IBM® DB2 Universal Database™

アプリケーション開発ガイド
クライアント・アプリケーションのプログラミング

バージョン 8.2

SC88-9138-01
(英文原典：SC09-4826-01)

IBM® DB2 Universal Database™

アプリケーション開発ガイド
クライアント・アプリケーションのプログラミング

バージョン 8.2

SC88-9138-01
(英文原典：SC09-4826-01)

ご注意！

本書および本書で紹介する製品をご使用になる前に、『*特記事項*』に記載されている情報をお読みください。

本マニュアルに関するご意見やご感想は、次の URL からお送りください。今後の参考にさせていただきます。

`http://www.ibm.com/jp/manuals/main/mail.html`

なお、日本 IBM 発行のマニュアルはインターネット経由でもご購入いただけます。詳しくは

`http://www.ibm.com/jp/manuals/` の「ご注文について」をご覧ください。

(URL は、変更になる場合があります)

お客様の環境によっては、資料中の円記号がバックスラッシュと表示されたり、バックスラッシュが円記号と表示されたりする場合があります。

原 典：	SC09-4826-01 IBM® DB2 Universal Database ™ Application Development Guide: Programming Client Applications Version 8.2
発 行：	日本アイ・ビー・エム株式会社
担 当：	ナショナル・ランゲージ・サポート

第1刷 2004.8

この文書では、平成明朝体™W3、平成明朝体™W9、平成角ゴシック体™W3、平成角ゴシック体™W5、および平成角ゴシック体™W7を使用しています。この(書体*)は、(財) 日本規格協会と使用契約を締結し使用しているものです。フォントとして無断複製することは禁止されています。

注* 平成明朝体™W3、平成明朝体™W9、平成角ゴシック体™W3、
平成角ゴシック体™W5、平成角ゴシック体™W7

目次

本書について xiii

第 1 部 入門 1

第 1 章 サポートされているプログラミング・インターフェースの概説 3
アプリケーション開発用の DB2 Universal Database ツール 3
DB2 Development Add-In の概要. 4
サポートされているプログラミング・インターフェース 6
DB2 がサポートしているプログラミング・インターフェース 6
DB2 アプリケーション・プログラミング・インターフェース 8
組み込み SQL 8
DB2 コール・レベル・インターフェース . . . 10
DB2 CLI と組み込み動的 SQL の比較 11
Java Database Connectivity (JDBC) 13
組み込み SQL for Java (SQLJ) 14
ADO (ActiveX Data Object) および RDO (Remote Data Object) 15
Perl DBI 16
ODBC エンド・ユーザー・ツール 16
DB2 .NET Data Provider 16
Web アプリケーション 17
Web アプリケーションの構築のためのツール . . 17
WebSphere Studio 17
XML Extender 18
MQSeries 使用可能性 19
Net.Data. 19
プログラミング機能 20
DB2 プログラミング機能. 20
DB2 ストアード・プロシージャー. 21
DB2 ユーザー定義関数およびメソッド 22
Development Center 22
ユーザー定義タイプ (UDT) およびラージ・オブジェクト (LOB) 24
OLE オートメーション・ルーチン. 25
OLE DB 表関数 25
DB2 トリガー 26

第 2 章 DB2 アプリケーションのコーディング 29
プログラミングの前提条件 29
DB2 アプリケーションのコーディングの概説 . . . 30
スタンドアロン・アプリケーションのプログラミング. 30
スタンドアロン・アプリケーションの宣言セクションの作成 31
データベース・マネージャーと対話する変数の宣言 31
SQL オブジェクトを表す変数の宣言 32
db2dclgn 宣言生成プログラムを使用したホスト変数の宣言 34
ホスト変数と SQL ステートメントの関連付け. . 35
エラー処理のための SQLCA の宣言 36
WHENEVER ステートメントを用いたエラー処理 37
アプリケーションへの非実行ステートメントの追加 38
アプリケーションのデータベースへの接続 . . . 38
トランザクションのコーディング 39
COMMIT ステートメントによるトランザクションの終了 40
ROLLBACK ステートメントによるトランザクションの終了 41
アプリケーション・プログラムの終了 42
スタンドアロン・アプリケーションのトランザクションの暗黙的な終了. 43
疑似コードによるアプリケーションの枠組み . . 44
SQL ステートメントのプロトタイプのための機能 45
組み込み SQL または DB2 CLI プログラムにおける管理 API. 46
データ値とリレーションシップの制御 46
データ値の制御 46
データ・タイプを使用したデータ値の制御 . . . 47
ユニーク制約を使用したデータ値の制御. . . . 47
表チェック制約を使用したデータ値の制御 . . . 48
参照保全制約を使用したデータ値の制御. . . . 48
CHECK OPTION のあるビューを使用したデータ値の制御 48
アプリケーションのロジックとプログラム変数のタイプを使用したデータ値の制御 49
データ・リレーションシップの制御 49
参照保全制約を使用したデータ・リレーションシップの制御 49
トリガーを使用したデータ・リレーションシップの制御 50
BEFORE トリガーを使用したデータ・リレーションシップの制御 50
AFTER トリガーを使用したデータ・リレーションシップの制御 51
アプリケーション・ロジックを使用したデータ・リレーションシップの制御 51
サーバーにおけるアプリケーションのロジック . 51
SQL および API における許可に関する考慮事項. . 53
組み込み SQL における許可に関する考慮事項. . 53
動的 SQL における許可に関する考慮事項 . . . 54
静的 SQL における許可に関する考慮事項 . . . 55
API における許可に関する考慮事項 55
アプリケーションのテスト 56

アプリケーションのテスト環境のセットアップ . 56
アプリケーションのデバッグと最適化 60

第 2 部 組み込み SQL 61

第 3 章 組み込み SQL の概説 63
ホスト言語における組み込み SQL ステートメントの使用 63
ソース・ファイルの作成と準備 65
パッケージ、バインディング、および組み込み SQL 67
組み込み SQL 用のパッケージの作成. 67
組み込み SQL を含むソース・ファイルのプリコンパイル 69
組み込み SQL アプリケーション用のソース・ファイル要件 71
組み込み SQL を含むソース・ファイルのコンパイルとリンケージ 72
BIND コマンドを使用したパッケージの作成 . . 73
パッケージにバージョンを付ける 74
バインドされる動的 SQL における特殊レジスターの影響 75
パッケージ・スキーマ用 CURRENT PACKAGE PATH 特殊レジスター. 76
非修飾表名の解決 79
バインディング時のその他の考慮事項 80
実行据え置きバインドの利点 81
ファイル内容のバインド 81
アプリケーション、バインド・ファイル、およびパッケージの関係 82
プリコンパイラー生成タイム・スタンプ. . . . 82
パッケージの再バインド 84

第 4 章 静的 SQL プログラムの作成 . . 87
静的 SQL を使用する場合の特性とそれを使用する理由. 87
静的 SQL の利点 88
静的 SQL プログラムの例 89
静的 SQL プログラムにおけるデータ検索 90
静的 SQL における REOPT の影響 91
静的 SQL プログラムにおけるホスト変数 91
静的 SQL におけるホスト変数 91
静的 SQL プログラムにおけるホスト変数の宣言 93
静的 SQL プログラムにおけるホスト変数の参照 94
静的 SQL プログラムにおける標識変数 94
静的 SQL プログラムにおける標識変数の組み込み 95
静的 SQL プログラムにおける標識変数のデータ・タイプ 97
静的 SQL プログラムにおける標識変数の例 . . 99
カーソルを用いた複数行の選択 100
カーソルを用いた複数行の選択 100
静的 SQL プログラムにおけるカーソルの宣言と使用 101
カーソル・タイプおよび作業単位に関する考慮事項 102
静的 SQL プログラムにおけるカーソルの例 . . 104
検索されたデータの取り扱い 105
静的 SQL プログラムにおける検索データの更新と削除. 105
カーソル・タイプ 106
静的 SQL プログラムにおける取り出しの例 . . 107
検索されたデータのスクロールと取り扱い. . . . 108
以前に検索されたデータのスクロール 108
データのコピーを保持する方法 109
データを 2 度検索する方法 109
最初の結果表と 2 番目の結果表の行の順序の違い 110
カーソルの位置を表の最後にする. 111
以前に検索されたデータの更新 112
静的 SQL プログラムにおける挿入、更新、および削除の例 112
診断情報 114
戻りコード 114
SQLCODE、SQLSTATE、および SQLWARN フィールドのエラー情報 114
SQLCA 構造体におけるトークンの切り捨て . . 115
例外、シグナル、および割り込みハンドラーについての考慮事項. 115
出口リスト・ルーチンに関する考慮事項 . . . 116
アプリケーション中でのエラー・メッセージの検索 116

第 5 章 動的 SQL プログラムの作成 119
動的 SQL を使用する場合の特性とそれを使用する理由 119
動的 SQL を使用する理由 119
動的 SQL サポート・ステートメント 120
動的 SQL と静的 SQL 121
動的 SQL におけるカーソル 123
動的 SQL プログラムにおけるカーソルの宣言と使用 123
動的 SQL プログラムにおけるカーソルの例 . . 124
動的 SQL における REOPT の影響 126
動的 SQL における DYNAMICRULES BIND オプションの影響 126
動的 SQL プログラムにおける SQLDA. 128
動的 SQL におけるホスト変数と SQLDA . . . 128
動的 SQL プログラムにおける SQLDA 構造体の宣言. 129
最小の SQLDA 構造体を用いた動的 SQL におけるステートメントの準備 131
十分な数の SQLVAR 項目を指定した動的 SQL プログラム用の SQLDA の割り振り. 133
動的 SQL プログラムにおける SELECT ステートメントの記述. 134
行を保持するためのストレージの獲得 134
動的 SQL プログラムにおけるカーソルの処理 135
動的 SQL プログラムにおける SQLDA 構造体の割り振り 136
SQLDA 構造体を使用した動的 SQL プログラムにおけるデータ転送 139

動的 SQL プログラムにおける対話式 SQL ステートメントの処理 140
動的 SQL プログラムにおけるステートメント・タイプの判別 140
動的 SELECT プログラムにおける可変リスト SQL ステートメントの処理. 141
エンド・ユーザーからの SQL 要求の保管. . . . 142
動的 SQL プログラムにおけるパラメーター・マーカー 142
パラメーター・マーカーを使用する動的 SQL への変数入力の提供 142
動的 SQL プログラムにおけるパラメーター・マーカーの例 143
DB2 コール・レベル・インターフェース (CLI) と動的 SQL との比較 145
DB2 コール・レベル・インターフェース (CLI) と組み込み動的 SQL の比較 145
組み込み SQL と比較した DB2 CLI の利点 . . 146
DB2 CLI または組み込み SQL をいつ使用するか 148

