー表の所有者。パブリックビュー表の場合は PUBLIC。
2	VIEW_NAME	VARCHAR(30)，又は MVARCHAR(30)	ビュー表の名称。
3	BASE_OWNER	VARCHAR(30)，又は MVARCHAR(30)	基になる表，又は利用資源の所有者。パブリックビュー表，又はパブリックルーチンの場合は PUBLIC。
4	BASE_TABLE_NAME	VARCHAR(30)，又は MVARCHAR(30)	基になる表，又は利用資源の名称。
5	BASE_TYPE	CHAR(1)	基になる表，又は利用資源の種別。 R：永続実表 V：ビュー表 P：ユーザ定義関数（プラグイン提供関数を除く） G：一時表
6	BASE_ROUTINE_CREATOR	VARCHAR(30)，又は MVARCHAR(30)	利用資源がパブリック関数の場合の，パブリック関数定義者。

項番	列　　名	データ型	内　　容
			利用資源がパブリック関数以外の場合はナル値となります。

(10)　SQL_VIEWS 表の内容

この表では，ビュー表の定義情報を管理します（1 行で 1 ビュー表分）。

SQL_VIEWS 表の内容を次の表に示します。

表 G–12　SQL_VIEWS 表の内容

項番	列　　名	データ型	内　　容
1	VIEW_SCHEMA	VARCHAR(30)，又は MVARCHAR(30)	ビュー表の所有者。パブリックビュー表の場合は PUBLIC。
2	VIEW_NAME	VARCHAR(30)，又は MVARCHAR(30)	ビュー表の名称。
3	SOURCE_ORDER	INTEGER	複数行に分けて格納した場合の順序(1〜n)。
4	IS_UPDATABLE	CHAR(3)	更新の可能性。 YES：可能性がある NO：可能性がない
5	VIEW_DEFINITION	VARCHAR (32000)，又は MVARCHAR(32000)	ビュー定義のソース文。
6	VIEW_ID	INTEGER	ビュー ID。

(11)　SQL_DIV_TABLE 表の内容

この表では，データベース内の表の分割情報を管理します（n 行で 1 表分）。

SQL_DIV_TABLE 表の内容を次の表に示します。

表 G–13　SQL_DIV_TABLE 表の内容

項番	列　　名	データ型	内　　容
1	TABLE_SCHEMA	VARCHAR(30)，又は MVARCHAR(30)	表の所有者。
2	TABLE_NAME	VARCHAR(30)，又は MVARCHAR(30)	表の名称。
3	DIV_NO	INTEGER	分割条件指定順序。 該当する表内で 1 から始まるユニークな値で，分割条件の指定順に 1 を加えた値となります。

項番	列　　名	データ型	内　　容
4	TABLE_ID	INTEGER	表 ID。
5	DCOND	CHAR(2)	分割条件コード。 表の分割格納条件値を文字形式で格納(格納される値は，=，^=，<，<=，>，>=のどれか)します。ユーザが< >，及び!=を指定した場合，^=で格納します。 マトリクス分割表の場合は<=が格納されます。 分割格納条件指定なし，及びハッシュ分割の場合は空白となります。
6	DCVALUES	VARCHAR(256)，又は MVARCHAR(256)	分割条件値。 分割キーに文字集合の指定がある場合でも，格納される値は変換されません。※1，※2 分割格納条件指定なし，又はハッシュ分割の場合はナル値となります。
7	RDAREA_NAME	VARCHAR(30)，又は MVARCHAR(30)	分割格納先 RD エリア名。
8	DCVALUES2	VARCHAR(255)，又は MVARCHAR(255)	第 2 次元キーの分割条件値（格納形式は DCVALUES と同じ）。 分割キーに文字集合の指定がある場合でも，格納される値は変換されません。※1，※2 マトリクス分割表以外の場合，及びマトリクス分割表で境界値指定がない場合はナル値となります。
9	DCVALUES_TYPE	CHAR(1)	分割条件値（DCVALUES 列の値）の指定形式。 X：16 進文字列定数 DCVALUES 列の値がナル値の場合，及び分割条件に指定する定数が 16 進文字列定数以外の場合はナル値となります。
10	DCVALUES2_TYPE	CHAR(1)	第 2 次元キーの分割条件値（DCVALUES2 列の値）の指定形式。 X：16 進文字列定数 DCVALUES2 列の値がナル値の場合，及び分割条件に指定する定数が 16 進文字列定数以外の場合はナル値となります。

注※1

分割条件値を 16 進文字列定数で指定した場合，指定した 16 進文字列定数の値がそのまま格納されます。

分割条件値に指定した 16 進文字列を確認する場合，スカラ関数 HEX を使用して 16 進文字列表現に変換して確認してください。16 進文字列表現に変換しないで参照すると，16 進文字列のコードに対応する文字が存在しない場合，文字化けが発生することがあります。

注※2

分割キーのデータ型が CHAR 又は MCHAR 型で分割条件値を 16 進文字列定数で指定した場合，分割条件値長（16 進文字数÷2）が分割キーの定義長に満たないとき，分割キーの定義長に満たない部分は半角スペースで埋めないで分割条件値長分格納されます。なお，16 進文字列定数以外で指定した場合は，定義長に満たない部分は半角スペースで埋めて格納されます。

(12) SQL_INDEX_COLINF 表の内容

この表は，インデクスの構成列情報を管理します（n 行で 1 インデクス分）。

SQL_INDEX_COLINF 表の内容を次の表に示します。

表 G–14 SQL_INDEX_COLINF 表の内容

項番	列　　名	データ型	内　　容
1	TABLE_SCHEMA	VARCHAR(30)，又は MVARCHAR(30)	表の所有者。
2	TABLE_NAME	VARCHAR(30)，又は MVARCHAR(30)	インデクスを含む表の名称。
3	INDEX_NAME	VARCHAR(30)，又は MVARCHAR(30)	インデクスの名称。
4	INDEX_ID	INTEGER	インデクス ID。
5	INDEX_ORDER	INTEGER	インデクスを構成する列の順序(インデクスを構成する列名順を識別する番号で，1 から始まる整数) 。
6	COLUMN_NAME	VARCHAR(30)，又は MVARCHAR(30)	列名(インデクス構成列名) 。
7	ASC_DESC	CHAR(1)	昇順，又は降順。 Ａ：昇順 Ｄ：降順 空白：昇降順なし（プラグインインデクスの場合） 単一列インデクスの降順指定時には，昇順として格納します。

(13) SQL_DIV_INDEX 表の内容

この表では，インデクスの分割情報（CREATE TABLE 時に指定した分割条件，及び格納 RD エリア名）を管理します（n 行で 1 インデクス分）。

SQL_DIV_INDEX 表の内容を次の表に示します。

表 G–15　SQL_DIV_INDEX 表の内容

項番	列　　名	データ型	内　　容
1	TABLE_SCHEMA	VARCHAR(30)，又は MVARCHAR(30)	表の所有者。
2	TABLE_NAME	VARCHAR(30)，又は MVARCHAR(30)	インデクスを含む表の名称。
3	INDEX_NAME	VARCHAR(30)，又は MVARCHAR(30)	インデクスの名称。
4	DIV_NO	INTEGER	RD エリア定義順序。 該当するインデクス内で 1 から始まるユニークな値で，RD エリアの定義順序に 1 を加えた値※となります。
5	INDEX_ID	INTEGER	インデクス ID。
6	RDAREA_NAME	VARCHAR(30)，又は MVARCHAR(30)	(分割)格納先 RD エリア名。

注※　該当する列の値と SQL_DIV_TABLE の DIV_NO とは関連性はありません。

(14)　SQL_DIV_COLUMN 表の内容

この表では，BLOB 型列の分割情報（CREATE TABLE 時に指定した格納 RD エリア名）を管理します（n 行で 1 列分）。

SQL_DIV_COLUMN 表の内容を次の表に示します。

表 G–16　SQL_DIV_COLUMN 表の内容

項番	列　　名	データ型	内　　容
1	TABLE_SCHEMA	VARCHAR(30)，又は MVARCHAR(30)	表の所有者。
2	TABLE_NAME	VARCHAR(30)，又は MVARCHAR(30)	表の名称。
3	COLUMN_NAME	VARCHAR(30)，又は MVARCHAR(30)	列名。
4	DIV_NO	INTEGER	格納順序。
5	RDAREA_NAME	VARCHAR(30)，又は MVARCHAR(30)	ユーザ LOB 用 RD エリアの名称。
6	STORE_NO	INTEGER	常に 1。
7	MASTER_RDAREA_NAME	VARCHAR(30)，又は MVARCHAR(30)	対応する表のユーザ用 RD エリアの名称。

項番	列　　名	データ型	内　　容
8	N_LEVEL	SMALLINT	レベル数。 BLOB 型の列の場合はナル値となります。
9	COMPONENT_NAME	VARCHAR(30)，又は MVARCHAR(30)	コンポーネント名。 BLOB 型の列の場合はナル値となります。
10	LOB_NO	SMALLINT	LOB 属性番号。 BLOB 型の列の場合はナル値となります。

(15)　SQL_ROUTINES 表の内容

この表では，ルーチンの定義情報を管理します（1 行で 1 ルーチン分）。

SQL_ROUTINES 表の内容を次の表に示します。

表 G–17　SQL_ROUTINES 表の内容

項番	列　　名	データ型	内　　容
1	ROUTINE_SCHEMA	VARCHAR(30)，又は MVARCHAR(30)	ルーチンの所有者。パブリックルーチンの場合は PUBLIC。
2	ROUTINE_NAME	VARCHAR(30)，又は MVARCHAR(30)	ルーチン名称。※9
3	OBJECT_ID	INTEGER	オブジェクト ID。
4	SPECIFIC_NAME	VARCHAR(30)，又は MVARCHAR(30)	特定名称。※2
5	ROUTINE_TYPE	CHAR(1)	ルーチン種別。 P：手続き F：関数
6	ROUTINE_VALID	CHAR(1)	有効フラグ。 Y：有効ルーチン N：無効ルーチン
7	INDEX_VALID	CHAR(1)	インデクス状態変化フラグ。 Y：インデクス状態有効 N：インデクス状態無効※1
8	CREATE_TIME	CHAR(14)	ルーチン作成時の時刻 (YYYYMMDDHHMMSS)。 SQL 手続き文の場合は SQL 解析時刻，外部ルーチンの場合はルーチン定義作成時刻となります。
9	ALTER_TIME	CHAR(14)	ルーチン再作成時刻 (YYYYMMDDHHMMSS)。

項番	列　　名	データ型	内　　容
			初期値はナル値となります。
10	OBJECT_SIZE	INTEGER	オブジェクトサイズ(単位:バイト)。 外部ルーチンの場合は 0 となります。
11	SOURCE_SIZE	INTEGER	定義ソースサイズ(単位:バイト)。 外部ルーチン及びレジストリ操作用プロシジャの場合は 0 となります。
12	ISOLATION_LEVEL	SMALLINT	データ保証レベル(0〜2)。 手続きの場合に有効となります。 ルーチン中に指定する SQL でだけ有効となります。
13	OPTIMIZE_LEVEL	INTEGER	SQL 最適化オプション（10 進数に変換した形式)。 CREATE PROCEDURE，ALTER PROCEDURE，CREATE TYPE，及び ALTER ROUTINE の OPTIMIZE LEVEL 指定値を設定します。
14	SQL_LEVEL	SMALLINT	システムで使用する情報
15	N_PARAM	INTEGER	パラメタの数。
16	N_RESOURCE	INTEGER	オブジェクト内で使用するリソースの数。
17	PARAM_LOCATION	INTEGER	定義ソース文中の手続き文の開始位置。※7
18	ROUTINE_COMMENT	VARCHAR(255)，又は MVARCHAR(255)	コメント。 初期値はナル値となります。
19	DEF_SOURCE	BLOB	定義ソース文(コンパイルオプション部分は除きます)。 システム定義関数，レジストリ操作用プロシジャ，及びトリガ動作手続きの場合はナル値になります。
20	ROUTINE_ADT_OWNER	VARCHAR(30)	ルーチンを定義した抽象データ型の所有者。 抽象データ型内で定義したルーチン以外の場合はナル値となります。
21	ROUTINE_ADT_NAME	VARCHAR(30)	ルーチンを定義した抽象データ型の名称。 抽象データ型内で定義したルーチン以外の場合はナル値となります。
22	ROUTINE_BODY	CHAR(1)	関数のルーチン本体種別。 S：SQL 手続き E：外部ルーチン T：トリガ動作手続き

項番	列　　名	データ型	内　　容
			外部ルーチン以外のプロシジャ（トリガ動作手続きは除く）の場合はナル値となります。
23	FUNCTION_TYPE	CHAR(1)	関数種別。 C：システム定義関数のコンストラクタ 空白：ユーザ定義関数 手続きの場合はナル値となります。
24	EXTERNAL_NAME	VARCHAR(255)	外部ルーチン名(ライブラリ名!オペレーション名)。 Java で定義した場合は，Java メソッド名。 外部ストアドルーチン以外の場合はナル値となります。
25	EXTERNAL_LANGUAGE	CHAR(20)	外部記述言語種別。 C：C 言語 Java：Java 言語 外部ストアドルーチン以外の場合はナル値となります。
26	PARAMETER_STYLE	VARCHAR(20)	パラメタスタイル(外部ストアドルーチン種別) 。 PLUGIN：プラグイン RDSQL：RDSQL Java：Java 外部ストアドルーチン以外の場合はナル値となります。
27	ENCAPSULATION_LEVEL	VARCHAR(10)	隠蔽レベル(PUBLIC，PRIVATE，又は PROTECTED)。 抽象データ型で定義されたルーチン以外の場合はナル値となります。
28	RETURN_UDT_OWNER	VARCHAR(30)，又は MVARCHAR(30)	戻り値のデータ型の所有者。 戻り値がユーザ定義型の関数以外の場合はナル値となります。
29	RETURN_UDT_NAME	VARCHAR(30)，又は MVARCHAR(30)	戻り値のデータ型の名称。 戻り値がユーザ定義型の関数以外の場合はナル値となります。
30	RETURN_UDT_TYPE_ID	INTEGER	戻り値のデータ型の ID。 戻り値がユーザ定義型の関数以外の場合はナル値となります。
31	RETURN_DATA_TYPE	CHAR(24)	戻り値のデータ型。 格納形式は，SQL_COLUMNS 表の DATA_TYPE 列を参照してください。 関数以外の場合はナル値となります。

項番	列　　名	データ型	内　　容
32	RETURN_DATA_TYPE_CODE	SMALLINT	戻り値のデータ型コード。 関数以外の場合はナル値となります。 格納形式は，SQL_COLUMNS 表の DATA_TYPE_CODE 列を参照してください。
33	RETURN_DATA_LENGTH_CODE	SMALLINT	戻り値のデータ長コード。※3 手続きの場合はナル値となります。
34	RETURN_DATA_LENGTH	CHAR(7)	戻り値のデータ長。 右詰めで文字形式にして格納します(上位の 0 は空白にします)。 手続きの場合はナル値となります。
35	RETURN_LOB_LENGTH_CODE	CHAR(8)	戻り値の BLOB データ長コード。※4※8 手続きの場合，又は戻り値が BLOB，BINARY 以外の関数の場合はナル値となります。
36	RETURN_LOB_LENGTH	CHAR(20)	戻り値の BLOB データ長指定値。 右詰めで文字形式にして格納します(上位の 0 は空白にします)。 手続きの場合，又は戻り値が BLOB，BINARY 以外の関数の場合はナル値となります。
37	RETURN_LOB_LENGTH_TYPE	CHAR(1)	戻り値の BLOB データ長種別。 K：K 指定時 M：M 指定時 G：G 指定時 空白：省略時 手続きの場合，又は戻り値が BLOB 以外の関数の場合はナル値となります。
38	ADDITIONAL_OPTIMIZE_LEVEL	INTEGER	SQL 拡張最適化オプション（10 進数に変換した形式）。 CREATE PROCEDURE，ALTER PROCEDURE，CREATE TYPE，及び ALTER ROUTINE の ADD OPTIMIZE LEVEL 指定値を設定します。 バージョンが 06-00 より前の HiRDB で作成したルーチンの場合はナル値となります。
39	CLASS_NAME	VARCHAR(255)	パッケージ名．クラス名。※5 Java で記述した外部ルーチン以外の場合はナル値となります。
40	JAR_NAME	VARCHAR(255)	Java アーカイブファイル名。

項番	列　　名	データ型	内　　容
			Java で記述した外部ルーチン以外の場合はナル値となります。
41	DYNAMIC_RESULT_SETS	SMALLINT	返却する結果集合の最大数。 結果集合の最大数を指定していない場合はナル値となります。
42	SQL_SPECIFICATION	CHAR(1)	システムが使用する情報。
43	RETURNS_JAVA_DATA_TYPE	VARCHAR(255)	戻り値のデータ型に対応する Java の戻り値のデータ型。※6 Java で記述した外部ルーチン以外の場合はナル値となります。
44	RETURNS_JAVA_DATA_TYPE_CODE	INTEGER	戻り値のデータ型に対応する Java の戻り値のデータ型コード。※6 Java で記述した外部ルーチン以外の場合はナル値となります。
45	RETURN_DATA_MAX_ELM	SMALLINT	戻り値のデータ型の最大要素数。 戻り値のデータ型に ARRAY を指定していない場合はナル値となります。
46	N_JAVA_RESULT_SETS	INTEGER	Java.sql.ResultSet[]の指定数。 Java.sql.ResultSet[]を指定していない場合はナル値となります。
47	FOR_UPDATE_EXCLUSIVE_LOCK	CHAR(1)	FOR UPDATE EXCLUSIVE の指定有無 Y：次のどちらかの場合 • FOR UPDATE EXCLUSIVE 指定あり • 推奨モードを適用し，FOR UPDATE を省略した場合 N：次のどちらかの場合 • FOR UPDATE COMPATIBLE 指定あり • 0904 互換モードを適用し，FOR UPDATE を省略した場合 ナル値：次のどれかの場合 • バージョン 07-00 以前に作成したルーチンの場合 • バージョン 09-50 より前に作成した ISOLATION LEVEL が 2 のルーチンの場合 • バージョン 09-50 より前に作成した FOR UPDATE EXCLUSIVE 未指定のルーチンの場合 ルーチン中に指定する SQL でだけ有効となります。

項番	列　　名	データ型	内　　容
48	SUBSTR_LENGTH	SMALLINT	SQL コンパイルオプションの SUBSTR LENGTH の指定値。 バージョン 08-00 より前の HiRDB で作成したルーチンの場合，及び文字コード種別が Unicode（UTF-8），Unicode（IVS 対応 UTF-8）でない場合はナル値となります。
49	RETCSET_SCHEMA	VARCHAR(30)，又は MVARCHAR(30)	戻り値のデータ型の文字集合所有者（常に"MASTER"）。 戻り値のデータ型の文字集合を指定していない場合はナル値となります。
50	RETCSET_NAME	VARCHAR(30)，又は MVARCHAR(30)	戻り値のデータ型の文字集合名。 EBCDIK：文字集合に EBCDIK を指定した場合 UTF16：文字集合に UTF16 を指定した場合 戻り値のデータ型の文字集合を指定していない場合はナル値となります。
51	RETCSET_ID	INTEGER	戻り値のデータ型の文字集合 ID。 戻り値のデータ型の文字集合を指定していない場合はナル値となります。 文字集合 ID については，マニュアル「HiRDB システム導入・設計ガイド」を参照してください。
52	ROUTINE_CREATOR	VARCHAR(30)，又は MVARCHAR(30)	パブリックルーチンの定義者。 パブリックルーチン以外の場合はナル値となります。

注※1

ルーチン内のインデクス情報が無効になった状態(ルーチンは実行できません)。この場合，ALTER ROUTINE 又は ALTER PROCEDURE で SQL オブジェクトを再作成する必要があります。

注※2

手続きの場合は，ルーチン名称と同じになります。関数の場合は，システムが内部的にルーチン名称とオブジェクト ID から生成した，次のような名称を付けます。

'F' ルーチン名称(最大 19 バイト) オブジェクト ID(10 バイト)

注※3

DECIMAL 型，INTERVAL YEAR TO DAY 型，INTERVAL HOUR TO SECOND 型の場合は，精度，位取りをそれぞれ 1 バイトに格納し，それ以外の場合は，2 バイトの 2 進形式で長さ（NCHAR 型，NVARCHAR 型の場合は文字数）を格納します。ただし，BLOB 型及び抽象データ型の場合は 0 になります。

注※4

列長の指定値を 4 バイトごとに区切られた 8 バイトに 2 進形式で格納します。

注※5

パッケージ名．クラス名の格納形式を次に示します。

- パッケージ名の指定がある場合
  パッケージ名．クラス名
- パッケージ名の指定がない場合
  クラス名

注※6

RETURNS_JAVA_DATA_TYPE には，次の Java データ型を文字列として格納します。
RETURNS_JAVA_DATA_TYPE_CODE には，次の Java データ型の 16 進数の値が格納されます。

Java データ型	16 進数の値
byte[]	1000
byte[][]	100A
short	1002
short[]	1003
int	1004
int[]	1005
float	1006
float[]	1007
double	1008
double[]	1009
java.match.BigDecimal	2000
java.match.BigDecimal[]	2001
java.lang.String	2002
java.lang.String[]	2003
java.sql.Date	2004
java.sql.Date[]	2005
java.sql.Time	2006
java.sql.Time[]	2007
java.lang.Double	2008
java.lang.Double[]	2009
java.lang.Float	200A
java.lang.Float[]	200B

Java データ型	16 進数の値
java.lang.Integer	200C
java.lang.Integer[]	200D
java.lang.Short	200E
java.lang.Short[]	200F
java.sql.Timestamp	2010
java.sql.Timestamp[]	2011
void	0000

注※7

SQL 文の先頭から手続き文開始位置までを，1 から数えた値が設定されます。ただし，外部ルーチン（Java ストアドルーチン）の場合は，SQL 文の先頭から外部ルーチン指定（EXTERNAL NAME 句）の開始位置までの値となります。

なお，次の場合は 0 が設定されます。

- 外部ルーチン（Java ストアドルーチンを除く）
- レジストリ操作用プロシジャ
- トリガ動作手続き

注※8

エンディアンが異なる接続形態でも，SQL の結果はエンディアン変換されません。したがって，アクセスする場合は，アプリケーション側でエンディアンを考慮して変換する必要があります。

注※9

トリガ動作手続きの場合，次のルーチン名称（長さ 22 バイト）が格納されます。

'(TRIGyyyymmddhhmmssth)'

yyyymmddhhmmssth：トリガ定義時のタイムスタンプ（100 分の 1 秒単位）

(16)　SQL_ROUTINE_RESOURCES 表の内容

この表では，ルーチンで使用するリソース情報を管理します（n 行で 1 ルーチン分）。

SQL_ROUTINE_RESOURCES 表の内容を次の表に示します。

表 G–18　SQL_ROUTINE_RESOURCES 表の内容

項番	列　　名	データ型	内　　容
1	ROUTINE_SCHEMA	VARCHAR(30)，又は MVARCHAR(30)	ルーチンの所有者。パブリックルーチンの場合は PUBLIC。
2	ROUTINE_NAME	VARCHAR(30)，又は MVARCHAR(30)	ルーチン名称。

項番	列　　名	データ型	内　　容
3	SPECIFIC_NAME	VARCHAR(30)，又は MVARCHAR(30)	特定名称。※1
4	BASE_OWNER	VARCHAR(30)，又は MVARCHAR(30)	使用資源所有者。パブリックビュー表，又はパブリックルーチンの場合は PUBLIC。
5	BASE_NAME	VARCHAR(30)，又は MVARCHAR(30)	使用資源識別子。
6	BASE_TYPE	CHAR(1)	使用資源種別。 R：永続実表 V：更新できるビュー表 U：読み込み専用のビュー表 I：インデクス（一時インデクス含む） D：データ型 P：ルーチン T：トリガ Q：順序数生成子 G：一時表
7	ROUTINE_TYPE_OWNER	VARCHAR(30)，又は MVARCHAR(30)	抽象データ型内のルーチンの場合，その抽象データ型の所有者。 抽象データ型内で定義したルーチン以外の場合はナル値となります。
8	ROUTINE_TYPE_NAME	VARCHAR(30)，又は MVARCHAR(30)	抽象データ型内のルーチンの場合，その抽象データ型の名称。 抽象データ型内で定義したルーチン以外の場合はナル値となります。
9	SELECT_OPERATION※2	CHAR(1)	検索対象の指定有無。 Y：指定しています。 ナル値：指定していません。 使用資源種別が R，V，及び G 以外の場合はナル値となります。※3
10	INSERT_OPERATION※2	CHAR(1)	データの挿入対象の有無。 Y：指定しています。 ナル値：指定していません。 使用資源種別が R，V，及び G 以外の場合はナル値となります。※3
11	UPDATE_OPERATION※2	CHAR(1)	データの更新対象の有無。 Y：指定しています。 ナル値：指定していません。 使用資源種別が R，V，及び G 以外の場合はナル値となります。※3
12	DELETE_OPERATION※2	CHAR(1)	データの削除対象の有無。

項番	列　　名	データ型	内　　容
			Y：指定しています。 ナル値：指定していません。 使用資源種別が R，V，及び G 以外の場合はナル値となります。※3
13	LOCK_OPERATION※2	CHAR(1)	データの挿入対象の有無。 Y：指定しています。 ナル値：指定していません。 使用資源種別が R，V，及び G 以外の場合はナル値となります。※3
14	PURGE_OPERATION※2	CHAR(1)	PURGE TABLE 文でのデータの削除対象の有無。 Y：指定しています。 ナル値：指定していません。 使用資源種別が R，V，及び G 以外の場合はナル値となります。※3
15	BASE_ROUTINE_CREATOR	VARCHAR(30)，又は MVARCHAR(30)	使用資源がパブリックルーチンの場合の，パブリックルーチン定義者。 使用資源がパブリックルーチン以外の場合はナル値となります。
16	ROUTINE_CREATOR	VARCHAR(30)，又は MVARCHAR(30)	パブリックルーチンの定義者。 パブリックルーチン以外の場合はナル値となります。

注※1

手続きの場合は，ルーチン名称と同じになります。関数の場合は，システムが内部的にルーチン名称とオブジェクト ID から生成した，次のような名称を付けます。

'F' ルーチン名称(最大 19 バイト) オブジェクト ID(10 バイト)

注※2

SQL オブジェクトにビュー表を使用している場合，このビュー表の基表である実表（基表がビュー表の場合も最上位の基表である実表）には，使用しているすべてのビュー表の操作種別をマージした情報が設定されます。

注※3

使用資源種別がビュー表（V）の場合，実際に SQL オブジェクト中に含まれないビュー表についてはナル値が設定されます。

(17)　SQL_ROUTINE_PARAMS 表の内容

この表では，ルーチンのパラメタ情報を管理します（n 行で 1 ルーチン分）。

SQL_ROUTINE_PARAMS 表の内容を次の表に示します。

表 G–19　SQL_ROUTINE_PARAMS 表の内容

項番	列　　名	データ型	内　　容
1	ROUTINE_SCHEMA	VARCHAR(30)，又は MVARCHAR(30)	ルーチンの所有者。パブリックルーチンの場合は PUBLIC。
2	ROUTINE_NAME	VARCHAR(30)，又は MVARCHAR(30)	ルーチン名称。
3	SPECIFIC_NAME	VARCHAR(30)，又は MVARCHAR(30)	特定名称。
4	PARAMETER_NAME	VARCHAR(30)，又は MVARCHAR(30)	パラメタ名称。※4
5	PARAMETER_NO	INTEGER	パラメタ指定順序（該当するルーチン内で 1 から始まるユニークな番号）。
6	DATA_TYPE	CHAR(24)	データ型。 格納形式は，SQL_COLUMNS 表の DATA_TYPE 列を参照してください。
7	DATA_LENGTH	CHAR(7)	データ長。 右詰めで文字形式にして格納します(上位の 0 は空白にします)。 データ型が BLOB，BINARY の場合，又はユーザ定義型の場合はナル値となります。
8	LOB_LENGTH	CHAR(20)	列長の指定値。 右詰めで文字形式にして格納します(上位の 0 は空白にします)。 データ型が BLOB，BINARY 以外の場合はナル値となります。
9	LOB_LENGTH_TYPE	CHAR(1)	列長の種別。 K：K 指定時 M：M 指定時 G：G 指定時 空白：省略時 データ型が BLOB 以外の場合はナル値となります。
10	PARAMETER_MODE	CHAR(5)	パラメタの入出力モード。 IN：入力モード NOUT：出力モード INOUT：入出力モード NONE：上記のどれにも該当しない場合
11	DATA_TYPE_CODE	SMALLINT	データ型コード。 格納形式は，SQL_COLUMNS 表の DATA_TYPE_CODE 列を参照してください。

項番	列　　名	データ型	内　　容
12	DATA_LENGTH_CODE	SMALLINT	データ長コード。※1 データ型が BLOB，BINARY の場合，又はユーザ定義型の場合はナル値となります。
13	LOB_LENGTH_CODE	CHAR(8)	列長の指定値。※2※3 データ型 BLOB，BINARY 以外の場合はナル値となります。
14	UDT_OWNER	VARCHAR(30)，又は MVARCHAR(30)	パラメタのデータ型の所有者。 パラメタがシステム定義型の場合はナル値となります。
15	UDT_NAME	VARCHAR(30)，又は MVARCHAR(30)	パラメタのデータ型の名称。 パラメタがシステム定義型の場合はナル値となります。
16	UDT_TYPE_ID	INTEGER	パラメタのデータ型の ID。 パラメタがシステム定義型の場合はナル値となります。
17	JAVA_DATA_TYPE	VARCHAR(255)	対応する Java パラメタのデータ型。 格納形式については，SQL_ROUTINES 表の RETURNS_JAVA_DATA_TYPE 列を参照してください。 Java で記述した外部ルーチン以外の場合はナル値となります。
18	JAVA_DATA_TYPE_CODE	INTEGER	対応する Java パラメタのデータ型コード。 格納形式については，SQL_ROUTINES 表の RETURNS_JAVA_DATA_TYPE_CODE 列を参照してください。 Java で記述した外部ルーチン以外の場合はナル値となります。
19	MAX_ELM	SMALLINT	パラメタの最大要素数。 パラメタの要素数の指定がない場合はナル値となります。
20	TRIGGER_COLUMN	CHAR(1)	トリガ動作手続きの新旧値相関名で指定した列に対するパラメタ情報。 O：旧値相関名で参照された列 N：新値相関名で参照された列 ナル値：上記以外 トリガ動作手続き以外，新旧値相関名で指定した列に対応するパラメタ以外の場合はナル値となります。
21	TRIGGER_TABLE_ID	INTEGER	パラメタに置き換える前の列を定義している表 ID。

項番	列　　名	データ型	内　　容
			トリガ動作手続き以外，新旧値相関名で指定した列に対応するパラメタ以外の場合はナル値となります。
22	TRIGGER_COLUMN_NAME	VARCHAR(30)，又は MVARCHAR(30)	パラメタに置き換える前の列名。 トリガ動作手続き以外，新旧値相関名で指定した列に対応するパラメタ以外の場合はナル値となります。
23	PARMCSET_SCHEMA	VARCHAR(30)，又は MVARCHAR(30)	データ型の文字集合所有者（常に"MASTER"）。 データ型の文字集合を指定していない場合はナル値となります。
24	PARMCSET_NAME	VARCHAR(30)，又は MVARCHAR(30)	データ型の文字集合名。 EBCDIK：文字集合に EBCDIK を指定した場合 UTF16：文字集合に UTF16 を指定した場合 データ型の文字集合を指定していない場合はナル値となります。
25	PARMCSET_ID	INTEGER	データ型の文字集合 ID。 データ型の文字集合を指定していない場合はナル値となります。 文字集合 ID については，マニュアル「HiRDB システム導入・設計ガイド」を参照してください。

注※1

DECIMAL 型，INTERVAL YEAR TO DAY 型，INTERVAL HOUR TO SECOND 型の場合は，精度，位取りをそれぞれ 1 バイトに格納し，それ以外の場合は，2 バイトの 2 進形式で長さ（NCHAR 型，NVARCHAR 型の場合は文字数）を格納します。ただし，BLOB 型及び抽象データ型の場合は 0 になります。

注※2

列長の指定値を 4 バイトごとに区切られた 8 バイトに 2 進形式で格納します。

注※3

エンディアンが異なる接続形態でも，SQL の結果はエンディアン変換されません。したがって，アクセスする場合は，アプリケーション側でエンディアンを考慮して変換する必要があります。

注※4

トリガ動作手続きの場合，パラメタ名（長さ 27 バイト）は次のようになります。

```
'(T#tbl_id#col_id#nnnnn)'
```

tbl_id：

表 ID（16 進数，8 けた（8 けたに満たない場合は前に 0 を格納））

col_id：

列 ID（16 進数，8 けた（8 けたに満たない場合は前に 0 を格納））

nnnnn：

00001：旧値相関名で修飾した列に対応するパラメタ

00002：新値相関名で修飾した列に対応するパラメタ

(18)　SQL_TABLE_STATISTICS 表の内容

この表では，表の統計情報を管理します（1 行で 1 表分）。

なお，統計情報がない場合（CREATE TABLE 直後など），この表の内容は空となります。

SQL_TABLE_STATISTICS 表の内容を次の表に示します。

表 G–20　SQL_TABLE_STATISTICS 表の内容

項番	列　　名	データ型	内　　容
1	TABLE_SCHEMA	VARCHAR(30)，又は MVARCHAR(30)	表の所有者。
2	TABLE_NAME	VARCHAR(30)，又は MVARCHAR(30)	表の名称。
3	N_PAGE	FLOAT	格納ページ数（統計情報）。 pdgetcst の-c オプションに lvl1 を指定した場合は，ナル値となります。
4	N_ROW	FLOAT	全行数（統計情報）。
5	UPDATE_TIME	CHAR(14)	更新日時（YYYYMMDDHHMMSS）。

(19)　SQL_COLUMN_STATISTICS 表の内容

この表では，列の統計情報を管理します（1 行で 1 列分）。

なお，統計情報がない場合（CREATE TABLE 直後など），この表の内容は空となります。

SQL_COLUMN_STATISTICS 表の内容を次の表に示します。

表 G–21　SQL_COLUMN_STATISTICS 表の内容

項番	列　　名	データ型	内　　容
1	TABLE_SCHEMA	VARCHAR(30)，又は MVARCHAR(30)	表の所有者。
2	TABLE_NAME	VARCHAR(30)，又は MVARCHAR(30)	列を含む表の名称。

項番	列　　名	データ型	内　　容
3	COLUMN_NAME	VARCHAR(30)，又は MVARCHAR(30)	列名。
4	N_UNIQUE	FLOAT	ユニーク値数（統計情報）。
5	N_MAX_DUP_KEY	FLOAT	最大重複キー値数（統計情報）。
6	N_MIN_DUP_KEY	FLOAT	最小重複キー値数（統計情報）。
7	N_NULL	FLOAT	ナル値の数。
8	UPDATE_TIME	CHAR(14)	更新日時（YYYYMMDDHHMMSS）。
9	RANGE_VALUES	VARCHAR(2872)	列値度数分布情報（統計情報）。※

注※

pdgetcst のパラメタファイルで設定した列値の最大値，最小値は RANGE_VALUES 列の中に内部形式に変換して格納しています。このため，最大値，及び最小値を参照するには次に示す SQL を実行する必要があります。ただし，検索結果は 16 進表示となります。

・列値の最大値を検索する SQL

```
SELECT HEX(SUBSTR(”RANGE_VALUES”,33,a))
    FROM ”MASTER”.SQL_COLUMN_STATISTICS
    WITHOUT LOCK NOWAIT
```

a には列のデータ長をバイト単位で指定します。ただし，文字列型データの場合，16 バイトで切り捨てているため，16 以下の値を指定します。

・列値の最小値を検索する SQL

・列のデータ型が DECIMAL 又は NUMERIC 型で，かつ精度が 32 けた以上の場合

```
SELECT HEX(SUBSTR(”RANGE_VALUES”,33+a,a))
    FROM ”MASTER”.SQL_COLUMN_STATISTICS
    WITHOUT LOCK NOWAIT
```

・列のデータ型が上記以外の場合

```
SELECT HEX(SUBSTR(”RANGE_VALUES”),49,a)
    FROM ”MASTER”.SQL_COLUMN_STATISTICS
    WITHOUT LOCK NOWAIT
```

a には列のデータ長をバイト単位で指定します。ただし，文字列型データの場合，16 バイトで切り捨てているため，16 以下の値を指定します。

＜例＞

INT 型の列の列値の最大値を参照する場合

```
SELECT HEX(SUBSTR(”RANGE_VALUES”),33,4)
    FROM ”MASTER”.SQL_COLUMN_STATISTICS
    WITHOUT LOCK NOWAIT
```

出力結果（列値の最大値が 10 の場合）

'0000000A'

(20)　SQL_INDEX_STATISTICS 表の内容

この表では，インデクスの統計情報を管理します（1 行で 1 インデクス分）。

なお，統計情報がない場合（CREATE TABLE 直後など），この表の内容は空となります。

SQL_INDEX_STATISTICS 表の内容を次の表に示します。

表 G–22　SQL_INDEX_STATISTICS 表の内容

項番	列　　名	データ型	内　　容
1	TABLE_SCHEMA	VARCHAR(30)，又は MVARCHAR(30)	表の所有者。
2	TABLE_NAME	VARCHAR(30)，又は MVARCHAR(30)	インデクスを含む表の名称。
3	INDEX_NAME	VARCHAR(30)，又は MVARCHAR(30)	インデクスの名称。
4	N_ENTRY	FLOAT	キーエントリ数（統計情報）。
5	N_IXPG	FLOAT	リーフページ数（統計情報）。
6	N_LEVEL	SMALLINT	レベル数（統計情報）。
7	SEQ_RATIO	INTEGER	シーケンシャル度（統計情報）。
8	UPDATE_TIME	CHAR(14)	更新日時（YYYYMMDDHHMMSS）。

(21)　SQL_DATATYPES 表の内容

この表では，ユーザ定義型の情報を管理します（1 行で 1 ユーザ定義型分）。

SQL_DATATYPES 表の内容を次の表に示します。

表 G–23　SQL_DATATYPES 表の内容

項番	列　　名	データ型	内　　容
1	TYPE_SCHEMA	VARCHAR(30)，又は MVARCHAR(30)	ユーザ定義型の所有者。
2	TYPE_NAME	VARCHAR(30)，又は MVARCHAR(30)	ユーザ定義型の名称。
3	META_TYPE	CHAR(1)	ユーザ定義型の種別。 A：抽象データ型
4	TYPE_ID	INTEGER	ユーザ定義型の ID。

項番	列　　名	データ型	内　　容
5	N_ATTR	SMALLINT	属性の数。
6	CREATE_TIME	CHAR(14)	作成時刻(YYYYMMDDHHMMSS)。
7	N_SUBTYPE	INTEGER	サブタイプの数。
8	SOURCE_SCHEMA	VARCHAR(30)，又は MVARCHAR(30)	スーパタイプの抽象データ型の所有者。 スーパタイプの抽象データ型がない場合はナル値となります。
9	SOURCE_NAME	VARCHAR(30)，又は MVARCHAR(30)	スーパタイプの抽象データ型の名称。 スーパタイプの抽象データ型がない場合はナル値となります。
10	SOURCE_TYPE_ID	INTEGER	スーパタイプの抽象データ型の ID。 スーパタイプの抽象データ型がない場合はナル値となります。
11	ROOT_TYPE_ID	INTEGER	スーパタイプの抽象データ型が，更にスーパタイプを持っている場合，その最上位の抽象データ型の ID。
12	LEVEL_NO	SMALLINT	スーパタイプの抽象データ型が，更にスーパタイプを持っている場合，その最上位の抽象データ型からの世代数。
13	TYPE_COMMENT	VARCHAR(255)	コメント。 初期値，及びコメントがない場合はナル値となります。
14	N_LOB_ATTR	SMALLINT	BLOB 型の属性数。
15	N_ADT_ATTR	SMALLINT	抽象データ型の属性数。
16	N_LARGE_BINARY_ATTR	SMALLINT	32,001 バイト以上の BINARY 型の属性数。

(22)　SQL_DATATYPE_DESCRIPTORS 表の内容

この表では，ユーザ定義型の構成属性の情報を管理します（1 行で 1 属性分）。

SQL_DATATYPE_DESCRIPTORS 表の内容を次の表に示します。

表 G–24　SQL_DATATYPE_DESCRIPTORS 表の内容

項番	列　　名	データ型	内　　容
1	TYPE_SCHEMA	VARCHAR(30)，又は MVARCHAR(30)	ユーザ定義型の所有者。
2	TYPE_NAME	VARCHAR(30)，又は MVARCHAR(30)	ユーザ定義型の名称。
3	OBJECT_NAME	VARCHAR(30)，又は	属性名。

項番	列　　名	データ型	内　　容
		MVARCHAR(30)	
4	TYPE_ID	INTEGER	ユーザ定義型の ID。
5	META_TYPE	CHAR(1)	ユーザ定義型の種別。 S：システム定義型 A：抽象データ型
6	ORDINAL_POSITION	SMALLINT	順序位置。
7	ENCAPSULATION_LEVEL	VARCHAR(10)	隠蔽レベル(PUBLIC，PRIVATE，又は PROTECTED) 。
8	IS_NULLABLE	CHAR(3)	列のナル値情報。 YES：ナル値を許します。 NO：ナル値を許しません。
9	DATA_TYPE	CHAR(24)	データ型。 格納形式は，SQL_COLUMNS 表の DATA_TYPE 列を参照してください。
10	DATA_TYPE_CODE	SMALLINT	データ型コード。 格納形式は，SQL_COLUMNS 表の DATA_TYPE_CODE 列を参照してください。
11	DATA_LENGTH_CODE	SMALLINT	データ長コード。※1
12	DATA_LENGTH	CHAR(7)	データ長。 右詰めで文字形式にして格納します(上位の 0 は空白にします)。
13	LOB_LENGTH_CODE	CHAR(8)	BLOB 属性長コード。※2※3 BLOB 及び BINARY 以外の場合はナル値となります。
14	LOB_LENGTH	CHAR(20)	BLOB 属性長指定値。 右詰めで文字形式にして格納します(上位の 0 は空白にします)。 BLOB 及び BINARY 以外の場合はナル値となります。
15	LOB_LENGTH_TYPE	CHAR(1)	BLOB 属性長の種別(単位)。 K：K 指定時 M：M 指定時 G：G 指定時 空白：省略時 BLOB 以外の場合はナル値となります。
16	UDT_OWNER	VARCHAR(30)，又は MVARCHAR(30)	抽象データ型の属性に更に抽象データ型がある場合，その属性の抽象データ型の所有者。 システム定義型の場合はナル値となります。

項番	列　　名	データ型	内　　容
17	UDT_NAME	VARCHAR(30)，又は MVARCHAR(30)	抽象データ型の属性に更に抽象データ型がある場合，その属性の抽象データ型の名称。 システム定義型の場合はナル値となります。
18	DATA_COMMENT	VARCHAR(255)	コメント。 初期値，及びコメントがない場合はナル値となります。
19	NO_SPLIT	CHAR(1)	NO SPLIT の指定有無。 Y：指定あり ナル値：指定なし

注※1

DECIMAL 型，INTERVAL YEAR TO DAY 型，INTERVAL HOUR TO SECOND 型の場合は，精度，位取りをそれぞれ 1 バイトに格納し，それ以外の場合は，2 バイトの 2 進形式で長さ（NCHAR 型，NVARCHAR 型の場合は文字数）を格納します。ただし，BLOB 型及び抽象データ型の場合は 0 になります。

注※2

列長の指定値を 4 バイトごとに区切られた 8 バイトに 2 進形式で格納します。

注※3

エンディアンが異なる接続形態でも，SQL の結果はエンディアン変換されません。したがって，アクセスする場合は，アプリケーション側でエンディアンを考慮して変換する必要があります。

(23)　SQL_TABLE_RESOURCES 表の内容

この表では，表で使用するリソース情報を管理します（1 行で 1 リソース分）。

SQL_TABLE_RESOURCES 表の内容を次の表に示します。

表 G–25　SQL_TABLE_RESOURCES 表の内容

項番	列　　名	データ型	内　　容
1	TABLE_SCHEMA	VARCHAR(30)，又は MVARCHAR(30)	表の所有者。
2	TABLE_NAME	VARCHAR(30)，又は MVARCHAR(30)	表の名称。
3	BASE_OWNER	VARCHAR(30)，又は MVARCHAR(30)	使用している資源の所有者。
4	BASE_NAME	VARCHAR(30)，又は MVARCHAR(30)	使用している資源の識別子。
5	BASE_TYPE	CHAR(1)	使用している資源の種別。 A：抽象データ型

(24)　SQL_PLUGINS 表の内容

この表では，プラグイン情報を管理します（1 行で 1 プラグイン分）。

SQL_PLUGINS 表の内容を次の表に示します。

表 G–26　SQL_PLUGINS 表の内容

項番	列　　名	データ型	内　　容
1	PLUGIN_SCHEMA	VARCHAR(30)，又は MVARCHAR(30)	プラグイン所有者。
2	PLUGIN_NAME	VARCHAR(30)，又は MVARCHAR(30)	プラグイン名称。
3	PLUGIN_TYPE	CHAR(1)	プラグイン種別。 D：データ型プラグイン I：インデクス型プラグイン
4	TYPE_SCHEMA	VARCHAR(30)，又は MVARCHAR(30)	抽象データ型，又はインデクス型の所有者。
5	TYPE_NAME	VARCHAR(30)，又は MVARCHAR(30)	抽象データ型，又はインデクス型の名称。
6	CREATE_TIME	CHAR(14)	プラグイン組み込み時刻。
7	PLUGIN_LIB_NAME	VARCHAR(255)	ライブラリパス名。
8	PLUGIN_COMMENT	VARCHAR(255)	コメント。 初期値，及びコメントがない場合はナル値となります。
9	PLUGIN_VERSION	VARCHAR(10)	プラグインのバージョン。 プラグインが初期バージョンの場合はナル値となります。
10	PLUGIN_EXT_FUNC	VARCHAR(255)	プラグイン拡張機能コード（システムが使用する情報）。
11	PLUGIN_HOLDABLE	CHAR(1)	ホールダブルカーソルの対応状況。 Y：対応済み ナル値：未対応

(25)　SQL_PLUGIN_ROUTINES 表の内容

この表では，プラグインのルーチン情報を管理します（1 行で 1 プラグインルーチン分）。

SQL_PLUGIN_ROUTINES 表の内容を次の表に示します。

表 G–27　SQL_PLUGIN_ROUTINES 表の内容

項番	列　　名	データ型	内　　容
1	ROUTINE_SCHEMA	VARCHAR(30)，又は MVARCHAR(30)	ルーチンの所有者。
2	PLUGIN_NAME	VARCHAR(30)，又は MVARCHAR(30)	プラグイン名称。
3	OPERATION_NAME	VARCHAR(255)	オペレーション名。
4	SPECIFIC_NAME	VARCHAR(30)，又は MVARCHAR(30)	特定名称。※
5	N_PARAM	INTEGER	パラメタ数。
6	TIMING_DESCRIPTOR	VARCHAR(30)	契機指示子。
7	OPERATION_DESCRIPTOR	VARCHAR(255)	オペレーション修飾情報。

注※　次の形式で名称を付けます。

'P' 関数名 登録日時

P：プラグイン実装の関数であることを示すコードです。

関数名：特定名称が 30 文字以下になるように前部の文字を切り捨てます(最大 15 文字)。

登録日時：年月日時分秒を 14 文字で表します。

(26)　SQL_PLUGIN_ROUTINE_PARAMS 表の内容

この表では，プラグインのルーチンのパラメタ情報を管理します（1 行で 1 パラメタ分）。

SQL_PLUGIN_ROUTINE_PARAMS 表の内容を次の表に示します。

表 G–28　SQL_PLUGIN_ROUTINE_PARAMS 表の内容

項番	列　　名	データ型	内　　容
1	ROUTINE_SCHEMA	VARCHAR(30)，又は MVARCHAR(30)	所有者。
2	PLUGIN_NAME	VARCHAR(30)，又は MVARCHAR(30)	プラグイン名称。
3	SPECIFIC_NAME	VARCHAR(30)，又は MVARCHAR(30)	特定名称。
4	PARAMETER_NAME	VARCHAR(30)，又は MVARCHAR(30)	パラメタ名称。
5	PARAMETER_MODE	CHAR(7)	パラメタ入出力属性。 IN：入力属性

項番	列　　名	データ型	内　　容
			OUT：出力属性 INOUT：入出力属性 RETURNS：戻り値属性 PICKUP：行識別子（ROWID）出力属性
6	PARAMETER_DESCRIPTOR	VARCHAR(255)	パラメタ修飾情報。 プラグイン IDL で記述されているパラメタ修飾情報を，そのまま文字列として保持します。パラメタ修飾情報の指定がない場合はナル値となります。
7	SPECIFIC_BIND_OPERATION_NAME	VARCHAR(30)，又は MVARCHAR(30)	特化済みバインドオペレーション名。 バインドオペレーション指定がない場合はナル値となります。
8	PARAMETER_TYPE	CHAR(1)	パラメタモード。 空白：normal(SQL で扱えるデータ型) I：indicator N：new data C：current data D：dbifb K：index key inf P：pointer R：rowid U：utlifb T：pointer* normal 以外はプラグイン固有のパラメタです。
9	PARAMETER_NO	INTEGER	抽象データ型関数のパラメタ指定順序位置。
10	DATA_TYPE	CHAR(24)	パラメタのデータ型。 格納形式は，SQL_COLUMNS 表の DATA_TYPE 列を参照してください。 パラメタモードが D，K，P，R，U，又は T の場合はナル値となります。
11	DATA_TYPE_CODE	SMALLINT	パラメタのデータ型コード。 パラメタモードが D，K，P，R，U，又は T の場合はナル値となります。 格納形式は，SQL_COLUMNS 表の DATA_TYPE_CODE 列を参照してください。
12	DATA_LENGTH_CODE	SMALLINT	パラメタのデータ型定議長コード。※1 パラメタモードが D，K，P，R，U，又は T の場合はナル値となります。
13	DATA_LENGTH	CHAR(7)	パラメタのデータ型定議長。

項番	列　　名	データ型	内　　容
			右詰めで文字形式にして格納します(上位の 0 は空白にします)。パラパラメタモードが D，K，P，R，U，又は T の場合はナル値となります。
14	LOB_LENGTH_CODE	CHAR(8)	LOB 列長コード又は BINARY 列長コード。※2※3 パラメタモードが normal で，かつデータ型が BLOB 及び BINARY 以外の場合はナル値となります。
15	LOB_LENGTH	CHAR(20)	LOB 列長指定値又は BINARY 列長指定値。 右詰めで文字形式にして格納します(上位の 0 は空白にします)。パラメタモードが normal で，かつデータ型が BLOB 及び BINARY 以外の場合はナル値となります。
16	LOB_LENGTH_TYPE	CHAR(1)	LOB 列長の種別(単位)。 K：K 指定時 M：M 指定時 G：G 指定時 空白：省略時 パラメタモードが normal で，かつデータ型が BLOB 以外の場合はナル値となります。
17	UDT_OWNER	VARCHAR(30)，又は MVARCHAR(30)	パラメタデータ型の所有者。 データ型がユーザ定義型以外の場合はナル値となります。
18	UDT_NAME	VARCHAR(30)，又は MVARCHAR(30)	パラメタデータ型の名称。 データ型がユーザ定義型以外の場合はナル値となります。
19	UDT_TYPE_ID	INTEGER	パラメタデータ型の ID。 データ型がユーザ定義型以外の場合はナル値となります。

注※1

DECIMAL 型，INTERVAL YEAR TO DAY 型，INTERVAL HOUR TO SECOND 型の場合は，精度，位取りをそれぞれ 1 バイトに格納し，それ以外の場合は，2 バイトの 2 進形式で長さ（NCHAR 型，NVARCHAR 型の場合は文字数）を格納します。ただし，BLOB 型及び抽象データ型の場合は 0 になります。

注※2

列長の指定値を 4 バイトごとに区切られた 8 バイトに 2 進形式で格納します。

注※3

エンディアンが異なる接続形態でも，SQL の結果はエンディアン変換されません。したがって，アクセスする場合は，アプリケーション側でエンディアンを考慮して変換する必要があります。

(27)　SQL_INDEX_TYPES 表の内容

この表では，インデクス型情報を管理します（1 行で 1 インデクス型分）。

SQL_INDEX_TYPES 表の内容を次の表に示します。

表 G–29　SQL_INDEX_TYPES 表の内容

項番	列　　名	データ型	内　　容
1	INDEX_TYPE_SCHEMA	VARCHAR(30)，又は MVARCHAR(30)	インデクス型の所有者。
2	INDEX_TYPE_NAME	VARCHAR(30)，又は MVARCHAR(30)	インデクス型の名称。
3	INDEX_TYPE_ID	INTEGER	インデクス型の ID。
4	CREATE_TIME	CHAR(14)	作成時刻。
5	ADT_OWNER	VARCHAR(30)，又は MVARCHAR(30)	抽象データ型の所有者。
6	ADT_NAME	VARCHAR(30)，又は MVARCHAR(30)	抽象データ型の名称。
7	N_FUNCTION	INTEGER	インデクス型で定義したインデクスで利用できる抽象データ型関数の数。

(28)　SQL_INDEX_RESOURCES 表の内容

この表では，インデクスで使用するリソース情報を管理します（1 行で 1 リソース分）。

SQL_INDEX_RESOURCES 表の内容を次の表に示します。

表 G–30　SQL_INDEX_RESOURCES 表の内容

項番	列　　名	データ型	内　　容
1	TABLE_SCHEMA	VARCHAR(30)，又は MVARCHAR(30)	インデクス定義表の所有者。
2	INDEX_NAME	VARCHAR(30)，又は MVARCHAR(30)	インデクスの名称。
3	BASE_OWNER	VARCHAR(30)，又は MVARCHAR(30)	使用している資源の所有者。
4	BASE_NAME	VARCHAR(30)，又は MVARCHAR(30)	使用している資源の識別子。
5	BASE_TYPE	CHAR(1)	使用している資源の種別。 I：インデクス型

(29)　SQL_INDEX_DATATYPE 表の内容

この表では，インデクスの対象項目情報を管理します（1 行で 1 対象項目分（1 段分））。

SQL_INDEX_DATATYPE 表の内容を次の表に示します。

表 G–31　SQL_INDEX_DATATYPE 表の内容

項番	列　　名	データ型	内　　容
1	TABLE_SCHEMA	VARCHAR(30)，又は MVARCHAR(30)	表の所有者。
2	TABLE_NAME	VARCHAR(30)，又は MVARCHAR(30)	インデクスを含む表の名称。
3	INDEX_NAME	VARCHAR(30)，又は MVARCHAR(30)	インデクスの名称。
4	INDEX_ID	INTEGER	インデクス ID。
5	COLUMN_NAME	VARCHAR(30)，又は MVARCHAR(30)	列名(インデクス構成列名) 。
6	N_LEVEL	SMALLINT	レベル数(抽象データ型を構成する属性名の順序を識別する番号。1 から始まる整数) 。
7	ADT_OWNER	VARCHAR(30)，又は MVARCHAR(30)	抽象データ型の所有者。
8	ADT_NAME	VARCHAR(30)，又は MVARCHAR(30)	抽象データ型の名称。
9	ADT_ATTR_NAME	VARCHAR(30)，又は MVARCHAR(30)	抽象データ型の属性名。
10	ADT_ATTR_ID	SMALLINT	属性の位置。

(30)　SQL_INDEX_FUNCTION 表の内容

この表では，インデクスで利用する抽象データ型関数の情報を管理します（1 行で一つの抽象データ型関数分）。

SQL_INDEX_FUNCTION 表の内容を次の表に示します。

表 G–32　SQL_INDEX_FUNCTION 表の内容

項番	列　　名	データ型	内　　容
1	TABLE_SCHEMA	VARCHAR(30)，又は MVARCHAR(30)	表の所有者。
2	TABLE_NAME	VARCHAR(30)，又は	インデクスを含む表の名称。

項番	列　　名	データ型	内　　容
		MVARCHAR(30)	
3	INDEX_NAME	VARCHAR(30)，又は MVARCHAR(30)	インデクスの名称。
4	INDEX_ID	INTEGER	インデクス ID。
5	COLUMN_NAME	VARCHAR(30)，又は MVARCHAR(30)	列名称(インデクス構成列名)。
6	ADT_OWNER	VARCHAR(30)，又は MVARCHAR(30)	抽象データ型関数の所有者名。
7	ADT_FUNCTION_NAME	VARCHAR(30)，又は MVARCHAR(30)	抽象データ型関数の名称(ルーチン名称) 。
8	ADT_FUNCTION_OBJECT_ID	INTEGER	抽象データ型関数のオブジェクト ID。

(31)　SQL_TYPE_RESOURCES 表の内容

この表では，ユーザ定義型で使用するリソース情報を管理します（1 行で 1 リソース分)。

SQL_TYPE_RESOURCES 表の内容を次の表に示します。

表 G–33　SQL_TYPE_RESOURCES 表の内容

項番	列　　名	データ型	内　　容
1	TYPE_SCHEMA	VARCHAR(30)，又は MVARCHAR(30)	ユーザ定義型の所有者。
2	TYPE_NAME	VARCHAR(30)，又は MVARCHAR(30)	ユーザ定義型の名称。
3	BASE_OWNER	VARCHAR(30)，又は MVARCHAR(30)	使用している資源の所有者。
4	BASE_NAME	VARCHAR(30)，又は MVARCHAR(30)	使用している資源の識別子。
5	BASE_TYPE	CHAR(1)	使用している資源の識別子。 A：抽象データ型

(32)　SQL_INDEX_TYPE_FUNCTION 表の内容

この表では，インデクス型を定義したインデクスで利用できる抽象データ型関数の情報を管理します（n 行で 1 インデクス型分)。

SQL_INDEX_TYPE_FUNCTION 表の内容を次の表に示します。

表 G–34　SQL_INDEX_TYPE_FUNCTION 表の内容

項番	列　　名	データ型	内　　容
1	INDEX_TYPE_SCHEMA	VARCHAR(30)，又は MVARCHAR(30)	インデクス型の所有者。
2	INDEX_TYPE_NAME	VARCHAR(30)，又は MVARCHAR(30)	インデクス型の名称。
3	ADT_OWNER	VARCHAR(30)，又は MVARCHAR(30)	抽象データ型関数の所有者。
4	ADT_FUNCTION_NAME	VARCHAR(30)，又は MVARCHAR(30)	抽象データ型関数の識別子。※
5	ADT_FUNCTION_OBJECT_ID	INTEGER	抽象データ型関数のオブジェクト ID。

注※ 特定名称ではありません。

(33)　SQL_EXCEPT 表の内容

この表では，インデクスの除外キー値の情報を管理します（n 行で 1 インデクスの除外キー群分）。また，1 行では 1 インデクスの一つの除外値（複数列インデクスの場合は，除外値群）を管理します。

SQL_EXCEPT 表の内容を次の表に示します。

表 G–35　SQL_EXCEPT 表の内容

項番	列　　名	データ型	内　　容
1	TABLE_SCHEMA	VARCHAR(30)，又は MVARCHAR(30)	インデクスの所有者。
2	TABLE_NAME	VARCHAR(30)，又は MVARCHAR(30)	インデクスがある表の名称。
3	INDEX_NAME	VARCHAR(30)，又は MVARCHAR(30)	インデクスの名称。
4	INDEX_ID	INTEGER	インデクス ID。
5	TABLE_ID	INTEGER	表 ID。
6	EXCEPT_VALUE	VARCHAR(573)，又は MVARCHAR(573)	除外キー値の内容。 構成列ごとの指定値を，文字形式でコンマで区切ります。 初期値はナル値となります。

(34)　SQL_IOS_GENERATIONS 表の内容

この表では，インナレプリカ機能使用時の HiRDB ファイルシステム領域の世代情報を管理します（1 行で 1HiRDB ファイルシステム領域分）。

HiRDB Staticizer Option を組み込んでいない場合，この表の内容は空となります。ただし，HiRDB Staticizer Option を組み込んでデータベースを構築した後に，HiRDB Staticizer Option を取り外した場合は，表の中のデータは残った状態となります。

Windows 版の場合は，SQL_IOS_GENERATIONS 表の内容は空となります。

SQL_IOS_GENERATIONS 表の内容を次の表に示します。

表 G–36　SQL_IOS_GENERATIONS 表の内容

項番	列　　名	データ型	内　　容
1	FILE_SYSTEM_NAME	VARCHAR(165)	HiRDB ファイルシステム領域名（絶対パス名）。
2	GENERATION_NUMBER	SMALLINT	世代番号。
3	SERVER_NAME	CHAR(8)	サーバ名（BES 又は SDS）。※
4	ORIGINAL_FILE_SYSTEM_NAME	VARCHAR(165)	オリジナル HiRDB ファイルシステム領域名（絶対パス名）。

注※

HiRDB/パラレルサーバのディクショナリ表を，そのまま HiRDB/シングルサーバで使用する場合でも，サーバ名は変更されません。

左詰めで 8 文字に満たない場合，残りの部分には空白が入ります。

(35)　SQL_TRIGGERS 表の内容

この表では，スキーマ内にあるトリガの情報を管理します（1 行で 1 トリガ分）。

SQL_TRIGGERS 表の内容を次の表に示します。

表 G–37　SQL_TRIGGERS 表の内容

項番	列　　名	データ型	内　　容
1	TRIGGER_SCHEMA	VARCHAR(30)，又は MVARCHAR(30)	トリガの所有者。
2	TRIGGER_NAME	VARCHAR(30)，又は MVARCHAR(30)	トリガ名称。
3	OBJECT_ID	INTEGER	オブジェクト ID。
4	TABLE_SCHEMA	VARCHAR(30)，又は MVARCHAR(30)	トリガを定義した表の所有者。
5	TABLE_NAME	VARCHAR(30)，又は MVARCHAR(30)	トリガを定義した表の名称。
6	TRIGGER_VALID	CHAR(1)	トリガ有効フラグ。

項番	列　　名	データ型	内　　容
			Y：有効 N：無効 トリガ動作手続きの SQL_ROUTINES 表の ROUTINE_VALID 列と同じ値です。
7	INDEX_VALID	CHAR(1)	インデクス有効フラグ。 Y：有効 N：無効 トリガ動作手続きの SQL_ROUTINES 表の INDEX_VALID 列と同じ値です。
8	ACTION_TIME	CHAR(1)	トリガ動作時期。 A：AFTER B：BEFORE
9	EVENT	CHAR(1)	トリガ契機種別。 I：INSERT D：DELETE U：UPDATE
10	ACTION_TYPE	CHAR(1)	トリガ動作単位。 R：ROW S：STATEMENT
11	OLD_ROW_NAME	VARCHAR(30)，又は MVARCHAR(30)	旧値相関名（OLD ROW に指定した相関名）。 OLD ROW を指定していない場合はナル値となります。
12	NEW_ROW_NAME	VARCHAR(30)，又は MVARCHAR(30)	新値相関名（NEW ROW に指定した相関名）。 NEW ROW を指定していない場合はナル値となります。
13	CREATE_TIME	VARCHAR(16)	トリガ定義の作成時刻。
14	ALTER_TIME	CHAR(14)	トリガの SQL オブジェクトの再作成時刻。 トリガ動作手続きの SQL_ROUTINES 表の ALTER_TIME 列と同じ値です。 トリガの SQL オブジェクトを再作成していない場合はナル値となります。
15	DEF_SOURCE_LEN	INTEGER	トリガ定義のソース長。
16	SPECIFIC_NAME	VARCHAR(30)，又は MVARCHAR(30)	トリガ動作手続きの特定名。
17	N_UPDATE_COLUMNS	SMALLINT	トリガ契機列の数。 INSERT トリガ，DELETE トリガ，及びトリガ契機列を指定していない UPDATE トリガの場合は 0 となります。

項番	列　　名	データ型	内　　容
18	REFERENCING_TABLE_ID	INTEGER	参照表の表 ID。 参照制約動作で作成されたトリガ以外の場合はナル値となります。
19	REFERENCE_ACTION	CHAR(2)	参照制約動作種別。 DC：ON DELETE CASCADE UC：ON UPDATE CASCADE 参照制約動作で作成されたトリガ以外の場合はナル値となります。
20	CONSTRAINT_NAME	VARCHAR(30)，又は MVARCHAR(30)	参照制約の制約名。 参照制約動作で作成されたトリガ以外の場合はナル値となります。

(36)　SQL_TRIGGER_COLUMNS 表の内容

この表では，UPDATE トリガの契機列のリスト情報を管理します（1 行で 1 契機列分）。

SQL_TRIGGER_COLUMNS 表の内容を次の表に示します。

表 G–38　SQL_TRIGGER_COLUMNS 表の内容

項番	列　　名	データ型	内　　容
1	TRIGGER_SCHEMA	VARCHAR(30)，又は MVARCHAR(30)	トリガの所有者。
2	TRIGGER_NAME	VARCHAR(30)，又は MVARCHAR(30)	トリガの名称。
3	TABLE_SCHEMA	VARCHAR(30)，又は MVARCHAR(30)	トリガを定義した表の所有者。
4	TABLE_NAME	VARCHAR(30)，又は MVARCHAR(30)	トリガを定義した表の名称。
5	COLUMN_NAME	VARCHAR(30)，又は MVARCHAR(30)	列リストに指定した列名。
6	TABLE_ID	INTEGER	トリガを定義した表の ID。

(37)　SQL_TRIGGER_DEF_SOURCE 表の内容

この表では，トリガ定義のソース情報を管理します（1 行で 1 トリガ定義ソース分）。

SQL_TRIGGER_DEF_SOURCE 表の内容を次の表に示します。

表 G–39　SQL_TRIGGER_DEF_SOURCE 表の内容

項番	列　　名	データ型	内　　容
1	TRIGGER_SCHEMA	VARCHAR(30)，又は MVARCHAR(30)	トリガの所有者。
2	TRIGGER_NAME	VARCHAR(30)，又は MVARCHAR(30)	トリガの名称。
3	TABLE_SCHEMA	VARCHAR(30)，又は MVARCHAR(30)	トリガを定義した表の所有者。
4	TABLE_NAME	VARCHAR(30)，又は MVARCHAR(30)	トリガを定義した表の名称。
5	SOURCE_NO	INTEGER	定義ソース通し番号。
6	DEF_SOURCE	VARCHAR(32000)，又は MVARCHAR(32000)	定義ソース（SQL コンパイルオプション，WITH PROGRAM は含みません）。

(38)　SQL_TRIGGER_USAGE 表の内容

この表では，トリガ動作条件中で参照している資源情報を管理します（1 行でトリガ動作条件中で参照している 1 資源名称分）。

SQL_TRIGGER_USAGE 表の内容を次の表に示します。

表 G–40　SQL_TRIGGER_USAGE 表の内容

項番	列　　名	データ型	内　　容
1	TRIGGER_SCHEMA	VARCHAR(30)，又は MVARCHAR(30)	トリガの所有者。
2	TRIGGER_NAME	VARCHAR(30)，又は MVARCHAR(30)	トリガの名称。
3	TABLE_SCHEMA	VARCHAR(30)，又は MVARCHAR(30)	トリガを定義した表の所有者。
4	TABLE_NAME	VARCHAR(30)，又は MVARCHAR(30)	トリガを定義した表の名称。
5	BASE_SCHEMA	VARCHAR(30)，又は MVARCHAR(30)	使用資源の所有者。パブリックルーチンの場合は PUBLIC。
6	BASE_TABLE	VARCHAR(30)，又は MVARCHAR(30)	使用資源の表名。 使用資源種別が F（関数）の場合はナル値となります。
7	BASE_NAME	VARCHAR(30)，又は	使用資源名称（特定名又は列名）。

項番	列　　名	データ型	内　　容
		MVARCHAR(30)	
8	BASE_TYPE	CHAR(1)	使用資源種別。 F：関数 C：列名
9	TABLE_ID	INTEGER	表 ID。 使用資源種別が F（関数）の場合はナル値となります。
10	BASE_ID	INTEGER	使用資源 ID（オブジェクト ID 又は列 ID）。
11	BASE_ROUTINE_CREATOR	VARCHAR(30)，又は MVARCHAR(30)	使用資源がパブリック関数の場合の，パブリック関数定義者。 使用資源がパブリック関数以外の場合はナル値となります。

(39)　SQL_PARTKEY 表の内容

この表では，マトリクス分割表の分割キーの情報を管理します（1 行で 1 分割キー分）。

HiRDB Advanced High Availability を組み込んでいない場合，この表の内容は空となります。ただし，HiRDB Advanced High Availability を組み込んでデータベースを構築した後に，HiRDB Advanced High Availability を取り外した場合は，表の中のデータは残った状態となります。

SQL_PARTKEY 表の内容を次の表に示します。

表 G–41　SQL_PARTKEY 表の内容

項番	列　　名	データ型	内　　容
1	TABLE_SCHEMA	VARCHAR(30)，又は MVARCHAR(30)	表の所有者。
2	TABLE_NAME	VARCHAR(30)，又は MVARCHAR(30)	表の名称。
3	KEY_NO	SMALLINT	分割キー番号（次元番号 1 又は 2）。
4	KEY_NAME	VARCHAR(30)，又は MVARCHAR(30)	分割キー列名。
5	COLUMN_ID	SMALLINT	分割キー列 ID。
6	N_DIVISION	SMALLINT	キー内分割数。
7	HASH_KEY_NO	SMALLINT	ハッシュキー列内通番。 境界値分割の次元の場合，ナル値となります。

(40)　SQL_PARTKEY_DIVISION 表の内容

この表では，マトリクス分割表の分割条件値の情報を管理します（1 行で 1 分割条件値分）。

HiRDB Advanced High Availability を組み込んでいない場合，この表の内容は空となります。ただし，HiRDB Advanced High Availability を組み込んでデータベースを構築した後に，HiRDB Advanced High Availability を取り外した場合は，表の中のデータは残った状態となります。

SQL_PARTKEY_DIVISION 表の内容を次の表に示します。

表 G–42　SQL_PARTKEY_DIVISION 表の内容

項番	列　　名	データ型	内　　容
1	TABLE_SCHEMA	VARCHAR(30)，又は MVARCHAR(30)	表の所有者。
2	TABLE_NAME	VARCHAR(30)，又は MVARCHAR(30)	表の名称。
3	KEY_NO	SMALLINT	分割キー番号（次元番号 1 又は 2）。
4	IN_DIM_NO	SMALLINT	分割キー内通番。
5	DCVALUES	VARCHAR(255)，又は MVARCHAR(255)	分割条件値（指定した分割条件値が文字形式で格納されます）。 分割キーに文字集合の指定がある場合でも，格納される値は変換されません。※1，※2 分割キー内の最後の境界値の場合，ハッシュ分割の次元の場合，ナル値となります。
6	DCVALUES_TYPE	CHAR(1)	分割条件値（DCVALUES 列の値）の指定形式。 X：16 進文字列定数 DCVALUES 列の値がナル値の場合，及び分割条件に指定する定数が 16 進文字列定数以外の場合はナル値となります。