第 6 章 C および C++ でのプログラミング 151
C/C++ に関するプログラミング考慮事項 151
C および C++ での 3 文字表記 151
C および C++ の入出力ファイル. 152
組み込みファイル 152
C および C++ の組み込みファイル 152
C および C++ における組み込みファイル. . . 155
C および C++ での組み込み SQL ステートメント 156
C および C++ でのホスト変数 158
C および C++ でのホスト変数 158
C および C++ でのホスト変数名. 159
C および C++ でのホスト変数の宣言 160
C および C++ における数値ホスト変数の構文 160
C および C++ における固定および NULL 終了文字ホスト変数の構文 162
C または C++ における可変長文字ホスト変数の構文 162
C および C++ での標識変数 164
C および C++ でのグラフィック・ホスト変数 164
C および C++ における単純グラフィックおよび NULL 終了グラフィック書式のグラフィック宣言の構文 165
C または C++ における VARGRAPHIC 構造書式のグラフィック宣言の構文 166
C または C++ におけるラージ・オブジェクト (LOB) ホスト変数の構文 167
C または C++ におけるラージ・オブジェクト (LOB) ロケーター・ホスト変数の構文 169
C または C++ におけるファイル参照ホスト変数宣言の構文 170
C および C++ におけるホスト変数の初期化 . . 171
C マクロ展開 171
C および C++ でのホスト構造体サポート. . . 172
C および C++ での標識表 174
C および C++ でのヌル終了ストリング . . . 175
C および C++ でポインター・データ・タイプとして使用されるホスト変数 176
C および C++ でホスト変数として使用されるクラス・データ・メンバー. 177
C および C++ での修飾およびメンバー演算子 178
C および C++ でのマルチバイト文字のエンコード 179
C および C++ での wchar_t および sqldbchar データ・タイプ. 179
C および C++ での WCHARTYPE プリコンパイラー・オプション 180
C および C++ での日本語または中国語 (繁体字) EUC、および UCS-2 に関する考慮事項 . . 183
C および C++ 用のホスト変数がある SQL 宣言セクション 184
C および C++ におけるデータ・タイプに関する考慮事項. 186
C および C++ においてサポートされている SQL データ・タイプ 186
C および C++ における FOR BIT DATA . . . 190
プロシージャー、関数、およびメソッド用の C/C++ データ・タイプ 190
C および C++ における SQLSTATE および SQLCODE 変数 192

第 7 章 C および C++ アプリケーション用のマルチスレッド・データベース・アクセス. 193
マルチスレッド・データベース・アクセスの目的 193
マルチスレッドの使用に際しての推奨事項. . . . 194
マルチスレッド UNIX アプリケーション用のコード・ページおよび国別/地域別コードに関する考慮事項 195
マルチスレッド・アプリケーションのトラブルシューティング 196
複数のスレッドの使用に関する潜在的な問題 . . 196
複数コンテキストにおけるデッドロックの回避方法 196

第 8 章 COBOL でのプログラミング 199
COBOL に関するプログラミング考慮事項. . . . 199
COBOL での言語制限 199
COBOL におけるマルチスレッド・データベース・アクセス 199
COBOL の入力および出力ファイル 200
COBOL の組み込みファイル 200
COBOL における組み込み SQL 203
COBOL のホスト変数 205
COBOL のホスト変数 205
COBOL におけるホスト変数の名前 205
COBOL におけるホスト変数宣言. 206
COBOL における数値ホスト変数の構文 . . . 206
COBOL における固定長ホスト変数の構文. . . 207
COBOL における固定長グラフィック・ホスト変数の構文 209

COBOL の標識変数 209
COBOL における LOB ホスト変数の構文. . . 210
COBOL における LOB ロケーター・ホスト変数の構文. 211
COBOL におけるファイル参照ホスト変数の構文 211
COBOL でのホスト構造サポート. 212
COBOL の標識表 214
COBOL グループ・データ項目での REDEFINES 214
COBOL 用のホスト変数がある SQL 宣言セクション 215
COBOL におけるデータ・タイプに関する考慮事項 215
COBOL でサポートされている SQL データ・タイプ 215
BINARY/COMP-4 COBOL データ・タイプ . . 218
COBOL での FOR BIT DATA. 218
COBOL での SQLSTATE および SQLCODE 変数 218
COBOL での日本語または中国語 (繁体字) EUC、および UCS-2 に関する考慮事項. 219
オブジェクト指向 COBOL 219

第 9 章 FORTRAN でのプログラミング 221
FORTRAN でのプログラミングに関する考慮事項 221
FORTRAN における言語制約事項 221
FORTRAN での参照による呼び出し. 221
FORTRAN でのデバッグとコメント行 222
FORTRAN でのプリコンパイルに関する考慮事項 222
FORTRAN におけるマルチスレッド・データベース・アクセス. 222
FORTRAN の入力および出力ファイル 222
組み込みファイル. 222
FORTRAN の組み込みファイル 222
FORTRAN アプリケーションの組み込みファイル 225
FORTRAN における組み込み SQL ステートメント 226
FORTRAN のホスト変数 227
FORTRAN のホスト変数 227
FORTRAN におけるホスト変数の名前 228
FORTRAN におけるホスト変数宣言. 228
FORTRAN における数値ホスト変数の構文 . . 229
FORTRAN における文字ホスト変数の構文 . . 229
FORTRAN の標識変数 231
FORTRAN におけるラージ・オブジェクト (LOB) ホスト変数の構文 231
FORTRAN におけるラージ・オブジェクト (LOB) ロケーター・ホスト変数の構文 232
FORTRAN におけるファイル参照ホスト変数の構文 232
FORTRAN 用のホスト変数がある SQL 宣言セクション 233
FORTRAN でサポートされている SQL データ・タイプ 233
FORTRAN でのマルチバイト文字セットに関する考慮事項. 235
FORTRAN での日本語または中国語 (繁体字) EUC、および UCS-2 に関する考慮事項 235
FORTRAN の SQLSTATE および SQLCODE 変数 235

第 3 部 ADO.NET、OLE DB、および ODBC. 237

第 10 章 DB2 .NET Data Provider 239
DB2 .NET Data Provider の概要 239
DB2 .NET Data Provider のシステム要件 239
DB2 .NET Data Provider を使用するアプリケーションのプログラミング 240
DB2 .NET Data Provider を使用したアプリケーションからのデータベースへの接続 240
DB2 .NET Data Provider を使用したアプリケーションからの SQL ステートメントの実行. . . 240
DB2 .NET Data Provider を使用したアプリケーションからの結果セットの読み取り 242
DB2 .NET Data Provider を使用したアプリケーションからのストアード・プロシージャーの呼び出し 242
DB2 .NET Data Provider でサポートされている SQL データ・タイプ 244

第 11 章 IBM OLE DB Provider for DB2 247
IBM OLE DB Provider for DB2 の目的. 247
IBM OLE DB Provider for DB2 でサポートされているアプリケーション・タイプ 248
OLE DB サービス. 249
IBM OLE DB Provider でサポートされているスレッド・モデル. 249
IBM OLE DB Provider によるラージ・オブジェクトの操作 249
IBM OLE DB Provider でサポートされているスキーマ行セット. 249
IBM OLE DB Provider で自動的に使用可能になる OLE DB サービス 251
データ・サービス 251
IBM OLE DB Provider でサポートされているカーソル・モード. 252
DB2 と OLE DB の間のデータ・タイプ・マッピング. 252
OLE DB タイプから DB2 タイプにデータを設定するためのデータ変換. 253
DB2 タイプから OLE DB タイプにデータを設定するためのデータ変換. 255
IBM OLE DB Provider の制限. 256
IBM OLE DB Provider での OLE DB コンポーネントおよびインターフェースのサポート 257
IBM OLE DB Provider での OLE DB プロパティーのサポート 259
IBM OLE DB Provider によるデータ・ソースへの接続 262
ADO アプリケーション 263
ADO 接続ストリング・キーワード 263

Visual Basic ADO アプリケーションによるデータ・ソースへの接続 264
ADO アプリケーションにおける更新可能な両方向スクロール・カーソル. 264
ADO アプリケーションに関する制限 264
IBM OLE DB Provider での ADO メソッドおよびプロパティーのサポート 264
C および C++ アプリケーション. 268
C/C++ アプリケーションのコンパイルおよびリンクと IBM OLE DB Provider. 269
IBM OLE DB Provider による、C/C++ アプリケーションでのデータ・ソースへの接続 269
MTS および COM+ 分散トランザクション . . . 269
MTS および COM+ 分散トランザクションのサポートと IBM OLE DB Provider 269
DB2 Universal Database における、C/C++ アプリケーション用の MTS サポートの使用可能化 . 270

第 12 章 OLE DB .NET Data Provider 271

| OLE DB .NET Data Provider 271
| OLE DB .NET Data Provider の制約事項 272
| OLE DB .NET Data Provider アプリケーションでの接続プーリング 276
| OLE DB .NET Data Provider アプリケーションの時刻列. 277
| OLE DB .NET Data Provider アプリケーションの ADORecordset オブジェクト 278

第 13 章 ODBC .NET Data Provider 279

| ODBC .NET Data Provider 279
| ODBC .NET Data Provider の制約事項 279

第 4 部 Java. 287

第 14 章 Java アプリケーション・サポートの概要. 289

第 15 章 JDBC アプリケーション・プログラミング. 293

JDBC アプリケーション・プログラミングの基本的な概念. 293
JDBC アプリケーション作成時の基本的なステップ 293
JDBC サポート用の Java パッケージ 296
JDBC アプリケーションでの変数. 296
JDBC アプリケーションによるデータ・ソースへの接続の方法 297
DB2 アプリケーションによる DriverManager インターフェースと DB2 JDBC Type 2 ドライバーを使用したデータ・ソースへの接続 298
DriverManager インターフェースと DB2 Universal JDBC ドライバーを使用したデータ・ソースへの接続. 300
DataSource インターフェースを使用したデータ・ソースへの接続 303
| JDBC トランザクションの分離レベルの設定 . . 305
JDBC 接続オブジェクト. 306
JDBC トランザクションのコミットまたはロールバック. 306
JDBC データ・ソースへの接続のクローズ. . . 307
SQL を実行するための JDBC インターフェース 307
DB2 オブジェクトを作成および変更するための Statement.executeUpdate メソッドの使用. . . . 308
DB2 表からデータを検索するための Statement.executeQuery メソッドの使用 309
DB2 表のデータを更新するための PreparedStatement.executeUpdate メソッドの使用 . 310
DB2 からデータを検索するための PreparedStatement.executeQuery メソッドの使用 . 312
CallableStatement メソッドを使用したストアード・プロシージャーの呼び出し 313
DB2 Universal JDBC ドライバー使用時の SQLException の処理 315
DB2 JDBC Type 2 ドライバー使用時の SQLException の処理 318
DB2 Universal JDBC ドライバー使用時の SQLWarning の処理 319
DB2 JDBC Type 2 ドライバーの使用時の SQLWarning の処理 320
JDBC アプリケーション・プログラミングの高度な概念 321
DB2 Universal JDBC ドライバーがある JDBC アプリケーションでの LOB 321
| JDBC アプリケーションでの LOB 列データの検索または更新のための Java データ・タイプ . . 323
DB2 Universal JDBC ドライバーを使用した JDBC での ROWID 325
JDBC アプリケーションでの特殊タイプ . . . 326
| JDBC アプリケーションでのセーブポイント . . 327
| JDBC アプリケーションでの識別列の値の検索 328
JDBC アプリケーションでのストアード・プロシージャーからの複数の結果セットの検索 . . . 330
ResultSetMetaData を使用した ResultSet の情報入手 333
DatabaseMetaData を使用したデータ・ソースの情報入手 334
| PreparedStatement のパラメーターの情報入手のための ParameterMetaData の使用. 336
JDBC アプリケーションでのバッチ更新の作成 337
BatchUpdateException からの情報の取り出し . . 339
DB2 Universal JDBC ドライバー使用時の JDBC ResultSet の特性 341
JDBC アプリケーションでの ResultSet の更新可能性、スクロール可能性、および保持可能性の指定 342
DataSource オブジェクトの作成およびデプロイ 345
DB2 Universal JDBC ドライバーのクライアント転送サポート 347