注※1

分割条件値を 16 進文字列定数で指定した場合，指定した 16 進文字列定数の値がそのまま格納されます。

分割条件値に指定した 16 進文字列を確認する場合，スカラ関数 HEX を使用して 16 進文字列表現に変換して確認してください。16 進文字列表現に変換しないで参照すると，16 進文字列のコードに対応する文字が存在しない場合，文字化けが発生することがあります。

注※2

分割キーのデータ型が CHAR 又は MCHAR 型で分割条件値を 16 進文字列定数で指定した場合，分割条件値長（16 進文字数÷2）が分割キーの定義長に満たないとき，分割キーの定義長に満たない部分は半角スペースで埋めないで分割条件値長分格納されます。なお，16 進文字列定数以外で指定した場合は，定義長に満たない部分は半角スペースで埋めて格納されます。

(41)　SQL_AUDITS 表の内容

この表では，監査対象の情報を管理します（1 行で 1 オブジェクト又は 1 ユーザに対する 1 イベント分）。なお，この表は，監査人だけが参照できます。

SQL_AUDITS 表の内容を次の表に示します。

表 G–43　SQL_AUDITS 表の内容

項番	列　　名	データ型	内　　容
1	EVENT_TYPE	VARCHAR(30)	CREATE AUDIT FOR 操作種別で指定したイベントタイプの名称※1又は'ANY'。
2	EVENT_SUBTYPE	VARCHAR(30)	イベントサブタイプの名称※2又は'ANY'。 CREATE AUDIT FOR ANY を指定した場合はナル値となります。
3	OBJECT_TYPE	VARCHAR(30)	CREATE AUDIT 選択オプションで指定したオブジェクトの種別。※3 オブジェクトを指定していない場合，及び HiRDB のバージョンが 07-03 より前の場合はナル値となります。
4	OBJECT_SCHEMA	VARCHAR(30)，又は MVARCHAR(30)	CREATE AUDIT 選択オプションで指定したオブジェクトの所有者。 オブジェクトを指定していない場合，及び HiRDB のバージョンが 07-03 より前の場合はナル値となります。
5	OBJECT_NAME	VARCHAR(30)，又は MVARCHAR(30)	CREATE AUDIT 選択オプションで指定したオブジェクトの名称。 オブジェクトを指定していない場合，及び HiRDB のバージョンが 07-03 より前の場合はナル値となります。
6	USER_NAME	VARCHAR(30)，又は MVARCHAR(30)	イベント実行者の認可識別子。イベント実行者を指定していない場合，及び HiRDB のバージョンが 09-03-02 以前の場合はナル値となります。
7	ANY_VALID	CHAR(1)	CREATE AUDIT WHENEVER ANY の指定有無。 Y：指定あり N：指定なし
8	SUCCESSFUL_VALID	CHAR(1)	CREATE AUDIT WHENEVER SUCCESSFUL の指定有無。 Y：指定あり N：指定なし
9	UNSUCCESSFUL_ANY_VALID	CHAR(1)	CREATE AUDIT WHENEVER UNSUCCESSFUL の指定有無。

項番	列　　名	データ型	内　　容
			Y：指定あり N：指定なし
10	AUDIT_TYPE	CHAR(1)	取得情報種別。 E：CREATE AUDIT AUDITTYPE EVENT 指定の場合 A：CREATE AUDIT AUDITTYPE ANY 指定の場合 ただし，CREATE AUDIT AUDITTYPE PRIVILEGE 指定の場合，及び AUDITTYPE を省略している場合，ナル値となります。

注※1

イベントタイプを次に示します。

SESSION，PRIVILEGE，DEFINITION，ACCESS，及び UTILITY

注※2

イベントサブタイプを次に示します。

CONNECT，AUTHORIZATION，DISCONNECT，ROLE，GRANT，REVOKE，CREATE，DROP，ALTER，SELECT，INSERT，UPDATE，DELETE，PURGE，CALL，ASSIGN LIST，LOCK，NEXT VALUE，PDLOAD，PDRORG，PDEXP，及び PDCONSTCK

注※3

オブジェクトの種別を次に示します。

FUNCTION，INDEX，PROCEDURE，SCHEMA，SERVER，TABLE，TRIGGER，DATA TYPE，VIEW，LIST，COMMENT，ROLE，及び SEQUENCE

(42)　SQL_REFERENTIAL_CONSTRAINTS 表の内容

この表では，参照制約の対応状況を管理します（1 行で 1 制約分）。

SQL_REFERENTIAL_CONSTRAINTS 表の内容を次の表に示します。

表 G–44　SQL_REFERENTIAL_CONSTRAINTS 表の内容

項番	列　　名	データ型	内　　容
1	CONSTRAINT_NAME	VARCHAR(30)，又は MVARCHAR(30)	制約名。
2	CONSTRAINT_SCHEMA	VARCHAR(30)，又は MVARCHAR(30)	制約の所有者。
3	TABLE_NAME	VARCHAR(30)，又は MVARCHAR(30)	制約を定義した表の名称。
4	TABLE_SCHEMA	VARCHAR(30)，又は	制約を定義した表の所有者。

項番	列　　名	データ型	内　　容
		MVARCHAR(30)	
5	COLUMN_COUNT	SMALLINT	外部キーの構成列数。
6	COLUMN_NAME	VARCHAR(2111)，又は MVARCHAR(2111)	外部キーがある表の列名。 各列を引用符で囲んで，コンマで連結します。
7	COLUMN_NO	VARCHAR(128)	外部キーがある表の列 ID（64 個分）※。
8	R_OWNER	VARCHAR(30)，又は MVARCHAR(30)	参照する表の所有者。
9	R_TABLE_NAME	VARCHAR(30)，又は MVARCHAR(30)	参照する表の名称。
10	DELETE_RULE	CHAR(11)	削除規則（RESTRICT，又は CASCADE）。
11	UPDATE_RULE	CHAR(11)	更新規則（RESTRICT，又は CASCADE）。
12	CONSTRAINT_TIME	CHAR(14)	制約を定義した日時 （YYYYMMDDHHMMSS）。
13	CHECK_PEND	CHAR(1)	検査保留状態の種別。 C：保留 ナル値：非保留
14	DELETE_TRIGGER_NAME	VARCHAR(30)，又は MVARCHAR(30)	ON DELETE 参照制約動作で作成されたトリガの名称（DRAYYYYMMDDHHMMSSth）。 ON DELETE 参照制約動作でトリガが作成されていない場合はナル値となります。
15	UPDATE_TRIGGER_NAME	VARCHAR(30)，又は MVARCHAR(30)	ON UPDATE 参照制約動作で作成されたトリガの名称（URAYYYYMMDDHHMMSSth）。 ON UPDATE 参照制約動作でトリガが作成されていない場合はナル値となります。
16	R_COLUMN_NAME	VARCHAR(2111)，又は MVARCHAR(2111)	主キー構成列の列名。 各列を引用符で囲んで，コンマで連結します。
17	R_COLUMN_NO	VARCHAR(128)	主キー構成列の列 ID（64 個分）※。

注※

エンディアンが異なる接続形態でも，SQL の結果はエンディアン変換されません。したがって，アクセスする場合は，アプリケーション側でエンディアンを考慮して変換する必要があります。

(43)　SQL_KEYCOLUMN_USAGE 表の内容

この表では，外部キーを構成する列情報を管理します（1 行で 1 列分）。

SQL_KEYCOLUMN_USAGE 表の内容を次の表に示します。

表 G–45　SQL_KEYCOLUMN_USAGE 表の内容

項番	列　　名	データ型	内　　容
1	CONSTRAINT_SCHEMA	VARCHAR(30)，又は MVARCHAR(30)	制約の所有者。
2	CONSTRAINT_NAME	VARCHAR(30)，又は MVARCHAR(30)	制約名。
3	TABLE_SCHEMA	VARCHAR(30)，又は MVARCHAR(30)	制約を定義した表の所有者。
4	TABLE_NAME	VARCHAR(30)，又は MVARCHAR(30)	制約を定義した表の名称。
5	COLUMN_NAME	VVARCHAR(30)，又は MVARCHAR(30)	制約を定義した列名。
6	COLUMN_ORDER	SMALLINT	制約を定義した列の位置。

(44)　SQL_TABLE_CONSTRAINTS 表の内容

この表では，スキーマ内にある整合性制約の情報を管理します（1 行で 1 整合性制約分）。

SQL_TABLE_CONSTRAINTS 表の内容を次の表に示します。

表 G–46　SQL_TABLE_CONSTRAINTS 表の内容

項番	列　　名	データ型	内　　容
1	CONSTRAINT_SCHEMA	VARCHAR(30)，又は MVARCHAR(30)	制約の所有者。
2	CONSTRAINT_NAME	VARCHAR(30)，又は MVARCHAR(30)	制約名。
3	CONSTRAINT_TYPE	VARCHAR(30)	制約の種類。 FOREIGN KEY：外部キー CHECK：検査制約
4	TABLE_SCHEMA	VARCHAR(30)，又は MVARCHAR(30)	制約を定義した表の所有者。
5	TABLE_NAME	VARCHAR(30)，又は MVARCHAR(30)	制約を定義した表の名称。

(45)　SQL_CHECKS 表の内容

この表では，検査制約の情報を管理します（1 行で 1 検査制約分）。

SQL_CHECKS 表の内容を次の表に示します。

表 G–47　SQL_CHECKS 表の内容

項番	列　　名	データ型	内　　容
1	CONSTRAINT_SCHEMA	VARCHAR(30)，又は MVARCHAR(30)	検査制約の所有者。
2	CONSTRAINT_NAME	VARCHAR(30)，又は MVARCHAR(30)	制約名。
3	TABLE_SCHEMA	VARCHAR(30)，又は MVARCHAR(30)	制約を定義した表の所有者。
4	TABLE_NAME	VARCHAR(30)，又は MVARCHAR(30)	制約を定義した表の名称。
5	CHK_SOURCE_LEN	INTEGER	検査制約の探索条件の長さ。
6	CHK_SOURCE	BINARY(2000000)	検査制約の探索条件。
7	CREATE_TIME	CHAR(14)	検査制約を定義した時刻（YYYYMMDDHHMMSS）。
8	CHECK_PEND2	CHAR(1)	検査保留状態の種別。 C：保留 ナル値：非保留
9	N_CHK_COLUMN	INTEGER	検査制約定義中に指定した制約の列数（重複排除した数）。

(46)　SQL_CHECK_COLUMNS 表の内容

この表では，検査制約で使用している列の情報を管理します（1 行で一つの検査制約で使用している 1 列分）。

SQL_CHECK_COLUMNS 表の内容を次の表に示します。

表 G–48　SQL_CHECK_COLUMNS 表の内容

項番	列　　名	データ型	内　　容
1	CONSTRAINT_SCHEMA	VARCHAR(30)，又は MVARCHAR(30)	検査制約の所有者。
2	CONSTRAINT_NAME	VARCHAR(30)，又は MVARCHAR(30)	制約名。
3	TABLE_SCHEMA	VARCHAR(30)，又は MVARCHAR(30)	制約を定義した表の所有者。
4	TABLE_NAME	VARCHAR(30)，又は MVARCHAR(30)	制約を定義した表の名称。
5	COLUMN_NAME	VARCHAR(30)，又は MVARCHAR(30)	制約で使用している列名。

(47)　SQL_DIV_TYPE 表の内容

この表では，キーレンジ分割とハッシュ分割を組み合わせたマトリクス分割表の分割キーの情報を管理します（1 行で 1 分割キー数分）。

SQL_DIV_TYPE 表の内容を次の表に示します。

表 G–49　SQL_DIV_TYPE 表の内容

項番	列　　名	データ型	内　　容
1	TABLE_SCHEMA	VARCHAR(30)，又は MVARCHAR(30)	表の所有者。
2	TABLE_NAME	VARCHAR(30)，又は MVARCHAR(30)	表の名称。
3	KEY_NO	SMALLINT	分割キー番号（次元番号）。
4	DIV_TYPE	CHAR(1)	次元内分割種別。 P：境界値分割 F：FIX ハッシュ分割 H：フレキシブルハッシュ分割
5	HASH_NAME	VARCHAR(30)，又は MVARCHAR(30)	ハッシュ関数名。 "HASH1" "HASH2" "HASH3" "HASH4" "HASH5" "HASH6" "HASH0" "HASHZ" ハッシュ分割以外の次元の場合，ナル値となります。
6	N_DIV_COLUMN	SMALLINT	次元内分割列数。

(48)　SQL_SYSPARAMS 表の内容

この表では，連続認証失敗回数制限，及びパスワードの文字列制限の情報を管理します（1 行で 1 設定項目数分，n 行で一つの連続認証失敗回数制限の情報分，又は一つのパスワードの文字列制限の情報分）。なお，SQL_SYSPARAMS 表は，DBA 権限所有者，及び監査人だけが参照できます。

SQL_SYSPARAMS 表の内容を次の表に示します。

表 G–50　SQL_SYSPARAMS 表の内容

項番	列　　名	データ型	内　　容
1	PARAM_KIND	VARCHAR(20)	パラメタ種別 （CONNECTION_SECURITY）。
2	FUNCTION_KEY	VARCHAR(20)	機能名。 CONNECT： 連続認証失敗回数制限の場合 PASSWORD： パスワードの文字列制限の場合
3	PARAM_KEY	VARCHAR(20)	指定項目。 機能名が CONNECT の場合は次のどれかです。 PERMISSION_COUNT：連続認証失敗許容回数 LOCK_MINUTE：アカウントロック期間（単位：分） LOCK_MINUTE_CODE：アカウントロック期間コード 機能名が PASSWORD の場合は次のどれかです。 MIN_LENGTH：最小許容バイト数 USER_IDENTIFIER：認可識別子指定禁止 SIMILAR：単一文字種禁止 FORCE_CHARCTER：必須文字種 REUSE_MAX：再利用禁止の個数
4	INT_VALUE	INTEGER	INT 型データ値※。
5	CHAR_VALUE	VARCHAR(30)	CHAR 型データ値※。

注※

INT 型データ値，及び CHAR 型データ値に格納される値を次の表に示します。

PARAM_KEY の設定値	SQL での指定値	INT_VALUE	CHAR_VALUE
PERMISSION_COUNT	定数	定数	定数
	指定なし	2	2
LOCK_MINUTE	定数	定数	定数
	UNLIMITED	ナル値	UNLIMITED
	指定なし	1440	1440
LOCK_MINUTE_CODE	定数	定数	定数
	UNLIMITED	ナル値	UNLIMITED

PARAM_KEY の設定値	SQL での指定値	INT_VALUE	CHAR_VALUE
	指定なし	1000000	1000000
MIN_LENGTH	定数	定数	定数
	指定なし	8	8
USER_IDENTIFIER	RESTRICT	ナル値	RESTRICT
	UNRESTRICT	ナル値	UNRESTRICT
	指定なし	ナル値	RESTRICT
SIMILAR	RESTRICT	ナル値	RESTRICT
	UNRESTRICT	ナル値	UNRESTRICT
	指定なし	ナル値	RESTRICT
FORCE_CHARCTER	UPPER	ナル値	UPPER
	LOWER	ナル値	LOWER
	NUMERIC	ナル値	NUMERIC
	ALL	ナル値	ALL
REUSE_MAX	定数	定数	定数

(49)　SQL_INDEX_XMLINF 表の内容

この表では，部分構造インデクスの構成部分構造パス情報を管理します（1 行で 1 インデクスの情報分）。

SQL_INDEX_XMLINF 表の内容を次の表に示します。

表 G–51　SQL_INDEX_XMLINF 表の内容

項番	列　　名	データ型	内　　容
1	TABLE_SCHEMA	VARCHAR(30)，又は MVARCHAR(30)	表の所有者。
2	TABLE_NAME	VARCHAR(30)，又は MVARCHAR(30)	表名。
3	INDEX_NAME	VARCHAR(30)，又は MVARCHAR(30)	インデクス名。
4	PARTIAL_STRUCTURE_PATH_ORDER	SMALLINT	常に 1。
5	PARTIAL_STRUCTURE_PATH	VARCHAR(32000)，又は MVARCHAR(32000)	部分構造パス。
6	ASC_DESC	CHAR(1)	昇降順種別。 'A'：昇順 'D'：降順

項番	列　　名	データ型	内　　容
7	DATA_TYPE	CHAR(24)	部分構造パスのデータ型。 格納形式は，SQL_COLUMNS 表の DATA_TYPE 列を参照してください。
8	DATA_TYPE_CODE	SMALLINT	部分構造パスのデータ型コード。 格納形式は，SQL_COLUMNS 表の DATA_TYPE_CODE 列を参照してください。
9	DATA_LENGTH	CHAR(7)	部分構造パスのデータ長（文字形式）。 格納形式は，SQL_COLUMNS 表の DATA_LENGTH 列を参照してください。
10	DATA_LENGTH_CODE	SMALLINT	部分構造パスのデータ長。 格納形式は，SQL_COLUMNS 表の DATA_LENGTH_CODE 列を参照してください。

(50)　SQL_SEQUENCES 表の内容

この表では，順序数生成子の情報を管理します（1 行で 1 順序数生成子分）。

SQL_SEQUENCES 表の内容を次の表に示します。

表 G–52　SQL_SEQUENCES 表の内容

項番	列　　名	データ型	内　　容
1	SEQUENCE_SCHEMA	VARCHAR(30)，又は MVARCHAR(30)	順序数生成子の所有者。
2	SEQUENCE_NAME	VARCHAR(30)，又は MVARCHAR(30)	順序数生成子識別子。
3	SEQUENCE_ID	INT	順序数生成子 ID。
4	SEQUENCE_TYPE	CHAR(1)	システムで使用する情報。 常に E となります。
5	PUBLIC_USAGE	CHAR(1)	PUBLIC USAGE 指定の有無。 Y：指定あり 指定しない場合，ナル値となります。
6	CREATE_TIME	CHAR(14)	順序数生成子作成時間。
7	ALTER_TIME	CHAR(14)	常にナル値となります。
8	DATA_TYPE	CHAR(24)	順序数生成子のデータ型。 INTEGER：省略時 格納形式は，SQL_COLUMNS 表の DATA_TYPE 列を参照してください。

項番	列　　名	データ型	内　　容
9	DATA_TYPE_CODE	SMALLINT	順序数生成子のデータ型コード。 格納形式は，SQL_COLUMNS 表の DATA_TYPE_CODE 列を参照してください。
10	DATA_LENGTH	CHAR(7)	順序数生成子のデータ長（文字形式）。 格納形式は，SQL_COLUMNS 表の DATA_LENGTH 列を参照してください。
11	DATA_LENGTH_CODE	SMALLINT	順序数生成子のデータ長。 格納形式は，SQL_COLUMNS 表の DATA_LENGTH_CODE 列を参照してください。
12	START_VALUE	VARCHAR(255)	開始値。 省略時，ナル値となります。
13	MAXIMUM_VALUE	VARCHAR(255)	最大値。 NO MAXVALUE：NO MAXVALUE 指定時 省略時，ナル値となります。
14	MINIMUM_VALUE	VARCHAR(255)	最小値。 NO MINVALUE：NO MINVALUE 指定時 省略時，ナル値となります。
15	INCREMENT	VARCHAR(255)	増分値。 省略時，ナル値となります。
16	CYCLE_OPTION	CHAR(1)	循環オプション。 Y：CYCLE N：NO CYCLE，又は省略時
17	LOGINTERVAL	INT	順序数生成子ログ出力間隔。 省略時は 1 を設定します。 初期値は 1 となります。
18	RDAREA_NAME	VARCHAR(30)，又は MVARCHAR(30)	順序数生成子格納先 RD エリア名称。
19	RDAREA_ID	INT	順序数生成子格納先 RD エリア ID。

(51)　SQL_ACCESS_SECURITY 表の内容

この表では，IP アドレスによる接続制限の情報を管理します（1 行で 1 接続制約分）。

なお，この表を参照できるのは，DBA 権限所有者，及び監査人だけです。

SQL_ACCESS_SECURITY 表の内容を次の表に示します。

表 G–53　SQL_ACCESS_SECURITY 表の内容

項番	列　　名	データ型	内　　容
1	CONSTRAINT_NAME	VARCHAR(30)，又は MVARCHAR(30)	接続制約名
2	ALLOW_DENY	CHAR(1)	接続許可拒否指定 'A'：許可 'D'：拒否
3	IP_ADDR	VARCHAR(18)	IP アドレス
4	CREATE_TIME	CHAR(14)	接続制約の作成時間 （YYYYMMDDHHMMSS）
5	USER_ID	VARCHAR(30)，又は MVARCHAR(30)	接続制約を適用するユーザの認可識別子。 NULL の場合は認可識別子指定なし。

(52)　SQL_ROLES 表の内容

この表では，ロールの情報を管理します（1 行で 1 ロール分）。

なお，この表を参照できるのは，DBA 権限所有者，及び監査人だけです。SQL_ROLES 表の内容を次の表に示します。

表 G–54　SQL_ROLES 表の内容

項番	列名	データ型	内容
1	ROLE_ID	INT	ロールの通し番号
2	ROLE_NAME	VARCHAR(30)，又は MVARCHAR(30)	ロール名
3	ROLE_CREATOR	VARCHAR(30)，又は MVARCHAR(30)	ロール定義者
4	CREATE_TIME	CHAR(14)	登録時間 （YYYYMMDDHHMMSS）

(53)　SQL_ROLE_PRIVILEGES 表の内容

この表では，ロール利用権限の情報を管理します（1 行で 1 認可識別子に付与された 1 ロール利用権限分）。

なお，この表を参照できるのは，DBA 権限所有者，監査人，及びロール利用権限を受ける認可識別子（GRANTEE）が自分である行だけです。SQL_ROLE_PRIVILEGES 表の内容を次の表に示します。

表 G–55　SQL_ROLE_PRIVILEGES 表の内容

項番	列名	データ型	内容
1	ROLE_NAME	VARCHAR(30)，又は MVARCHAR(30)	ロール名
2	GRANTOR	VARCHAR(30)，又は MVARCHAR(30)	ロール利用権限を付与する認可識別子の名称
3	GRANTEE	VARCHAR(30)，又は MVARCHAR(30)	ロール利用権限を受ける認可識別子の名称
4	GRANTEE_TYPE	CHAR(1)	ロール利用権限を付与された認可識別子の種別 'R'：ロール ナル値：ユーザ
5	GRANT_TIME	CHAR(14)	登録時間（YYMMDDHHMMSS）

付録 H　HiRDB が提供する関数

ここでは，HiRDB が提供する次の関数について説明します。

- 表分割ハッシュ関数
- 空白変換関数
- DECIMAL 型符号正規化関数
- 文字コード種別設定関数

付録 H.1　表分割ハッシュ関数

表分割ハッシュ関数とは，分割キーの値から，表分割に指定した分割条件指定順序を取得するものです。この表分割ハッシュ関数を使用した UAP を実行すると，ハッシュ分割した表であっても，表にデータを格納する前に格納先 RD エリアが分かります。格納先 RD エリアが分かるので，次のような場合に使用すると有効です。

- ハッシュ分割するためのハッシュ関数，及び分割キーを決定するときに，格納するデータが均等に分割されるかどうかを評価する場合
- ハッシュ分割した表に，データベース作成ユティリティで RD エリア単位に並列にデータロードする場合，RD エリアごとの入力データファイルを作成するとき

(1)　表分割ハッシュ関数を使用する場合の前提条件

表分割ハッシュ関数を使用する場合の前提条件を次に示します。

(a)　プログラム言語

表分割ハッシュ関数を使用して UAP を作成する場合，次の高級言語で記述できます。

- C，又は C++言語

(b)　実行できる環境

HiRDB サーバ，又は HiRDB クライアントをインストールしたサーバマシンで実行できます。

ただし，HiRDB クライアントで実行する場合，HiRDB サーバと HiRDB クライアントの OS の組み合わせによって，正しい結果が得られない場合があります。

HiRDB クライアントでの実行可否を次の表に示します。

表 H–1　HiRDB クライアントでの実行可否

HiRDB サーバの OS	HiRDB クライアントの OS	
	AIX	Linux，Windows
AIX	○	×
Linux，Windows	×	○

（凡例）

○：実行できます。

×：バイトオーダが異なるため，分割条件指定順序又は分割キー内通番が不正となります。

(2)　表分割ハッシュ関数を使用した UAP の作成，実行

次の手順で UAP を作成し，実行します。

1. ソースプログラムの作成
2. コンパイル，リンケージ
3. ロードモジュールの実行

(a)　プログラムの作成

C，又は C++言語で記述したソースプログラム中に，表分割ハッシュ関数の関数呼出しを記述します。表分割ハッシュ関数は共用ライブラリの形で提供されるので，共用ライブラリをリンクすると使用できます。

表分割ハッシュ関数を使用するときには，提供されるヘッダファイルをソースプログラム作成時にインクルードする必要があります。表分割ハッシュ関数が必要とするすべてのヘッダファイルをインクルードします。表分割ハッシュ関数が必要とするヘッダファイルについては，「表分割ハッシュ関数の詳細」を参照してください。

(b)　コンパイル，リンケージ

HiRDB サーバがインストールされているサーバマシン，又は HiRDB クライアントがインストールされているクライアントマシンでコンパイル，リンケージします。

なお，ソースプログラム中に SQL 文を埋め込む場合は，コンパイル，リンケージする前に，プリプロセスする必要があります。

コンパイル，リンケージ，及びプリプロセスについては，「UAP 実行前の準備」を参照してください。

UNIX 版の HiRDB サーバ側でのコンパイル，リンケージ

HiRDB サーバでコンパイル，リンケージする場合の指定例について説明します。

＜例＞（C 言語の場合）

- ソースファイルの名称が sample.c で，実行形式ファイルの名称を指定しない場合

```
＜32bitモードで動作するUAPを作成する場合＞
  cc -I $PDDIR/include sample.c -L$PDDIR/client/lib -l sqlauxf
＜64bitモードで動作するUAPを作成する場合＞
  cc +DD64 -I $PDDIR/include sample.c -L$PDDIR/client/lib -l sqlauxf64
```

＜例＞（C++言語の場合）

- ソースファイルの名称が sample.C で，実行形式ファイルの名称を指定しない場合

```
＜32bitモードで動作するUAPを作成する場合＞
  CC -I $PDDIR/include sample.C -L$PDDIR/client/lib -l sqlauxf
＜64bitモードで動作するUAPを作成する場合＞
  CC +DD64 -I $PDDIR/include sample.C -L$PDDIR/client/lib -l sqlauxf64
```

UNIX 版の HiRDB クライアント側でのコンパイル，リンケージ

HiRDB クライアントでコンパイル，リンケージする場合の指定例について説明します。

＜例＞（C 言語の場合）

- ソースファイルの名称が sample.c で，実行形式ファイルの名称を指定しない場合

```
＜32bitモードで動作するUAPを作成する場合＞
  cc -I /HiRDB/include sample.c -L/HiRDB/client/lib -l sqlauxf
＜64bitモードで動作するUAPを作成する場合＞
  cc +DD64 -I /HiRDB/include sample.c -L/HiRDB/client/lib -l sqlauxf64
```

注

下線で示す部分は，HiRDB のインストールディレクトリを指定します。

＜例＞（C++言語の場合）

- ソースファイルの名称が sample.C で，実行形式ファイルの名称を指定しない場合

```
＜32bitモードで動作するUAPを作成する場合＞
  CC -I /HiRDB/include sample.C -L/HiRDB/client/lib -l sqlauxf
＜64bitモードで動作するUAPを作成する場合＞
  CC +DD64 -I /HiRDB/include sample.C -L/HiRDB/client/lib -l sqlauxf64
```

注

下線で示す部分は，HiRDB のインストールディレクトリを指定します。

Windows 版の HiRDB サーバ側でのコンパイル，リンケージ

C 言語のソースプログラムは，ANSI-C に従ったコンパイラでコンパイルをします。また，C++言語のソースプログラムは，C++に従ったコンパイラでコンパイルをします。

Microsoft Visual C++ Version 1.0 を使用してコンパイル，リンケージをする場合のオプションの設定は，オプションメニューから「プロジェクトの設定」を選択します。

Microsoft Visual C++ Version 2.0 を使用してコンパイル，リンケージをする場合のオプションの設定は，プロジェクトメニューから「設定」を選択します。

「プロジェクトの設定」，又は「設定」で設定する項目（HiRDB サーバ側）を次の表に示します。

表 H–2 「プロジェクトの設定」，又は「設定」で設定する項目（HiRDB サーバ側）

項目	カテゴリ	カテゴリの設定	設定値
コンパイラ	コード生成	構造体メンバのアライメント	8 バイト
		使用するランタイムライブラリ	マルチスレッド
	プリプロセサ	インクルードのファイルパス	¥HiRDB¥client¥include
リンカ	インプット	ライブラリ	¥HiRDB¥client¥lib ¥pdsqlauxf.lib

注　下線で示す部分は，HiRDB のインストールディレクトリを指定してください。

Windows 版の HiRDB クライアント側でのコンパイル，リンケージ

C 言語のソースプログラムは，ANSI-C に従ったコンパイラでコンパイルをします。また，C++言語のソースプログラムは，C++に従ったコンパイラでコンパイルをします。

Microsoft Visual C++ Version 1.0 を使用してコンパイル，リンケージをする場合のオプションの設定は，オプションメニューから「プロジェクトの設定」を選択します。

Microsoft Visual C++ Version 2.0 を使用してコンパイル，リンケージをする場合のオプションの設定は，プロジェクトメニューから「設定」を選択します。

「プロジェクトの設定」，又は「設定」で設定する項目（HiRDB クライアント側）を次の表に示します。

表 H–3 「プロジェクトの設定」，又は「設定」で設定する項目（HiRDB クライアント側）

項目	カテゴリ	カテゴリの設定	設定値
コンパイラ	コード生成	構造体メンバのアライメント	8 バイト
		使用するランタイムライブラリ	マルチスレッド
	プリプロセサ	インクルードのファイルパス	¥HiRDB¥include
リンカ	インプット	ライブラリ	¥HiRDB¥lib ¥pdsqlauxf.lib

注　下線で示す部分は，HiRDB のインストールディレクトリを指定してください。

(3)　表分割ハッシュ関数の詳細

(a)　入力に必要な情報

表分割ハッシュ関数を呼び出すには，次の 1〜8 の情報を取得して，引数に設定する必要があります。

1. 分割に指定したハッシュ関数名
2. 分割キーに指定した列数
3. 分割キーの指定順序とデータ型コード，データ長コード
4. 表の分割数
5. 分割キーに格納するデータ値
6. HiRDB サーバで使用する各国文字の全角空白文字
7. 空白変換レベルの値
8. DECIMAL 型の符号正規化機能の使用有無

この中の 1〜4 は，CREATE TABLE の次の部分に対応します。

```
CREATE TABLE TABLE1 ( C1  CHAR(10) NOT NULL,
                                   3
                      C2  NVARCHAR(4) NOT NULL,
                                   3
                      C3  DEC(5,2) NOT NULL,
                                   3
                     C4   INT,

                      C5  SMALLINT NOT NULL )
                                   3
        FIX HASH  HASH6   BY  C1,C3,C5,C2
                    1           2,3
            in ( RU01,RU02,RU03,RU04,RU05,RU06 )
                                4
```

なお，既に表を定義している場合，1〜4 はディクショナリ表を検索すれば情報を取得できます。ディクショナリ表の検索例については，「ディクショナリ表からの検索（ハッシュ分割の場合）」を参照してください。

また，空白変換レベル（空白変換機能），及び DECIMAL 型の符号正規化機能については，マニュアル「HiRDB システム運用ガイド」を参照してください。

(b)　記述構成

表分割ハッシュ関数の詳細を次のように説明します。

機能

概要について説明します。

ヘッダファイル

表分割ハッシュ関数を使用するために必要な，ヘッダについて説明します。

形式

実際に指定する形式について説明します。

引数

形式で記述している引数とその意味について説明します。

戻り値

表分割ハッシュ関数の戻り値の型（データ型として記述）について説明します。

(c)　表分割ハッシュ関数（p_rdb_dbhash）

機能

分割キーの値が格納される分割条件指定順序（1〜表の分割数），又は分割キー内通番を取得できます。ただし，正常終了しなかった場合，分割条件指定順序の値は不正になります。

分割条件指定順序を取得する行が複数ある場合には，行ごとに分割キーのデータを変更して，表分割ハッシュ関数を呼ぶ必要があります。この場合，分割キーのデータ値以外の引数は変更する必要はありません。

分割キー内通番から分割条件指定順序を求める方法については，「ディクショナリ表からの検索（マトリクス分割の場合）」を参照してください。

ヘッダファイル

#include<pddbhash.h>

表分割ハッシュ関数を使用する場合に必ず指定します。

#include<pdbsqlda.h>

分割キーのデータ型コードを設定する場合，マクロ（PDSQL_で始まるマクロ）を使用して設定するときに指定します。なお，ディクショナリ表からデータ型コードを検索して設定する場合には，指定する必要はありません。

形式 1(分割キーに文字集合の指定がない場合)

```
int p_rdb_dbhash(short           hashcode,
                 short           ncol,
                 p_rdb_collst_t  *collst,
                 p_rdb_dadlst_t  *dadlst,
                 unsigned int    ndiv,
                 unsigned char   ncspace[2],
                 int             flags,
                 int             *rdno);
```

形式 2(分割キーに文字集合の指定がある場合)

```
int p_rdb_dbhash_cs(short           hashcode,
                    short           ncol,
                    p_rdb_collst_t  *collst,
                    p_rdb_csidlst_t *csidlst,
                    p_rdb_dadlst_t  *dadlst,
                    unsigned int    ndiv,
                    unsigned char   ncspace[2],
                    int             flags,
                    int             *rdno);
```

引数

hashcode（入力）

ハッシュ関数名に対応するハッシュ関数コードを指定します。ハッシュ関数コードについては，「ハッシュ関数コード」を参照してください。

ncol（入力）

表定義時に，分割キーとして指定した列数を指定します。

collst（入力）

分割キーリストへのポインタを指定します。分割キーリストは，分割キーのデータ型コード，データ長コードから成る構造体で，分割キー数分連続する領域です。分割キーリストについては，「分割キーリスト」を参照してください。

分割キーのデータ型コード，データ長コードについては，ディクショナリ表を検索すれば情報を取得できます。ディクショナリ表の検索例については，「ディクショナリ表からの検索（ハッシュ分割の場合）」を参照してください。

csidlst（入力）

形式 2（分割キーに文字集合の指定がある場合）の場合にだけ指定します。分割キーの文字集合 ID リストへのポインタを指定します。

文字集合 ID リストは，文字集合 ID から成る構造体が分割キー数分連続する領域です。文字集合 ID リストについては，「文字集合 ID リスト」を参照してください。

分割キーの文字集合 ID は，ディクショナリ表を検索することで取得できます。ディクショナリ表の検索については，「ディクショナリ表からの検索（ハッシュ分割の場合）」を参照してください。

dadlst（入力）

データアドレスリストへのポインタを指定します。データアドレスリストは，分割キーのデータを格納する領域へのアドレスから成る構造体で，分割キー数分連続する領域です。詳細については，「データアドレスリスト」を参照してください。

ndiv（入力）

ハッシュ分割の分割数を指定します。

ncspace（入力）

HiRDB サーバで使用する各国文字コードの全角空白文字を，2 バイトの領域で指定します。分割キーのデータ型が NVARCHAR の場合，文字列に続く空白を取り除いてハッシングするために使用しま

す。また，flags に空白変換レベル 1，又は 3 を指定した場合の分割キー値（NCHAR，NVARCHAR，MCHAR，又は MVARCHAR）の空白変換のときにも使用します。

次の場合に，ncspace の領域が未設定のときはエラーとなります。

- 分割キーが NVARCHAR の場合
- flags に空白変換レベル 1，又は 3 を指定していて，かつ分割キーが NCHAR，NVARCHAR，MCHAR，又は MVARCHAR の場合

ncspace に指定する全角空白文字を次の表に示します。

表 H–4　ncspace に指定する全角空白文字

pdsetup で指定した文字コード種別※2	ncspace	
	ncspace[0]	ncspace[1]
sjis（シフト JIS 漢字コード）	0x81	0x40
chinese（EUC 中国語漢字コード）	0xA1	0xA1
ujis（EUC 日本語漢字コード）	0xA1	0xA1
lang-c（単一バイト文字コード）※1	0x00	0x00
utf-8（Unicode（UTF-8））※3	0x00	0x00
utf-8_ivs（Unicode（IVS 対応 UTF-8））※3	0x00	0x00
chinese-gb18030（中国語漢字コード GB18030）※3	0xA1	0xA1
省略時（AIX の場合は sjis）	0x81	0x40
省略時（Linux の場合は ujis）	0xA1	0xA1
省略時（Windows の場合は sjis）	0x81	0x40

注※1

lang-c の場合，列のデータ型に NCHAR，NVARCHAR，MCHAR，及び MVARCHAR は使用できません。

注※2

Windows 環境の場合は，pdntenv コマンドで指定した文字コード種別の空白文字コードを指定してください。

注※3

列のデータ型には，NCHAR 及び NVARCHAR は使用できません。

flags（入力）

空白変換レベル，及び DECIMAL 型の符号正規化機能の指定に従って指定します（これらの機能を使用しない場合でも指定する必要があります）。空白変換レベル，及び DECIMAL 型の符号正規化機能については，マニュアル「HiRDB システム運用ガイド」を参照してください。

flags に指定する値を次に示します。

HiRDB の動作環境		flags 指定値
空白変換レベル※	省略	p_rdb_FLG_SPLVL_0
	0	
	1	p_rdb_FLG_SPLVL_1
	3	p_rdb_FLG_SPLVL_3
DECIMAL 型の符号正規化機能	省略	p_rdb_FLG_DECNRM_N
	N	
	Y	p_rdb_FLG_DECNRM_Y

注※

HiRDB サーバの文字コードが Unicode（UTF-8）の場合，あらかじめ空白変換をしてからこの関数を実行する必要があるため，flags の指定値には p_rdb_FLG_SPLVL_0 以外は指定しないでください。

rdno（出力）

分割条件指定順序（1〜表の分割数），又は分割キー内通番が設定されます。

戻り値

データ型：int

p_rdb_RC_RTRN(0)

正常終了

p_rdb_RC_ERRHASH(-1)

ハッシュ関数コード不正（p_rdb_HASH1〜p_rdb_HASH6，p_rdb_HASHZ，p_rdb_HASH0，p_rdb_HASHA〜p_rdb_HASHF）

p_rdb_RC_ERRNCOL(-2)

分割キー数不正（１〜p_rdb_MXDCL）

p_rdb_RC_ERRCLST(-3)

分割キーのデータ型，データ長の領域不正

p_rdb_RC_ERRCTYP(-31)

分割キーのデータ型が不正

p_rdb_RC_ERRCLEN(-32)

分割キーのデータ長が不正

p_rdb_RC_ERRCSID(-33)

分割キーの文字集合 ID が不正

p_rdb_RC_ERRDLST(-4)

データアドレスの領域不正

p_rdb_RC_ERRDADR(-41)

データアドレスが未設定

p_rdb_RC_ERRDLEN(-42)

ハッシュ関数が HASH3，HASHC，HASH4，HASHD，HASH5，又は HASHE の場合：データの実長が指定されたハッシュ関数でハッシング可能な最小長より短い

ハッシュ関数が HASH0 又は HASHZ で，分割キーのデータ型が CHAR 型の場合：日付データの文字列表現形式が不正

p_rdb_RC_ERRNDIV(-5)

表の分割数不正（１～p_rdb_MNCND）

p_rdb_RC_ERRRADR(-6)

分割条件指定順序，又は分割キー内通番の格納領域が未設定

p_rdb_RC_ERRNCSC(-7)

全角空白文字の領域が未設定

注意事項

注意事項を次に示します。

1. 分割キーが NCHAR，NVARCHAR，MCHAR，又は MVARCHAR の場合，flags に空白変換レベルに対応した値を指定しないと，rdno の値が不正になることがあります。
2. 分割キーが NCHAR，NVARCHAR，MCHAR，又は MVARCHAR で，かつ空白変換レベルに 1，又は 3 を指定した場合，次に示すどちらかのことをしてください。

   ・setlocale 関数で LC_CTYPE カテゴリ，又は LC_ALL カテゴリに適切なロケールを設定する

   ・p_rdb_set_lang 関数を呼び出す

   キー値の文字コード種別と setlocale 関数又は p_rdb_set_lang 関数で指定したロケールが矛盾する場合，動作は保証されません。Windows の UAP，文字コード種別が SJIS の Linux の UAP，又は文字コード種別が CHINESE の UAP からこの関数が呼び出される場合は，setlocale 関数ではなく p_rdb_set_lang 関数を使用します。p_rdb_set_lang 関数については，「文字コード種別設定関数」を参照してください。
3. 分割キー値が DECIMAL，INTERVAL YEAR TO DAY，又は INTERVAL HOUR TO SECOND の場合，flags に DECIMAL 型の符号正規化機能に対応した値を指定しないと，rdno の値が不正になることがあります。
4. バージョン 05-05 より前の HiRDB クライアントで表分割ハッシュ関数を使用する場合，flags の空白変換レベル，及び DECIMAL 型の符号正規化機能は使用できません。したがって，flags の指定は無効となり，空白変換レベル，及び DECIMAL 型の符号正規化機能が省略時の場合と同じ動作になります。flags の指定を有効にしたい場合，HiRDB サーバ側で実行するか，又はバージョン 05-05 以降の HiRDB クライアントで実行してください。
5. HiRDB サーバの文字コードが Unicode（UTF-8）の場合，この関数では空白変換をしません。dadlst に指定する分割キーの値は，事前に空白変換関数 p_rdb_conv_space_utf8 を使用して変換しておく必要があります。
6. 文字集合に UTF16 の指定がある場合，分割キーのデータはビッグエンディアン形式にする必要があります。ビッグエンディアンにしないと，rdno の値が不正になるおそれがあります。

(4)　データ型とマクロ

(a)　ハッシュ関数コード

CREATE TABLE，又は ALTER TABLE で指定するハッシュ関数に対応するハッシュ関数コードを次の表に示します。

表 H–5　ハッシュ関数に対応するハッシュ関数コード

ハッシュ関数名	ハッシュ関数コード（値）
HASH1（ハッシュ関数名省略時）	p_rdb_HASH1 (1)
HASH2	p_rdb_HASH2 (2)
HASH3	p_rdb_HASH3 (3)
HASH4	p_rdb_HASH4 (4)
HASH5	p_rdb_HASH5 (5)
HASH6	p_rdb_HASH6 (6)
HASHZ	p_rdb_HASHZ (99)
HASH0	p_rdb_HASH0 (100)
HASHA	p_rdb_HASHA(101)
HASHB	p_rdb_HASHB(102)
HASHC	p_rdb_HASHC(103)
HASHD	p_rdb_HASHD(104)
HASHE	p_rdb_HASHE(105)
HASHF	p_rdb_HASHF(106)

(b)　分割キーリスト

分割キーリストは，分割キーのデータ型コード，データ長コードから成る構造体で，分割キー数分連続する領域です。分割キーを設定する領域を次の表に示します。分割キーが複数ある場合，分割キーとして指定した列数分の配列にする必要があります。

表 H–6　分割キーを設定する領域

データ型	データ型の詳細	説明
p_rdb_collst_t	struct p_rdb_TG_collst { unsigned short datatype ; short datalen ; } p_rdb_collst_t ;	データ型コード データ長コード

データ型コード，データ長コードを次の表に示します。

表 H–7　データ型コード，データ長コード

データ型	データ型コード	データ長コード
INTERVAL YEAR TO DAY	PDSQL_YEARTODAY	8×256
INTERVAL HOUR TO SECOND	PDSQL_HOURTOSEC	6×256
DATE	PDSQL_DATE	4
TIME	PDSQL_TIME	3
TIMESTAMP[(p)]	PDSQL_TIMESTAMP	7 ＋↑p/2↑（p を省略した場合は 0 を仮定）
MVARCHAR(n)	PDSQL_MVARCHAR	n
MCHAR[(n)]	PDSQL_MCHAR	n(n を省略した場合は 1 を仮定)
NVARCHAR(n)	PDSQL_NVARCHAR	n
NCHAR[(n)]	PDSQL_NCHAR	n(n を省略した場合は 1 を仮定)
VARCHAR(n)	PDSQL_VARCHAR	n
CHAR[(n)]	PDSQL_CHAR	n(n を省略した場合は 1 を仮定)
FLOAT	PDSQL_FLOAT	8
SMALLFLT	PDSQL_SMALLFLT	4
DECIMAL[(p[,q])]	PDSQL_DECIMAL	p×256+q(p を省略した場合は 15 を仮定，q を省略した場合は 0 を仮定)
INTEGER	PDSQL_INTEGER	4
SMALLINT	PDSQL_SMALLINT	2

(c)　文字集合 ID リスト

分割キーに文字集合の指定がある場合にだけ指定します。文字集合 ID を設定する領域を次の表に示します。分割キーが複数ある場合，分割キーとして指定した列数分の配列にする必要があります。

表 H–8　文字集合 ID を設定する領域

データ型	データ型の詳細	説明
p_rdb_csidlst_t	struct p_rdb_TG_csidlst { int charset_id ; } p_rdb_csidlst_t ;	文字集合 ID コード

文字集合 ID コードを次の表に示します。

表 H–9　文字集合 ID コード

文字集合	文字集合 ID コード（値）
文字集合の指定なし	p_rdb_CSDEF

文字集合	文字集合 ID コード（値）
	(0x00000000)
EBCDIK	p_rdb_CSEBK (0x01000004)
UTF16	p_rdb_CSU16 (0x11030004)

(d)　データアドレスリスト

データアドレスリストは，分割キーのデータを格納する領域へのアドレスから成る構造体で，分割キー数分連続する領域です。分割キーのデータアドレスを設定する領域を次の表に示します。分割キーが複数ある場合，分割キーとして指定した列数分の配列にする必要があります。

データ領域は，バイナリ形式で記述してください。バイナリ形式については，マニュアル「HiRDB コマンドリファレンス」を参照してください。

表 H–10　分割キーのデータアドレスを設定する領域

データ型	データ型の詳細	説明

p_rdb_dadlst_t	`struct p_rdb_TG_dadlst {`
	`unsigned char * dataaddr ;`
	`} p_rdb_dadlst_t ;`
		データ領域へのアドレス

注意事項

すべてのデータ型共通の注意事項

- データアドレスリストの実データは，列で定義したデータ型の形式に変換してください。
- データアドレスリストの実データ領域は，特に境界調整をする必要はありません。

DECIMAL，INTERVAL YEAR TO DAY，又は INTERVAL HOUR TO SECOND の場合の注意事項

- 正の値の場合，データアドレスリストの実データの符号部は'C'，又は'F'を使用してください。なお，DECIMAL 型の符号正規化機能に Y を指定している場合は，'A'，及び'E'も使用できます。
- 負の値の場合，データアドレスリストの実データの符号部は'D'を使用してください。なお，DECIMAL 型の符号正規化機能に Y を指定している場合は，'B'も使用できます。

CHAR，NCHAR，又は MCHAR の場合の注意事項

- CHAR，MCHAR の場合，データアドレスリストのデータ領域には，定義長まで 1 バイトの空白を埋めてください。
- NCHAR の場合，データアドレスリストのデータ領域には，定義長まで 2 バイトの空白を埋めてください。2 バイトの空白は，HiRDB サーバのセットアップ時に指定した文字コードの空白を指定する必要があります。

- データアドレスリストのデータ領域は，HiRDB サーバで使用する文字コードで指定する必要があります。

VARCHAR，NVARCHAR，又は MVARCHAR の場合の注意事項

- データアドレスリストのデータ領域の実長部分は，文字列長ではなくバイトで表してください。
- データアドレスリストのデータ領域の実長が，分割キーリストの定義長に満たない場合，後続する文字列に空白を埋めないでください。
- データアドレスリストのデータ領域は，HiRDB サーバで使用する文字コードで指定してください。

(e)　最大値に関するマクロ

最大値に関するマクロを次の表に示します。

表 H–11　最大値に関するマクロ

マクロ名	意味（値）
p_rdb_MXDCL	分割キーの列数の最大値（16）
p_rdb_MNCND	表の分割数の最大値（4096）
p_rdb_MNCND_0902	表の分割数の最大値（1024） （バージョン 07-02 から 09-02 の HiRDB を使用する場合）
p_rdb_MNCND_0701	表の分割数の最大値（512） （バージョン 07-01 までの HiRDB を使用する場合）

(5)　コーディング例

ハッシュ分割の場合の，C 言語で記述した部分的なコーディング例を次に示します。このコーディング例を，ユーザの用途に合うようにカスタマイズして使用してください。ただし，SQL 文実行時のエラー処理については記述していないので，必要に応じて記述するようにしてください。エラー処理については，「SQL のエラーの判定と処置」を参照してください。

(a)　宣言部

```
/*********************************************************************/
/* ALL RIGHTS RESERVED, COPYRIGHT (C) 1999,2000, HITACHI, LTD.       */
/* LICENSED MATERIAL OF HITACHI,LTD.                                 */
/* 表分割ハッシュ関数を使用したサンプルプログラム                    */
/*********************************************************************/
#include <stdio.h>
#include <string.h>
#include <pdbsqlda.h>
#include <pddbhash.h>

union data_area {                          /* データ格納領域       */
    short data_smallint ;
    int data_int ;
    unsigned char data_dec[15] ;
    float data_smallflt ;
```

```
    double data_float ;
    unsigned char data_char[255] ;
    struct {
        short length ;
        unsigned char data[255] ;
    } data_varchar ;
    unsigned char data_date[4] ;
    unsigned char data_time[3] ;
    unsigned char data_timestamp[10] ;
    unsigned char data_iytd[5] ;
    unsigned char data_ihts[4] ;
} ;

void print_data(short , p_rdb_collst_t * , union data_area *) ;


/*********************************************************************/
/* メイン関数                                                        */
/*********************************************************************/
int main(int argc , char *argv[])
{

  short           hashcode ;              /* ハッシュ関数コード   */
  short           ncol ;                  /* 分割キーの列数       */
  p_rdb_collst_t  collst[p_rdb_MXDCL] ;   /* 分割キーリスト       */
  p_rdb_csidlst_t csidlst[p_rdb_MXDCL] ;  /* 文字集合IDリスト※   */
  p_rdb_dadlst_t  dadlst[p_rdb_MXDCL] ;   /* データアドレスリスト */
  union data_area data[p_rdb_MXDCL] ;     /* データ格納領域       */
  unsigned int    ndiv ;                  /* 格納先RDエリア数     */
  unsigned char   ncspace[2] ;            /*各国文字スペースコード*/
  int             flags ;                 /* エンハンス用フラグ   */
  int             rdno ;                  /* 分割条件指定順序     */
  int             rc ;                    /* リターン値           */
  short           i, j, k ;               /* カウンタ用変数       */

  struct rdarea {                         /* RDエリアリスト       */
  int             rdareaid ;
  char            rdareaname[31] ;
  } rdarealst [p_rdb_MNCND] ;

  EXEC SQL BEGIN DECLARE SECTION ;
    struct {                        /* ハッシュ関数名の埋込み変数    */
      short length ;
      char data[9] ;
    } xhashname ;
    short xncol ;                  /* 分割キーの列数の埋込み変数     */
    short xndiv ;                  /* 表の分割数の埋込み変数         */
    short xdatatype ;              /* データ型コードの埋込み変数     */
    short xdatalen ;               /* データ長コードの埋込み変数     */
    int   xcharset_id ;            /* 文字集合IDコードの埋込み変数※ */
    struct {                       /* 格納先RDエリア名の埋込み変数   */
      short length ;
      char data[31] ;
    } xrdname ;
  EXEC SQL END DECLARE SECTION ;

  EXEC SQL CONNECT ;
```

注※　分割キーに文字集合の指定がある場合にだけ指定します。

(b)　データ格納領域，各国文字コードのスペースの設定

```
for  (k = 0 ; k < p_rdb_MXDCL ; k ++) {
dadlst[k].dataaddr = (unsigned char *)&data[k] ;
}
ncspace[0] = 0x81 ;                    /* スペースコード          */
ncspace[1] = 0x40 ;                    /* シフトJIS漢字コードの例*/
flags = 0 ;
```

(c)　flags の設定

```
/*********************************************************************/
/* (a) 明示的に指定する場合                                          */
/*     空白変換レベル１・DECIMAL型の符号正規化機能Ｙの場合           */
/*********************************************************************/
flags=p_rdb_FLG_SPLVL_1+p_rdb_FLG_DECNRM_Y;
```

(d)　ハッシュ関数名，分割キーの列数，格納先 RD エリア数の設定

```
/*********************************************************************/
/* (a) コードで設定する場合                                          */
/*********************************************************************/
hashcode = p_rdb_HASH6 ;            /* HASH6を指定する場合       */
ncol = 4 ;                          /* ４列で分割する場合         */
ndiv = 6 ;                          /* ６分割する場合             */

/*********************************************************************/
/* (b) ディクショナリ表から検索する場合                              */
/*********************************************************************/
EXEC SQL
    select  HASH_NAME,
          value(N_DIV_COLUMN,1) ,
          N_RDAREA
        into  :xhashname , :xncol, :xndiv
        from MASTER.SQL_TABLES
        where TABLE_SCHEMA=USER
          and TABLE_NAME='TABLE1' ;

xhashname.data[xhashname.length] = '¥0' ;
if (strcmp(xhashname.data,"HASH1") == 0) {
    hashcode=p_rdb_HASH1 ;                          /* HASH1設定  */
} else if (strcmp(xhashname.data,"HASH2") == 0) {
    hashcode=p_rdb_HASH2 ;                          /* HASH2設定  */
} else if (strcmp(xhashname.data,"HASH3") == 0) {
    hashcode=p_rdb_HASH3 ;                          /* HASH3設定  */
} else if (strcmp(xhashname.data,"HASH4") == 0) {
    hashcode=p_rdb_HASH4 ;                          /* HASH4設定  */
} else if (strcmp(xhashname.data,"HASH5") == 0) {
    hashcode=p_rdb_HASH5 ;                          /* HASH5設定  */
} else if (strcmp(xhashname.data,"HASH6") == 0) {
    hashcode=p_rdb_HASH6 ;                          /* HASH6設定  */
} else if (strcmp(xhashname.data,"HASHZ") == 0) {
```

```
    hashcode=p_rdb_HASHZ ;                          /* HASHZ設定  */
} else if (strcmp(xhashname.data,"HASH0") == 0) {
    hashcode=p_rdb_HASH0 ;                          /* HASH0設定  */
} else if (strcmp(xhashname.data,"HASHA") == 0) {
    hashcode=p_rdb_HASHA ;                          /* HASHA設定  */
} else if (strcmp(xhashname.data,"HASHB") == 0) {
    hashcode=p_rdb_HASHB ;                          /* HASHB設定  */
} else if (strcmp(xhashname.data,"HASHC") == 0) {
    hashcode=p_rdb_HASHC ;                          /* HASHC設定  */
} else if (strcmp(xhashname.data,"HASHD") == 0) {
    hashcode=p_rdb_HASHD ;                          /* HASHD設定  */
} else if (strcmp(xhashname.data,"HASHE") == 0) {
    hashcode=p_rdb_HASHE ;                          /* HASHE設定  */
} else if (strcmp(xhashname.data,"HASHF") == 0) {
    hashcode=p_rdb_HASHF ;                          /* HASHF設定  */
} else {
    /* 将来，ハッシュ関数が追加になった時に追加する。              */
}
ncol = xncol ;
ndiv = xndiv ;

/* ========================================================== */
/* 表定義情報表示                                             */
/* ========================================================== */
printf("ハッシュ関数コード：%d¥n",hashcode) ;
printf("分割キーの列数     ：%d¥n",ncol) ;
printf("表の分割数         ：%d¥n",ndiv) ;
printf("¥n") ;
```

(e) 分割キーの指定順序，データ型コード，データ長コードの設定（分割キーに文字集合の指定がない場合）

```
/*********************************************************************/
/* (a) コードで設定する場合                                          */
/*********************************************************************/
collst[0].datatype=PDSQL_CHAR ;             /* CHAR(10)        */
collst[0].datalen=10 ;
collst[1].datatype=PDSQL_DECIMAL ;          /* DEC(5,2)        */
collst[1].datalen=5*256+2 ;
collst[2].datatype=PDSQL_SMALLINT ;         /* SMALLINT        */
collst[2].datalen=2 ;
collst[3].datatype=PDSQL_NVARCHAR ;         /* NVARCHAR(4)     */
collst[3].datalen=4 ;

/*********************************************************************/
/* (b) ディクショナリ表から検索する場合                              */
/*********************************************************************/
EXEC SQL
    declare cr1 cursor for
        select  value(DIVCOL_ORDER,1) ,
                DATA_TYPE_CODE,
                DATA_LENGTH_CODE
        from MASTER.SQL_COLUMNS
        where TABLE_SCHEMA=USER
          and TABLE_NAME='TABLE1'
          and DIVIDED_KEY='Y'
```

```
        order by 1 asc ;

EXEC SQL open cr1 ;
EXEC SQL whenever not found goto fetch_end1 ;

for  (i = 0 ; ; i++) {
    EXEC SQL fetch cr1 into :xncol , : xdatatype , : xdatalen ;
    collst[i].datatype = xdatatype ;
    collst[i].datalen = xdatalen ;
}

fetch_end1 :
EXEC SQL close cr1 ;
```

(f) 分割キーの指定順序，データ型コード，データ長コード，文字集合 ID の設定（分割キーに文字集合の指定がある場合）

```
/*********************************************************************/
/* (a) コードで設定する場合                                          */
/*********************************************************************/
collst[0].datatype=PDSQL_CHAR ;         /* CHAR(10)              */
collst[0].datalen=10 ;
csidlst[0].charset_id=p_rdb_CSEBK ;     /* CHARACTER SET EBCDIK  */
collst[1].datatype=PDSQL_DECIMAL ;      /* DEC(5,2)              */
collst[1].datalen=5*256+2 ;
csidlst[1].charset_id=p_rdb_CSDEF ;
collst[2].datatype=PDSQL_SMALLINT ;     /* SMALLINT              */
collst[2].datalen=2 ;
csidlst[2].charset_id=p_rdb_CSDEF ;
collst[3].datatype=PDSQL_NVARCHAR ;     /* NVARCHAR(4)           */
collst[3].datalen=4 ;
csidlst[3].charset_id=p_rdb_CSDEF ;

/*********************************************************************/
/* (b) ディクショナリ表から検索する場合                              */
/*********************************************************************/
EXEC SQL
    declare cr1_cs cursor for
        select  value(DIVCOL_ORDER,1),
                DATA_TYPE_CODE,
                DATA_LENGTH_CODE,
                value(CHARSET_ID,0)
           from MASTER.SQL_COLUMNS
          where TABLE_SCHEMA=USER
            and TABLE_NAME='TABLE1'
            and DIVIDED_KEY='Y'
          order by 1 asc ;

EXEC SQL open cr1_cs ;
EXEC SQL whenever not found goto fetch_end1_cs ;

for  (i = 0 ; ; i++) {
     EXEC SQL fetch cr1_cs into :xncol , : xdatatype , : xdatalen, :xcharset_id ;
     collst[i].datatype = xdatatype ;
     collst[i].datalen = xdatalen ;
     csidlst[i].charset_id = xcharset_id ;
```

```
}

fetch_end1_cs :
EXEC SQL close cr1_cs ;
```

(g)　格納先 RD エリア名の設定

```
/*********************************************************************/
/* ディクショナリ表から検索する場合                                  */
/*********************************************************************/
EXEC SQL
    declare cr2 cursor for
        select RDAREA_NAME
            from MASTER.SQL_DIV_TABLE
            where TABLE_SCHEMA=USER
              and TABLE_NAME='TABLE1'
            order by DIV_NO asc ;

EXEC SQL open cr2 ;
EXEC SQL whenever not found goto fetch_end2 ;

for (j = 0 ; ; j++) {
    EXEC SQL fetch cr2 into :xrdname ;
    strncpy(rdarealst[j].rdareaname,
            xrdname.data,
            xrdname.length) ;
    rdarealst[j].rdareaname[xrdname.length] = '¥0' ;
}

fetch_end2 :
EXEC SQL close cr2 ;

EXEC SQL DISCONNECT ;

/* ========================================================= */
/* ＲＤエリア情報表示                                        */
/* ========================================================= */
printf("ＲＤエリア名：[")  ;
for (j = 0 ; j<ndiv ; j++) {
    printf("%s",rdarealst[j].rdareaname) ;
    if (j != ndiv-1) {
             printf(",") ;
    } else ;
}
printf("]¥n") ;
printf("¥n") ;
```

(h)　分割キーに格納するデータ設定

```
/*********************************************************************/
/* バイナリ形式で代入する。                                          */
/*  行ごとにデータを設定してハッシュ関数を呼ぶ                       */
/*********************************************************************/
memcpy((char *)data[0].data_char,"abcdefg   ",10); /* "abcdefg   " */

data[1].data_dec[0] = 0x04 ;
```

```
  data[1].data_dec[1] = 0x32 ;
  data[1].data_dec[2] = 0x1D ;                    /* -43.21       */

  data[2].data_smallint = 12345 ;                 /* 12345        */

  /* NCHAR,NVARCHARは，HiRDBサーバの文字コードで指定する。    */
  data[3].data_varchar.length = 6 ;
  data[3].data_varchar.data[0] = 0x82 ;    /* あ                        */
  data[3].data_varchar.data[1] = 0xa0 ;    /* シフトJIS漢字コードの例*/
  data[3].data_varchar.data[2] = 0x82 ;    /* い                        */
  data[3].data_varchar.data[3] = 0xa2 ;    /* シフトJIS漢字コードの例*/
  data[3].data_varchar.data[4] = 0x82 ;    /* う                        */
  data[3].data_varchar.data[5] = 0xa4 ;    /* シフトJIS漢字コードの例*/

  /* ========================================================== */
  /* データ型コード，データ長コード，データ領域表示              */
  /* ========================================================== */
  print_data(ncol , collst , data) ;

  /* ========================================================== */
  /* ハッシュ関数コール                                         */
  /* ========================================================== */
  rc = p_rdb_dbhash(hashcode,ncol,collst,dadlst,ndiv,ncspace,flags,&rdno);

  switch (rc) {
  case p_rdb_RC_RTRN :
    /*****************************************************************/
    /* 正常処理                                                      */
    /*****************************************************************/
    printf("分割条件指定順序 : %d -> %s¥n",
           rdno,rdarealst[rdno-1].rdareaname) ;
    break ;
  default :
    /*****************************************************************/
    /* エラー処理を追加する                                          */
    /*****************************************************************/
    printf("RETURN CODE=%d¥n",rc) ;
    break ;
  }

  return ;
}

/*************************************************************************/
/* データ型コード，データ長コード，データ領域表示関数            */
/*************************************************************************/
void print_data( short           ncol ,
                 p_rdb_collst_t  *pcollst ,
                 union data_area *pdata )
{
  int i , j ;                                  /*  カウンタ用変数  */
  int len;
  p_rdb_collst_t   *ccollst ;
  union data_area  *cdata ;

  printf("分割キー指定順序 データ型コード データ長コード バイナリ形式のデータ値¥n") ;
  for  (i = 0 , ccollst = pcollst , cdata = pdata ;
        i<ncol ;
```

```
        i++ , ccollst++ , cdata++) {
      printf("             %2d         %#.4x         %#.4x  ",
              i+1,ccollst->datatype, ccollst->datalen) ;

    switch (ccollst->datatype) {
      case PDSQL_CHAR :
      case PDSQL_MCHAR :
      case PDSQL_INTEGER :
      case PDSQL_SMALLFLT :
      case PDSQL_FLOAT :
      case PDSQL_SMALLINT :
      case PDSQL_DATE :
      case PDSQL_TIME :
      case PDSQL_TIMESTAMP :
          len=ccollst->datalen ;
          break ;
      case PDSQL_VARCHAR :
      case PDSQL_MVARCHAR :
      case PDSQL_NVARCHAR :
          len=cdata->data_varchar.length+2 ;
          break ;
      case PDSQL_NCHAR :
          len=ccollst->datalen*2 ;
          break ;
      case PDSQL_DECIMAL :
      case PDSQL_YEARTODAY :
      case PDSQL_HOURTOSEC :
          len=ccollst->datalen/256/2+1 ;
          break ;
      default :
          break ;
      }
      for (j=0 ; j<len ;j++){
          printf("%.2X",cdata->data_char[j]) ;
      }
      printf("¥n") ;
  }
  printf("¥n") ;
return ;
}
```

(i)　シフト JIS 漢字コードの場合の実行結果

```
ハッシュ関数コード：6
分割キーの列数     ：4
表の分割数         ：6

ＲＤエリア名：[RU01,RU02,RU03,RU04,RU05,RU06]

分割キー指定順序 データ型コード データ長コード バイナリ形式のデータ値
              1         0x00c4         0x000a  61626364656667202020
              2         0x00e4         0x0502  04321D
              3         0x00f4         0x0002  3039
              4         0x00b0         0x0004  000682A082A282A4

分割条件指定順序 ： 1 -> RU01
```

(6) ディクショナリ表からの検索（ハッシュ分割の場合）

ハッシュ分割の場合の，ディクショナリ表からの検索例を次に示します。

(a) ハッシュ分割表のハッシュ関数名，分割キーの列数，表の分割数の取得

```
SELECT HASH_NAME,                          /*ハッシュ関数名    */
       VALUE(N_DIV_COLUMN,1),              /*分割キーの列数    */
       N_RDAREA                            /*表の分割数        */
    FROM MASTER.SQL_TABLES
      WHERE TABLE_SCHEMA=認可識別子        /*認可識別子が一致するもの */
        AND TABLE_NAME=表識別子            /*表識別子が一致するもの    */
```

(b) 分割キー指定順序とデータ型コード，データ長コードの取得（分割キーに文字集合の指定がない場合）

```
SELECT VALUE(DIVCOL_ORDER,1),              /*分割キー指定順序   */
       DATA_TYPE_CODE,                     /*データ型コード     */
       DATA_LENGTH_CODE                    /*データ長コード     */
    FROM MASTER.SQL_COLUMNS
      WHERE TABLE_SCHEMA=認可識別子        /*認可識別子が一致するもの */
        AND TABLE_NAME=表識別子            /*表識別子が一致するもの    */
        AND DIVIDED_KEY='Y'                /*分割キーのもの            */
    ORDER BY 1 ASC
```

(c) 分割キー指定順序とデータ型コード，データ長コード，文字集合 ID コードの取得（分割キーに文字集合の指定がある場合）

```
SELECT VALUE(DIVCOL_ORDER,1),              /*分割キー指定順序    */
     DATA_TYPE_CODE,                       /*データ型コード      */
     DATA_LENGTH_CODE,                     /*データ長コード      */
     VALUE(CHARSET_ID,0)                   /*文字集合IDコード   */
   FROM MASTER.SQL_COLUMNS
     WHERE TABLE_SCHEMA=認可識別子         /*認可識別子が一致するもの */
       AND TABLE_NAME=表識別子             /*表識別子が一致するもの    */
       AND DIVIDED_KEY='Y'                 /*分割キーのもの            */
   ORDER BY 1 ASC
```

(d) 格納先 RD エリア名の取得

```
SELECT  DIV_NO,                           /*分割条件指定順序     */
        RDAREA_NAME                       /*格納先RDエリア名    */
    FROM MASTER.SQL_DIV_TABLE
      WHERE TABLE_SCHEMA=認可識別子       /*認可識別子が一致するもの */
        AND TABLE_NAME=表識別子           /*表識別子が一致するもの    */
    ORDER BY 1 ASC
```

(7) ディクショナリ表からの検索（マトリクス分割の場合）

マトリクス分割の場合の，ディクショナリ表からの検索例を次に示します。

(a) ハッシュ分割表のハッシュ関数名，分割キーの列数，分割キー番号，表の分割数の取得

- **ハッシュ関数名と分割キーの列数の取得**

```
select HASH_NAME,                    /*ハッシュ関数名          */
       value(N_DIV_COLUMN,1),        /*分割キーの列数          */
       KEY_NO                        /*分割キー番号            */
 from MASTER.SQL_DIV_TYPE
     where TABLE_SCHEMA=認可識別子   /*認可識別子が一致するもの*/
       and TABLE_NAME=表識別子       /*表識別子が一致するもの  */
```

- **キー内分割数の取得**

```
select distinct N_DIVISION           /*キー内分割数            */
from MASTER.SQL_PARTKEY
     where TABLE_SCHEMA=認可識別子   /*認可識別子が一致するもの*/
       and TABLE_NAME=表識別子       /*表識別子が一致するもの  */
       and KEY_NO=分割キー番号       /*ハッシュ分割の          */
                                     /*分割キー番号を設定する  */
```

(b) 分割キー指定順序とデータ型コード，データ長コードの取得（分割キーに文字集合の指定がない場合）

```
select DIVCOL_ORDER,                 /*キー内分割数            */
       DATA_TYPE_CODE,               /*データ型コード          */
       DATA_LENGTH_CODE              /*データ長コード          */
 from MASTER.SQL_COLUMNS X,
      MASTER.SQL_PARTKEY Y
     where X.TABLE_SCHEMA=Y.TABLE_SCHEMA
       and X.TABLE_NAME=Y.TABLE_NAME
       and X.COLUMN_ID=Y.COLUMN_ID
       and Y.TABLE_SCHEMA=認可識別子 /*認可識別子が一致するもの*/
       and Y.TABLE_NAME=表識別子     /*表識別子が一致するもの  */
       and Y.KEY_NO=分割キー番号     /*ハッシュ分割の          */
                                     /*分割キー番号を設定する  */
  order by DIVCOL_ORDER asc
```

(c) 分割キー指定順序とデータ型コード，データ長コード，文字集合 ID の取得（分割キーに文字集合の指定がある場合）

```
select DIVCOL_ORDER,                 /*キー内分割数            */
      DATA_TYPE_CODE,                /*データ型コード          */
      DATA_LENGTH_CODE,              /*データ長コード          */
      value(CHARSET_ID,0)            /*文字集合IDコード       */
 from MASTER.SQL_COLUMNS X,
      MASTER.SQL_PARTKEY Y
     where X.TABLE_SCHEMA=Y.TABLE_SCHEMA
       and X.TABLE_NAME=Y.TABLE_NAME
       and X.COLUMN_ID=Y.COLUMN_ID
       and Y.TABLE_SCHEMA=認可識別子 /*認可識別子が一致するもの */
       and Y.TABLE_NAME=表識別子     /*表識別子が一致するもの   */
       and Y.KEY_NO=分割キー番号     /*ハッシュ分割の           */
```

```
                                        /*分割キー番号を設定する     */
  order by DIVCOL_ORDER asc
```