DB2 Universal JDBC ドライバーを使用した DB2 サーバーへの拡張クライアント情報の提供 . 348

第 16 章 SQLJ アプリケーション・プログラミング 351

基本的な SQLJ アプリケーション・プログラミングの概念. 351
SQLJ アプリケーション作成の基本ステップ . . 351
SQLJ サポート用の Java パッケージ 354
SQLJ アプリケーションでの変数 355
SQLJ アプリケーションでのコメント 356
SQLJ を使用したデータ・ソースへの接続 . . . 357
SQLJ トランザクションの分離レベルの設定 . . 362
SQLJ トランザクションのコミットまたはロールバック. 363
SQLJ アプリケーションにおけるセーブポイント 363
SQLJ アプリケーションでのデータ・ソースへの接続のクローズ. 364
SQLJ アプリケーションにおける SQL ステートメント. 365
SQLJ アプリケーションでの DB2 オブジェクトの作成および変更 366
SQLJ アプリケーションによる DB2 表からのデータの検索方法. 366
SQLJ アプリケーションでの名前指定イテレーターの使用 367
SQLJ アプリケーションでの位置指定イテレーターの使用 370
SQLJ アプリケーションでの位置指定 UPDATE および DELETE 操作の実行 373
SQLJ アプリケーションにおける同じ SQL ステートメントの複数のオープン・イテレーター . . 377
SQLJ アプリケーションにおけるイテレーターの複数のオープン・インスタンス 378
SQLJ アプリケーションでのストアード・プロシージャーの呼び出し 379
SQLJ アプリケーションでの SQL エラーの処理 380
SQLJ アプリケーションでの SQL 警告の処理 380
高度な SQLJ アプリケーション・プログラミングの概念 381
同じアプリケーションでの SQLJ と JDBC の使用 381
DB2 Universal JDBC ドライバーを使用した SQLJ アプリケーションでの LOB 384
SQLJ アプリケーションでの LOB 列データの検索および更新のための Java データ・タイプ . . 385
DB2 Universal JDBC ドライバーを使用した SQLJ での ROWID 387
SQLJ アプリケーションにおける特殊タイプ . . 389
SQLJ での SQL ステートメントの実行の制御 390
SQLJ アプリケーションでのストアード・プロシージャーの複数の結果セットの検索 391
SQLJ アプリケーションでのバッチ更新の実行 392
SQLJ アプリケーションでの、位置指定 UPDATE または DELETE 操作の変数としてのイテレーターの受け渡し. 396
SQLJ アプリケーションでのスクロール可能イテレーターの使用. 399

第 17 章 JDBC および SQLJ リファレンス 403

Java、JDBC、および SQL のデータ・タイプ . . . 403
DB2 Universal JDBC ドライバーのプロパティー 408
JDBC API 用ドライバー・サポートの比較. . . . 416
SQLJ ステートメントのリファレンス 437
SQLJ 文節 437
SQLJ ホスト式 437
SQLJ インプリメント文節 438
SQLJ with 文節. 439
SQLJ 接続宣言文節 441
SQLJ イテレーター宣言文節 442
SQLJ 実行可能文節 443
SQLJ コンテキスト文節 444
SQLJ ステートメント文節 445
SQLJ SET-TRANSACTION 文節 446
SQLJ 代入文節 447
SQLJ イテレーター変換文節 448
選択済み sqlj.runtime クラスおよびインターフェース 449
DB2 Universal JDBC ドライバーの参照情報 . . . 457
JDBC に対する DB2 Universal JDBC ドライバーの拡張のサマリー 457
DB2 Universal JDBC ドライバーと他の DB2 JDBC ドライバーの JDBC との相違点 470
DB2 Universal JDBC ドライバーと他の DB2 JDBC ドライバーとの間の SQLJ の相違 . . . 477
DB2 Universal JDBC ドライバーが発行するエラー・コード 479
DB2 Universal JDBC ドライバーが発行する SQLSTATE 479

第 18 章 JDBC ドライバーのインストール 481

DB2 Universal JDBC ドライバーのインストール 481

第 19 章 JDBC および SQLJ セキュリティー 487

DB2 JDBC Type 2 ドライバー使用時のセキュリティー 487
DB2 Universal JDBC ドライバー使用時のセキュリティー. 488
DB2 Universal JDBC ドライバー使用時のユーザー ID およびパスワード・セキュリティーの使用 . . 489
DB2 Universal JDBC ドライバー使用時のユーザー ID 専用セキュリティーの使用. 491
DB2 Universal JDBC ドライバー使用時の暗号化されたユーザー ID セキュリティーまたは暗号化されたパスワード・セキュリティー 492
DB2 Universal JDBC ドライバー使用時の Kerberos セキュリティーの使用 493

第 20 章 JDBC および SQLJ 問題の診断 497

DB2 Universal JDBC ドライバーでの JDBC および SQLJ 問題の診断 497
DB2 Universal JDBC ドライバーでの JDBC および SQLJ 問題の診断 497
DB2 Universal JDBC ドライバー使用時のトレースの例. 500
DB2 JDBC Type 2 ドライバーでの JDBC および SQLJ 問題の診断 504
CLI/ODBC/JDBC トレース機能 504
CLI および JDBC トレース・ファイル. . . . 511

第 21 章 Java 2 Platform Enterprise Edition 521

Java 2 Platform Enterprise Edition (J2EE) の概要 521
Java 2 Platform Enterprise Edition. 521
Java 2 Platform Enterprise Edition コンテナー. . . 522
Java 2 Platform Enterprise Edition サーバー . . . 523
Java 2 Enterprise Edition のデータベース要件. . . 523
Java Naming and Directory Interface (JNDI) . . . 524
Java トランザクション管理. 524
| JTA メソッドを使用する分散トランザクションの例 525
Enterprise Java Beans 530

第 5 部 他のプログラミング・インターフェース 533

第 22 章 Perl でのプログラミング. . . 535

Perl でのプログラミングに関する考慮事項 . . . 535
Perl の制約事項 535
Perl でのマルチスレッド・データベース・アクセス 535
Perl でのデータベース接続. 536
Perl での取り出し結果 536
Perl のパラメーター・マーカー 537
Perl の SQLSTATE および SQLCODE 変数 . . . 537
Perl プログラムの例 538

第 23 章 REXX でのプログラミング 539

REXX でのプログラミングに関する考慮事項. . . 539
REXX の言語制限. 540
REXX の言語制限. 540
REXX における SQLEXEC、SQLDBS、および SQLDB2 の登録 540
REXX でのマルチスレッド・データベース・アクセス. 541
REXX の日本語または中国語 (繁体字) EUC に関する考慮事項. 541
REXX アプリケーションでの組み込み SQL . . . 541
REXX のホスト変数 543
REXX のホスト変数 543
REXX でのホスト変数名 544
REXX でのホスト変数の参照 544
REXX の標識変数. 544
事前定義された REXX 変数 545
REXX の LOB ホスト変数. 547
REXX における LOB ロケーター宣言の構文 547
REXX における LOB ファイル参照宣言の構文 548
REXX での LOB ホスト変数のクリア 549
REXX でのカーソル 549
REXX でサポートされている SQL データ・タイプ 549
REXX の実行要件. 551
REXX アプリケーションの構築と実行 551
REXX のバインド・ファイル 552
REXX の API 構文 553
REXX からのストアード・プロシージャーの呼び出し 554
REXX でのストアード・プロシージャー . . . 554
REXX におけるストアード・プロシージャーの呼び出し 555
REXX におけるクライアントによるストアード・プロシージャーの呼び出しに関する考慮事項 556
REXX におけるサーバーによるストアード・プロシージャーの呼び出しに関する考慮事項. . . 556
SQLDA 10 進フィールドの精度と SCALE 値の検索 556

第 24 章 DB2 WebSphere MQ 関数を使用したアプリケーションの作成 . . . 559

| WebSphere MQ 機能の概要. 559
| WebSphere MQ メッセージング 561
| WebSphere MQ 関数を使用したメッセージ送信 . . 564
| WebSphere MQ 関数を使用したメッセージ取り出し 566
| WebSphere MQ アプリケーション間の接続 . . . 568
| WebSphere MQ 関数を使用した要求/応答通信 . . 569
| WebSphere MQ 関数を使用したパブリッシュ/サブスクライブ 571

第 25 章 WebSphere 575

企業データへの接続 575
WebSphere の接続プールとデータ・ソース . . . 575
WebSphere 接続プールの利点 576
WebSphere でのステートメント・キャッシング . . 577

第 6 部 セキュリティー・プラグイン 579

第 26 章 セキュリティー・プラグイン 581

| セキュリティー・プラグイン 581
| セキュリティー・プラグイン・ライブラリーの位置 585
| セキュリティー・プラグインの命名規則 586
| セキュリティー・プラグインの 2 パーツのユーザー ID のサポート. 587
| セキュリティー・プラグインの 32 ビットと 64 ビットに関する考慮事項 590
| セキュリティー・プラグインの問題判別 590
| グループ検索プラグインの展開 592
| ユーザー ID/パスワード・プラグインのデプロイ 593
| GSS-API プラグインのデプロイ 595
| Kerberos プラグインのデプロイ 597

第 27 章 セキュリティー・プラグインの作成 599
DB2 によるセキュリティー・プラグインのロード方法 599
セキュリティー・プラグイン・ライブラリーに関する制約事項 601
セキュリティー・プラグインの戻りコード. 603
セキュリティー・プラグインのエラー・メッセージ 605
セキュリティー・プラグイン API の呼び出し順序 606

第 28 章 セキュリティー・プラグイン API. 611
セキュリティー・プラグイン API 611
グループ・プラグイン API 612
グループ検索プラグイン用の API 612
db2secGroupPluginInit - グループ・プラグインの初期化. 615
db2secPluginTerm - グループ・プラグイン・リソースのクリーンアップ 616
db2secGetGroupsForUser - ユーザーのグループのリストの取得 616
db2secDoesGroupExist - グループの存在のチェック 620
db2secFreeGroupListMemory - グループ・リストのメモリーの解放 621
db2secFreeErrormsg - エラー・メッセージのメモリーの解放 622
ユーザー認証プラグイン API 622
ユーザー ID/パスワード認証プラグインの API 622
db2secClientAuthPluginInit - クライアント認証プラグインの初期化 629
db2secClientAuthPluginTerm - クライアント認証プラグイン・リソースのクリーンアップ . . . 630
db2secRemapUserid - ユーザー ID およびパスワードの再マップ. 631
db2secGetDefaultLoginContext - デフォルト・ログイン・コンテキストの取得 633
db2secGenerateInitialCred - 初期証明書の生成 635
db2secValidatePassword - パスワードの検証 . . 637
db2secProcessServerPrincipalName - サーバーから戻されたサービス・プリンシパル名の処理. . . 639
db2secFreeToken - トークンが保持しているメモリーの解放 640
db2secFreeInitInfo - db2secGenerateInitialCred() が保持しているリソースのクリーンアップ . . . 641
db2secServerAuthPluginInit - サーバー認証プラグインの初期化 641
db2secServerAuthPluginTerm - サーバー認証プラグイン・リソースのクリーンアップ 643
db2secGetAuthIDs - 認証 ID の取得 644
db2secDoesAuthIDExist - 認証 ID の存在の検査 646
GSS-API プラグイン API 647
GSS-API 認証プラグインに必要な API および定義 647
GSS-API 認証プラグインに関する制約事項 . . 648
セキュリティー・プラグイン API のバージョン管理 649