(d)　格納先 RD エリア名の取得

```
select DIV_NO,                          /*分割条件指定順序          */
       RDAREA_NAME                      /*格納先RDエリア名          */
 from MASTER.SQL_DIV_TABLE
     where TABLE_SCHEMA=認可識別子      /*認可識別子が一致するもの*/
       and TABLE_NAME=表識別子          /*表識別子が一致するもの    */
       order by 1 asc
```

注

分割条件指定順序は，分割キー内通番から求められます。式を次に示します。

```
N×m-(N-n)
  N:第2次元の分割数
  m:第1分割キーの分割キー内通番
  n:第2分割キーの分割キー内通番
```

付録 H.2　空白変換関数

空白変換関数とは，文字列中の半角空白を全角空白に，又は全角空白を半角空白に変換する関数です。文字列データをデータベースに格納しなくても（空白変換レベルを指定してデータベースに格納してなくても），変換結果が分かるので，次のような場合に使用すると有効です。

- 表をキーレンジ分割するための分割キーを決定するときに，格納するデータが均等に分割されるかどうかを評価する場合
- データベース作成ユティリティで，キーレンジ分割した表へ RD エリア単位に並列にデータロードする場合，RD エリアごとに入力データファイルを作成する場合

空白変換関数を使用する場合の前提条件

表分割ハッシュ関数と同じです。詳細については，「表分割ハッシュ関数を使用する場合の前提条件」を参照してください。

空白変換関数を使用した UAP の作成，実行

表分割ハッシュ関数と同じです。詳細については，「表分割ハッシュ関数を使用した UAP の作成，実行」を参照してください。

(1)　空白変換関数の詳細

(a)　記述構成

記述構成については，「記述構成」を参照してください。

(b)　空白変換関数（p_rdb_conv_space）

機能

指定した変換種別に従って，次のように空白変換をします。

半角空白→全角空白の場合：

文字列中の半角空白 2 バイトを全角空白 1 文字に変換します。

全角空白→半角空白の場合：

文字列中の全角空白 1 文字を半角空白 2 バイトに変換します。

srcp が示す文字列中の空白文字に対して変換処理をします。変換結果は destp に格納されます。

引数 stype，flags の組み合わせと変換内容を次に示します。

stype（データ型）	flags（変換種別）	
	半角空白→全角空白	全角空白→半角空白
NCHAR NVARCHAR	先頭から 2 バイトずつチェックをして，半角空白が二つ連続であった場合，全角空白※に変換します。 この場合，半角空白が 1 文字だけある場合は変換しません。	先頭から 2 バイトずつチェックをして，全角空白※があった場合に半角空白に変換します。
MCHAR MVARCHAR	エラーになります。	先頭から文字コードを意識してチェックし，全角空白※があった場合に半角空白に変換します。

注※　引数 ncspace で指定した値を全角空白の文字コードとして扱います。

ヘッダファイル

#include<pdauxcnv.h>

空白変換機能を使用する場合に必ず指定します。

#include<pdbsqlda.h>

データ型コードを設定する場合，マクロ（PDSQL_で始まる名称のマクロ）を使用するときに指定します。ディクショナリ表からデータ型コードを検索して設定する場合には必要ありません。

形式

```
int p_rdb_conv_space(char           *srcp,
                     unsigned char  stype,
                     unsigned int   srcl,
                     char           *destp,
                     unsigned char  ncspace[2],
                     int            flags);
```

引数

srcp（入力）

文字列格納領域の先頭アドレスを指定します。

stype（入力）

変換前のデータ型を指定します。指定できるデータ型を次に示します。

マクロ名	データ型
PDSQL_NCHAR	NCHAR
PDSQL_NVARCHAR	NVARCHAR
PDSQL_MCHAR	MCHAR
PDSQL_MVARCHAR	MVARCHAR

srcl（入力）

srcp で指定した文字列の長さを指定します。可変長文字列の場合は，実際に srcp が示す領域に格納されている文字列の長さ（単位：バイト）を指定します。

destp（出力）

変換後の文字列格納領域の先頭アドレスが設定されます。destp が示す領域は空白変換関数の呼び出し側で確保してください。

ncspace（入力）

HiRDB サーバで使用する各国文字コードの全角空白文字を，2 バイトの領域で指定します。ncspace に指定する全角空白文字については，表「ncspace に指定する全角空白文字」を参照してください。

flags（入力）

変換種別を指定します。変換種別を次に示します。

マクロ名	変換種別
p_rdb_HALF_TO_FULL_SPACE	半角空白→全角空白
p_rdb_FULL_TO_HALF_SPACE	全角空白→半角空白

戻り値

データ型：int

p_rdb_RC_RTRN(0)

正常終了

p_rdb_RC_ERRINVF(-8)

flags 引数不正

p_rdb_RC_ERRTYPC(-9)

データ型不正

注意事項

注意事項を次に示します。

1. 全角空白→半角空白の空白変換は，OS が提供する母国語サポート機能（NLS）を使用して実現しています。したがって，空白変換関数を呼び出す前に，setlocale 関数で LC_CTYPE カテゴリ，又は LC_ALL カテゴリに適切なロケールを設定しておく必要があります。また，Windows の UAP，文字

コード種別が SJIS の Linux の UAP，又は文字コード種別が CHINESE の UAP の場合は，空白変換関数を呼び出す前に，p_rdb_set_lang 関数を呼び出す必要があります。p_rdb_set_lang 関数については，「文字コード種別設定関数」を参照してください。

引数 srcp が示す文字列の文字コード種別と，setlocale 関数又は p_rdb_set_lang 関数で指定したロケールが矛盾する場合，動作の保証はされません。

2. データの入力領域と出力領域が同一の場合，又は出力領域が入力領域の前にあり，出力領域の後半部と入力領域の前半部が重複している場合は，動作は保証されます。
3. 文字列の長さに関するエラーチェックはしないので，srcl には妥当な値を設定しておく必要があります。
4. 半角空白の文字コードは 0x20，全角空白の文字コードは引数 ncspace に設定した文字コードを使用します。
5. 入力に設定できるデータ型は，NCHAR，NVARCHAR，MCHAR，及び MVARCHAR です。
6. 可変長文字列の場合は，変換する文字列の長さとして srcl を参照します。また，srcl には実長部を除いた長さを指定してください。
7. 可変長文字列の実長部は，空白変換後でも変わりません。
8. 文字コード種別が Unicode の場合，この関数の動作は保証されません。文字コード種別が Unicode の場合は p_rdb_conv_space_utf8 を使用してください。p_rdb_conv_space_utf8 については，「空白変換関数（p_rdb_conv_space_utf8）」を参照してください。

(c)　空白変換関数（p_rdb_conv_space_utf8）

機能

文字コードが Unicode（UTF-8）の場合に，全角空白を半角空白に変換する関数です。文字列中の全角空白 1 文字を半角空白 2 文字に変換します。

srcp が示す文字列中の空白文字に対して変換処理をします。変換結果は destp に格納され，変換後の文字列の長さが destl に格納されます。

引数 stype，flags の組み合わせと変換内容を次に示します。

stype（データ型）	flags（変換種別）	
	半角空白→全角空白	全角空白→半角空白
MCHAR	エラーになります。	先頭から文字コードを意識してチェックし，全角空白※があった場合に半角空白に変換します。
MVARCHAR		

注※　0xE38080 を全角空白の文字コードとして扱います。

ヘッダファイル

#include <pdauxcnv.h>

空白変換関数を使用する場合に必ず指定する。

#include <pdbsqlda.h>

データ型コードを設定する場合，マクロ（PDSQL_で始まるマクロ）を使用して設定するときに指定します。ディクショナリ表からデータ型コードを検索して設定する場合には必要ありません。

形式

```
int p_rdb_conv_space_utf8(char          *srcp,
                  unsigned char         stype,
                  unsigned int          srcl,
                  char                  *destp,
                  unsigned int          *destl,
                  int                   flags) ;
```

引数

srcp（入力）

文字列格納領域の先頭アドレスを指定します。

stype（入力）

変換前のデータ型を指定します。指定できるデータ型を次に示します。

マクロ名	データ型
PDSQL_MCHAR	MCHAR
PDSQL_MVARCHAR	MVARCHAR

srcl（入力）

srcp で指定した文字列の長さを指定します。可変長文字列の場合は，実際に srcp が示す領域に格納されている文字列の長さ（単位：バイト）を指定します。

destp（出力）

変換後の文字列格納領域の先頭アドレスが設定されます。destp が示す領域は空白変換関数の呼び出し側で確保してください。

destl（出力）

destp で指定した文字列の長さが設定されます。可変長文字列の場合は，実際に destp が示す領域に格納されている文字列の長さ（単位：バイト）が設定されます。

flags（入力）

変換種別を指定します。変換種別を次に示します。

マクロ名	変換種別
p_rdb_FULL_TO_HALF_SPACE	全角空白→半角空白

戻り値

データ型：int

p_rdb_RC_RTRN(0)

正常終了

p_rdb_RC_ERRINVF(-8)

flags 引数不正

p_rdb_RC_ERRTYPC(-9)

データ型不正

注意事項

1. この関数は，Unicode（UTF-8）専用の空白変換関数です。
2. この関数を呼び出す前に，必ず引数 lang に UTF8 を設定して p_rdb_set_lang 関数を呼び出してください。p_rdb_set_lang 関数については，「文字コード種別設定関数」を参照してください。
3. データの入力領域と出力領域が同一の場合，又は出力領域が入力領域の前にあり，出力領域の後半部と入力領域の前半部が重複している場合は，動作が保証されます。
4. 文字列の長さに関するエラーチェックはされないため，srcl には妥当な値を入れておく必要があります。
5. 半角空白コードは 0x20，全角空白コードは 0xE38080 を使用します。
6. 入力に設定できるデータ型は，MCHAR 及び MVARCHAR です。
7. 可変長文字列の場合，変換する文字列の長さとして srcl を参照します。また，srcl には実長部を除いた長さを指定してください。
8. 空白変換がされると，全角空白 1 文字（3 バイト）が半角空白 2 文字（2 バイト）に変換されるため，変換後の文字列の長さは変換前の文字列の長さより短くなります。
9. 可変長文字列の場合，変換後の文字列格納領域の実長部には，変換後のデータの長さが設定されます。
10. 変換後の文字列格納領域内のデータは，destl の長さ分だけ保証されます。
11. Unicode（UTF-8）以外の文字コードで，この関数を呼び出した場合の動作は保証されません。

付録 H.3　DECIMAL 型符号正規化関数

DECIMAL 型符号正規化関数とは，DECIMAL 型データの符号を X'C'，又は X'D'に統一（値が 0 の場合は X'C'に統一）する関数です。DECIMAL 型データをデータベースに格納しなくても符号を正規化した結果が分かるので，次のような場合に使用すると有効です。

- システム定義 pd_dec_sign_normalize=Y の場合，表をキーレンジ分割するための分割キーを決定するときに，格納するデータが均等に分割されるかどうかを評価する場合
- システム定義 pd_dec_sign_normalize=Y の場合，データベース作成ユティリティで，キーレンジ分割した表へ RD エリア単位に並列にデータロードする場合，RD エリアごとに入力データファイルを作成する場合

DECIMAL 型符号正規化関数を使用する場合の前提条件

表分割ハッシュ関数と同じです。詳細については，「表分割ハッシュ関数を使用する場合の前提条件」を参照してください。

DECIMAL 型符号正規化関数を使用した UAP の作成，実行

表分割ハッシュ関数と同じです。詳細については，「表分割ハッシュ関数を使用した UAP の作成，実行」を参照してください。

(1)　DECIMAL 型符号正規化関数の詳細

(a)　記述構成

記述構成については，「記述構成」を参照してください。

(b)　DECIMAL 型符号正規化関数（p_rdb_dec_sign_norm）

機能

DECIMAL 型データの符号を正規化します。

srcp が示す DECIMAL 型データの符号部を，次のように正規化します。

正規化前	正規化後
X'A'	X'C'
X'B'	X'D'※
X'C'	X'C'
X'D'	X'D'※
X'E'	X'C'
X'F'	X'C'
X'0'～X'9'	エラー

注※　データの絶対値が 0 の場合，符号部は X'C'に統一されます。

ヘッダファイル

#include<pdauxcnv.h>

DECIMAL 型符号正規化関数を使用する場合に必ず指定します。

形式

```
int p_rdb_dec_sign_norm(unsigned char   *srcp,
                        short           srcl,
                        unsigned char   *destp);
```

引数

srcp（入力）

正規化する DECIMAL 型データの先頭アドレスを指定します。

srcl（入力）

引数 srcp が示す DECIMAL データのデータ長コードを指定します。指定できるデータ長コードを次に示します。

データ型	データ長コード
INTERVAL YEAR TO DAY	8×256
INTERVAL HOUR TO SECOND	6×256
DECIMAL[(p[,q])]	p×256 ＋ q （p を省略した場合は 15 を仮定，q を省略した場合は 0 を仮定）

destp（出力）

正規化後の DECIMAL 型データが格納されます。destp が示す領域は DECIMAL 型符号正規化関数の呼び出し側で確保してください。

戻り値

データ型：int

p_rdb_RC_RTRN(0)

正常終了

p_rdb_RC_ERRDFRM(-12)

データの符号部が不正

注意事項

1. DECIMAL 型データの符号部以外はエラーチェックされません。したがって，不正な DECIMAL 型データの場合，又は引数 srcl で設定したデータ長コードが DECIMAL 型データと矛盾する場合は，動作は保証されません。
2. データの入力領域と出力領域が同一の場合，又は出力領域が入力領域の前にあり，出力領域の後半部と入力領域の前半部が重複している場合は，動作は保証されます。

付録 H.4　文字コード種別設定関数

文字コード種別設定関数とは，表分割ハッシュ関数，及び空白変換関数に対して，UAP から文字コード種別を通知するために使用する関数です。

この関数で文字コードの種別を設定すると，表分割ハッシュ関数，及び空白変換関数など，文字コード種別に依存する処理が実行できます。

文字コード種別設定関数を使用する場合の前提条件

表分割ハッシュ関数と同じです。詳細については，「表分割ハッシュ関数を使用する場合の前提条件」を参照してください。

文字コード種別設定関数を使用した UAP の作成，実行

表分割ハッシュ関数と同じです。詳細については，「表分割ハッシュ関数を使用した UAP の作成，実行」を参照してください。

(1)　文字コード種別設定関数の詳細

(a)　記述構成

記述構成については，「記述構成」を参照してください。

(b)　文字コード種別設定関数 p_rdb_set_lang

機能

表分割ハッシュ関数，及び空白変換関数が扱う文字コードの種別を設定します。

ヘッダファイル

#include<pdauxcnv.h>

文字コード種別設定関数を使用する場合に必ず指定します。

形式

```
int p_rdb_set_lang(char *lang);
```

引数

lang（入力）

表分割ハッシュ関数，及び空白変換関数が扱う文字コードの種別を設定します。

この引数には，次の文字コードが指定できます。

文字コード種別	引数 lang の指定値
シフト JIS 漢字コード※1	"SJIS"
EUC 中国語漢字コード	"CHINESE"
単一バイト文字コード※2	"C"
Unicode（UTF-8）	"UTF8"
Unicode（IVS 対応 UTF-8）	"UTF8"
中国語漢字コード（GB18030）	"GB18030"

注※1　Linux，及び Windows の場合に指定できます。

注※2　Windows の場合に指定できます。

空の文字列を指定した場合（例：p_rdb_set_lang ("")），次のような動作となります。

- UNIX 環境の場合

  この関数の前に実行した setlocale 関数で，LC_ALL カテゴリに設定したロケールに対応する文字コード種別が設定されます。setlocale 関数を実行していない場合は，LC_ALL カテゴリのデフォルトのロケールに対応する文字コード種別が設定されます。

- Windows 環境の場合

  OS のデフォルトとなる文字コード種別が設定されます。ただし，OS のデフォルトを上記の表以外の文字コード種別に設定している場合は，動作が保証されません。

戻り値

データ型：int

p_rdb_RC_RTRN(0)

正常終了

p_rdb_RC_ERRIVLG(-10)

文字コード種別不正

注意事項

1. p_rdb_set_lang は，次のどれかに該当する場合，必ず実行してください。

   ・Windows 環境の UAP で文字コード種別を設定するとき

   ・UNIX 環境の UAP で p_rdb_conv_space_utf8 を呼び出すとき※

   ・Linux 環境の UAP で，文字コード種別を SJIS に設定するとき

   ・UNIX 環境の UAP で，文字コード種別を CHINESE に設定するとき

   注※

   p_rdb_conv_space_utf8 を呼び出す前に p_rdb_set_lang を実行してください。ただし，空白変換関数 p_rdb_conv_space を呼び出す場合は，この関数ではなく，OS 提供の関数 setlocale を使用してください。

2. UNIX 環境の場合，この関数で文字コード種別を設定した後に，使用できない文字コード種別を使用して別の関数を使用するときは，p_rdb_set_lang("")を発行してから，setlocale 関数を呼び出して適切な文字コード種別を設定し直す必要があります。

付録 I　エスケープ句で指定できるスカラ関数

エスケープ句で指定できるスカラ関数を次の表に示します。

表 I–1　エスケープ句で指定できるスカラ関数

スカラ関数	スカラ関数の標準形式	形式変換※1		
		Type2	Type4	Type4（XDM/RD E2 接続）
数学関数	ABS(number)	無	無	無
	ACOS(float)	無	MASTER.ACOS(float)	無
	ASIN(float)	無	MASTER.ASIN(float)	無
	ATAN(float)	無	MASTER.ATAN(float)	無
	ATAN2(float1, float2)	無	MASTER.ATAN2(float1, float2)	無
	CEILING(number)※2	CEIL(number) 08-02	MASTER.CEIL(number)	無
	COS(float)	無	MASTER.COS(float)	無
	COT(float)※3，※4	無	無	無
	DEGREES(number)	無	MASTER.DEGREES(number)	無
	EXP(float)	無	MASTER.EXP(float)	無
	FLOOR(number)	無	MASTER.FLOOR(number)	無
	LOG(float)※2	LN(float) 08-02	MASTER.LN(float)	LN(float)
	LOG10(float)	無	MASTER.LOG10(float)	無
	MOD(integer1, integer2)	無	無	無
	PI()	無	MASTER.PI()	無
	POWER(number, power)	無	MASTER.POWER(number, power)	無
	RADIANS(number)	無	MASTER.RADIANS(number)	無
	RAND(integer)※3，※4	無	無	無
	ROUND(number, places)	無	MASTER.ROUND(number, places)	無
	SIGN(number)	無	MASTER.SIGN(number)	無
	SIN(float)	無	MASTER.SIN(float)	無
	SQRT(float)	無	MASTER.SQRT(float)	無

スカラ関数	スカラ関数の標準形式	形式変換※1		
		Type2	Type4	Type4（XDM/RD E2 接続）
	TAN(float)	無	MASTER.TAN(float)	無
	TRUNCATE(number, places)※2	TRUNC(number, places) 08-02	MASTER.TRUNC(number, places)	無
文字列関数	ASCII(string)	無	MASTER.ASCII(string)	無
	BIT_LENGTH(string)※3	無	無	無
	CHAR(code)※2	CHR(code) 08-02	MASTER.CHR(code)	無
	CHAR〔ACTER〕_LENGTH(string)※3	無	無	無
	CONCAT(string1, string2)※3	無	無	無
	DIFFERENCE(string1, string2)※3	無	無	無
	INSERT(string1, start, length, string2)※2	INSERTSTR(string1, start, length, string2) 08-02	MASTER.INSERTSTR(string1, start, length, string2)	無
	LCASE(string)※2	LOWER(string) 08-02	同左	同左
	LEFT(string, count)※2	LEFTSTR(string, count) 08-02	MASTER.LEFTSTR(string, count)	無
	LENGTH(string)	無	無	無
	LOCATE(string1, string2〔, start〕)※2	POSITION(string1 IN string2〔FROM start〕) 08-02	同左	同左
	LTRIM(string)	無	MASTER.LTRIM(string)	TRIM(LEADING FROM string)
	OCTET_LENGTH(string)※3	無	無	無
	POSITION(character IN character)	無	無	無

スカラ関数	スカラ関数の標準形式	形式変換※1		
		Type2	Type4	Type4（XDM/RD E2 接続）
	REPEAT(string, count)※3	無	無	無
	REPLACE(string1, string2, string3)	無	MASTER.REPLACE(string1, string2, string3)	無
	RIGHT(string, count)※2	RIGHTSTR(string, count) 08-02	MASTER.RIGHTSTR(string, count)	無
	RTRIM(string)	無	MASTER.RTRIM(string)	TRIM(TRAILING FROM string)
	SOUNDEX(string)※3	無	無	無
	SPACE(count)※3	無	無	無
	SUBSTRING(string, start, length)※2	SUBSTR(string, start, length)	同左	同左
	UCASE(string)※2	UPPER(string)	同左	同左
時刻と日付の関数	CURDATE()※2	CURRENT DATE	同左	同左
	CURRENT_DATE()※2	CURRENT DATE	同左	同左
	CURTIME()※2	CURRENT TIME	同左	同左
	CURRENT_TIME	無	無	無
	CURRENT_TIME(time-precision)※2，※5 引数 time-precision は，戻り値の小数秒の精度を指定する。	CURRENT TIME 08-02	同左	無
	CURRENT_TIMESTAMP[(timestamp-precision)] 引数 timestamp-precision は，返されるタイムスタンプの小数秒の精度を指定する。	無	無	無
	DAYNAME(date)	無	MASTER.DAYNAME(date)	無
	DAYOFMONTH(date)※3	無	無	無
	DAYOFWEEK(date)	無	MASTER.DAYOFWEEK(date)	無
	DAYOFYEAR(date)	無	MASTER.DAYOFYEAR(date)	無

スカラ関数	スカラ関数の標準形式	形式変換※1		
		Type2	Type4	Type4（XDM/RD E2 接続）
	EXTRACT(extract-field FROM extract-source)※3	無	無	無
	HOUR(time)	無	無	無
	MINUTE(time)	無	無	無
	MONTH(time)	無	無	無
	MONTHNAME(date)	無	MASTER.MONTHNAME(date)	無
	NOW()※2	CURRENT TIMESTAMP(6) 08-02	同左	同左
	QUARTER(date)	無	MASTER.QUARTER(date)	無
	SECOND(time)	無	無	無
	TIMESTAMPADD(interval, count, timestamp)※3	無	無	無
	TIMESTAMPDIFF(interval, timestamp1, timestamp2)※3	無	無	無
	WEEK(date)	無	MASTER.WEEK(date)	無
	YEAR(date)	無	無	無
システム関数	DATABASE()※3	無	無	無
	IFNULL(expression, value)※3	無	無	無
	USER()※4	USER 08-02	同左	同左
データ型変換関数	CONVERT(value, SQLtype)※2，※4	無	無	無

注※1

Statement オブジェクトのエスケープ構文解析での，スカラ関数変換後の形式を示します。関数変換しない場合は，「無」と示します。また，xx-xx は追加したバージョンを示します。

注※2

標準形式と HiRDB 形式，又は XDM/RD E2 形式が異なります。

注※3

HiRDB，又は XDM/RD E2 では該当するスカラ関数がありません。

注※4

HiRDB，又は XDM/RD E2 で未サポートの関数のため，スカラ関数（標準形式）をエスケープ構文に指定すると HiRDB サーバ，又は XDM/RD E2 でエラーになります。また，xx-xx は Type2 JDBC ドライバに追加したバージョンを示します。

注※5

Type4 JDBC ドライバでは，CURRENT TIME に変換するため，小数秒の精度=0 として扱います。引数で指定された秒の精度は無効になります。

付録 J　文字集合を使用した場合の文字コード変換規則

文字集合を使用した場合の文字コード変換規則について説明します。

付録 J.1　シフト JIS 漢字コードを EBCDIK に変換する場合

シフト JIS 漢字コードを EBCDIK に変換する場合について次に示します。

下位	上位															
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	[NUL]	[DLE]	空白	0	@	P	`	p				ｰ	ﾀ	ﾐ		
	0	10	20	30	40	50	60	70	80	90	A0	B0	C0	D0	E0	F0
	0	10	40	F0	7C	D7	79	76	20	30	57	58	91	A5	B2	DC
1	[SOH]	[DC1]	!	1	A	Q	a	q			。	ｱ	ﾁ	ﾑ		
	1	11	21	31	41	51	61	71	81	91	A1	B1	C1	D1	E1	F1
	1	11	4F	F1	C1	D8	59	77	21	31	41	81	92	A6	B3	DD
2	[STX]	[DC2]	"	2	B	R	b	r			「	ｲ	ﾂ	ﾒ		
	2	12	22	32	42	52	62	72	82	92	A2	B2	C2	D2	E2	F2
	2	12	7F	F2	C2	D9	62	78	22	1A	42	82	93	A7	B4	DE
3	[ETX]	[DC3]	#	3	C	S	c	s			」	ｳ	ﾃ	ﾓ		
	3	13	23	33	43	53	63	73	83	93	A3	B3	C3	D3	E3	F3
	3	13	7B	F3	C3	E2	63	80	23	33	43	83	94	A8	B5	DF
4	[EOT]	[DC4]	$	4	D	T	d	t			、	ｴ	ﾄ	ﾔ		
	4	14	24	34	44	54	64	74	84	94	A4	B4	C4	D4	E4	F4
	37	3C	E0	F4	C4	E3	64	8B	24	34	44	84	95	A9	B6	EA
5	[ENQ]	[NAK]	%	5	E	U	e	u			・	ｵ	ﾅ	ﾕ		
	5	15	25	35	45	55	65	75	85	95	A5	B5	C5	D5	E5	F5
	2D	3D	6C	F5	C5	E4	65	9B	25	35	45	85	96	AA	B7	EB
6	[ACK]	[SYN]	&	6	F	V	f	v			ヲ	ｶ	ﾆ	ﾖ		
	6	16	26	36	46	56	66	76	86	96	A6	B6	C6	D6	E6	F6
	2E	32	50	F6	C6	E5	66	9C	6	36	46	86	97	AC	B8	EC
7	[BEL]	[ETB]	'	7	G	W	g	w			ｧ	ｷ	ﾇ	ﾗ		
	7	17	27	37	47	57	67	77	87	97	A7	B7	C7	D7	E7	F7
	2F	26	7D	F7	C7	E6	67	A0	17	8	47	87	98	AD	B9	ED
8	[BS]	[CAN]	(	8	H	X	h	x			ｨ	ｸ	ﾈ	ﾘ		
	8	18	28	38	48	58	68	78	88	98	A8	B8	C8	D8	E8	F8
	16	18	4D	F8	C8	E7	68	AB	28	38	48	88	99	AE	CA	EE
9	[HT]	[EM]	)	9	I	Y	i	y			ｩ	ｹ	ﾉ	ﾙ		
	9	19	29	39	49	59	69	79	89	99	A9	B9	C9	D9	E9	F9
	5	19	5D	F9	C9	E8	69	B0	29	39	49	89	9A	AF	CB	EF
A	[NL]	[SUB]	*	:	J	Z	j	z			ｪ	ｺ	ﾊ	ﾚ		
	0A	1A	2A	3A	4A	5A	6A	7A	8A	9A	AA	BA	CA	DA	EA	FA
	15	3F	5C	7A	D1	E9	70	B1	2A	3A	51	8A	9D	BA	CC	FA
B	[VT]	[ESC]	+	;	K	[	k	{			ｫ	ｻ	ﾋ	ﾛ		
	0B	1B	2B	3B	4B	5B	6B	7B	8B	9B	AB	BB	CB	DB	EB	FB
	0B	27	4E	5E	D2	4A	71	C0	2B	3B	52	8C	9E	BB	CD	FB
C	[FF]	[FS]	,	<	L	\	l	|			ｬ	ｼ	ﾌ	ﾜ		
	0C	1C	2C	3C	4C	5C	6C	7C	8C	9C	AC	BC	CC	DC	EC	FC
	0C	1C	6B	4C	D3	5B	72	6A	2C	4	53	8D	9F	BC	CE	FC
D	[CR]	[GS]	-	=	M	]	m	}			ｭ	ｽ	ﾍ	ﾝ		
	0D	1D	2D	3D	4D	5D	6D	7D	8D	9D	AD	BD	CD	DD	ED	FD
	0D	1D	60	7E	D4	5A	73	D0	9	14	54	8E	A2	BD	CF	FD
E	[SO]	[RS]	.	>	N	^	n	~			ｮ	ｾ	ﾎ	ﾞ		
	0E	1E	2E	3E	4E	5E	6E	7E	8E	9E	AE	BE	CE	DE	EE	FE
	0E	1E	4B	6E	D5	5F	74	A1	0A	3E	55	8F	A3	BE	DA	FE
F	[SI]	[US]	/	?	O	_	o	[DEL]			ｯ	ｿ	ﾏ	ﾟ		E0
	0F	1F	2F	3F	4F	5F	6F	7F	8F	9F	AF	BF	CF	DF	EF	FF
	0F	1F	61	6F	D6	6D	75	7	1B	E1	56	90	A4	BF	DB	FF

＜表の見方＞

1 行目：変換する文字

2 行目：変換前のコード

3 行目：変換後のコード

全角文字（1 バイト目が 0x81～0x9F，0xE0～0xEF，0xF0～0xFC で始まる 2 バイト文字）のコードは，半角文字 2 文字として扱います。

付録 J.2　EBCDIK をシフト JIS 漢字コードに変換する場合

EBCDIK をシフト JIS 漢字コードに変換する場合について次に示します。

下位	上位															
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	[NUL]	[DLE]			空白	&	-	j	s	ソ	w	y	{	}	$	0
	0	10	20	30	40	50	60	70	80	90	A0	B0	C0	D0	E0	F0
	0	10	80	90	20	26	2D	6A	73	BF	77	79	7B	7D	24	30
1	[SOH]	[DC1]			。	ｪ	/	k	ｱ	ﾀ	~	z	A	J	0	1
	1	11	21	31	41	51	61	71	81	91	A1	B1	C1	D1	E1	F1
	1	11	81	91	A1	AA	2F	6B	B1	C0	7E	7A	41	4A	9F	31
2	[STX]	[DC2]		[SYN]	「	ｫ	b	l	ｲ	ﾁ	ヘ		B	K	S	2
	2	12	22	32	42	52	62	72	82	92	A2	B2	C2	D2	E2	F2
	2	12	82	16	A2	AB	62	6C	B2	C1	CD	E0	42	4B	53	32
3	[ETX]	[DC3]			」	ｬ	c	m	ｳ	ﾂ	ホ		C	L	T	3
	3	13	23	33	43	53	63	73	83	93	A3	B3	C3	D3	E3	F3
	3	13	83	93	A3	AC	63	6D	B3	C2	CE	E1	43	4C	54	33
4					、	ｭ	d	n	ｴ	ﾃ	マ		D	M	U	4
	4	14	24	34	44	54	64	74	84	94	A4	B4	C4	D4	E4	F4
	9C	9D	84	94	A4	AD	64	6E	B4	C3	CF	E2	44	4D	55	34
5	[HT]	[NL]			・	ｮ	e	o	ｵ	ﾄ	ミ		E	N	V	5
	5	15	25	35	45	55	65	75	85	95	A5	B5	C5	D5	E5	F5
	9	0A	85	95	A5	AE	65	6F	B5	C4	D0	E3	45	4E	56	35
6		[BS]	[ETB]		ヲ	ｯ	f	p	ｶ	ﾅ	ム		F	O	W	6
	6	16	26	36	46	56	66	76	86	96	A6	B6	C6	D6	E6	F6
	86	8	17	96	A6	AF	66	70	B6	C5	D1	E4	46	4F	57	36
7	[DEL]		[ESC]	[EOT]	ｧ		g	q	ｷ	ﾆ	メ		G	P	X	7
	7	17	27	37	47	57	67	77	87	97	A7	B7	C7	D7	E7	F7
	7F	87	1B	4	A7	A0	67	71	B7	C6	D2	E5	47	50	58	37
8		[CAN]			ｨ	ｰ	h	r	ｸ	ﾇ	モ		H	Q	Y	8
	8	18	28	38	48	58	68	78	88	98	A8	B8	C8	D8	E8	F8
	97	18	88	98	A8	B0	68	72	B8	C7	D3	E6	48	51	59	38
9		[EM]			ｩ	a	i	`	ｹ	ﾈ	ヤ		I	R	Z	9
	9	19	29	39	49	59	69	79	89	99	A9	B9	C9	D9	E9	F9
	8D	19	89	99	A9	61	69	60	B9	C8	D4	E7	49	52	5A	39
A					[	]	|	:	ｺ	ﾉ	ュ	レ				
	0A	1A	2A	3A	4A	5A	6A	7A	8A	9A	AA	BA	CA	DA	EA	FA
	8E	92	8A	9A	5B	5D	7C	3A	BA	C9	D5	DA	E8	EE	F4	FA
B	[VT]				.	\	,	#	t	u	x	ﾛ				
	0B	1B	2B	3B	4B	5B	6B	7B	8B	9B	AB	BB	CB	DB	EB	FB
	0B	8F	8B	9B	2E	5C	2C	23	74	75	78	DB	E9	EF	F5	FB
C	[FF]	[FS]		[DC4]	<	*	%	@	ｻ	v	ﾖ	ﾜ				
	0C	1C	2C	3C	4C	5C	6C	7C	8C	9C	AC	BC	CC	DC	EC	FC
	0C	1C	8C	14	3C	2A	25	40	BB	76	D6	DC	EA	F0	F6	FC
D	[CR]	[GS]	[ENQ]	[NAK]	(	)	_	'	ｼ	ﾊ	ﾗ	ﾝ				
	0D	1D	2D	3D	4D	5D	6D	7D	8D	9D	AD	BD	CD	DD	ED	FD
	0D	1D	5	15	28	29	5F	27	BC	CA	D7	DD	EB	F1	F7	FD
E	[SO]	[RS]	[ACK]		+	;	>	=	ｽ	ﾋ	ﾘ	ﾞ				
	0E	1E	2E	3E	4E	5E	6E	7E	8E	9E	AE	BE	CE	DE	EE	FE
	0E	1E	6	9E	2B	3B	3E	3D	BD	CB	D8	DE	EC	F2	F8	FE
F	[SI]	[US]	[BEL]	[SUB]	!	^	?	"	ｾ	ﾌ	ﾙ	ﾟ				E0
	0F	1F	2F	3F	4F	5F	6F	7F	8F	9F	AF	BF	CF	DF	EF	FF
	0F	1F	7	1A	21	5E	3F	22	BE	CC	D9	DF	ED	F3	F9	FF

＜表の見方＞

1 行目：変換する文字

2 行目：変換前のコード

3 行目：変換後のコード

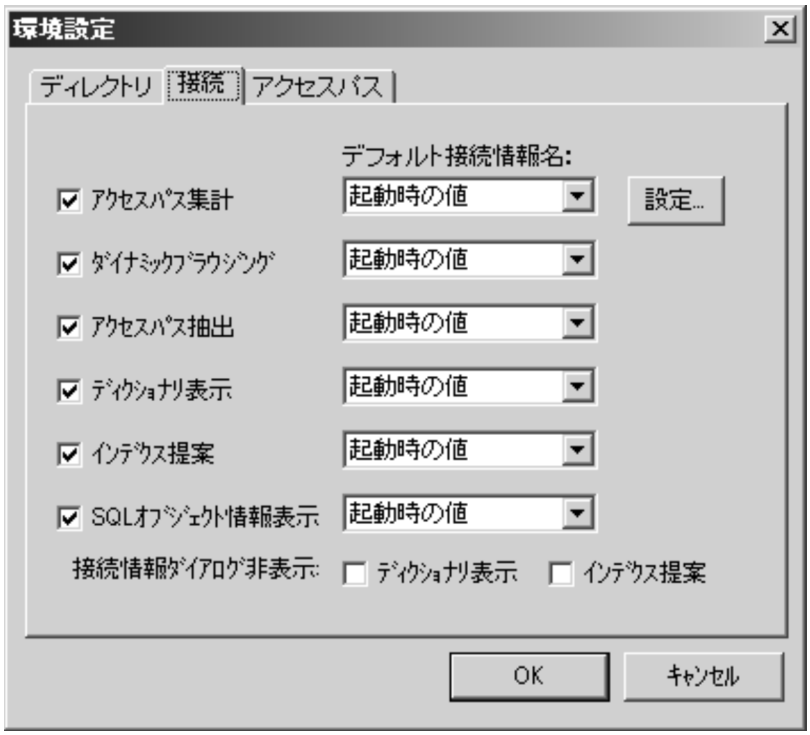
付録 K　HiRDB SQL Tuning Advisor の環境設定

HiRDB SQL Tuning Advisor では，解析処理を行うために，HiRDB サーバに接続して情報を取得します。HiRDB SQL Tuning Advisor を使用するには，接続のための環境設定をしてください。HiRDB SQL Tuning Advisor の環境設定の手順を次に示します。

1. ［スタート］−［プログラム］−［HiRDB SQL Tuning Advisor］−［HiRDB SQL Tuning Advisor］を選択し，HiRDB SQL Tuning Advisor を起動します。

2. ［オプション］メニューから［環境設定］を選択します。
   ［環境設定］画面が表示されます。

3. ［接続］タブを選択します。



4. 接続情報名を登録するため［設定］ボタンをクリックします。
   「接続情報設定」画面が表示されます。
   接続情報は，表の所有者と DBA 権限保持者の 2 種類を登録します。接続する HiRDB サーバの PDHOST，PDNAMEPORT，接続する認可識別子，パスワード（PDUSER），及び接続する HiRDB サーバの文字コード種別を設定します。

5. ［追加］ボタンをクリックします。
   次の画面は，表の所有者（USER1）の接続情報を追加した例です。

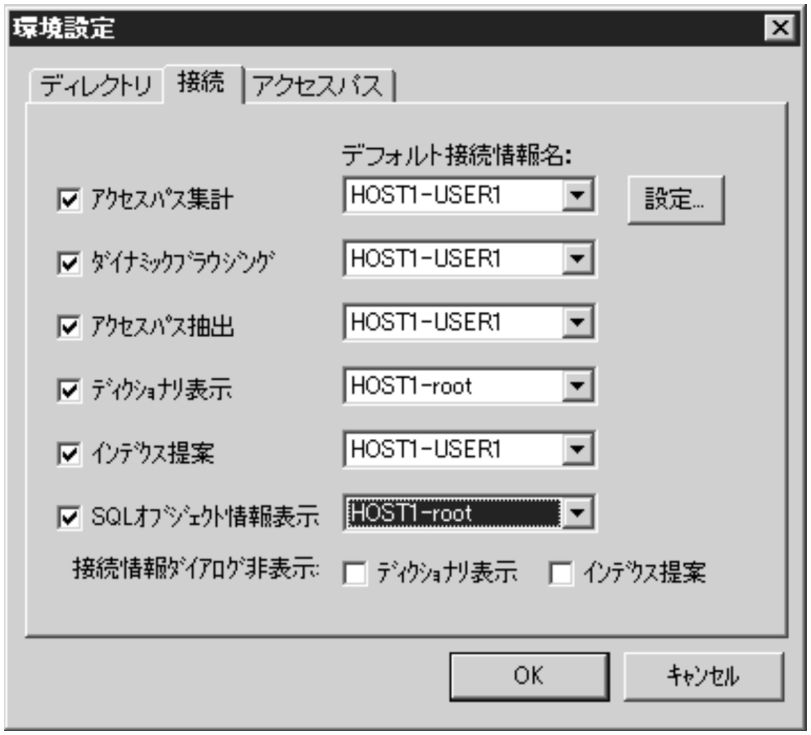


次の画面は，DBA 権限保持者（"root"）の接続情報を追加した例です。



6. すべての情報の追加が完了したら，［OK］ボタンをクリックします。

7. ［環境設定］画面で，各操作の接続情報を設定します。
   - ［ディクショナリ表示］に DBA 権限保持者の接続情報名を設定します。
   - ［SQL オブジェクト情報表示］に DBA 権限保持者の接続情報名を設定します。
   - 上記以外には，表の所有者の接続情報名を設定します。



8. [OK] ボタンをクリックして終了します。

9. [オプション] メニューから [環境設定] を選択します。
   [環境設定] 画面が表示されます。

10. [環境設定] 画面の [アクセスパス] タブを選択し，次の項目を設定します。
    - [接続情報名] に表の所有者の接続情報名を設定します。
    - [中間結果情報] の [データ件数の取得を行なう] をチェックします。



11. [OK] ボタンをクリックして終了します。

付録 L　HiRDB の最大値・最小値

HiRDB の最大値・最小値を次の表に示します。

表 L–1　HiRDB の最大値・最小値

分類	項　　目	最小値	最大値	単位
データベース操作	検索項目数	1	30,000	個
	更新列数	1	30,000	
	ソート列数	1	255	
	グループ化列数	0 又は 1 ※1	255	
	重複排除列数	1	255	
	論理演算のネスト数	0	255	
	IN 述語の右辺の行値構成子数	1	30,000	
	スカラ関数のネスト数	0	255	
	SQL 中の文字列定数，混在文字列定数，16 進文字列定数の長さ	0	32,000	バイト
	SQL 中の各国文字列定数の長さ	0	16,000	字
	1SQL 文の長さ	1	2,000	キロバイト
	1SQL 文中に指定できる表数※2	1	128	個
	1SQL 文中に指定できる相関名の数※3	0	129	
	LOCK 文中の排他制御実表数	1	128	
	CALL 文の引数の数	0	30,000	
	作業表の行長※4	6	32,720	バイト
UAP	1UAP 中に指定する SQL 文数	1	4,095	個
	1UAP 中のカーソル数	0	1,023	
	SQL 文中の?パラメタ数	0	30,000	
	SQL 文中の埋込み変数	0	30,000	

注※1

GROUP BY 句を指定する場合の最小値は 1 です。GROUP BY 句を指定しないで HAVING 句を指定する場合，又は SELECT 句に集合関数を指定する場合の最小値は 0 です。

注※2

更新対象の表は数に含みません。

注※3

更新対象の表に対して指定する相関名は数に含みません。

注※4

SQL 文によって，作業表用ファイルが必要となる場合があります。作業表用ファイルについては，マニュアル「HiRDB システム導入・設計ガイド」を参照してください。

付録 M　UAP のサンプル一覧

UAP のサンプルが掲載されている箇所を次の表に示します。

表 M–1　UAP のサンプル一覧

分類	内容
ストアドルーチンの作成	ストアドプロシジャを使用した採番の例
	SQL ストアドプロシジャの例（定義と呼び出し）
	結果集合返却機能の使用例（SQL ストアドプロシジャの定義と呼び出し）
	SQL ストアドファンクションの例（定義と呼び出し）
	外部 Java ストアドルーチンのプログラム例
	結果集合返却機能の使用例（Java ストアドプロシジャの定義と呼び出し）
	外部 C ストアドルーチンのプログラム例
配列の使用	配列を使用した FETCH 機能の使用例
	配列を使用した INSERT 機能の使用例
	配列を使用した UPDATE 機能の使用例
	配列を使用した DELETE 機能の使用例
複数接続機能の使用	複数接続機能のコーディング例
位置付け子機能の使用	位置付け子機能の使用例
C 言語での UAP の作成	基本的な操作の例
	ユーザ定義の SQL 記述領域を使用した例
	LOB データを操作する例
	埋込み SQL 宣言節の不要化の使用例
	C 言語の構造体の使用例
	ハッシュ分割のコーディング例
COBOL 言語での UAP の作成	基本的な操作の例
	行インタフェースを使用した例
	TYPE 句，TYPEDEF 句，及び SAME AS 句を使用した例
UAP からのコマンド実行	COMMAND EXECUTE からコマンドを実行するための準備（データロードを実行する UAP の例）
HiRDB データプロバイダ for .NET Framework の使用	データベースへの接続
	SQL 文の実行
	トランザクションの実行

分類	内容
	検索文の実行
	配列を使用した INSERT 機能の実行
	繰返し列の実行
	SQL 文のエラー判定とエラー情報の取得
DataSource と JNDI の使用（Type2 JDBC ドライバ）	JNDI への DataSource の登録
	JNDI からの DataSource の取得
	DB 接続
DataSource と JNDI の使用（Type4 JDBC ドライバ）	JNDI への DataSource の登録
	JNDI からの DataSource の取得
	DB 接続
JDBC ドライバの使用	JDBC ドライバを使用した UAP 例
SQLJ の使用	HiRDB サーバとの接続，切り離しの記述
	カーソルによる検索の記述
	動的結果セットの受け取り
	JDBC との相互運用
	ネイティブインタフェースを使用したコーディング例

付録 N　メインフレームからのマイグレーション

付録 N.1　次の行取り出しで排他を解除する機能

VOSK システムからマイグレーションするときに必要となる機能です。VOSK システムからマイグレーションするときだけ，この機能を使用してください。通常はこの機能を使用しないでください。

(1)　概要

次の行取り出しで排他を解除する機能を適用すると，カーソルを使用した検索で，更新のない行の排他を解除します。これによって，更新のなかった行をほかのユーザが参照，更新できるため，トランザクションの同時実行性が向上します。

(2)　前提条件

次の行取り出しで排他を解除する機能は，HiRDB/シングルサーバ限定の機能です。HiRDB/パラレルサーバでは使用できません。

1. HiRDB サーバ以外で UAP を実行する場合，次の PP が必要です。
   HiRDB/Run Time Version 6 06-01 以降（Windows の場合は HiRDB/Run Time Version 6 06-00-/B 以降）
2. HiRDB サーバ以外で UAP を開発する場合，次の PP が必要です。
   HiRDB/Developer's Kit Version 6 06-01 以降（Windows の場合は HiRDB/Developer's Kit Version 6 06-00-/B 以降）
3. UAP については，「注意事項」に記載のすべての注意事項について問題ないことを確認してください。

(3)　指定方法

1. UAP 中の SQL 文に適用する場合
   クライアント環境変数 PDISLLVL に 12 を指定します。
2. ストアドプロシジャ中の SQL 文に適用する場合
   CREATE PROCEDURE 文，ALTER PROCEDURE 文，又は ALTER ROUTINE 文の ISOLATION 指定で 12 を指定します。

(4)　機能

SQL 文のデータ保証レベルを 12 に指定します。

データ保証レベルに 12 を指定すると，カーソルを使用する検索の場合，カーソルが位置づけられている間，排他を保持します。次の行を取り出すとき，又はカーソルを閉じるときに，更新が行われていない一つ前の行の排他を解除します。これによって，カーソルの現在位置の行は，ほかのユーザが更新することを許可しません。さらに，次の行の取り出し，又はカーソルクローズの後，検索済みの更新のない行を，

ほかのユーザが更新できます。カーソルを使用しない検索の場合，検索が終了していれば，トランザクションが終了していなくてもほかのユーザからの参照および更新を許可します。

(5)　注意事項

1. データ保証レベルの指定よりも，SQL 文中の排他オプションの指定が優先されます。データ保証レベルについては，「データ保証レベル」を参照してください。排他オプションについては，マニュアル「HiRDB SQL リファレンス」の「排他オプション」を参照してください。
2. 次の条件に該当する場合は，データ保証レベル 2 相当の動作になります。
   - 配列を使用した FETCH 機能を利用する場合
3. ブロック転送機能は無効になります
4. この機能の実行中に，作業表を作成する SQL 文を実行しても，行の排他が解放されません。そのため，更新のない行でもほかのユーザが更新できません。

   なお，UAP が COBOL85 の「RDB ファイル入出力機能」又は COBOL2002 の「HiRDB による索引ファイル入出力機能」を利用する場合，次の条件をすべて満たしているときは作業表を作成しません。
   - インデクスキー値無排他機能を使用している

     システム定義でインデクスキー値無排他機能の利用を指定するか，又はシステム定義でインデクスキー値無排他機能を省略すれば，デフォルト設定でインデクスキー値無排他機能が適用されます。
   - クライアント環境変数又はシステム定義で，SQL 最適化オプションに「更新 SQL の作業表作成抑止機能」を指定している
   - キー項目をクラスタキー定義している，又はインデクスを定義し，最適化情報収集ユティリティ（pdgetcst）によってインデクスシーケンシャル度（SEQ_RATIO）を 100 としてインデクスの最適化情報を登録している

     クラスタキーを定義する場合は，表を横分割していないこと，インデクスを定義する場合は，インデクスを横分割していないことが前提条件となります。インデクスを横分割しないためには，表を横分割しないか，又は表を横分割する場合はキー項目を分割キーにしないようにする必要があります。
5. 次のどれかの条件を満たしている場合，行の排他又はほかの排他資源が解放されず，ほかのユーザが更新できないことがあります。
   - SQL 最適化オプションに「グループ分け高速化処理」を指定し，かつ「グループ分け高速化処理」対象の SQL 文を実行する

     「グループ分け高速化処理」対象の SQL 文については，「グループ分け高速化機能」を参照してください。
   - 選択式に COUNT(*)，COUNT(列指定)，COUNT_FLOAT(*)，COUNT_FLOAT (列指定)，SUM(列指定)，及び AVG(列指定)だけを指定し，かつ GROUP BY 句を指定しない SQL 文を実行する
   - 最小排他資源単位がページ単位に定義されている表（表オプションに LOCK PAGE を指定して定義した表）を検索する
   - LOCK TABLE によって排他を取得した表を検索する

- BLOB 型又は定義長 32,001 バイト以上の BINARY 型の列をユーザ定義関数（システム定義スカラ関数を含む）の引数に指定する
- プラグイン提供関数を指定する
- 抽象データ型の列又は抽象データ型の列のコンポネント指定を指定する
- クライアント環境変数 PDDBLOG に NO を指定し，FOR UPDATE 指定のある SQL を実行する
- プラグイン提供関数を使用した SQL を実行する
- アクセスパスで，表の結合方式がキースキャンマージジョインになる SQL を実行する

6. 上記 4 及び 5 に該当しない場合で，かつ GROUP BY 句を含む SQL を実行する場合，行取り出し時に次のグループに含まれる一行に対して排他が掛かります。この排他は次の行取り出し時に解除されます。
7. 上記 4 及び 5 に該当しない場合で，かつ GROUP BY 句を含まないで，集合関数又は HAVING 句を含む SQL を実行する場合，検索結果の行を取り出した時点で行排他は掛かりません。
8. インデクスキー値排他を適用する場合，カーソルの現在位置の行のインデクスキー値に排他が掛かります。そのため，ほかのトランザクションで同じキー値を持つ行を更新できないことがあります。

索引

数字

1 行 SELECT 文　121
1 行検索　51
1 個の表からの検索　52
2 個の表からの検索　56
64 ビットモードでの UAP の作成　760

A

absolute(int row)〔ResultSet インタフェース〕 1554
acceptsURL(String url)〔Driver インタフェース〕 1369
addBatch()〔PreparedStatement インタフェース〕 1445
addBatch(String sql)〔Statement インタフェース〕 1409
addConnectionEventListener(ConnectionEventListener listener)〔PooledConnection インタフェース〕 1817
addStatementEventListener (StatementEventListener listener)〔PooledConnection インタフェース〕　1819
ADO.NET 対応アプリケーションプログラムからの HiRDB アクセス　1112
afterLast()〔ResultSet インタフェース〕　1556
allProceduresAreCallable()〔DatabaseMetaData インタフェース〕　1642
allTablesAreSelectable()〔DatabaseMetaData インタフェース〕　1643
AND　75
AND PLURAL INDEXES SCAN〔SQL の最適化〕　321
AND の複数インデクス利用〔SQL の最適化〕　318
AND の複数インデクス利用の抑止〔SQL 最適化オプション〕　617
Array インタフェース〔JDBC1.2 コア API〕　1790
Array クラス〔Type2 JDBC ドライバ〕　1278, 1333
autoCommitFailureClosesAllResultSets()〔DatabaseMetaData インタフェース〕　1759
AVG　83

B

beforeFirst()〔ResultSet インタフェース〕　1556
beginRequest()〔Connection インタフェース〕 1401
BETWEEN 述語　71
BETWEEN 述語を使用したデータの探索　72
Blob インタフェース〔JDBC1.2 コア API〕　1784
BLOB 型を使用する場合の注意事項〔Type2 JDBC ドライバ〕　1285
Blob クラス〔Type2 JDBC ドライバ〕　1333
BLOB データ，BINARY データの後方削除更新　401
BLOB データ，BINARY データの追加更新　401
BLOB データ，BINARY データの部分抽出　401
BLOB データ，BINARY データの部分的な更新・検索　401
BLOB データのファイル出力機能　397

C

C++言語による UAP の作成　758
CallableStatement インタフェース〔JDBC1.2 コア API〕　1476
CallableStatement クラス〔JDBC1.0 機能〕　1248
CallableStatement クラス〔Type2 JDBC ドライバ〕　1325
cancel()〔Statement インタフェース〕　1409
checkSession(int waittime)〔Connection インタフェース〕　1398
Class ファイルの作成　860
clearBatch()〔Statement インタフェース〕　1411
clearParameters()〔PreparedStatement インタフェース〕　1446
clearWarnings()〔Connection インタフェース〕 1377
clearWarnings()〔ResultSet インタフェース〕　1557
clearWarnings()〔Statement インタフェース〕 1411
close()〔Connection インタフェース〕　1377
close()〔PooledConnection インタフェース〕　1818
close()〔ResultSet インタフェース〕　1558
close()〔Statement インタフェース〕　1411

closeOnCompletion()〔Statement インタフェース〕 1430
CNF 変換での探索高速化条件の導出 341
COBOL2002 の Unicode 機能を使用した UAP の実行 844
COBOL 言語で作成した UAP の実行手順 765
COBOL 言語での SQL を記述できる部 730
COBOL のビッグエンディアンオプションへの対応 811
COMMIT_BEHAVIOR についての注意事項 1243
commit()〔Connection インタフェース〕 1378
connect(String url, Properties info)〔Driver インタフェース〕 1369
ConnectionPoolDataSource インタフェース〔JDBC2.0 Optional Package〕 1812
Connection インタフェース〔JDBC1.2 コア API〕 1374
Connection クラス〔JDBC1.0 機能〕 1246
Connection クラス〔Type2 JDBC ドライバ〕 1323
COUNT 83
CREATE PUBLIC FUNCTION 271
CREATE PUBLIC PROCEDURE 263
createStatement()〔Connection インタフェース〕 1378
createStatement(int resultSetType, int resultSetConcurrency, int resultSetHoldability)〔Connection インタフェース〕 1380
createStatement(int resultSetType, int resultSetConcurrency)〔Connection インタフェース〕 1379
CROSS JOIN〔SQL の最適化〕 317
C 言語で作成した UAP の実行手順 764
C 言語による UAP の作成 694
C ストアドファンクション 894
C ストアドプロシジャ 894
C ファイルの作成 898
C プログラム作成時の制限事項 907
C プログラムの記述 898
C ライブラリファイルの作成 900
C ライブラリファイルの作成例 901
C ライブラリファイルの新規登録 902

D

DatabaseMetaData インタフェース〔JDBC1.2 コア API〕 1629
DatabaseMetaData クラス〔JDBC1.0 機能〕 1251
DatabaseMetaData クラス〔Type2 JDBC ドライバ〕 1327
dataDefinitionCausesTransactionCommit()〔DatabaseMetaData インタフェース〕 1643
dataDefinitionIgnoredInTransactions()〔DatabaseMetaData インタフェース〕 1644
DataSource インタフェース〔JDBC2.0 Optional Package〕 1807
DataSource と JNDI を使用した DB 接続〔Type4 JDBC ドライバ〕 1364
DbProviderFactory を使用したプロバイダに依存しないコード 1203
DbType プロパティと HiRDBType プロパティ 1191
DECIMAL 型符号正規化関数 2233
deletesAreDetected(int type)〔DatabaseMetaData インタフェース〕 1644
DELETE 文の WHERE 句 64
doesMaxRowSizeIncludeBlobs()〔DatabaseMetaData インタフェース〕 1645
DriverManager クラスによる DB 接続〔Type4 JDBC ドライバ〕 1339
Driver インタフェース〔JDBC1.2 コア API〕 1368
Driver クラス〔JDBC1.0 機能〕 1238

E

endRequest()〔Connection インタフェース〕 1401
enquoteIdentifier(String identifier, boolean alwaysQuote)〔Statement インタフェース〕 1435
enquoteLiteral(String val)〔Statement インタフェース〕 1434
enquoteNCharLiteral(String val)〔Statement インタフェース〕 1437
Exception トレースログ〔Type4 JDBC ドライバ〕 1951
Exception トレースログのファイルサイズ 1972
execute()〔PreparedStatement インタフェース〕 1446

execute(String sql)〔PreparedStatement インタフェース〕 1447
execute(String sql)〔Statement インタフェース〕 1412
executeBatch()〔Statement インタフェース〕 1413
EXECUTE IMMEDIATE 文で前処理と実行が一度に実行できる SQL 123
executeLargeBatch()〔Statement インタフェース〕 1433
executeLargeUpdate(String sql)〔Statement インタフェース〕 1433
executeLargeUpdate〔PreparedStatement インタフェース〕 1473
executeQuery()〔PreparedStatement インタフェース〕 1447
executeQuery(String sql)〔PreparedStatement インタフェース〕 1448
executeQuery(String sql)〔Statement インタフェース〕 1414
executeUpdate()〔PreparedStatement インタフェース〕 1449
executeUpdate(String sql)〔PreparedStatement インタフェース〕 1450
executeUpdate(String sql)〔Statement インタフェース〕 1415
EXISTS 述語 79
EXISTS 述語を使用した副問合せ 79
EX モード 139

F

FALSE 76
FETCH 文 54
findColumn(String columnName)〔ResultSet インタフェース〕 1558
first()〔ResultSet インタフェース〕 1559
FIX 属性の表 250
FIX 属性の表の検索 56
FIX 属性の表の更新 60
FIX 属性の表への行の挿入 68
FOR UPDATE 句，及び FOR READ ONLY 句を指定するとき考慮する内容 219
FOR UPDATE 句と FOR READ ONLY 句の使い分け 218
free()〔Blob インタフェース〕 1789
FROM 句の導出表のマージ適用〔SQL 拡張最適化オプション〕 630

G

generatedKeyAlwaysReturned()〔DatabaseMetaData インタフェース〕 1766
getArray()〔Array インタフェース〕 1791
getArray(int i)〔ResultSet インタフェース〕 1560
getArray(long index, int count)〔Array インタフェース〕 1792
getArray(String colName)〔ResultSet インタフェース〕 1560
getAsciiStream(int columnIndex)〔ResultSet インタフェース〕 1561
getAsciiStream(String columnName)〔ResultSet インタフェース〕 1562
getAttributes(String catalog,String schemaPattern,String typeNamePattern,String attributeNamePattern)〔DatabaseMetaData インタフェース〕 1645
getAutoCommit()〔Connection インタフェース〕 1381
getBaseType()〔Array インタフェース〕 1793
getBaseTypeName()〔Array インタフェース〕 1794
getBatchExceptionBehavior〔Type4 JDBC ドライバ〕 1897
getBestRowIdentifier(String catalog, String schema, String table, int scope, boolean nullable)〔DatabaseMetaData インタフェース〕 1647
getBigDecimal(int columnIndex, int scale)〔ResultSet インタフェース〕 1564
getBigDecimal(int columnIndex)〔ResultSet インタフェース〕 1563
getBigDecimal(int parameterIndex, int scale)〔CallableStatement インタフェース〕 1484
getBigDecimal(int parameterIndex)〔CallableStatement インタフェース〕 1482
getBigDecimal(String columnName, int scale)〔ResultSet インタフェース〕 1566

getBigDecimal(String columnName)〔ResultSet インタフェース〕 1565
getBinaryStream()〔Blob インタフェース〕 1785
getBinaryStream(int columnIndex)〔ResultSet インタフェース〕 1567
getBinaryStream(String columnName)〔ResultSet インタフェース〕 1568
getBlob(int i)〔ResultSet インタフェース〕 1569
getBlob(int parameterIndex)〔CallableStatement インタフェース〕 1485
getBlob(String colName)〔ResultSet インタフェース〕 1570
getBlob(String parameterName)〔CallableStatement インタフェース〕 1486
getBlockUpdate〔Type2 JDBC ドライバ〕1283,1310
getBoolean(int columnIndex)〔ResultSet インタフェース〕 1571
getBoolean(int parameterIndex)〔CallableStatement インタフェース〕 1486
getBoolean(String columnName)〔ResultSet インタフェース〕 1572
getBoolean(String parameterName)〔CallableStatement インタフェース〕 1488
getByte(int columnIndex)〔ResultSet インタフェース〕 1573
getByte(int parameterIndex)〔CallableStatement インタフェース〕 1489
getByte(String columnName)〔ResultSet インタフェース〕 1575
getByte(String parameterName)〔CallableStatement インタフェース〕 1491
getBytes(int columnIndex)〔ResultSet インタフェース〕 1576
getBytes(int parameterIndex)〔CallableStatement インタフェース〕 1492
getBytes(long pos, int length)〔Blob インタフェース〕 1786
getBytes(String columnName)〔ResultSet インタフェース〕 1577
getBytes(String parameterName)〔CallableStatement インタフェース〕 1492
getCatalog()〔Connection インタフェース〕 1381
getCatalogName(int column)〔ResultSetMetaData インタフェース〕 1770
getCatalogs()〔DatabaseMetaData インタフェース〕 1648
getCatalogSeparator()〔DatabaseMetaData インタフェース〕 1648
getCatalogTerm()〔DatabaseMetaData インタフェース〕 1649
getCharacterStream(int columnIndex)〔ResultSet インタフェース〕 1578
getCharacterStream(String columnName)〔ResultSet インタフェース〕 1579
getClear_Env〔Type2 JDBC ドライバ〕 1318
getClientInfoProperties()〔DatabaseMetaData インタフェース〕 1759
getColumnClassName(int column)〔ResultSetMetaData インタフェース〕 1770
getColumnCount()〔ResultSetMetaData インタフェース〕 1772
getColumnDisplaySize(int column)〔ResultSetMetaData インタフェース〕 1772
getColumnLabel(int column)〔ResultSetMetaData インタフェース〕 1773
getColumnName(int column)〔ResultSetMetaData インタフェース〕 1774
getColumnPrivileges (String catalog,String schema,String table,String columnNamePattern)〔DatabaseMetaData インタフェース〕 1649
getColumns(String catalog, String schemaPattern, String tableNamePattern, String columnNamePattern)〔DatabaseMetaData インタフェース〕 1651
getColumnType(int column)〔ResultSetMetaData インタフェース〕 1774
getColumnTypeName(int column)〔ResultSetMetaData インタフェース〕 1775
getCommit_Behavior〔Type2 JDBC ドライバ〕 1309
getConcurrency()〔ResultSet インタフェース〕 1579
getConnection()〔DatabaseMetaData インタフェース〕 1653
getConnection()〔DataSource インタフェース〕 1808

getConnection()〔PooledConnection インタフェース〕 1817
getConnection()〔Statement インタフェース〕 1417
getConnection(String username, String password)〔DataSource インタフェース〕 1809
getCrossReference (String primaryCatalog,String primarySchema,String primaryTable,String foreignCatalog,String foreignSchema,String foreignTable)〔DatabaseMetaData インタフェース〕 1654
getCursorName()〔ResultSet インタフェース〕 1580
getDatabaseMajorVersion()〔DatabaseMetaData インタフェース〕 1656
getDatabaseMinorVersion()〔DatabaseMetaData インタフェース〕 1657
getDatabaseProductName()〔DatabaseMetaData インタフェース〕 1657
getDatabaseProductVersion()〔DatabaseMetaData インタフェース〕 1658
getDate(int columnIndex, Calendar cal)〔ResultSet インタフェース〕 1582
getDate(int columnIndex)〔ResultSet インタフェース〕 1581
getDate(int parameterIndex, java.util.Calendar cal)〔CallableStatement インタフェース〕 1494
getDate(int parameterIndex)〔CallableStatement インタフェース〕 1493
getDate(String columnName, Calendar cal)〔ResultSet インタフェース〕 1583
getDate(String columnName)〔ResultSet インタフェース〕 1583
getDate(String parameterName, java.util.Calendar cal)〔CallableStatement インタフェース〕 1496
getDate(String parameterName)〔CallableStatement インタフェース〕 1495
getDBHostName〔Type2 JDBC ドライバ〕 1295
getDBHostName〔Type4 JDBC ドライバ〕 1855
getDefaultTransactionIsolation()〔DatabaseMetaData インタフェース〕 1658
getDescription〔Type2 JDBC ドライバ〕 1294
getDescription〔Type4 JDBC ドライバ〕 1854
getDouble(int columnIndex)〔ResultSet インタフェース〕 1584
getDouble(int parameterIndex)〔CallableStatement インタフェース〕 1497
getDouble(String columnName)〔ResultSet インタフェース〕 1586
getDouble(String parameterName)〔CallableStatement インタフェース〕 1499
getDriverMajorVersion()〔DatabaseMetaData インタフェース〕 1659
getDriverMinorVersion()〔DatabaseMetaData インタフェース〕 1659
getDriverName()〔DatabaseMetaData インタフェース〕 1660
getDriverVersion()〔DatabaseMetaData インタフェース〕 1660
getEncodeLang〔Type2 JDBC ドライバ〕 1297
getEncodeLang〔Type4 JDBC ドライバ〕 1885
getEnvironmentVariables〔Type4 JDBC ドライバ〕 1883
getExportedKeys (String catalog,String schema,String table)〔DatabaseMetaData インタフェース〕 1661
getExtraNameCharacters()〔DatabaseMetaData インタフェース〕 1663
getFetchDirection()〔ResultSet インタフェース〕 1587
getFetchDirection()〔Statement インタフェース〕 1417
getFetchSize()〔ResultSet インタフェース〕 1588
getFetchSize()〔Statement インタフェース〕 1418
getFloat(int columnIndex)〔ResultSet インタフェース〕 1588
getFloat(int parameterIndex)〔CallableStatement インタフェース〕 1500
getFloat(String columnName)〔ResultSet インタフェース〕 1591
getFloat(String parameterName)〔CallableStatement インタフェース〕 1502
getFunctionColumns(String catalog,String schemaPattern,String functionNamePattern,String

columnNamePattern)〔DatabaseMetaData インタフェース〕 1762
getFunctions(String catalog, String schemaPattern, String functionNamePattern)〔DatabaseMetaData インタフェース〕 1760
getHiRDBCursorMode〔Type4 JDBC ドライバ〕1879
getHiRDBINI〔Type4 JDBC ドライバ〕 1896
getHoldability()〔Connection インタフェース〕1381
getHoldability()〔ResultSet インタフェース〕 1621
getIdentifierQuoteString()〔DatabaseMetaData インタフェース〕 1664
getImportedKeys (String catalog,String schema,String table)〔DatabaseMetaData インタフェース〕 1664
getIndexInfo(String catalog, String schema, String table, boolean unique, boolean approximate)〔DatabaseMetaData インタフェース〕 1666
getInt(int columnIndex)〔ResultSet インタフェース〕 1591
getInt(int parameterIndex)〔CallableStatement インタフェース〕 1503
getInt(String columnName)〔ResultSet インタフェース〕 1593
getInt(String parameterName)〔CallableStatement インタフェース〕 1505
getJDBC_IF_TRC〔Type4 JDBC ドライバ〕 1857
getJDBCMajorVersion()〔DatabaseMetaData インタフェース〕 1668
getJDBCMinorVersion()〔DatabaseMetaData インタフェース〕 1669
getLargeMaxRows()〔Statement インタフェース〕 1432
getLargeUpdateCount()〔Statement インタフェース〕 1431
getLoginTimeout()〔ConnectionPoolDataSource インタフェース〕 1812
getLoginTimeout()〔DataSource インタフェース〕 1809
getLogWriter()〔ConnectionPoolDataSource インタフェース〕 1813
getLogWriter()〔DataSource インタフェース〕 1810
getLong(int columnIndex)〔ResultSet インタフェース〕 1594
getLong(int parameterIndex)〔CallableStatement インタフェース〕 1505
getLong(String columnName)〔ResultSet インタフェース〕 1596
getLong(String parameterName)〔CallableStatement インタフェース〕 1507
getLONGVARBINARY_AccessSize〔Type4 JDBC ドライバ〕 1891
getLONGVARBINARY_Access〔Type2 JDBC ドライバ〕 1312
getLONGVARBINARY_Access〔Type4 JDBC ドライバ〕 1869
getLONGVARBINARY_TruncError〔Type4 JDBC ドライバ〕 1893
getMajorVersion()〔Driver インタフェース〕 1370
getMaxBinaryLiteralLength()〔DatabaseMetaData インタフェース〕 1669
getMaxBinarySize〔Type4 JDBC ドライバ〕 1887
getMaxCatalogNameLength()〔DatabaseMetaData インタフェース〕 1670
getMaxCharLiteralLength()〔DatabaseMetaData インタフェース〕 1670
getMaxColumnNameLength()〔DatabaseMetaData インタフェース〕 1671
getMaxColumnsInGroupBy()〔DatabaseMetaData インタフェース〕 1671
getMaxColumnsInIndex()〔DatabaseMetaData インタフェース〕 1672
getMaxColumnsInOrderBy()〔DatabaseMetaData インタフェース〕 1672
getMaxColumnsInSelect()〔DatabaseMetaData インタフェース〕 1673
getMaxColumnsInTable()〔DatabaseMetaData インタフェース〕 1673
getMaxConnections()〔DatabaseMetaData インタフェース〕 1674
getMaxCursorNameLength()〔DatabaseMetaData インタフェース〕 1674
getMaxFieldSize()〔Statement インタフェース〕 1418