第 7 部 一般的な DB2 アプリケーションの概念 651

第 29 章 各国語サポート. 653
照合順序の概説. 653
照合順序 653
照合順序に基づく文字比較 655
TRANSLATE 関数による、大文字小文字に無関係な文字比較 656
EBCDIC および ASCII 照合順序におけるソート順序の相違 657
データベースの作成時に指定される照合順序 . . 658
照合順序のサンプル 660
コード・ページとロケール 661
コード・ページ値の導出. 661
アプリケーション・プログラム中のロケールの導出 661
DB2 によるロケールの導出方法 662
アプリケーションに関する考慮事項 662
各国語サポートとアプリケーション開発に関する考慮事項 662
各国語サポートと SQL ステートメント . . . 664
リモート・ルーチン 665
混合コード・ページ環境でのパッケージ名の考慮事項 665
プリコンパイルおよびバインド用のアクティブ・コード・ページ. 666
アプリケーション実行用のアクティブ・コード・ページ. 666
異なるコード・ページ間での文字変換 666
コード・ページ変換はいつ行われるか 667
コード・ページ変換時の文字置換. 668
サポートされるコード・ページ変換 668
コード・ページ変換の拡張係数 669
DBCS 文字セット 670
拡張 UNIX コード (EUC) 文字セット 671
DBCS 環境での CLI、ODBC、JDBC、および SQLJ プログラム 672
日本語および中国語 (繁体字) EUC および UCS-2 コード・セットに関する考慮事項. 673
日本語および中国語 (繁体字) EUC および UCS-2 コード・セットに関する考慮事項 . . . 673
EUC および 2 バイトが混在するクライアントおよびデータベースに関する考慮事項 674
中国語 (繁体字) のユーザーへの文字変換に関する考慮事項 675
日本語または中国語 (繁体字) EUC アプリケーションにおけるグラフィック・データ 675
コード・ページが異なる状況におけるアプリケーション開発 677
混合コード・セット環境におけるクライアント・ベースのパラメーターの検証 680

混合コード・セット環境における DESCRIBE ステートメント 681
混合コード・セット環境における固定長および可変長データ 683
混合コード・セット環境におけるコード・ページ変換によるストリング長のオーバーフロー. . . 683
Unicode データベースに接続されるアプリケーション 685

第 30 章 トランザクションの管理 . . . 687
リモート作業単位 687
マルチサイト更新に関する考慮事項 687
マルチサイト更新 688
マルチサイト更新をいつ使用するか 688
マルチサイト更新アプリケーションでの SQL ステートメント 689
マルチサイト更新アプリケーションのプリコンパイル 691
マルチサイト更新アプリケーションの構成パラメーターに関する考慮事項. 692
ホスト、AS/400、または iSeries サーバーへのアクセス 694
並行トランザクション 694
並行トランザクション 694
並行トランザクションを使用する際に気を付けるべき問題 695
並行トランザクションのデッドロックの回避 . . 696
セーブポイントとトランザクション 697
セーブポイントによるトランザクションの管理 697
アプリケーションのセーブポイントとコンパウンド SQL ブロックの比較. 699
セーブポイントの作成や制御に使用する SQL ステートメント 701
セーブポイントの使用に関する制約事項 . . . 702
セーブポイントおよびデータ定義言語 (DDL) 702
| セーブポイントのネスト. 704
セーブポイントおよびバッファー化挿入 . . . 704
セーブポイントとカーソル・ブロック化 . . . 704
セーブポイントおよび XA に準拠したトランザクション・マネージャー. 705
X/Open XA インターフェース・プログラミングに関する考慮事項. 705
アプリケーション・リンケージと X/Open XA インターフェース 709
MTS および COM+ トランザクション管理 . . . 709
| トランザクション・マネージャーとしての
| Microsoft Transaction Server (MTS) および
| Microsoft Component Services (COM+) 709
| Microsoft Component Services (COM+) での疎結
| 合サポート 711
| Microsoft Transaction Server (MTS) および
| Microsoft Component Services (COM+) のトラン
| ザクション・タイムアウト 712
| Microsoft Transaction Server (MTS) および
| Microsoft Component Services (COM+) を使用し
| た ODBC および ADO 接続プール 713

第 31 章 パーティション・データベース環境におけるプログラミング上の考慮事項 715
パーティション・データベース環境における FOR READ ONLY カーソル 715
指示 DSS およびローカル・バイパス 715
パーティション・データベース環境における指示 DSS およびローカル・バイパス 715
パーティション・データベース環境における指示 DSS 716
パーティション・データベース環境におけるローカル・バイパス. 717
バッファー化挿入 717
パーティション・データベース環境におけるバッファー化挿入 717
バッファー化挿入の使用に関する考慮事項. . . 720
バッファー化挿入の使用に関する制限 722
パーティション・データベース環境における大量データの抽出の例. 723
シミュレートされたパーティション・データベース環境の作成 728
トラブルシューティング. 728
パーティション・データベース環境におけるエラー処理. 728
パーティション・データベース環境における重大エラー. 729
マージされた複数の SQLCA 構造 730
エラーを戻すパーティション 730
ループ状態または延期状態のアプリケーション 731

第 32 章 一般的な DB2 アプリケーションの技法. 733
Windows Local System Account からのアプリケーションの実行 733
生成列. 733
ID 列 734
| SQL データ変更ステートメントからの結果セットの検索. 735
| 中間結果表 736
| ターゲット表とビュー 737
| INPUT SEQUENCE をベースにした結果セットのソート 737
| カーソルを使用した SQL データ変更ステートメントからの結果セットの検索 738
| INCLUDE 列 739
| INSERT 操作における INCLUDE 列. 739
| UPDATE および DELETE 操作における
| INCLUDE 列 740
全選択に対する検索
UPDATE、INSERT、DELETE、および MERGE 操作 740
順次値とシーケンス・オブジェクト 741
順次値の生成 741
シーケンスの振る舞いの管理 742

アプリケーションのパフォーマンスとシーケンス・オブジェクト 743
シーケンス・オブジェクトと ID 列の比較. . . 744
宣言済み一時表とアプリケーションのパフォーマンス 744
ネットワークを通じた大量データの伝送 746

第 8 部 付録 749

付録 A. サポートされる SQL ステートメント 751

付録 B. セキュリティー・プラグインのデプロイメントに関する制限 755

付録 C. ホストまたは iSeries 環境でのプログラミング 757
ホストまたは iSeries 環境におけるアプリケーション 757
ホストまたは iSeries 環境におけるデータ定義言語 758
ホストまたは iSeries 環境におけるデータ操作言語 759
ホストまたは iSeries 環境におけるデータ制御言語 760
DB2 Connect によるデータベース接続の管理. . . 760
割り込み要求の処理 761
パッケージ属性、PREP、および BIND 761
IBM のリレーショナル・データベース・システム間でのパッケージ属性の相違点. 761
C ヌル終了ストリング用の CNULREQD BIND オプション 762
スタンドアロン SQLCODE および SQLSTATE 変数 762
DB2 Connect でサポートされている分離レベル 763
ユーザー定義のソート順序 764
IBM のリレーショナル・データベース・システム間での参照保全の相違点. 764
ロックとアプリケーションの移植性 764
IBM のリレーショナル・データベース・システム間での SQLCODE と SQLSTATE の相違点 . . . 765
IBM のリレーショナル・データベース・システム間でのシステム・カタログの相違点 765
検索割り当て時の数値変換のオーバーフロー . . . 765
ホストまたは iSeries 環境におけるストアード・プロシージャー 766
DB2 Connect によるコンパウンド SQL のサポート 767
DB2 Connect によるマルチサイト更新 768
DB2 Connect がサポートするホストおよび iSeries サーバー SQL ステートメント 769
DB2 Connect が拒否するホストおよび iSeries サーバー SQL ステートメント 769

付録 D. EBCDIC バイナリー照合のシミュレート. 771

付録 E. DB2 Universal Database 技術情報 775
DB2 資料とヘルプ. 775
DB2 資料の更新 775
DB2 インフォメーション・センター. 776
DB2 インフォメーション・センターのインストール・シナリオ 778
DB2 セットアップ・ウィザードを使用した DB2 インフォメーション・センターのインストール (UNIX) 780
DB2 セットアップ・ウィザードを使用した DB2 インフォメーション・センターのインストール (Windows) 783
DB2 インフォメーション・センターの呼び出し 786
コンピューターまたはイントラネット・サーバーへの DB2 インフォメーション・センターの更新インストール 787
DB2 インフォメーション・センターにおける特定の言語でのトピックの表示 788
DB2 PDF 資料および印刷された資料 789
DB2 の基本情報 789
管理情報 790
アプリケーション開発情報 791
ビジネス・インテリジェンス情報. 792
DB2 Connect 情報. 792
入門情報 792
チュートリアル情報 793
オプショナル・コンポーネント情報 793
リリース・ノート 794
PDF ファイルからの DB2 資料の印刷方法 . . . 795
DB2 の印刷資料の注文方法. 796
DB2 ツールからコンテキスト・ヘルプを呼び出す 797
コマンド行プロセッサーからメッセージ・ヘルプを呼び出す 798
コマンド行プロセッサーからコマンド・ヘルプを呼び出す. 798
コマンド行プロセッサーから SQL 状態ヘルプを呼び出す. 799
DB2 チュートリアル 799
DB2 トラブルシューティング情報 800
アクセス支援 801
キーボードによる入力およびナビゲーション . . 801
アクセスしやすい表示 802
支援テクノロジーとの互換性 802
アクセスしやすい資料 802
ドット 10 進シンタックス・ダイアグラム. . . . 803
DB2 Universal Database 製品の共通基準認証 . . . 805

付録 F. 特記事項 807
商標 809

索引 811

IBM と連絡をとる. 833
製品情報 833

本書について

「*アプリケーション開発ガイド*」は、DB2 アプリケーションのコーディング、デバッグ、ビルド、および実行に関して知っておくべきことを説明した 3 冊からなるガイドです。

- 「*アプリケーション開発ガイド クライアント・アプリケーションのプログラミング*」では、DB2 クライアントで実行されるスタンドアロン DB2 クライアントをコーディングする際に知っておくべきことを説明しています。以下の情報を扱います。
  - DB2 でサポートされているプログラミング・インターフェース。DB2 Developer's Edition、サポートされているプログラミング・インターフェース、Web アプリケーションを作成するための機能、および DB2 が提供するルーチンやトリガーなどのプログラミング機能についてハイレベルな説明がなされています。
  - DB2 アプリケーションが従うべき一般的な構造。データベース中のデータ値やリレーションシップの推奨される保守方法や、許可に関する考慮事項が説明されており、アプリケーションのテストとデバッグ方法に関する情報もあります。
  - 動的組み込み SQL と静的組み込み SQL。組み込み SQL に関する一般的な考慮事項、および DB2 アプリケーションで静的 SQL と動的 SQL を使用する際の特有な考慮事項。
  - C/C++、COBOL、Perl、および REXX などのサポートされているホストおよびインタープリター言語とこれらの言語で書かれたアプリケーションでの組み込み SQL の使用方法。
  - DB2 .NET Data Provider、および OLE DB .NET および ODBC .NET Data Provider。
  - Java (JDBC と SQLJ) および WebSphere Application Server で使用する Java アプリケーションを構築する際の考慮事項。
  - IBM OLE DB Provider for DB2 Server。 IBM OLE DB Provider による OLE DB サービス、コンポーネント、およびプロパティーのサポートに関する一般情報。 ActiveX Data Objects (ADO) 用の OLE DB インターフェースを使用する Visual Basic および Visual C++ アプリケーション特有の情報があります。
  - 各国語サポートの問題。照合シーケンス、コード・ページとロケールから派生する問題、および文字変換などの一般トピックが説明されています。 DBCS コード・ページ、EUC 文字セット、および日本語と中国語 (繁体字) EUC および UCS-2 環境に適用される問題についても説明されます。
  - トランザクション管理。マルチサイト更新を実行するアプリケーション、および並行トランザクションを実行するアプリケーションに適用される問題が説明されています。
  - パーティション・データベース環境におけるアプリケーション。パーティション・データベース環境における指示 DSS、ローカル・バイパス、バッファー挿入、アプリケーションのトラブルシューティングについて説明します。