getMaxIndexLength()〔DatabaseMetaData インタフェース〕 1675
getMaxLogicalLobSize()〔DatabaseMetaData インタフェース〕 1766
getMaxProcedureNameLength()〔DatabaseMetaData インタフェース〕 1675
getMaxRows()〔Statement インタフェース〕 1419
getMaxRowSize()〔DatabaseMetaData インタフェース〕 1676
getMaxSchemaNameLength()〔DatabaseMetaData インタフェース〕 1676
getMaxStatementLength()〔DatabaseMetaData インタフェース〕 1677
getMaxStatements()〔DatabaseMetaData インタフェース〕 1678
getMaxTableNameLength()〔DatabaseMetaData インタフェース〕 1678
getMaxTablesInSelect()〔DatabaseMetaData インタフェース〕 1679
getMaxUserNameLength()〔DatabaseMetaData インタフェース〕 1679
getMetaData()〔Connection インタフェース〕 1382
getMetaData()〔PreparedStatement インタフェース〕 1451
getMetaData()〔ResultSet インタフェース〕 1597
getMinorVersion()〔Driver インタフェース〕 1370
getMoreResults()〔Statement インタフェース〕1419
getNotErrorOccurred〔Type4 JDBC ドライバ〕 1881
getNumericFunctions()〔DatabaseMetaData インタフェース〕 1680
getObject(int columnIndex)〔ResultSet インタフェース〕 1598
getObject(int parameterIndex)〔CallableStatement インタフェース〕 1508
getObject(String columnName)〔ResultSet インタフェース〕 1599
getObject(String parameterName)〔CallableStatement インタフェース〕 1510
getParameterClassName(int param)〔ParameterMetaData インタフェース〕 1831
getParameterCount()〔ParameterMetaData インタフェース〕 1826
getParameterMetaData()〔PreparedStatement インタフェース〕 1452
getParameterMode(int param)〔ParameterMetaData インタフェース〕 1832
getParameterType(int param)〔ParameterMetaData インタフェース〕 1829
getParameterTypeName(int param)〔ParameterMetaData インタフェース〕 1830
getPassword〔Type2 JDBC ドライバ〕 1300
getPassword〔Type4 JDBC ドライバ〕 1864
getPooledConnection()〔ConnectionPoolDataSource インタフェース〕 1813
getPooledConnection(String user, String password)〔ConnectionPoolDataSource インタフェース〕 1814
getPrecision(int column)〔ResultSetMetaData インタフェース〕 1776
getPrecision(int param)〔ParameterMetaData インタフェース〕 1828
getPrimaryKeys(String catalog, String schema, String table)〔DatabaseMetaData インタフェース〕 1681
getProcedureColumns(String catalog,String schemaPattern,String procedureNamePattern, String columnNamePattern)〔DatabaseMetaData インタフェース〕 1682
getProcedures(String catalog,String schemaPattern,String procedureNamePattern)〔DatabaseMetaData インタフェース〕 1686
getProcedureTerm()〔DatabaseMetaData インタフェース〕 1688
getPropertyInfo(String url, Properties info)〔Driver インタフェース〕 1370
getPseudoColumns(String catalog, String schemaPattern, String tableNamePattern, String columnNamePattern)〔DatabaseMetaData インタフェース〕 1765
getQueryTimeout()〔Statement インタフェース〕 1420
getResultSet()〔Array インタフェース〕 1794
getResultSet()〔Statement インタフェース〕 1420
getResultSet(long index,int count)〔Array インタフェース〕 1795

getResultSetConcurrency()〔Statement インタフェース〕 1421
getResultSetHoldability()〔DatabaseMetaData インタフェース〕 1688
getResultSetHoldability()〔Statement インタフェース〕 1421
getResultSetType()〔Statement インタフェース〕 1422
getRMID〔Type2 JDBC ドライバ〕 1305
getRow()〔ResultSet インタフェース〕 1600
getRowIdLifetime()〔DatabaseMetaData インタフェース〕 1757
getScale(int column)〔ResultSetMetaData インタフェース〕 1777
getScale(int param)〔ParameterMetaData インタフェース〕 1829
getSchema()〔Connection インタフェース〕 1400
getSchemaName(int column)〔ResultSetMetaData インタフェース〕 1778
getSchemas()〔DatabaseMetaData インタフェース〕 1689
getSchemas(String catalog, String schemaPattern)〔DatabaseMetaData インタフェース〕 1757
getSchemaTerm()〔DatabaseMetaData インタフェース〕 1690
getSearchStringEscape()〔DatabaseMetaData インタフェース〕 1690
getShort(int columnIndex)〔ResultSet インタフェース〕 1600
getShort(int parameterIndex)〔CallableStatement インタフェース〕 1510
getShort(String columnName)〔ResultSet インタフェース〕 1603
getShort(String parameterName)〔CallableStatement インタフェース〕 1512
getSQLInNum〔Type2 JDBC ドライバ〕 1314
getSQLInNum〔Type4 JDBC ドライバ〕 1871
getSQLKeywords()〔DatabaseMetaData インタフェース〕 1691
getSQLOutNum〔Type2 JDBC ドライバ〕 1315
getSQLOutNum〔Type4 JDBC ドライバ〕 1872
getSQLStateType()〔DatabaseMetaData インタフェース〕 1693
getSQLWarningIgnore 1877
getSQLWarningLevel〔Type2 JDBC ドライバ〕 1317
getSQLWarningLevel〔Type4 JDBC ドライバ〕 1874
getStatement()〔ResultSet インタフェース〕 1603
getStatementCommitBehavior〔Type4 JDBC ドライバ〕 1889
getString(int columnIndex)〔ResultSet インタフェース〕 1604
getString(int parameterIndex)〔CallableStatement インタフェース〕 1513
getString(String columnName)〔ResultSet インタフェース〕 1606
getString(String parameterName)〔CallableStatement インタフェース〕 1515
getStringFunctions()〔DatabaseMetaData インタフェース〕 1694
getSuperTables(String catalog,String schemaPattern,String tableNamePattern)〔DatabaseMetaData インタフェース〕 1694
getSuperTypes(String catalog,String schemaPattern,String typeNamePattern)〔DatabaseMetaData インタフェース〕 1695
getSystemFunctions()〔DatabaseMetaData インタフェース〕 1696
getTableName(int column)〔ResultSetMetaData インタフェース〕 1778
getTablePrivileges(String catalog,String schemaPattern,String tableNamePattern)〔DatabaseMetaData インタフェース〕 1697
getTables(String catalog, String schemaPattern, String tableNamePattern, String[] types)〔DatabaseMetaData インタフェース〕 1698
getTableTypes()〔DatabaseMetaData インタフェース〕 1700
getTime(int columnIndex, Calendar cal)〔ResultSet インタフェース〕 1608
getTime(int columnIndex)〔ResultSet インタフェース〕 1606
getTime(int parameterIndex, java.util.Calendar cal)〔CallableStatement インタフェース〕 1517

getTime(int parameterIndex)〔CallableStatement インタフェース〕 1516
getTime(String columnName, Calendar cal)〔ResultSet インタフェース〕 1609
getTime(String columnName)〔ResultSet インタフェース〕 1608
getTime(String parameterName, java.util.Calendar cal)〔CallableStatement インタフェース〕 1518
getTime(String parameterName)〔CallableStatement インタフェース〕 1518
getTimeDateFunctions()〔DatabaseMetaData インタフェース〕 1700
getTimestamp(int columnIndex, Calendar cal)〔ResultSet インタフェース〕 1611
getTimestamp(int columnIndex)〔ResultSet インタフェース〕 1610
getTimestamp(int parameterIndex, java.util.Calendar cal)〔CallableStatement インタフェース〕 1520
getTimestamp(int parameterIndex)〔CallableStatement インタフェース〕 1519
getTimestamp(String columnName, Calendar cal)〔ResultSet インタフェース〕 1613
getTimestamp(String columnName)〔ResultSet インタフェース〕 1612
getTimestamp(String parameterName, java.util.Calendar cal)〔CallableStatement インタフェース〕 1522
getTimestamp(String parameterName)〔CallableStatement インタフェース〕 1521
getTransactionIsolation()〔Connection インタフェース〕 1382
getTRC_NO〔Type4 JDBC ドライバ〕 1859
getType()〔ResultSet インタフェース〕 1613
getTypeInfo()〔DatabaseMetaData インタフェース〕 1701
getTypeMap()〔Connection インタフェース〕 1383
getUapName〔Type4 JDBC ドライバ〕 1860
getUDTs(String catalog,String schemaPattern,String typeNamePattern,int[] types)〔DatabaseMetaData インタフェース〕 1703
getUpdateCount()〔Statement インタフェース〕 1423
getUpdateCountBehavior〔Type4 JDBC ドライバ〕 1899
getURL()〔DatabaseMetaData インタフェース〕 1704
getUserName()〔DatabaseMetaData インタフェース〕 1704
getUser〔Type2 JDBC ドライバ〕 1299
getUser〔Type4 JDBC ドライバ〕 1862
getVersionColumns(String catalog,String schema,String table)〔DatabaseMetaData インタフェース〕 1704
getWarnings()〔Connection インタフェース〕 1383
getWarnings()〔ResultSet インタフェース〕 1614
getWarnings()〔Statement インタフェース〕 1424
getXACloseString〔Type2 JDBC ドライバ〕 1303
getXACloseString〔Type4 JDBC ドライバ〕 1867
getXALocalCommitMode〔Type4 JDBC ドライバ〕 1876
getXAOpenString〔Type2 JDBC ドライバ〕 1302
getXAOpenString〔Type4 JDBC ドライバ〕 1866
getXAThreadMode〔Type2 JDBC ドライバ〕 1306

H

HASH JOIN〔SQL の最適化〕 311
HASH SUBQ〔SQL の最適化〕 327, 330
HiRDB_ConnectionRecove 1229
HiRDB_ConnectionString 1229
HiRDB_LifeTime 1229
HiRDB_PDHOST 516, 542
HiRDB_PDNAMEPORT 516, 543
HiRDB_PDTMID 516, 543
HiRDB_PDXAMODE 516, 544
HiRDB_Pooling 1229
HiRDB/Developer's Kit 446
HiRDB/Run Time 446
HiRDBCommand 1137
HiRDBCommandBuilder 1141
HiRDBCommandBuilder のメンバー一覧 1123
HiRDBCommand のメンバー一覧 1122

HiRDBConnection 1147
HiRDBConnection のメンバー一覧 1125
HiRDBDataAdapter 1153
HiRDBDataAdapter のメンバー一覧 1126
HiRDBDataReader 1154
HiRDBDataReader のメンバー一覧 1127
HiRDBException 1166
HiRDBException のメンバー一覧 1129
HiRDB JDBC ドライバの提供機能 1282
HiRDB OLE DB プロバイダ 1103
HiRDBParameter 1166
HiRDBParameterCollection 1172
HiRDBParameterCollection のメンバー一覧 1132
HiRDBParameter のメンバー一覧 1130
HiRDBProviderFactory 1179
HiRDBProviderFactory のメンバー一覧 1134
HiRDBRowUpdatedEventArgs 1181
HiRDBRowUpdatedEventArgs のメンバー一覧 1134
HiRDBRowUpdatingEventArgs 1182
HiRDBRowUpdatingEventArgs のメンバー一覧 1135
HiRDB SQL Tuning Advisor の環境設定 2246
HiRDB SQL Tuning Advisor 用アクセスパス情報ファイル 1001
HiRDBTransaction 1182
HiRDBTransaction メンバー一覧 1135
HiRDB が提供する ODBC 関数 1037
HiRDB が提供する関数 2205
HiRDB からの切り離し 117
HiRDB クライアント 446
HiRDB クライアント環境変数登録ツール 672
HiRDB クライアントの SELinux セキュリティコンテキスト設定及び解除 688
HiRDB クライアントのディレクトリ及びファイル構成 460
HiRDB システムへの接続と切り離し 132
HiRDB データプロバイダ for .NET Framework 1113
HiRDB データプロバイダ for .NET Framework と対応する ADO.NET のバージョン 1117
HiRDB データプロバイダ for .NET Framework のインストール 1119
HiRDB データプロバイダ for .NET Framework のインタフェース 1137
HiRDB データプロバイダ for .NET Framework の型変換 1193
HiRDB データプロバイダ for .NET Framework のクラス一覧 1121
HiRDB データプロバイダ for .NET Framework の前提プログラム 1113
HiRDB データプロバイダ for .NET Framework のデータ型 1191
HiRDB データプロバイダ for .NET Framework のトラブルシュート機能 1206
HiRDB データプロバイダ for .NET Framework のメンバー一覧 1122
HiRDB データプロバイダ for .NET Framework の留意事項 1185
HiRDB データプロバイダ for .NET Framework を使用した UAP 例 1210
HiRDB で使用できる SQL 一覧 35
HiRDB での問合せ処理方式〔SQL の最適化〕 297
HiRDB との接続 113
HiRDB の通信処理 522
hosts ファイル 491

I

INDEX SCAN〔SQL の最適化〕 319
insertsAreDetected(int type)〔DatabaseMetaData インタフェース〕 1705
INSERT 文 67
INSERT 文に ROW を指定 68
IN 述語 72
IN 述語を使用したデータの探索 73
isAfterLast()〔ResultSet インタフェース〕 1615
isAutoIncrement(int column)〔ResultSetMetaData インタフェース〕 1779
isBeforeFirst()〔ResultSet インタフェース〕 1615
isCaseSensitive(int column)〔ResultSetMetaData インタフェース〕 1779
isCatalogAtStart()〔DatabaseMetaData インタフェース〕 1706
isClosed()〔Connection インタフェース〕 1384

isClosed()〔ResultSet インタフェース〕 1622
isClosed()〔Statement インタフェース〕 1429
isCloseOnCompletion()〔Statement インタフェース〕 1430
isCurrency(int column)〔ResultSetMetaData インタフェース〕 1780
isDefinitelyWritable(int column)〔ResultSetMetaData インタフェース〕 1780
isFirst()〔ResultSet インタフェース〕 1616
isLast()〔ResultSet インタフェース〕 1616
isNullable(int column)〔ResultSetMetaData インタフェース〕 1781
isNullable(int param)〔ParameterMetaData インタフェース〕 1826
isPoolable〔Statement インタフェース〕 1429
isReadOnly()〔Connection インタフェース〕 1384
isReadOnly()〔DatabaseMetaData インタフェース〕 1707
isReadOnly(int column)〔ResultSetMetaData インタフェース〕 1781
isSearchable(int column)〔ResultSetMetaData インタフェース〕 1782
isSigned(int column)〔ResultSetMetaData インタフェース〕 1782
isSigned(int param)〔ParameterMetaData インタフェース〕 1827
isSimpleIdentifier(String identifier)〔Statement インタフェース〕 1436
isValid(int timeout)〔Connection インタフェース〕 1399
isWrapperFor(Class<?> iface)〔Wrapper インタフェース〕 1839
isWritable(int column)〔ResultSetMetaData インタフェース〕 1783

J

JAR 形式へのアーカイブ 861
JAR ファイルアクセス機能〔JDBC2.0 Optional Package〕 1275
JAR ファイルの作成 861
JAR ファイルの新規登録 861
Java ストアドファンクション 855
Java ストアドプロシジャ 855
Java ファイルの作成 859
Java プログラムの記述 859
JDBC1.0 機能 1238
JDBC1.2 コア API 1368
JDBC2.0 Optional Package〔Type2 JDBC ドライバ〕 1268
JDBC2.0 Optional Package〔Type4 JDBC ドライバ〕 1807
JDBC2.0 基本機能 1254
JDBC2.1 コア API 1799
JDBC3.0 API 1825
JDBC4.0 API 1834
JDBC4.3API 1844
jdbcCompliant()〔Driver インタフェース〕 1373
JDBC インタフェースメソッドトレース〔Type4 JDBC ドライバ〕 1948
JDBC ドライバを使用した UAP 例 1977

K

KEY SCAN MERGE JOIN〔SQL の最適化〕 309
KEY SCAN〔SQL の最適化〕 319

L

last()〔ResultSet インタフェース〕 1617
length()〔Blob インタフェース〕 1788
LIKE 述語 73
LIKE 述語を使用したデータの探索 74
LIST SCAN MERGE JOIN〔SQL の最適化〕 309
LIST SCAN〔SQL の最適化〕 322
locatorsUpdateCopy()〔DatabaseMetaData インタフェース〕 1707
L-KEY R-LIST MERGE JOIN〔SQL の最適化〕 310
L-KEY R-SORT MERGE JOIN〔SQL の最適化〕 310
L-LIST R-KEY MERGE JOIN〔SQL の最適化〕 310
L-LIST R-SORT MERGE JOIN〔SQL の最適化〕 310
L-SORT R-KEY MERGE JOIN〔SQL の最適化〕 310
L-SORT R-LIST MERGE JOIN〔SQL の最適化〕 310

M

MAX 83
MIN 83
MULTI COLUMNS INDEX SCAN〔SQL の最適化〕 320
MULTI COLUMNS KEY SCAN〔SQL の最適化〕 320

N

nativeSQL(String sql)〔Connection インタフェース〕 1385
NESTED LOOPS JOIN〔SQL の最適化〕 310
NESTED LOOPS ROW VALUE SUBQ〔SQL の最適化〕 329
NESTED LOOPS WORK TABLE SUBQ〔SQL の最適化〕 329
next()〔ResultSet インタフェース〕 1617
NOT 75
nullPlusNonNullIsNull()〔DatabaseMetaData インタフェース〕 1708
nullsAreSortedAtEnd()〔DatabaseMetaData インタフェース〕 1708
nullsAreSortedAtStart()〔DatabaseMetaData インタフェース〕 1709
nullsAreSortedHigh()〔DatabaseMetaData インタフェース〕 1709
nullsAreSortedLow()〔DatabaseMetaData インタフェース〕 1710
NULL 述語 74
NULL 述語と NOT を組み合わせて使用したデータの探索 75

O

ODBC2.0 ドライバのインストール 1023
ODBC3.5 ドライバが返却する SQLSTATE 1087
ODBC3.5 ドライバのインストールと環境変数の設定 1026
ODBC 関数で使用できる機能 1042
ODBC 対応アプリケーションプログラムからの HiRDB アクセス 1021
OLE DB 1103
OLE DB 対応アプリケーションプログラムからの HiRDB アクセス 1102
OOCOBOL 言語による UAP の作成 759
OR 75
OR PLURAL INDEXES SCAN〔SQL の最適化〕 322
OR の複数インデクス利用〔SQL の最適化〕 318
OR の複数インデクス利用の優先〔SQL 最適化オプション〕 616
othersDeletesAreVisible(int type)〔DatabaseMetaData インタフェース〕 1710
othersInsertsAreVisible(int type)〔DatabaseMetaData インタフェース〕 1711
othersUpdatesAreVisible(int type)〔DatabaseMetaData インタフェース〕 1711
ownDeletesAreVisible(int type)〔DatabaseMetaData インタフェース〕 1712
ownInsertsAreVisible(int type)〔DatabaseMetaData インタフェース〕 1713
ownUpdatesAreVisible(int type)〔DatabaseMetaData インタフェース〕 1713

P

p_rdb_conv_space 2229
p_rdb_conv_space_utf8 2231
p_rdb_dbhash 2210
p_rdb_dec_sign_norm 2234
p_rdb_set_lang 2236
ParameterMetaData インタフェース〔JDBC3.0 API〕 1825
PATH 492, 499
PDADDITIONALOPTLVL 521, 624
PDAGGR 521, 632
PDASTHOST 517, 574
PDASTPORT 518, 574
PDASTUSER 518, 575
PDAUTHTYPE 517, 568
PDAUTOCONNECT 521, 633
PDAUTORECONNECT 517, 566
PDBESCONHOLD 523, 648
PDBESCONHTI 523, 648
PDBINARYBLKF 522, 643
PDBINDRETRYCOUNT 522, 644

PDBINDRETRYINTERVAL 522, 644
PDBLKBUFFSIZE 522, 643
PDBLKF 522, 643
PDCALCMDWAITTIME 522, 641
pdcbl 780
PDCLTAPNAME 516, 545
PDCLTBINDLOOPBACKADDR 517, 570
PDCLTCNVBYTERATIO 517, 565
PDCLTCNVMODE 516, 548
PDCLTCNVUOCFUNC 517, 562
PDCLTCNVUOCLIB 517, 561
PDCLTGAIJIDLL 516, 557
PDCLTGAIJIFUNC 517, 558
PDCLTGRP 517, 566
PDCLTLANG 516, 545
PDCLTPATH 519, 589
PDCLTRCVADDR 515, 535
PDCLTRCVPORT 515, 534
PDCLTRDNODE 522, 646
PDCMDTRACE 518, 575
PDCMDWAITTIME 518, 575
PDCMMTBFDDL 521, 633
PDCNSTRNTNAME 523, 647
PDCONNECTWAITTIME 519, 588
PDCONREFRCOUNT 519, 586
PDCONREFRINTERVAL 519, 587
PDCONTYPE 516, 536
pdcpp 769
PDCURSORLVL 522, 638
PDCWAITTIME 518, 577
PDCWAITTIMEWRNPNT 518, 581
PDDBACCS 522, 644
PDDBBUFLRU 517, 569
PDDBLOG 516, 547
PDDBORGUAP 522, 645
PDDDLDEAPRP 521, 637
PDDDLDEAPRPEXE 521, 634
PDDEFAULTOPTION 517, 571
PDDELRSVWDFILE 522, 641
PDDFLNVAL 521, 632
PDDLKPRIO 521, 604
PDDNDPCOMPATIBLE 524, 667
PDDNDPTRACE 520, 599
PDERRSKIPCODE 519, 590
PDEXTDECCHECK 517, 571
PDEXWARN 516, 547
PDFESGRP 515, 531
PDFESHOST 515, 528
PDFORUPDATEEXLOCK 521, 605
PDGDATAOPT 523, 650
PDHASHTBLSIZE 521, 631
PDHATRNQUEUING 517, 570
PDHOST 515, 526
PDHSICOPTIONS 524, 665
PDIPC 518, 576
PDIPCFILEDIR 572
PDISLLVL 521, 606
PDJDBFILEDIR 524, 666
PDJDBFILEOUTNUM 524, 666
PDJDBFILESIZE 524, 667
PDJDBONMEMNUM 524, 667
PDJDBSQLTRCMEMNUM 520, 600
PDJDBTRACELEVEL 524, 667
PDJETCOMPATIBLE 523, 652
PDKALVL 518, 582
PDKATIME 519, 585
PDLANG 516, 546
PDLCKWAITTIME 522, 638
PDLOCKLIMIT 521, 604
PDLOCKSKIP 521, 605
PDNAMEPORT 515, 527
PDNBLOCKWAITTIME 519, 587
pdocb 780
pdocc 769
PDODBCWRNSKIP 523, 651
PDODBERRTRC 524, 659
PDODBERRTRCSIZE 524, 660
PDODBESCAPE 523, 649

PDODBGINFOSUPPRESS 523, 652
PDODBLOCATOR 523, 650
PDODBSPLITSIZE 523, 651
PDODBSTANDARDARGSIZE 523, 653
PDODBSTANDARDDESCCOL 523, 656
PDODBSTANDARDGTYPEINFO 524, 657
PDODBSTANDARDSQLSTATE 523, 655
PDODBSTATCACHE 523, 649
PDODBSYSKEYS 524, 664
PDODBSYSUNSIGNED 524, 660
PDPLGIXMK 524, 664
PDPLGPFSZ 524, 664
PDPLGPFSZEXP 524, 664
PDPLUGINNSUB 524, 664
PDPRMTRC 519, 591
PDPRMTRCSIZE 519, 592
PDPRPCRCLS 521, 633
PDRCCOUNT 517, 567
PDRCINTERVAL 517, 567
PDRCTIMING 517, 567
PDRCTRACE 520, 596
PDRDCLTCODE 522, 647
PDRECVMEMSIZE 518, 577
PDREPPATH 519, 593
PDSENDMEMSIZE 518, 576
PDSERVICEGRP 515, 529
PDSERVICEPORT 515, 530
PDSOCKETCRYPTO 517, 573
PDSPACELVL 522, 645
PDSQLEXECTIME 520, 595
PDSQLOPTLVL 521, 607
PDSQLTEXTSIZE 520, 595
PDSQLTRACE 519, 590
PDSQLTRCBUFFSIZE 520, 600
PDSQLTRCFMT 520, 599
PDSQLTRCOPENMODE 520, 594
PDSQLTRCTYPE 520, 599
PDSRVTYPE 515, 530
PDSTANDARDSQLSTATE 522, 642
PDSTJTRNOUT 521, 603
PDSUBSTRLEN 516, 547
PDSWAITTIME 518, 578
PDSWATCHTIME 518, 579
PDSYSTEMID 518, 575
PDTAAPINFMODE 520, 603
PDTAAPINFPATH 520, 602
PDTAAPINFSIZE 521, 603
PDTIMEDOUTRETRY 519, 586
PDTMID 516, 539
PDTMPTBLRDAREA 523, 647
PDTP1SERVICE 522, 646
pdtrcmgr 997
PDTRCMODE 519, 592
PDTRCPATH 520, 594
PDTXACANUM 516, 540
PDUAPENVFILE 517, 569
PDUAPERLOG 519, 590
PDUAPEXERLOGPRMSZ 520, 598
PDUAPEXERLOGUSE 520, 598
PDUAPREPLVL 519, 592
PDUSER 516, 544
PDVWOPTMODE 520, 601
PDWRTLNCOMSZ 520, 598
PDWRTLNFILSZ 520, 596
PDWRTLNPATH 520, 596
PDXAAUTORECONNECT 516, 541
PDXAMODE 516, 539
PDXARCVWTIME 516, 540
PDXATRCFILEMODE 516, 541
PLUGIN INDEX SCAN〔SQL の最適化〕 320
PLUGIN KEY SCAN〔SQL の最適化〕 321
PooledConnection インタフェース〔JDBC2.0 Optional Package〕 1816
position(Blob pattern, long start)〔Blob インタフェース〕 1788
position(byte[] pattern, long start)〔Blob インタフェース〕 1789

prepareCall(String sql, int resultSetType, int resultSetConcurrency, int resultSetHoldability)〔Connection インタフェース〕 1391
prepareCall(String sql, int resultSetType, int resultSetConcurrency)〔Connection インタフェース〕 1390
prepareCall(String sql)〔Connection インタフェース〕 1389
PreparedStatement インタフェース〔JDBC1.2 コア API〕 1442
PreparedStatement クラス〔JDBC1.0 機能〕 1247
PreparedStatement クラス〔Type2 JDBC ドライバ〕 1324
prepareStatement(String sql, int resultSetType, int resultSetConcurrency, int resultSetHoldability)〔Connection インタフェース〕 1393
prepareStatement(String sql, int resultSetType, int resultSetConcurrency)〔Connection インタフェース〕 1392
prepareStatement(String sql)〔Connection インタフェース〕 1392
PREPARE 文で前処理できる SQL 123
previous()〔ResultSet インタフェース〕 1618
PR モード 139
PURGE TABLE 文 65
PU モード 139
Python アプリケーションからの HiRDB アクセス 1097

R

RD エリア名を指定した検索，更新，又は削除 419
Red Hat Enterprise Linux 10，Oracle Linux 10 以降での UAP 作成 853
registerOutParameter(int parameterIndex, int sqlType, int scale)〔CallableStatement インタフェース〕 1523
registerOutParameter(int parameterIndex, int sqlType)〔CallableStatement インタフェース〕 1523
registerOutParameter(String parameterName, int sqlType, int scale)〔CallableStatement インタフェース〕 1525
registerOutParameter(String parameterName, int sqlType)〔CallableStatement インタフェース〕 1524
relative(int rows)〔ResultSet インタフェース〕 1619
removeConnectionEventListener(ConnectionEventListener listener)〔PooledConnection インタフェース〕 1818
removeStatementEventListener(StatementEventListener listener)〔PooledConnection インタフェース〕 1820
ResultSetMetaData インタフェース〔JDBC1.2 コア API〕 1768
ResultSetMetaData クラス〔JDBC1.0 機能〕 1249
ResultSetMetaData クラス〔Type2 JDBC ドライバ〕 1327
ResultSet インタフェース〔JDBC1.2 コア API〕 1547
ResultSet クラス〔JDBC1.0 機能〕 1249
ResultSet クラス〔Type2 JDBC ドライバ〕 1325
rollback()〔Connection インタフェース〕 1394
ROWID FETCH〔SQL の最適化〕 323
ROW VALUE SUBQ〔SQL の最適化〕 326
R-LIST NESTED LOOPS JOIN〔SQL の最適化〕 311

S

SELECT 文で検索 52
SELECT 文の FROM 句 55
SELECT 文の選択句 56
SELECT 文のハッシュジョイン最大数 334
SELECT-APSL〔SQL の最適化〕 308, 316, 318, 321
setArray(int i,Array x)〔PreparedStatement インタフェース〕 1453
setAsciiStream(int parameterIndex, java.io.InputStream x, int length)〔PreparedStatement インタフェース〕 1455
setAsciiStream(String parameterName, java.io.InputStream x, int length)〔CallableStatement インタフェース〕 1526
setAutoCommit(boolean autoCommit)〔Connection インタフェース〕 1395
setBatchExceptionBehavior〔Type4 JDBC ドライバ〕 1897
setBigDecimal(int parameterIndex, BigDecimal x)〔PreparedStatement インタフェース〕 1456

setBigDecimal(String parameterName, java.math.BigDecimal x)〔CallableStatement インタフェース〕 1527
setBinaryStream(int parameterIndex, java.io.InputStream x, int length)〔PreparedStatement インタフェース〕 1456
setBinaryStream(String parameterName, java.io.InputStream x, int length)〔CallableStatement インタフェース〕 1528
setBlob(int parameterIndex, Blob x)〔PreparedStatement インタフェース〕 1457
setBlockUpdate〔Type2 JDBC ドライバ〕 1282, 1309
setBoolean(int parameterIndex, boolean x)〔PreparedStatement インタフェース〕 1458
setBoolean(String parameterName, boolean x)〔CallableStatement インタフェース〕 1528
setByte(int parameterIndex, byte x)〔PreparedStatement インタフェース〕 1459
setByte(String parameterName, byte x)〔CallableStatement インタフェース〕 1529
setBytes(int parameterIndex, byte x[])〔PreparedStatement インタフェース〕 1459
setBytes(String parameterName,byte[] x)〔CallableStatement インタフェース〕 1530
setCatalog(String catalog)〔Connection インタフェース〕 1396
setCharacterStream(int parameterIndex, Reader reader, int length)〔PreparedStatement インタフェース〕 1460
setCharacterStream(String parameterName, Reader x, int length)〔CallableStatement インタフェース〕 1531
setClear_Env〔Type2 JDBC ドライバ〕 1317
setCommit_Behavior〔Type2 JDBC ドライバ〕 1307
setCursorName(String name)〔Statement インタフェース〕 1424
setDate(int parameterIndex, java.sql.Date x,Calendar cal)〔PreparedStatement インタフェース〕 1462
setDate(int parameterIndex, java.sql.Date x)〔PreparedStatement インタフェース〕 1461
setDate(String parameterName, java.sql.Date x,Calendar cal)〔CallableStatement インタフェース〕 1533
setDate(String parameterName, java.sql.Date x)〔CallableStatement インタフェース〕 1532
setDBHostName〔Type2 JDBC ドライバ〕 1294
setDBHostName〔Type4 JDBC ドライバ〕 1855
setDescription〔Type2 JDBC ドライバ〕 1292
setDescription〔Type4 JDBC ドライバ〕 1852
setDouble(int parameterIndex, double x)〔PreparedStatement インタフェース〕 1463
setDouble(String parameterName, double x)〔CallableStatement インタフェース〕 1533
setEncodeLang〔Type2 JDBC ドライバ〕 1296
setEncodeLang〔Type4 JDBC ドライバ〕 1884
setEnvironmentVariables〔Type4 JDBC ドライバ〕 1882
setEscapeProcessing(boolean enable)〔Statement インタフェース〕 1425
setFetchDirection(int direction)〔ResultSet インタフェース〕 1619
setFetchDirection(int direction)〔Statement インタフェース〕 1425
setFetchSize(int rows)〔ResultSet インタフェース〕 1620
setFetchSize(int rows)〔Statement インタフェース〕 1426
setFloat(int parameterIndex, float x)〔PreparedStatement インタフェース〕 1463
setFloat(String parameterName, float x)〔CallableStatement インタフェース〕 1534
setHiRDB_Audit_Info(int pos，String userinf)〔Connection インタフェース〕 1398
setHiRDBCursorMode〔Type4 JDBC ドライバ〕 1878
setHiRDBINI〔Type4 JDBC ドライバ〕 1895
setHoldability(int holdability)〔Connection インタフェース〕 1396
setInt(int parameterIndex, int x)〔PreparedStatement インタフェース〕 1464
setInt(String parameterName, int x)〔CallableStatement インタフェース〕 1535
setJDBC_IF_TRC〔Type4 JDBC ドライバ〕 1856

setLargeMaxRows(long max)〔Statement インタフェース〕 1432
setLoginTimeout(int seconds)〔ConnectionPoolDataSource インタフェース〕 1814
setLoginTimeout(int seconds)〔DataSource インタフェース〕 1810
setLogWriter(PrintWriter out)〔ConnectionPoolDataSource インタフェース〕 1815
setLogWriter(PrintWriter out)〔DataSource インタフェース〕 1811
setLong(int parameterIndex, long x)〔PreparedStatement インタフェース〕 1465
setLong(String parameterName, long x)〔CallableStatement インタフェース〕 1536
setLONGVARBINARY_AccessSize〔Type4 JDBC ドライバ〕 1890
setLONGVARBINARY_Access〔Type2 JDBC ドライバ〕 1311
setLONGVARBINARY_Access〔Type4 JDBC ドライバ〕 1868
setLONGVARBINARY_TruncError〔Type4 JDBC ドライバ〕 1892
setMaxBinarySize〔Type4 JDBC ドライバ〕 1886
setMaxFieldSize(int max)〔Statement インタフェース〕 1427
setMaxRows(int max)〔Statement インタフェース〕 1427
setNotErrorOccurred〔Type4 JDBC ドライバ〕 1880
setNull(int parameterIndex,int sqlType)〔PreparedStatement インタフェース〕 1465
setNull(String parameterName,int sqlType)〔CallableStatement インタフェース〕 1536
setObject(int parameterIndex, Object x, int targetSqlType, int scale)〔PreparedStatement インタフェース〕 1468
setObject(int parameterIndex, Object x, int targetSqlType)〔PreparedStatement インタフェース〕 1467
setObject(int parameterIndex, Object x)〔PreparedStatement インタフェース〕 1466
setObject(String parameterName, Object x, int targetSqlType, int scale)〔CallableStatement インタフェース〕 1539
setObject(String parameterName, Object x, int targetSqlType)〔CallableStatement インタフェース〕 1538
setObject(String parameterName, Object x)〔CallableStatement インタフェース〕 1537
setPassword〔Type2 JDBC ドライバ〕 1299
setPassword〔Type4 JDBC ドライバ〕 1863
setQueryTimeout(int seconds)〔Statement インタフェース〕 1428
setReadOnly(boolean readOnly)〔Connection インタフェース〕 1397
setRMID〔Type2 JDBC ドライバ〕 1304
setSchema(String schema)〔Connection インタフェース〕 1400
setShort(int parameterIndex, short x)〔PreparedStatement インタフェース〕 1469
setShort(String parameterName, short x)〔CallableStatement インタフェース〕 1540
setSQLInNum〔Type2 JDBC ドライバ〕 1313
setSQLInNum〔Type4 JDBC ドライバ〕 1870
setSQLOutNum〔Type2 JDBC ドライバ〕 1314
setSQLOutNum〔Type4 JDBC ドライバ〕 1871
setSQLWarningIgnore 1877
setSQLWarningLevel〔Type2 JDBC ドライバ〕 1316
setSQLWarningLevel〔Type4 JDBC ドライバ〕 1873
setStatementCommitBehavior〔Type4 JDBC ドライバ〕 1888
setString(int parameterIndex, String x)〔PreparedStatement インタフェース〕 1469
setString(String parameterName, String x)〔CallableStatement インタフェース〕 1541
setTime(int parameterIndex, java.sql.Time x,Calendar cal)〔PreparedStatement インタフェース〕 1471
setTime(int parameterIndex, java.sql.Time x)〔PreparedStatement インタフェース〕 1471
setTime(String parameterName, java.sql.Time x,Calendar cal)〔CallableStatement インタフェース〕 1543
setTime(String parameterName, Time x)〔CallableStatement インタフェース〕 1542

setTimestamp(int parameterIndex, java.sql.Timestamp x,Calendar cal)〔PreparedStatement インタフェース〕 1473
setTimestamp(int parameterIndex, java.sql.Timestamp x)〔PreparedStatement インタフェース〕 1472
setTimestamp(String parameterName, java.sql.Timestamp x,Calendar cal)〔CallableStatement インタフェース〕 1545
setTimestamp(String parameterName, java.sql.Timestamp x)〔CallableStatement インタフェース〕 1544
setTransactionIsolation(int level)〔Connection インタフェース〕 1397
setTRC_NO〔Type4 JDBC ドライバ〕 1858
setUapName〔Type4 JDBC ドライバ〕 1859
setUpdateCountBehavior〔Type4 JDBC ドライバ〕 1898
setUser〔Type2 JDBC ドライバ〕 1298
setUser〔Type4 JDBC ドライバ〕 1861
setXACloseString〔Type2 JDBC ドライバ〕 1303
setXACloseString〔Type4 JDBC ドライバ〕 1867
setXALocalCommitMode〔Type4 JDBC ドライバ〕 1875
setXAOpenString〔Type2 JDBC ドライバ〕 1301
setXAOpenString〔Type4 JDBC ドライバ〕 1865
setXAThreadMode〔Type2 JDBC ドライバ〕 1305
SORT MERGE JOIN〔SQL の最適化〕 309
SQL_ACCESS_SECURITY 表の内容 2202
SQL_AUDITS 表の内容 2193
SQL_CHECK_COLUMNS 表の内容 2197
SQL_CHECKS 表の内容 2196
SQL_COLUMN_STATISTICS 表の内容 2173
SQL_COLUMNS 表の内容 2138
SQL_DATATYPE_DESCRIPTORS 表の内容 2176
SQL_DATATYPES 表の内容 2175
SQL_DIV_COLUMN 表の内容 2159
SQL_DIV_INDEX 表の内容 2158
SQL_DIV_TABLE 表の内容 2156
SQL_DIV_TYPE 表の内容 2198
SQL_EXCEPT 表の内容 2186
SQL_INDEX_COLINF 表の内容 2158
SQL_INDEX_DATATYPE 表の内容 2184
SQL_INDEX_FUNCTION 表の内容 2184
SQL_INDEX_RESOURCES 表の内容 2183
SQL_INDEX_STATISTICS 表の内容 2175
SQL_INDEX_TYPE_FUNCTION 表の内容 2185
SQL_INDEX_TYPES 表の内容 2183
SQL_INDEX_XMLINF 表の内容 2200
SQL_INDEXES 表の内容 2148
SQL_IOS_GENERATIONS 表の内容 2186
SQL_KEYCOLUMN_USAGE 表の内容 2195
SQL_PARTKEY_DIVISION 表の内容 2192
SQL_PARTKEY 表の内容 2191
SQL_PHYSICAL_FILES 表の内容 2127
SQL_PLUGIN_ROUTINE_PARAMS 表の内容 2180
SQL_PLUGIN_ROUTINES 表の内容 2179
SQL_PLUGINS 表の内容 2179
SQL_RDAREA_PRIVILEGES 表の内容 2153
SQL_RDAREAS 表の内容 2127
SQL_REFERENTIAL_CONSTRAINTS 表の内容 2194
SQL_ROLE_PRIVILEGES 表の内容 2203
SQL_ROLES 表の内容 2203
SQL_ROUTINE_PARAMS 表の内容 2169
SQL_ROUTINE_RESOURCES 表の内容 2167
SQL_ROUTINES 表の内容 2160
SQL_SEQUENCES 表の内容 2201
SQL_SYSPARAMS 表の内容 2198
SQL_TABLE_CONSTRAINTS 表の内容 2196
SQL_TABLE_PRIVILEGES 表の内容 2153
SQL_TABLE_RESOURCES 表の内容 2178
SQL_TABLE_STATISTICS 表の内容 2173
SQL_TABLES 表の内容 2129
SQL_TRIGGER_COLUMNS 表の内容 2189
SQL_TRIGGER_DEF_SOURCE 表の内容 2189
SQL_TRIGGER_USAGE 表の内容 2190
SQL_TRIGGERS 表の内容 2187
SQL_TYPE_RESOURCES 表の内容 2185
SQL_USERS 表の内容 2151
SQL_VIEW_TABLE_USAGE 表の内容 2155