  - 一般的に使用されるアプリケーション技法。生成列と ID 列、宣言済み一時表の使用方法、およびトランザクションを管理するためのセーブポイントの使用法について説明されます。
  - 組み込み SQL アプリケーションの使用がサポートされている SQL ステートメント。
  - ホストおよび iSeries 環境にアクセスするアプリケーション。ホストおよび iSeries 環境にアクセスする組み込み SQL アプリケーションに関する問題。
  - EBCDIC バイナリー照合のシミュレーション。
- 「*アプリケーション開発ガイド サーバー・アプリケーションのプログラミング*」では、ルーチン、ラージ・オブジェクト、ユーザー定義タイプ、およびトリガーを含む、サーバー・サイド・オブジェクトを使用したプログラミングを行う上で知っておくべきことが説明されています。以下の情報を扱います。
  - ルーチン (ストアード・プロシージャー、ユーザー定義関数、およびメソッド)。以下のことも扱われています。
    - ルーチンのパフォーマンス、セキュリティー、ライブラリー管理上の考慮事項、および制限。
    - 外部ルーチン、および CREATE ステートメントを含む、ルーチンの作成。
    - プロシージャーのパラメーター・モードおよびパラメーターの取り扱い。
    - プロシージャーの結果セット。
    - デバッグおよび条件処理を含む SQL プロシージャー。
    - ユーザー定義のスカラー関数および表関数。
    - ユーザー定義のスカラーおよび表関数呼び出し (FIRST 呼び出し、FINAL 呼び出しなど) やスクラッチパッド。
    - メソッド。
    - 許可、および外部ルーチンのバインディング。
    - C、Java、.NET 共通言語ランタイム、および OLE オートメーション・ルーチンの言語特有の考慮事項。
    - ルーチンの呼び出し。
    - 関数選択。
    - 特殊タイプと LOB の関数への渡し方。
    - コード・ページとルーチン。
  - LOB の使用法とロケーター、参照変数、および CLOB データを含むラージ・オブジェクト。
  - ユーザー定義特殊タイプ (UDT) (以下のことの説明も含まれます)。強い型定義、UDT の定義とドロップ、構造型による表の作成、特定のアプリケーション用の特殊タイプと型付き表の使用、複数の特殊タイプの取り扱いとそれらの間のキャスト、特殊タイプ間の比較と代入、特殊タイプ列における UNION 操作。
  - ユーザー定義構造型 (以下のことも説明されています)。インスタンスの保管インスタンス生成、構造型の階層、構造型の動作の定義、メソッドの動的ディスパッチング、比較関数、cast 関数、コンストラクター関数、および構造型用の mutator メソッドと observer メソッド。

- 型付き表 (以下のことも説明されています)。オブジェクトの作成、ドロップ、置換、保管、システム生成オブジェクト ID の定義、およびオブジェクト ID 列における制約。
- 参照タイプ (以下のことも説明されています)。型付き表のオブジェクト間のリレーションシップ、参照のあるセマンティック・リレーションシップ、および参照保全と有効範囲参照。
- 型付き表と型付きビュー (以下のことも説明されています)。列型としての構造型、トランスフォーム関数とトランスフォーム・グループ、ホスト言語プログラムのマッピング、および構造型ホスト変数。
- トリガー (以下のことも説明されています)。 INSERT/UPDATE/DELETE トリガー、参照制約との相互作用、作成に関するガイドライン、細分性、活動化時間、遷移変数と表、トリガー・アクション、多重トリガー、および複数のトリガーと制約とルーチン間の協同。

• 「*アプリケーション開発ガイド アプリケーションの構築および実行*」では、DB2 でサポートされている以下のオペレーティング・システムにおいて DB2 アプリケーションをビルドして実行する上で知っておくべきことが説明されています。

- AIX
- HP-UX
- Linux
- Solaris
- Windows

以下の情報を扱います。

- DB2 がサポートするコンパイラーとインタープリターを含め、アプリケーションを作成するためにサポートされているサーバーとソフトウェア。
- DB2 サンプル・プログラム・ファイル、makefile、ビルド・ファイル、およびエラー・チェック・ユーティリティー・ファイル。
- Java および WebSphere MQ 関数用の特定の命令を含むアプリケーション開発環境のセットアップ方法。
- サンプル・データベースのセットアップ方法。
- 前のバージョンの DB2 からのアプリケーションの移行方法。
- Java アプレット、アプリケーション、およびルーチンのビルドと実行の仕方。
- SQL プロシージャーのビルドと実行の仕方。
- C/C++ アプリケーションとルーチンのビルドと実行の仕方。
- IBM COBOL および Micro Focus COBOL アプリケーションとルーチンのビルドと実行の仕方。
- AIX および Windows における REXX アプリケーションのビルドと実行の仕方。
- Windows での C# および Visual Basic .NET アプリケーションと、CLR .NET ルーチンの作成および実行方法。
- Windows における Visual Basic および Visual C++ を使用した ActiveX Data Object (ADO) のあるアプリケーションのビルドと実行の仕方。
- Windows における Visual C++ を使用した リモート・データ・オブジェクトのあるアプリケーションのビルドと実行の仕方。

第 1 部 入門

第 1 章 サポートされているプログラミング・インターフェースの概説

アプリケーション開発用の DB2 Universal Database ツール 3
DB2 Development Add-In の概要. 4
サポートされているプログラミング・インターフェース 6
DB2 がサポートしているプログラミング・インターフェース 6
DB2 アプリケーション・プログラミング・インターフェース 8
組み込み SQL 8
DB2 コール・レベル・インターフェース . . . 10
DB2 CLI と組み込み動的 SQL の比較 11
Java Database Connectivity (JDBC) 13
組み込み SQL for Java (SQLJ) 14
ADO (ActiveX Data Object) および RDO (Remote Data Object) 15
Perl DBI 16
ODBC エンド・ユーザー・ツール 16
DB2 .NET Data Provider 16
Web アプリケーション 17
Web アプリケーションの構築のためのツール . . 17
WebSphere Studio 17
XML Extender 18
MQSeries 使用可能性 19
Net.Data. 19
プログラミング機能 20
DB2 プログラミング機能. 20
DB2 ストアード・プロシージャー. 21
DB2 ユーザー定義関数およびメソッド 22
Development Center 22
ユーザー定義タイプ (UDT) およびラージ・オブジェクト (LOB) 24
OLE オートメーション・ルーチン. 25
OLE DB 表関数 25
DB2 トリガー 26

アプリケーション開発用の DB2 Universal Database ツール

アプリケーションを開発する際に、さまざまなツールを使用できます。 DB2® Universal Database は、アプリケーションでの SQL ステートメントの作成とテスト、およびパフォーマンスのモニターを行うのに役立つ、以下のツールを提供しています。

注: すべてのツールがすべてのプラットフォーム上で使用可能であるとは限りません。

コントロール・センター

データベース・オブジェクト (データベース、表、およびパッケージなど) とそれらのオブジェクト間のリレーションシップを表示するグラフィカル・インターフェース。システムの構成、ディレクトリーの管理、システムのバックアップとリカバリー、ジョブのスケジューリングおよびメディアの管理などの管理用タスクを実行するには、コントロール・センターを使用してください。

DB2 には、以下の機能が組み込まれています。

コマンド・エディター
: 対話式ウィンドウに DB2 コマンドや SQL ステートメントを入力したり、結果ウィンドウに実行結果を表示するのに使用します。 結果をスクロールさせたり、出力をファイルに保管することができます。

タスク・センター
: スクリプトを作成するのに使用します。スクリプトは保管して後に起動することができます。 スクリプトには、DB2 コマンド、SQL ステートメン

ト、またはオペレーティング・システム・コマンドを入れることができます。スクリプトが自動実行されるようスケジュールすることもできます。これらのジョブは、一度だけ実行するか、繰り返しスケジュールによって実行されるよう設定することができます。繰り返しスケジュールは、バックアップなどのタスクの場合に特に役立ちます。タスク・センターは、潜在的な問題を早期に警告するようにシステムをモニターしたり、問題を訂正するための処置を自動化するときにも使用できます。

ジャーナル
次のタイプの情報を表示するのに使用します。実行がペンディングになっているか、実行中であるか、実行を完了したジョブに関するすべての入手可能な情報、リカバリー履歴ログ、アラート・ログ、およびメッセージ・ログです。 また、ジャーナルを使用して、自動実行されたジョブの結果を検討することもできます。

ツール設定
タスク・センターの設定を変更するのに使用します。

イベント・モニター
一定期間におけるデータベースのアクティビティーのパフォーマンス情報を収集します。収集される情報は特定のデータベース・イベント、たとえば、データベース接続や SQLステートメントなどに関するアクティビティーの十分なサマリーになります。

Visual Explain

コントロール・センターのインストール可能オプション。 Visual Explain は、SQL ステートメントのオプティマイザーが選択したアクセス・プランの表示を含め、SQL ステートメントの分析と調整を行うことができるグラフィカル・インターフェースです。

DB2 Development Add-In の概要

IBM® DB2® Development Add-In は、 DB2 サーバーとの連携作業と DB2 ルーチンの開発のために、 Microsoft® Visual Studio .NET 開発環境にシームレスに統合される機能の集まりです。この Add-In を使って、以下を行うことができます。

- 各種の DB2 開発ツールや管理ツールの起動
- Solution Explorer での DB2 プロジェクトの作成および管理
- IBM Explorer での DB2 データ接続へのアクセスとその管理
- ストアード・プロシージャーおよびユーザー定義関数 (UDF) を作成するスクリプトなどの DB2 スクリプトの作成および変更

「DB2 ツール (DB2 Tools)」ツールバー:

「DB2 ツール (DB2 Tools)」ツールバーにより、さまざまな DB2 開発および管理ツールを起動できます。「DB2 ツール (DB2 Tools)」ツールバーを使うと、次のような DB2 ツールを起動することができます。

- デベロップメント・センター
- コントロール・センター
- レプリケーション・センター

- コマンド・エディター
- タスク・センター
- ヘルス・センター
- ジャーナル

DB2 プロジェクト・タイプ:

DB2 Development Add-In には新しい IBM プロジェクト・フォルダーが導入されています。それには、 DB2 データベース・サーバー・スクリプトの開発用の DB2 データベース・プロジェクト・タイプが組み込まれています。 DB2 プロジェクトを使って、以下を行うことができます。

- 新規または既存の SQL ストアード・プロシージャー・スクリプトの追加
- 新規または既存の SQL UDF スクリプトの追加
- 汎用テンプレートに基づいた新規または既存のスクリプトの追加
- スクリプトの作成順も含めた構成の構築オプションの指定
- Microsoft Visual Source Safe ソース制御管理システムへのスクリプト・ファイルのチェックイン

IBM Explorer の「データ接続 (Data Connections)」フォルダー:

DB2 Development Add-In では、 Visual Studio .NET Server Explorer に似たドッキング可能ウィンドウである IBM Explorer という名前の新ツールの追加によって、Visual Studio .NET 環境を拡張しています。 IBM Explorer は、 Visual Studio .NET ユーザーが「データ接続 (Data Connections)」フォルダーを使って IBM データベース接続にアクセスする手段になります。 IBM Explorer 内の「データ接続 (Data Connections)」フォルダーは、DB2 で管理されるプロバイダー接続のために特に設計されています。 IBM Explorer の「データ接続 (Data Connections)」フォルダーでは、以下を行うことができます。

- 要求時接続テクノロジーをサポートする複数の名前付き DB2 接続を使った処理
- パフォーマンスとスケーラビリティーの向上のためのデータベース・カタログ・フィルターとローカル・キャッシングの指定
- 表、ビュー、ストアード・プロシージャー、および関数などのサーバー・オブジェクトのプロパティーの表示
- 表およびビューからのデータの取り出し
- ストアード・プロシージャーおよび UDF のテスト実行
- ストアード・プロシージャーと関数のソース・コードの表示
- ドラッグ・アンド・ドロップを使った ADO .NET コードの生成

DB2 SQL Editor:

DB2 Development Add-In はまた、DB2 SQL Editor も備えています。このエディターを使って、DB2 ルーチンおよびスクリプト内のコードを変更および表示することができます。 DB2 SQL Editor には次のような機能が組み込まれています。

- SQL の読み易さの向上のためのテキストのカラー化
- DB2 スクリプトの入力時のインテリジェント・オートコンプリートに対応するための Microsoft Visual Studio .NET IntelliSense 機能との統合

サポートされているプログラミング・インターフェース

以下のセクションでは、サポートされているプログラミング・インターフェースについて概説します。

DB2 がサポートしているプログラミング・インターフェース

DB2® データベースを管理またはアクセスする際に、さまざまなプログラミング・インターフェースを使用することができます。 行うことができる操作は、以下のとおりです。

- データベースのバックアップおよびリストアなどの管理機能を実行する場合は、DB2 API を使用します。
- アプリケーションに静的および動的 SQL を組み込む。
- アプリケーションで DB2 コール・レベル・インターフェース (DB2 CLI) 関数をコーディングして、動的 SQL ステートメントを呼び出す。
- JDBC API (Java Database Connectivity Application Programming Interface) を呼び出す、 Java™ アプリケーションおよびアプレットを開発する。
- DAO (Data Access Object) および RDO (Remote Data Object) 仕様に準拠した Microsoft® Visual Basic と Visual C++ アプリケーション、および OLE DB Bridge を使用する ADO (ActiveX Data Object) アプリケーションを開発する。
- DB2 .NET Data Provider、OLE DB .NET Data Provider、または ODBC .NET Data Provider を使用して、 ADO.NET アプリケーションを開発する。
- IBM® のツール、または Net.Data®、Excel、Perl などの他社のツール、および Lotus® Approach などの Open Database Connectivity (ODBC) エンド・ユーザー・ツールと、そのプログラム言語である Lotus Script を使用して、アプリケーションを開発する。

アプリケーションが DB2 データベースにアクセスする方法は、開発するアプリケーションのタイプによって異なります。たとえば、データ入力アプリケーションの場合、アプリケーションに静的 SQL ステートメントを組み込むことができます。ワールド・ワイド・ウェブ (WWW) を介して照会を実行するアプリケーションの場合、 Net.Data、 Perl、または Java を選択できます。

アプリケーションがデータにアクセスする方法以外に、以下のことを考慮する必要があります。

- 以下の機能を利用したデータ値の制御:
  - データ・タイプ (組み込みまたはユーザー定義)
  - 表チェック制約
  - 参照保全制約
  - CHECK OPTION を使用したビュー
  - アプリケーションのロジックと変数のタイプ
- 以下の機能を利用したデータ値間のリレーションシップの制御:
  - 参照保全制約
  - トリガー
  - アプリケーションのロジック

- 以下の機能を利用した、サーバーでのプログラムの実行:
  - ストアード・プロシージャー
  - ユーザー定義関数
  - トリガー

前述のリストには、トリガーなどのように、複数の項目で使用されている機能があります。これは、複数の設計基準に対応するという、これらの機能の柔軟性を反映しています。

最も重要かつ基礎的な決定は、アプリケーションに関連したデータ規則を適用するためのロジックを、データベースに移動させるかどうかです。

データに着目した、ロジックをアプリケーションからデータベースに移すことの重要な利点は、アプリケーションのデータからの独立性が一層高まるということです。 データに関連するロジックは、1 箇所 (データベース) に集められます。つまり、データまたはデータ・ロジックの変更を一度に行うことができ、またそれがただちにすべてのアプリケーションに反映されます。

この後者の利点は非常に強力なものですが、データベース内に置かれるデータ・ロジックがデータのユーザーすべてに同じ影響を与えるということに気を付けてください。データに加えられる規則や制約が、そのデータの全ユーザーやアプリケーションのユーザーにも当てはまるかどうかを考慮しなければなりません。

アプリケーションの要件は、データ規則をデータベースで適用するか、アプリケーションで適用するかにも影響を与えます。たとえば、特定の順序でデータが入力されたときに妥当性検査エラーを処理しなければならないことがあります。一般に、これらのタイプのデータの妥当性検査は、アプリケーション・コードで行う必要があります。

アプリケーションを使用するコンピューティング環境も考慮してください。クライアント側のマシンでロジックを実行する場合と、ストアード・プロシージャー、UDF、またはその両方を組み合わせて使用して、通常のより強力なデータベース・サーバー側のマシンでロジックを実行する場合の差について考慮する必要があります。

場合によっては、アプリケーション (アプリケーション固有の理由のため) とデータベース (アプリケーション外の他の対話的な用途のため) の両方を考慮に入れて設計しなければならないということもあります。

関連概念:

- 145 ページの『DB2 コール・レベル・インターフェース (CLI) と組み込み動的 SQL の比較』
- 8 ページの『組み込み SQL』
- 10 ページの『DB2 コール・レベル・インターフェース』
- 8 ページの『DB2 アプリケーション・プログラミング・インターフェース』
- 15 ページの『ADO (ActiveX Data Object) および RDO (Remote Data Object)』
- 16 ページの『Perl DBI』
- 16 ページの『ODBC エンド・ユーザー・ツール』

- 17 ページの『Web アプリケーションの構築のためのツール』
- 13 ページの『Java Database Connectivity (JDBC)』

DB2 アプリケーション・プログラミング・インターフェース

アプリケーションでいくつかのデータベース管理作業 (データベースの作成、活動化、バックアップ、またはリストアなど) を実行しなければならない場合があります。 DB2® にはたくさんの API があるため、ご使用のアプリケーション (組み込み SQL および DB2 CLI アプリケーションを含む) から、これらの作業を実行できます。 この API により、コントロール・センターで使用可能な DB2 サーバー管理ツールで実行できる管理機能と同じ機能を、アプリケーションにプログラミングすることが可能になっています。

さらに、 DB2 API を使用してしか実行できない特定の作業を行わなければならない場合があります。たとえば、エラー・メッセージのテキストを取り出し、アプリケーションがエンド・ユーザーに対してそのテキストを表示できるようにしたい場合もあるかもしれません。メッセージを取り出すには、「Get Error Message (エラー・メッセージの入手)」という API を使用しなければなりません。

関連概念:

- 55 ページの『API における許可に関する考慮事項』
- 46 ページの『組み込み SQL または DB2 CLI プログラムにおける管理 API』

組み込み SQL

構造化照会言語 (SQL) は、 DB2® データベース内のデータへのアクセスおよび操作に使用するデータベース・インターフェース言語です。 アプリケーションに SQL ステートメントを組み込めば、そのアプリケーションで、 SQL がサポートしている作業 (データの検索または保管など) を行うことができます。 DB2 を使用して、C/C++、COBOL、FORTRAN、Java™ (SQLJ)、および REXX プログラム言語で、組み込み SQL アプリケーションをコーディングすることができます。

注: REXX および FORTRAN プログラム言語は、DB2 Universal Database バージョン 5 から拡張されていません。

SQL ステートメントの組み込み先のアプリケーションをホスト・プログラムと呼びます。ホスト・プログラムの作成に使用するプログラム言語をホスト言語と呼びます。このようにプログラムおよび言語を定義するのは、それらが SQL ステートメントに対してホストの役割を果たす、つまりそれらの機能を提供しているためです。

静的 SQL ステートメントの場合、コンパイルを行う前に、ステートメント・タイプおよび表名と列名が決まっています。ただし、ステートメントが検索または更新を行う特定のデータ値は決められていません。それらの値は、ホスト言語変数で指定することができます。静的 SQL ステートメントのプリコンパイル、コンパイル、およびバインドは、アプリケーションを実行する前に行います。静的 SQL は、統計が大幅に変更されないデータベースで実行するのに適しています。 それ以外の場合、静的ステートメントは、すぐに最新の情報に則したものではなくなってしまいます。

対照的に、動的 SQL ステートメントとは、ランタイムにアプリケーションが作成し、実行するステートメントです。エンド・ユーザーに対して SQL ステートメントの重要な部分 (検索する表および列の名前など) を求める対話式アプリケーションが、動的 SQL のよい例です。動的 SQL ステートメントはアプリケーションの実行中に作成され、それからステートメントの処理がサブミットされます。

静的 SQL ステートメントか動的 SQL ステートメント、あるいはその両方を組み込んで、アプリケーションを作成することができます。

一般的に、静的 SQL ステートメントは、トランザクションが事前定義されている高性能のアプリケーションに向いています。予約システムなどが、そのようなアプリケーションのよい例です。

一般的に、動的 SQL ステートメントは、ランタイムにトランザクションを指定する必要がある、頻繁に変更が行われるデータベースに対して実行するアプリケーションに向いています。対話式照会インターフェースなどが、そのようなアプリケーションのよい例です。

アプリケーションに SQL ステートメントを組み込む場合、以下のステップに従って、プリコンパイルを行い、アプリケーションをデータベースにバインドしなければなりません。

1. SQL ステートメントを組み込んだプログラムを含むソース・ファイルを作成する。
2. データベースに接続してから、各ソース・ファイルをプリコンパイルする。

   プリコンパイラーで各ソース・ファイル内の SQL ステートメントを、データベース・マネージャーへの DB2 ランタイム API 呼び出しに変換します。また、プリコンパイラーはデータベース内にアクセス・パッケージを作成します。バインド・ファイルを作成するように指定すれば、オプションでバインド・ファイルの作成も行います。

   アクセス・パッケージには、アプリケーション内の静的 SQL ステートメント用に、 DB2 オプティマイザーが選択したアクセス・プランが含まれています。アクセス・プランには、オプティマイザーが決定した最も有効な方法で、データベース・マネージャーが静的 SQL ステートメントを実行するのに必要な情報が含まれています。動的 SQL ステートメントの場合、アプリケーションの実行時に、オプティマイザーがアクセス・プランを作成します。

   バインド・ファイルには、アクセス・パッケージを作成するのに必要な、 SQL ステートメントと他のデータが含まれています。このバインド・ファイルを使用して、後からアプリケーションを再バインドすることができます。その際に最初にプリコンパイルをする必要はありません。再バインドにより、現在のデータベースの状態に合わせて最適化されたアクセス・プランが作成されます。アプリケーションのプリコンパイルを行ったデータベースとは別のデータベースへアクセスする場合、そのアプリケーションを再バインドする必要があります。最後のバインド以降にデータベース統計を変更した場合、アプリケーションを再バインドするようお勧めします。
3. ホスト言語コンパイラーを使用して、変更したソース・ファイル (および SQL ステートメントを含まない他のファイル) をコンパイルする。

4. オブジェクト・ファイルを DB2 およびホスト言語ライブラリーとリンクさせ、実行可能プログラムを作成する。
5. バインド・ファイルをバインドし、アクセス・パッケージを作成する (プリコンパイル時に行っていなかった場合、あるいは別のデータベースにアクセスする場合)。
6. アプリケーションを実行する。アプリケーションが、パッケージ内のアクセス・プランを使用してデータベースにアクセスします。