SQL_VIEWS 表の内容　2156
SQLClientInfoException　1841
SQLDataException　1840
SQLException インタフェース〔JDBC1.2 コア API〕　1796
SQLFeatureNotSupportedException　1840
SQLIntegrityConstraintViolationException　1840
SQLInvalidAuthorizationException　1840
SQLJ　1989
SQLJ トランスレータ　1990, 1993
SQLJ ランタイムライブラリ　1990
SQLNonTransientConnectionException　1840
SQLNonTransientException　1840
SQLRecoverableException　1841
SQLSyntaxErrorException　1840
SQLTimeoutException　1841
SQLTransactionRollbackException　1841
SQLTransientConnectionException　1840
SQLTransientException　1840
SQLWarning インタフェース〔JDBC1.2 コア API〕　1796
SQLWarning クラス〔JDBC1.0 機能〕　1252
SQL 拡張最適化オプション　282, 624
SQL 記述領域　2042
SQL 記述領域操作用マクロ　2056
SQL 記述領域との関係　2075
SQL 記述領域に設定するデータコードとデータの長さ　2046
SQL 記述領域の構成　2042
SQL 記述領域の使用例　2052
SQL 記述領域の展開　2050
SQL 記述領域の展開方法　2055
SQL 記述領域の内容　2043
SQL 最適化オプション　282, 607
SQL 最適化指定　282
SQL 最適化モード　283
SQL 実行時の中間結果情報　971
SQL 実行時の中間結果情報容量　982
SQL ストアドファンクションの実行　270
SQL ストアドファンクションの定義　270
SQL ストアドプロシジャの実行　262
SQL ストアドプロシジャの定義　262
SQL 単位の情報　969
SQL トレース機能　910
SQL トレース情報の取得方法　910
SQL トレース情報の見方　924
SQL トレース動的取得機能　997
SQL トレースの解析　921
SQL トレースファイル　910
SQL トレースファイルのバックアップの取得　945
SQL のエラーの判定と処置　231
SQL の記述　34
SQL の記述規則〔C++言語による UAP の作成〕　758
SQL の記述規則〔COBOL 言語による UAP の作成〕　727
SQL の記述規則〔C 言語による UAP の作成〕　694
SQL の記述規則〔OOCOBOL 言語による UAP の作成〕　759
SQL の機能体系　35
SQL の最適化　282
SQL の最適化の指定方法　296
SQL の実行時に与えられる値　122
SQL のデータ型と COBOL 言語のデータ記述　2097
SQL のデータ型と C 言語のデータ記述　2079
SQL プリプロセサがサポートしているロケール名　768
SQL プリプロセサの起動　769, 780, 785, 795
SQL プリプロセサのリターンコード（UNIX 環境での COBOL 言語の場合）　784
SQL プリプロセサのリターンコード（UNIX 環境での C 言語の場合）　779
SQL プリプロセサのリターンコード（Windows 環境での COBOL 言語の場合）　799
SQL プリプロセサのリターンコード（Windows 環境での C 言語の場合）　794
SQL 文の実行状態と変数に設定される値の関係　231
SQL 文を記述できる箇所　697
SQL 連絡領域　2035
SQL 連絡領域の構成　2035
SQL 連絡領域の展開　2039

SQL 連絡領域の内容　2036
SR モード　139
StatementEventListener　1836
Statement インタフェース〔JDBC1.2 コア API〕1404
Statement クラス〔JDBC1.0 機能〕　1246
Statement クラス〔Type2 JDBC ドライバ〕　1324
storesLowerCaseIdentifiers()〔DatabaseMetaData インタフェース〕　1714
storesLowerCaseQuotedIdentifiers()〔DatabaseMetaData インタフェース〕　1715
storesMixedCaseIdentifiers()〔DatabaseMetaData インタフェース〕　1715
storesMixedCaseQuotedIdentifiers()〔DatabaseMetaData インタフェース〕　1716
storesUpperCaseIdentifiers()〔DatabaseMetaData インタフェース〕　1716
storesUpperCaseQuotedIdentifiers()〔DatabaseMetaData インタフェース〕　1717
SUM　83
supportsAlterTableWithAddColumn()〔DatabaseMetaData インタフェース〕　1718
supportsAlterTableWithDropColumn()〔DatabaseMetaData インタフェース〕　1718
supportsANSI92EntryLevelSQL()〔DatabaseMetaData インタフェース〕　1719
supportsANSI92FullSQL()〔DatabaseMetaData インタフェース〕　1719
supportsANSI92IntermediateSQL()〔DatabaseMetaData インタフェース〕　1720
supportsBatchUpdates()〔DatabaseMetaData インタフェース〕　1720
supportsCatalogsInDataManipulation()〔DatabaseMetaData インタフェース〕　1721
supportsCatalogsInIndexDefinitions()〔DatabaseMetaData インタフェース〕　1721
supportsCatalogsInPrivilegeDefinitions()〔DatabaseMetaData インタフェース〕　1722
supportsCatalogsInProcedureCalls()〔DatabaseMetaData インタフェース〕　1722
supportsCatalogsInTableDefinitions()〔DatabaseMetaData インタフェース〕　1723
supportsColumnAliasing()〔DatabaseMetaData インタフェース〕　1723
supportsConvert()〔DatabaseMetaData インタフェース〕　1724
supportsConvert(int fromType, int toType)〔DatabaseMetaData インタフェース〕　1724
supportsCoreSQLGrammar()〔DatabaseMetaData インタフェース〕　1727
supportsCorrelatedSubqueries()〔DatabaseMetaData インタフェース〕　1728
supportsDataDefinitionAndDataManipulationTransactions()〔DatabaseMetaData インタフェース〕　1728
supportsDataManipulationTransactionsOnly()〔DatabaseMetaData インタフェース〕　1729
supportsDifferentTableCorrelationNames()〔DatabaseMetaData インタフェース〕　1729
supportsExpressionsInOrderBy()〔DatabaseMetaData インタフェース〕　1730
supportsExtendedSQLGrammar()〔DatabaseMetaData インタフェース〕　1731
supportsFullOuterJoins()〔DatabaseMetaData インタフェース〕　1731
supportsGetGeneratedKeys()〔DatabaseMetaData インタフェース〕　1732
supportsGroupBy()〔DatabaseMetaData インタフェース〕　1732
supportsGroupByBeyondSelect()〔DatabaseMetaData インタフェース〕　1733
supportsGroupByUnrelated()〔DatabaseMetaData インタフェース〕　1733
supportsIntegrityEnhancementFacility()〔DatabaseMetaData インタフェース〕　1734
supportsLikeEscapeClause()〔DatabaseMetaData インタフェース〕　1734
supportsLimitedOuterJoins()〔DatabaseMetaData インタフェース〕　1735
supportsMinimumSQLGrammar()〔DatabaseMetaData インタフェース〕　1735
supportsMixedCaseIdentifiers()〔DatabaseMetaData インタフェース〕　1736
supportsMixedCaseQuotedIdentifiers()〔DatabaseMetaData インタフェース〕　1736

supportsMultipleOpenResults()〔DatabaseMetaData インタフェース〕 1737
supportsMultipleResultSets()〔DatabaseMetaData インタフェース〕 1738
supportsMultipleTransactions()〔DatabaseMetaData インタフェース〕 1738
supportsNamedParameters()〔DatabaseMetaData インタフェース〕 1739
supportsNonNullableColumns()〔DatabaseMetaData インタフェース〕 1739
supportsOpenCursorsAcrossCommit()〔DatabaseMetaData インタフェース〕 1740
supportsOpenCursorsAcrossRollback()〔DatabaseMetaData インタフェース〕 1740
supportsOpenStatementsAcrossCommit()〔DatabaseMetaData インタフェース〕 1741
supportsOpenStatementsAcrossRollback()〔DatabaseMetaData インタフェース〕 1741
supportsOrderByUnrelated()〔DatabaseMetaData インタフェース〕 1742
supportsOuterJoins()〔DatabaseMetaData インタフェース〕 1742
supportsPositionedDelete()〔DatabaseMetaData インタフェース〕 1743
supportsPositionedUpdate()〔DatabaseMetaData インタフェース〕 1743
supportsRefCursors()〔DatabaseMetaData インタフェース〕 1767
supportsResultSetConcurrency(int type, int concurrency)〔DatabaseMetaData インタフェース〕 1744
supportsResultSetHoldability(int holdability)〔DatabaseMetaData インタフェース〕 1745
supportsResultSetType(int type)〔DatabaseMetaData インタフェース〕 1745
supportsSavepoints()〔DatabaseMetaData インタフェース〕 1746
supportsSchemasInDataManipulation()〔DatabaseMetaData インタフェース〕 1747
supportsSchemasInIndexDefinitions()〔DatabaseMetaData インタフェース〕 1747
supportsSchemasInPrivilegeDefinitions()〔DatabaseMetaData インタフェース〕 1748
supportsSchemasInProcedureCalls()〔DatabaseMetaData インタフェース〕 1748
supportsSchemasInTableDefinitions()〔DatabaseMetaData インタフェース〕 1749
supportsSelectForUpdate()〔DatabaseMetaData インタフェース〕 1749
supportsSharding()〔DatabaseMetaData インタフェース〕 1767
supportsStatementPooling()〔DatabaseMetaData インタフェース〕 1750
supportsStoredFunctionsUsingCallSyntax()〔DatabaseMetaData インタフェース〕 1758
supportsStoredProcedures()〔DatabaseMetaData インタフェース〕 1750
supportsSubqueriesInComparisons()〔DatabaseMetaData インタフェース〕 1751
supportsSubqueriesInExists()〔DatabaseMetaData インタフェース〕 1751
supportsSubqueriesInIns()〔DatabaseMetaData インタフェース〕 1752
supportsSubqueriesInQuantifieds()〔DatabaseMetaData インタフェース〕 1752
supportsTableCorrelationNames()〔DatabaseMetaData インタフェース〕 1753
supportsTransactionIsolationLevel(int level)〔DatabaseMetaData インタフェース〕 1753
supportsTransactions()〔DatabaseMetaData インタフェース〕 1754
supportsUnion()〔DatabaseMetaData インタフェース〕 1754
supportsUnionAll()〔DatabaseMetaData インタフェース〕 1755
SU モード 139

T

TABLE SCAN〔SQL の最適化〕 319
TRUE 76
Type2 JDBC ドライバ 1235
Type4 JDBC ドライバ 1335

U

UAP からのコマンド実行 1008

UAP 実行時の注意事項　832
UAP 実行前の準備　763
UAP 障害の回復　1006
UAP 中での SQL の基本構成　112
UAP で使用するデータ型とアクセサ　1192
UAP でできる排他制御　192
UAP と HiRDB とのインタフェース　36
UAP 統計レポート機能　963
UAP 統計レポートの取得方法　963
UAP 統計レポートの見方　965
UAP の開発の流れ　33
UAP の記述　118
UAP の記述言語　118
UAP の基本構成　692
UAP の形式　34
UAP の構成要素　692
UAP の作成　691
UAP の実行手順　764
UAP の障害対策　909
UAP の設計　111
UAP の動作環境　37
UAP の特長　34
Unicode　549
UNKNOWN　76
unwrap(Class<T> iface)〔Wrapper インタフェース〕　1839
updatesAreDetected(int type)〔DatabaseMetaData インタフェース〕　1755
UPDATE 文の SET 句　60
UPDATE 文の WHERE 句　59
usesLocalFilePerTable()〔DatabaseMetaData インタフェース〕　1756
usesLocalFiles()〔DatabaseMetaData インタフェース〕　1756
UTF-16　844
UTF-8　549

W

wasNull()〔CallableStatement インタフェース〕　1545
wasNull()〔ResultSet インタフェース〕　1621
WORK TABLE ATS SUBQ〔SQL の最適化〕　325
WORK TABLE SUBQ〔SQL の最適化〕　326
Wrapper インタフェース〔JDBC4.0 API〕　1838

X

X/Open に従った API（TX_関数）を使用した UAP の実行　836
XAConnection インタフェース〔JDBC2.0 Optional Package〕　1820
XADataSource インタフェース〔JDBC2.0 Optional Package〕　1821
XAException インタフェース〔JDBC2.0 Optional Package〕　1824
XAResource インタフェース〔JDBC2.0 Optional Package〕　1823
XDM/RD と UNIFY2000 で作成した UAP の移行性　845
XML 型全文検索用インデクスを使用した検索　106

あ

アウタジョイン　86
アクセスパス情報　970
アクセスパス情報の容量　971
値式に対する結合条件適用機能〔SQL 拡張最適化オプション〕　627
アプリケーション構成ファイル　1229
アンインストール　448

い

一意性制約　120
一時表〔環境変数〕　523
一時表行挿入情報　973
位置付け子機能　413
一括ハッシュジョイン〔ハッシュジョインの処理方式〕　312
意図共用モード　139
意図排他モード　139
インストール　448
インストール〔Type2 JDBC ドライバ〕　1236
インストール〔Type4 JDBC ドライバ〕　1336

インタフェース 36
インタフェース領域の種類と使用目的 119
インデクス型プラグイン専用関数 440
インデクスキー値の排他資源の作成方法 212
インデクスキー値無排他 150, 153, 174
インデクス検索時の留意事項 242
インデクススキャン〔SQL の最適化〕 318
インデクスの提案 237
インデクス利用の抑止〔SQL 最適化オプション〕 618
インナレプリカ機能を使用した場合の絞込み検索 395

う

ウィンドウ関数 417
埋込み SQL 宣言節内で記述できる項目 696
埋込み SQL 宣言節の不要化 801
埋込み型 34, 118
埋込み型 UAP の概要 692
埋込み変数及び標識変数の宣言 112

え

エスケープ句 1373
エスケープ句で指定できるスカラ関数 2238
エラー検出時の対処 233
エラー時の処置の指定 114
エラーの自動判定 234
エラーの判定 116, 231

お

オブジェクトファイルの作成 900
オブジェクトリレーショナルデータベースの表 47
オペレーションコード 934

か

カーソル宣言 114, 121
カーソル宣言と排他の関係 219
カーソルの効果 216
カーソルの更新可能性とカーソルを使用しない操作との関連 216
カーソルの使用例 228
カーソルの定義 53
カーソルの利用 49
カーソルライブラリの設定 1071
カーソルを使用した検索 49
カーソルを使用した検索行の更新 50
カーソルを使用した更新 58
カーソルを使用した削除 63
カーソルを使用しない検索 51
カーソルを閉じる 54
カーソルを開く 53
外結合内結合変換機能〔SQL 拡張最適化オプション〕 631
外部 C ストアドルーチン 895
外部 C ストアドルーチンの作成 898
外部 C ストアドルーチンの実行 904
外部 C ストアドルーチンの定義 903
外部 C ストアドルーチンの特長 895
外部 Java ストアドルーチン 856
外部 Java ストアドルーチン実行前の準備 858
外部 Java ストアドルーチンの作成 859
外部 Java ストアドルーチンの実行 863
外部 Java ストアドルーチンの定義 862
外部 Java ストアドルーチンの特長 858
拡張 SQL エラー情報出力機能 952
型名記述領域 2066
型名記述領域の構成 2066
型名記述領域の展開 2067
型名記述領域の内容 2066
環境設定〔Type2 JDBC ドライバ〕 1236
環境設定〔Type4 JDBC ドライバ〕 1337
環境変数〔HiRDB SQL Tuning Advisor 用アクセスパス情報ファイル〕 520
環境変数〔JDBC ドライバ〕 524
環境変数〔ODBC 関数〕 523
環境変数〔OLTP 下の X/Open に従った API を使用するクライアント〕 516
環境変数〔SQL 関連〕 521
環境変数〔UAP からのコマンド実行〕 517
環境変数〔UAP に関する統計情報の出力単位〕 521

環境変数〔XDM/RD E2 接続機能〕 522
環境変数〔アクセスパス表示ユティリティ用アクセスパス情報ファイル〕 520
環境変数〔インナレプリカ機能〕 522
環境変数グループ 671
環境変数〔更新可能なオンライン再編成〕 522
環境変数〔参照制約及び検査制約〕 523
環境変数〔システム監視〕 518
環境変数〔システム構成〕 515
環境変数〔データの空白変換〕 522
環境変数〔トラブルシュート〕 519
環境変数のグループ登録 671
環境変数の指定方法〔OpenTP1 下の UAP をクライアントとする場合〕 502
環境変数の指定方法〔TP1/EE 下の UAP をクライアントとする場合〕 512
環境変数の指定方法〔TP1/LiNK 下の UAP をクライアントとする場合〕 505
環境変数の指定方法〔TPBroker for C++下の UAP をクライアントとする場合〕 507
環境変数の指定方法〔TUXEDO 下の UAP をクライアントとする場合〕 510
環境変数の設定 492
環境変数〔排他制御〕 521
環境変数〔バックエンドサーバ接続保持機能〕 523
環境変数〔プラグイン〕 524
環境変数〔プロセス間メモリ通信機能〕 518
環境変数〔ブロック転送機能〕 522
環境変数〔ユーザ実行環境〕 516
関数 95
関数の作成方法 270

き

キースキャン〔SQL の最適化〕 318
記述規則 727, 758
行 45
行識別子を使用した検索〔SQL の最適化〕 323
行数 83
行単位の行の挿入 68
行単位の検索 57
行単位の更新 61
行値実行〔SQL の最適化〕 324
行に掛かる排他制御の順序 204
行の削除の処理手順 63
行排他 141, 143
共用意図排他モード 139
共用モード 139

く

空集合 79
空白変換関数 2228
クライアントエラーログ機能 949
クライアントエラーログ情報の取得方法 950
クライアントエラーログ情報の見方 950
クライアントエラーログファイル 949
クライアントエラーログファイルのバックアップの取得 951
クライアント環境定義 492
クライアント環境定義の一覧 515
クライアント環境定義の設定内容 526
クライアントの環境設定 445
繰返し列がある表の更新 61
繰返し列がある表への行の挿入 68
繰返し列を?パラメタにしたときの値の指定方法 1280
繰返し列を使用した表 45
グループの平均値 83
グループ分け高速化機能 375
グループ分け高速化処理〔SQL 最適化オプション〕617
グループ分け高速化処理〔SQL の最適化〕 304
グループ分け処理方式〔SQL の最適化〕 303
グローバルデッドロック 164
グローバルデッドロックの例 165

け

結果集合返却機能（Java ストアドプロシジャ限定） 882
結果集合返却機能（SQL ストアドプロシジャ限定）267
結果セットの拡張〔JDBC2.0 基本機能〕 1254
結果セットの拡張〔JDBC2.1 コア API〕 1799

結合処理情報　976
結合方式〔SQL の最適化〕　307
結合方式の種類〔SQL の最適化〕　307
検索，更新の SQL（実行文）　115
検索した内容を取り出して UAP の領域に格納　54
検索方式〔SQL の最適化〕　317
検査保留状態　155, 156
限定述語　78
限定述語を使用した副問合せ　78

こ

合計値　83
更新 SQL の作業表作成抑止〔SQL 最適化オプション〕　618
構成ファイル　1204, 1229
構造化繰返し述語を使用した検索　76
構造体の参照　804
高速接続機能　530
コーディング例　703, 711, 720, 736, 749
コストベース最適化モード 1　283
コストベース最適化モード 2　283
コストベース最適化モード 2 の適用〔SQL 拡張最適化オプション〕　627
コマンドトレース機能　995
コマンドトレース情報の取得方法　995
コマンドトレース情報の見方　995
コマンドトレースファイルのバックアップの取得　997
コミットしていない削除データの排他制御　177
コンパイル　816, 900
コンパイル〔Java ファイル〕　860
コンポネント指定　95

さ

サーバ間で発生するデッドロック　164
最小値　83
最小排他資源単位の設定　138
再接続トレース機能　999
最大値　83
最適化方法の種類　295
採番業務で使用する表　250
作業表 ATS 実行〔SQL の最適化〕　323
作業表実行〔SQL の最適化〕　323
作業表用バッファサイズの求め方　334
サポートしていないインタフェース〔JDBC1.2 コア API〕　1798
サポートしていないインタフェース〔JDBC2.1 コア API〕　1806

し

時刻データの演算　81
システムプロパティの設定〔Type2 JDBC ドライバ〕　1287
実表　46
実表検索処理情報　977
指定できるクライアント環境定義〔Type4 JDBC ドライバ〕　1921
自動再接続機能　407
自動採番機能　421
自バックエンドサーバでのグループ化，ORDER BY，DISTINCT 集合関数処理〔SQL 最適化オプション〕617
自バックエンドサーバでのグループ化，ORDER BY，DISTINCT 集合関数処理〔SQL の最適化〕　304
絞込み検索　390
集合演算処理情報　973
条件指定による更新　59
条件指定による削除　64
条件推移での探索高速化条件の導出　341

す

数値データの四則演算　81
スカラ演算を含むキー条件の適用〔SQL 最適化オプション〕　620
スカラ演算を含む条件に対するサーチ条件適用〔SQL 拡張最適化オプション〕　628
スカラ関数　82
ストアドファンクション　270
ストアドプロシジャ　261
ストアドルーチン　261

せ

整合性制約　119
静的 SQL　121
静的 SQL と動的 SQL の実行時の特徴　122
性能向上，操作性向上に関する UAP の設計　236
接続情報の優先順位〔Type4 JDBC ドライバ〕　1934
接続プーリング機能　1200
接続プール〔JDBC2.0 Optional Package〕　1271
先頭から n 行の検索結果を取得する機能　405

そ

総ヒット件数返却機能　417
ソースプログラムの記述　34
ソート　84
外結合　86
外への参照のある副問合せの実行方式〔SQL の最適化〕　328
外への参照のない副問合せの実行方式〔SQL の最適化〕　323

た

探索高速化条件の適用範囲　337
探索高速化条件の導出　337
探索高速化条件の導出〔SQL 最適化オプション〕　620
断続ハッシュジョイン〔ハッシュジョインの処理方式〕　312

ち

抽象データ型がある表の行の削除　108
抽象データ型がある表の検索　106
抽象データ型がある表の更新　107
抽象データ型がある表への行の挿入　109
抽象データ型を含む表のデータ操作　95
長時間 SQL の時間監視　578
重複したデータの排除　84

つ

追加されたデータ型〔JDBC2.0 基本機能〕　1260
追加されたデータ型〔JDBC2.1 コア API〕　1805

て

ディクショナリ表の検索方法　2124
ディクショナリ表の詳細　2126
データ型　45
データ型〔Type2 JDBC ドライバ〕　1320
データ型〔Type4 JDBC ドライバ〕　1901
データ収集用サーバの分離機能〔SQL 最適化オプション〕　618
データ収集用サーバの分離機能〔SQL の最適化〕　300
データディクショナリ一覧　2121
データディクショナリ表の検索　2116
データディクショナリ表を検索する SQL 文の例　2124
データの演算　81
データの加工　83
データのグループ分け　83
データの検索　52
データの更新　58
データの削除　63
データの挿入　67
データの取り出し　54
データの並べ替え　84
データベースの操作　44
データ保証レベル　344, 606
データ保証レベル 0　345
データ保証レベル 1　345
データ保証レベル 2　346
テーブルスキャン〔SQL の最適化〕　317
テスト〔外部 Java ストアドルーチン〕　860
手続きの作成方法　261
デッドロック　162
デッドロックが発生する処理と対策　169
デッドロックの回避策　170
デッドロックの検出　167
デッドロックの対策　168
デッドロックの発生要因　162
デッドロックの例　163
デッドロックプライオリティ値による排他制御　169
デバッグ〔外部 Java ストアドルーチン〕　860

と

問合せ処理情報　974
同期点の設定　132, 134
導出表の条件繰り込み機能〔SQL 最適化オプション〕　622
動的 SELECT 文　121
動的 SQL　121
動的 SQL の実行と留意点　122
特定データの探索　70
特定の文字パターンの探索　73
トランザクション制御　132
トランザクションの移行　135
トランザクションの開始　134
トランザクションの開始と終了　132
トランザクションの無効化　117
トランザクションの有効化　116
トリガ　281
トリガ SQL 文　281
トリガ動作の探索条件　281
トリガを引き起こす SQL　281
トレース取得コマンド　997

な

内部的に作成した作業表の検索〔SQL の最適化〕　322
並べ替え　84
ナル値でないデータの探索　74

ね

ネイティブランタイム　2023
ネストループ行値実行〔SQL の最適化〕　328
ネストループ作業表実行〔SQL の最適化〕　328
ネストループジョイン〔SQL の最適化〕　308
ネストループジョイン強制〔SQL 最適化オプション〕　614
ネストループジョイン優先〔SQL 最適化オプション〕　615

の

ノンブロックモード　587

は

バージョン，リビジョンによるクライアント環境定義及びプリプロセスオプションの変更点　846
バージョンアップした場合に必要な作業　846
バージョンによるプリプロセスオプションの変更点 810
排他資源とその包含関係　137
排他時の参照　162
排他制御　137
排他制御の単位　137
排他制御のモード　138
排他制御のモードの組み合わせ〔インデクスキー値無排他〕　150, 153
排他制御のモードの組み合わせ〔行排他〕　141, 143
排他制御のモードの組み合わせ〔検査保留状態〕　155, 156
排他制御のモードの組み合わせ〔ページ排他〕 145, 148
排他制御のモードの遷移規則　140
排他制御モードの違いによる 2 ユーザの同時実行性 139
排他の開始と解放　162
排他の期間　162
排他モード　139
配列を使用した DELETE 機能　372
配列を使用した FETCH 機能　352
配列を使用した INSERT 機能　359
配列を使用した UPDATE 機能　369
配列を使用した機能　352
バケット分割ハッシュジョイン〔ハッシュジョインの処理方式〕　312
ハッシュ実行〔SQL の最適化〕　324, 328
ハッシュジョイン，副問合せのハッシュ実行〔SQL 拡張最適化オプション〕　627
ハッシュジョイン，副問合せのハッシュ実行を適用する場合の準備　331
ハッシュジョイン〔SQL の最適化〕　308
ハッシュジョインの処理方式　312
ハッシュ表サイズ　332
ハッシュ表最大行長　332
バッチ更新〔JDBC2.0 基本機能〕　1256
バッチ更新〔JDBC2.1 コア API〕　1800
パブリックファンクション　271

パブリックプロシジャ　263
パブリックルーチン　261
パラメタのデータ長及びデータタイプの仮定　1222
パラメタを含む XMLEXISTS 述語への部分構造インデクスの有効化〔SQL 拡張最適化オプション〕　629

ひ

比較述語　70
比較述語を使用したデータの探索　71
比較条件　78
日付データの演算　82
非ナル値制約　120
ビュー表　46
ビュー表の操作　94
ビュー表の定義　89
ビュー表の定義と操作　89
表に対する操作　250
表の基本構成　45
表の更新の処理手順　58
表の全行削除　65
表の外結合　86
表分割ハッシュ関数　2205

ふ

複数インデクス利用の強制〔SQL 最適化オプション〕　618
複数接続機能　378
複数接続機能を使用する場合のコンパイルとリンケージ　829
複数トランザクションで検索と更新をする場合　213
複数の SQL オブジェクト作成〔SQL 最適化オプション〕　614
複数の条件を満たすデータの探索　75
複数の表からの検索　55
副問合せ　77
副問合せを使用した検索　77
不正電文トレース　1975
部分構造インデクスを使用した検索　105
プラグインインデクスでの他インデクス絞り込み結果の利用　431
プラグインインデクスでの他インデクス絞り込み結果の利用の抑止〔SQL 拡張最適化オプション〕　630
プラグインが使用する論理ファイルのデッドロック回避策　171
プラグイン提供関数　439
プラグイン提供関数からの一括取得機能〔SQL 最適化オプション〕　621
プリプロセサ宣言文の有効化　799
プリプロセス　767
プリプロセス〔UNIX 環境，COBOL 言語の場合〕　779
プリプロセス〔UNIX 環境，C 言語の場合〕　768
プリプロセス〔Windows 環境，COBOL 言語の場合〕　794
プリプロセス〔Windows 環境，C 言語の場合〕　785
プリプロセスの概要　767
フロータブルサーバ候補数の拡大〔SQL 最適化オプション〕　616
フロータブルサーバ候補数の拡大〔SQL の最適化〕301
フロータブルサーバ対象拡大（データ取り出しバックエンドサーバ）〔SQL 最適化オプション〕　614
フロータブルサーバ対象拡大（データ取り出しバックエンドサーバ）〔SQL の最適化〕　300
フロータブルサーバ対象限定（データ取り出しバックエンドサーバ）〔SQL 最適化オプション〕　618
フロータブルサーバ対象限定（データ取り出しバックエンドサーバ）〔SQL の最適化〕　300
フロータブルサーバの割り当て候補数に関する最適化　300
フロータブルサーバの割り当てに関する最適化　300
フロータブルサーバの割り当て方法〔SQL の最適化〕　297
フロータブルサーバを使用する SQL　299
プログラム例〔外部 C ストアドルーチン〕　905
プログラム例〔外部 Java ストアドルーチン〕　865
プログラム例題　701, 733
プロシジャ　261
プロセス間メモリ通信機能　576
プロセス残存の回避　579, 580
ブロック転送機能　348
ブロックモード　587
プロバイダ名　1203

分散トランザクション〔JDBC2.0 Optional Package〕 1272

へ

ページ排他 145, 148

ほ

ポインタでの埋込み変数指定 802
ポート番号 527, 534
ホールダブルカーソル 224
ホールダブルカーソルを使用した例 229

ま

マージジョイン〔SQL の最適化〕 308
マシン構成ファイル 1229
マルチスレッドを使用した UAP の SQL の実行 128

み

未使用インデクスの調査 240

む

無応答状態〔HiRDB クライアント〕 578
無排他条件判定 172

め

名標の付け方〔C++言語による UAP の作成〕 758
名標の付け方〔COBOL 言語による UAP の作成〕 727
名標の付け方〔OOCOBOL 言語による UAP の作成〕 759
名標の付け方の規則〔C 言語による UAP の作成〕 694
メモリ容量見積もり［クライアントライブラリ］ 680

も

モードの遷移〔排他制御〕 140
文字コード種別 768, 779
文字コード種別設定関数 2235
文字コード変換機能〔Type2 JDBC ドライバ〕 1321
文字コード変換機能〔Type4 JDBC ドライバ〕 1920
文字集合名記述領域 2069
文字集合名記述領域操作用マクロ 2077
文字集合名記述領域の SQLCSN に設定できる文字集合情報 2074
文字集合名記述領域の構成 2069
文字集合名記述領域の展開 2075
文字集合名記述領域の内容 2070
文字集合を使用した場合の文字コード変換規則 2243

ゆ

優先順位〔SQL の最適化〕 297

よ

呼び出す関数の決定規則 275

り

リターンコードの参照 231
リレーショナルデータベースの表 45
リンケージ 816, 900

れ

列 45
列単位の挿入 67
列名 45
列名記述領域 2062
列名記述領域の構成 2062
列名記述領域の展開 2064
列名記述領域の内容 2062
連続ハッシュジョイン〔ハッシュジョインの処理方式〕 312

ろ

ロールバックの設定 132, 134, 135
ログ取得モード 547
ログレスモード 547
論理述語を使用した検索 76