関連概念:

- 541 ページの『REXX アプリケーションでの組み込み SQL』
- 156 ページの『C および C++ での組み込み SQL ステートメント』
- 203 ページの『COBOL における組み込み SQL』
- 226 ページの『FORTRAN における組み込み SQL ステートメント』
- 14 ページの『組み込み SQL for Java (SQLJ)』

関連タスク:

- 63 ページの『ホスト言語における組み込み SQL ステートメントの使用』

DB2 コール・レベル・インターフェース

DB2® CLI は、 C および C++ アプリケーションが DB2 データベースにアクセスする際に使用できるプログラミング・インターフェースです。 DB2 CLI は、 Microsoft® Open Database Connectivity (ODBC) 仕様と、 ISO CLI 規格に基づいています。 DB2 CLI は業界標準に基づいているため、それらのデータベース・インターフェースにすでに精通しているアプリケーション・プログラマーであれば、より短期間で習熟することができます。

DB2 CLI を使用する場合、アプリケーションは動的 SQL ステートメントを関数の引き数としてデータベース・マネージャーに渡し、処理します。そのようにして、DB2 CLI は組み込み動的 SQL の代替としての役割を果たします。

CLI、ODBC、または JDBC アプリケーション内で静的 SQL として SQL ステートメントを実行することもできます。 CLI/ODBC/JDBC 静的プロファイル作成機能により、アプリケーションのエンド・ユーザーは、多くの場合に、動的 SQL の代わりに静的 SQL を使用することができるようになります。

ODBC Driver Manager を使用せずに、UNIX® 上の `libdb2` および Windows® オペレーティング・システム上の `db2cli.lib` とアプリケーションをリンクするだけで、いずれのプラットフォーム上でも DB2 の ODBC ドライバーを使用した ODBC アプリケーションを構築することができます。 DB2 CLI サンプル・プログラムは、ODBC アプリケーションの構築方法を例示しています。 DB2 CLI サンプル・プログラムは、 UNIX では `sqllib/samples/cli` に、 Windows オペレーティング・システムでは `sqllib¥samples¥cli` にあります。

DB2 CLI アプリケーションは DB2 とともに提供されている共通アクセス・ パッケージを使用するため、 DB2 CLI アプリケーションのプリコンパイルまたはバインドを行う必要はありません。必要なのは、アプリケーションのコンパイルとリンクだけです。

ただし、 DB2 AD Client に付属している DB2 CLI バインド・ファイルを、アクセスする DB2 データベースごとにバインドしてからでなければ、 DB2 CLI または ODBC アプリケーションは DB2 データベースにアクセスできません。これは最初のステートメントの実行の際に自動的に行われますが、データベース管理者が、各プラットフォームごとに、 DB2 データベースにアクセスするそれぞれのクライアントからバインド・ファイルをバインドするようお勧めします。

たとえば、AIX®、Solaris オペレーティング環境、および Windows 98 クライアントがあり、それぞれが 2 つの DB2 データベースにアクセスするとします。管理者は、アクセスするデータベースごとに、 1 つの AIX クライアントからバインド・ファイルをバインドしなければなりません。次に、管理者は、アクセスするデータベースごとに、 1 つの Solaris オペレーティング環境クライアントからバインド・ファイルをバインドします。最後に、管理者は Windows 98 クライアントで同様の作業を行います。

関連概念:

- 46 ページの『組み込み SQL または DB2 CLI プログラムにおける管理 API』
- 11 ページの『DB2 CLI と組み込み動的 SQL の比較』

関連タスク:

- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 1 巻*」の『CLI/ODBC/JDBC 静的プロファイル作成による静的 SQL の作成』

DB2 CLI と組み込み動的 SQL の比較

動的アプリケーションを開発する際に、組み込み動的 SQL か DB2® CLI のいずれかを使用できます。 どちらの場合でも、 SQL ステートメントはランタイムに準備され、処理されます。各方式には、それぞれユニークな利点があります。

DB2 CLI の利点は、以下のとおりです。

移植性
DB2 CLI アプリケーションはデータベースに SQL ステートメントを渡す際に、関数の標準セットを使用します。 DB2 CLI アプリケーションを実行する前に必要なのは、コンパイルとリンクだけです。対照的に、組み込み SQL アプリケーションは、プリコンパイルしてからコンパイルし、その後それらのアプリケーションをデータベースにバインドしてからでなければ実行できません。この処理により、アプリケーションを特定のデータベースに効果的に結び付けます。

バインドを行わない
DB2 CLI アプリケーションは、アクセスするデータベースごとに個別にバインドする必要がありません。必要なのは、すべての DB2 CLI アプリケーションに使用できる、 DB2 CLI に付属しているバインド・ファイルをバインドすることだけです。これにより、アプリケーションの管理に費やされる時間が大幅に減ります。

取り出しおよび入力の拡張
DB2 CLI 関数を使用すれば、一度の呼び出しで、データベース内の複数の行を 1 つの配列に取り出すことができます。また、入力変数の配列を使用して、 SQL ステートメントを何回も実行できます。

カタログに対する整合したインターフェース

データベース・システムには、データベースとそのユーザーに関する情報が入っているカタログ表が含まれています。これらのカタログの形式は、システム間で異なる場合があります。 DB2 CLI は、表、列、外部キー、主キー、およびユーザー特権などの、コンポーネントに関するカタログ情報を照会する整合性インターフェースを提供します。これにより、データベース・サーバーのリリース間でのカタログの変更、およびデータベース・サーバー間での相違からアプリケーションが保護されます。特定のサーバーまたは製品のバージョンに固有のカタログ照会を作成する必要はありません。

データ変換の拡張

DB2 CLI は、 SQL と C のデータ・タイプ間で自動的にデータを変換します。たとえば、 SQL データ・タイプを C 文字データ・タイプに取り出すと、文字ストリング表示に変換されます。そのため、 DB2 CLI は対話式照会アプリケーションに適しています。

グローバル・データ域がない

一般に、組み込み SQL アプリケーションには、アプリケーションが制御するグローバル・データ域 (SQLDA および SQLCA など) が関連付けられており、それらはしばしば複雑になることがあります。しかし、DB2 CLI にはグローバル・データ域が必要ではありません。その代わりに、 DB2 CLI が、必要なデータ構造を自動的に割り当てて制御し、それらを参照するためのアプリケーションのハンドルを提供します。

ストアード・プロシージャーからの結果セットの取り出し

DB2 CLI アプリケーションには、DB2 Universal Database™ サーバー、DB2 for MVS™/ESA サーバー (バージョン 5 またはそれ以降)、 OS/400® サーバー (バージョン 5 またはそれ以降) にあるストアード・プロシージャーから生成される複数行と結果セットを取り出す機能が備えられています。 OS/400 において複数の結果セットの取り出しをサポートするには、PTF (プログラム一時修正) SI01761 をサーバーに適用する必要があります。必要に応じて、使用する OS/400 のシステム管理者に、この PTF が適用されているかどうか問い合わせてください。

両方向スクロール・カーソル

DB2 CLI は、サーバー側の両方向スクロール・カーソルをサポートしています。これは、配列出力と併用することができます。これは、「Page Up」、「Page Down」、「Home」、および「End」キーを使用するスクロール・ボックスで、データベース情報を表示する GUI アプリケーションに役立ちます。カーソルをスクロール可能と宣言すれば、 1 行または複数行ごとに結果セット内を順方向または逆方向に移動できます。また、現在の行からの相対位置や、結果セットの最初または最後からの相対位置、あるいはブックマークが既に付けられている特定の行からの相対位置を指定して、行を取り出すこともできます。

組み込み動的 SQL の利点は、以下のとおりです。

きめ細かなセキュリティー

すべての DB2 CLI ユーザーは、同じ権限を共有します。組み込み SQL の利点は、パッケージ用の特定のユーザーに実行権限を付与することによってさらにきめ細かなセキュリティーを提供することです。

さらに多くの言語のサポート

組み込み SQL は、C および C++ 以外の言語もサポートしています。アプリケーションを別の言語でコーディングする場合、この点が利点と言えるかもしれません。

静的 SQL とかなり整合している

動的 SQL は一般的に言って静的 SQL とかなり整合性があります。すでに静的 SQL のプログラミング方法を知っている場合、動的 SQL に移行する方が DB2 CLI に移行するよりも簡単です。

関連概念:

- 145 ページの『DB2 コール・レベル・インターフェース (CLI) と組み込み動的 SQL の比較』
- 146 ページの『組み込み SQL と比較した DB2 CLI の利点』
- 148 ページの『DB2 CLI または組み込み SQL をいつ使用するか』

Java Database Connectivity (JDBC)

DB2® の Java™ サポートには JDBC が含まれます。 JDBC はベンダーに依存しない動的 SQL インターフェースで、標準化された Java メソッドを使用した、アプリケーションへのデータ・アクセスを提供します。 JDBC は、JDBC プログラムのプリコンパイルまたはバインドを必要としないという点で、 DB2 CLI と類似しています。ベンダーに依存しない規格であるため、 JDBC アプリケーションは高い移植性を持ちます。 JDBC を使用して書かれたアプリケーションは動的 SQL だけしか使用しません。

JDBC は、インターネットを介して DB2 データベースにアクセスする場合に特に便利です。 Java プログラム言語を使用して、ネットワーク接続を使ってリモート DB2 データベース内のデータにアクセスしたり、操作したりする JDBC アプリケーションおよびアプレットを開発できます。また、サーバー上に常駐し、データベース・サーバーにアクセスしたり、ストアード・プロシージャーを呼び出したリモート・クライアント・アプリケーションに情報を戻す、 JDBC ストアード・プロシージャーを作成することもできます。

JDBC API は CLI/ODBC API と類似していて、 Java コードからデータベースにアクセスする標準的な方法を提供しています。 Java コードは、メソッドの引き数として SQL ステートメントを DB2 JDBC ドライバーに渡します。ドライバーはクライアントの Java コードから JDBC API 呼び出しを処理します。

Java の移植性により、複数のプラットフォーム上のクライアントに DB2 アクセスを配置することができます。しかも、その際に必要なのは Java 対応 Web ブラウザーつまり Java runtime environment だけです。

JDBC Type 2

JDBC type 2 ドライバーに基づく Java アプリケーションでは、 DB2 に接続するために DB2 クライアントが必要です。他のアプリケーションと同様に、デスクトップまたはコマンド行からアプリケーションを開始します。 DB2 JDBC ドライバーは、アプリケーションからの JDBC API 呼び出しを処理し、クライアント接続を使用して要求をサーバーに通信し、その結果を受け取ります。 JDBC type 2 ドライバーを使用した Java アプレットは作成できません。

注: JDBC type 2 ドライバーは、WebSphere® Application Server で使用することが推奨されています。

JDBC Type 3

JDBC type 3 ドライバーを使用する場合、Java アプレットしか作成できません。Java アプレットは、クライアントのマシンに DB2 クライアントがインストールされていなくても動作します。通常 、アプレットはハイパーテキスト・マークアップ言語 (HTML) の Web ページに組み込みます。

JDBC type 3 ドライバーに基づくアプレットを実行するために必要なことは、クライアント・マシンに Java を使用できる Web ブラウザーつまりアプレット・ビューアーをインストールしておくことだけです。 HTML ページをロードすると、ブラウザーが Java アプレットをユーザーのマシンにダウンロードし、さらに Java クラス・ファイルおよび DB2 の JDBC ドライバーもダウンロードします。アプレットが DB2 に接続するために JDBC API を呼び出すと、 JDBC ドライバーは、Web サーバーにある JDBC アプレット・サーバーを介して、 DB2 データベースとの個別のネットワーク接続を確立します。

注: JDBC type 3 ドライバーは、バージョン 8 では使用できません。

JDBC Type 4

バージョン 8 用に新しく開発された JDBC type 4 ドライバーを使用して、Java アプリケーションとアプレットの両方を作成できます。Type 4 ドライバーに基づいたアプリケーションまたはアプレットを実行するために必要なものは、`db2jcc.jar` ファイルだけです。DB2 クライアントは必要ありません。

DB2 JDBC サポートの詳細については、以下のアドレスの Web ページを参照してください。

`http://www.ibm.com/software/data/db2/udb/ad/v8/java`

組み込み SQL for Java (SQLJ)

DB2® Java™ 組み込み SQL (SQLJ) サポートは、 DB2 AD Client によって提供されます。 DB2 SQLJ サポートと DB2 JDBC サポートによって、 SQLJ アプレット、アプリケーション、およびストアード・プロシージャーを構築し、実行できます。 これらには、静的 SQL が含まれ、 DB2 データベースにバインドされた組み込み SQL ステートメントを使用します。

DB2 SQLJ サポートの詳細については、以下のアドレスの Web ページを参照してください。

`http://www.ibm.com/software/data/db2/udb/ad/v8/java`

ADO (ActiveX Data Object) および RDO (Remote Data Object)

DAO (Data Access Object) および RDO (Remote Data Object) 仕様に準拠した、Microsoft® Visual Basic および Visual C++ データベース・アプリケーションを開発することができます。 さらに、 DB2® では ODBC Bridge に Microsoft OLE DB を使用する ADO (ActiveX Data Object) アプリケーションもサポートしています。

ADO (ActiveX Data Object) を使用すれば、 OLE DB Provider を使用して、データベース・サーバー内のデータにアクセスしたり、操作したりするアプリケーションを作成できます。 ADO の主要な利点は、開発速度が速く、使用が容易で、ディスク・フットプリントが小さいことです。

RDO (Remote Data Object) は、 ODBC を介してリモート・データ・ソースにアクセスするための情報モデルを提供します。 RDO が提供するオブジェクトのセットを使用すれば、データベースへの接続、照会の実行、ストアード・プロシージャーの実行、結果の操作、変更のサーバーへのコミットが容易になります。これは、リモート ODBC リレーショナル・データ・ソースへアクセスするために特別に設計されたものであり、複雑なアプリケーション・コードを使用せずに ODBC を使用することが容易になっています。

ADO および RDO 仕様を使用する DB2 アプリケーションの完全なサンプルについては、以下のディレクトリーを参照してください。

- Visual Basic で ActiveX Data Object を使用するサンプルについては、`sqllib¥samples¥VB¥ADO` をご覧ください。
- Visual Basic で Remote Data Object を使用するサンプルについては、`sqllib¥samples¥VB¥RDO` をご覧ください。
- Visual Basic で Microsoft Transaction Server を使用するサンプルについては、sqllib¥samples¥VB¥MTS をご覧ください。
- Visual C++ で ActiveX Data Object を使用するサンプルについては、`sqllib¥samples¥VC¥ADO` をご覧ください。

関連タスク:

- 「*アプリケーション開発ガイド アプリケーションの構築および実行*」の『Visual Basic による ADO アプリケーションの構築』
- 「*アプリケーション開発ガイド アプリケーションの構築および実行*」の『Visual Basic による RDO アプリケーションの構築』
- 「*アプリケーション開発ガイド アプリケーションの構築および実行*」の『Visual C++ による ADO アプリケーションの構築』

関連資料:

- 「*アプリケーション開発ガイド アプリケーションの構築および実行*」の『Visual Basic のサンプル』
- 「*アプリケーション開発ガイド アプリケーションの構築および実行*」の『Visual C++ のサンプル』

Perl DBI

DB2® は、 DBD::DB2 ドライバーを介したデータ・アクセスに使用する、 Perl Database Interface (DBI) 仕様をサポートしています。 DB2 Universal Database™ Perl DBI の Web サイトは、以下のアドレスにあります。

```
http://www.ibm.com/software/data/db2/perl/
```

このサイトで、最新の DBD::DB2 ドライバーと関連情報を入手できます。

Perl はインタープリター言語であり、 Perl DBI モジュールは動的 SQL を使用します。このため、DB2 アプリケーションのプロトタイプを短時間で作成および修正する上で Perl は 理想的な言語といえます。 Perl DBI モジュールは、 CLI および JDBC と大変よく似たインターフェースを使用します。それで、Perl プロトタイプを CLI および JDBC に簡単に移植することができます。

関連概念:

- 535 ページの『Perl でのプログラミングに関する考慮事項』

ODBC エンド・ユーザー・ツール

Lotus® Approach、Microsoft® Access、および Microsoft Visual Basic などの ODBC エンド・ユーザー・ツールを使用してアプリケーションを作成することもできます。ODBC ツールを使用すれば、高水準プログラム言語を使用する場合よりも簡単にアプリケーションを開発できます。

Lotus Approach には、DB2® データにアクセスする方法が 2 つあります。 1 つ目は、グラフィカル・インターフェースを使用して、照会の実行、レポートの作成、およびデータの分析を行う方法です。もう 1 つは、 LotusScript を使用してアプリケーションを開発する方法です。 LotusScript は完全仕様の、オブジェクト指向プログラム言語で、多岐にわたるオブジェクト、イベント、メソッド、およびプロパティーを備えており、プログラム・エディターが組み込まれています。

DB2 .NET Data Provider

DB2® .NET Data Provider は、DB2 の ADO.NET インターフェース・サポートを拡張したものです。 DB2 .NET Data Provider は、DB2 データへのハイ・パフォーマンスのセキュア・アクセスを提供します。

DB2 .NET Data Provider により、 .NET アプリケーションは、次のデータベース管理システムにアクセスできます。

- Windows®、UNIX®、および Linux ベースのコンピューター用の DB2 Universal Database™ バージョン 8
- DB2 Connect™ 経由で、DB2 Universal Database for OS/390® and z/OS™ バージョン 6 (以降)
- DB2 Connect 経由で、DB2 Universal Database for AS/400® and iSeries™ バージョン 5 リリース 1 (以降)
- DB2 Connect 経由で、DB2 Universal Database for VSE & VM バージョン 7.3 (以降)

DB2 .NET Data Provider を使用するアプリケーションを開発および実行するには、.NET Framework バージョン 1.0 または 1.1 が必要です。

DB2 .NET Data Provider のほかに、 Microsoft® Visual Studio .NET IDE 用のアドイン・セットがあります。これらのアドインを使用すれば、 ADO.NET インターフェースを使用する DB2 アプリケーションの作成が単純化されます。さらに、これらのアドインを使用して、サーバー側オブジェクト (たとえば SQL ストアード・プロシージャーやユーザー定義関数) を開発することもできます。

DB2 .NET Data Provider の機能を示す VB.NET および C#.NET のサンプル・アプリケーションは、以下のサイトで入手できます。

```
http://www.ibm.com/software/data/db2/udb/ad/v8/samples.html
```

Web アプリケーション

以下のセクションでは、Web アプリケーションを構築するために使用可能な製品と関数について説明します。

Web アプリケーションの構築のためのツール

DB2® Universal Database は主要なインターネット標準をすべてサポートする、Web での使用に理想的なデータベースです。 これには、インターネット検索を容易にするメモリー内速度と、リレーショナル・データベースの拡張容易性と可用性という特性を兼ね備えた複雑なテキスト・マッチングがあります。 DB2 Universal Database は WebSphere®、Java™ および XML Extender をサポートしているため、ユーザーによる e-business アプリケーションの展開を容易にします。

DB2 Universal Developer's Edition には、 Web 使用可能性サポートを提供するいくつかのツールが含まれています。 WebSphere Studio Application Developer, Version 4 は、 WebSphere Application Server や DB2 Universal Database への Java アプリケーションの構築、検査、および展開を可能にする統合開発環境 (IDE) です。 WebSphere Studio は、Web サイト開発のすべての局面を 1 つの共通インターフェース内にまとめるツール群です。 WebSphere Application Server Advanced Edition (シングル・サーバー) は、 e-business アプリケーションのための堅固な開発環境を備えています。このコンポーネントにより、ユーザーは個人設定した動的 Web コンテンツを素早く容易に構築および展開できます。

関連概念:

- 17 ページの『WebSphere Studio』
- 18 ページの『XML Extender』

WebSphere Studio

WebSphere® Studio は、Web サイト開発のすべての局面を 1 つの共通インターフェース内にまとめるツール群です。 WebSphere Studio は、動的な対話式 Web アプリケーションの協同的な作成、収集、保守をこれまでになく容易にしています。 Studio は、ワークベンチ、ページ・デザイナー、リモート・デバッガー、およびウィザードから成り、 Macromedia Flash、Fireworks、Freehand、および Director など

の一連の Web 開発製品の試用コピーが付随しています。 WebSphere Studio を使用すると、拡張ビジネス機能をサポートする対話式の Web サイトを作成するために必要なあらゆる作業を行えます。

WebSphere Application Server スタンダード版 (DB2® Universal Developer's Edition に付属) は、 WebSphere Studio のコンポーネントです。 これは、サーバー側のビジネス・アプリケーションの移植性と Java™ テクノロジーのパフォーマンスおよび管理の容易性を結び付けて、 Java ベースの Web アプリケーションを設計するための包括的なプラットフォームを提供しています。 これは、エンタープライズ・データベースやトランザクション・システムとの強力な対話を可能にします。 DB2 サーバーは、WebSphere Application Server と同じマシン上か、別の Web サーバー上で実行できます。

WebSphere Application Server アドバンスト版 (DB2 Universal Developer's Edition には付属していません) は、 Enterprise JavaBean アプリケーションの追加サポートを提供しています。 DB2 Universal Database™は WebSphere Application Server アドバンスト版に付属し、Administration Server リポジトリーとして使用されます。 これは、Sun Microsystems 社の EJB 仕様に基づいて構築されるアプリケーション用のサーバー機能を導入し、 Web アプリケーションを Web ではないビジネス・システムに統合するためのサポートを提供しています。

関連概念:

- 530 ページの『Enterprise Java Beans』

関連資料:

- 「*アプリケーション開発ガイド アプリケーションの構築および実行*」の『Java WebSphere のサンプル』

XML Extender

XML (Extensible Markup Language) は、アプリケーション間でデータを交換するための、一般に認められている標準技法です。 XML 文書は、人間が解読できる、タグの付けられた文書です。テキストは文字データとマークアップ・タグからなります。マークアップ・タグは、文書の作成者が定義できます。文書型定義 (DTD) は、マークアップ定義と制約を宣言するために使用されます。 DB2® XML Extender (Windows® 用の DB2 Universal Developer's Edition および Personal Developer's Edition に付属) は、プログラムが SQL 拡張機能を使用して XML データを操作するためのメカニズムを提供しています。

DB2 XML Extender は、 `XMLVARCHAR`、`XMLCLOB`、および `XMLFILE` という 3 つの新しいデータ・タイプを紹介しています。 エクステンダーは、単一または複数の列または表にある XML 文書を保管、抽出、および更新する UDF を提供しています。検索は、XML 文書全体について実行するか、または位置パスを使用して構造コンポーネントに基づいて実行することができます。ここでは、XSLT (Extensible Stylesheet Language Transformation) と XPath (XML Path Language) のサブセットが使用されます。

列セットとしての XML 文書の保管を容易にするため、 DB2 XML Extender は、設計者が XML からリレーショナル・データベースへのマッピングを行うための補助となる管理ツールを提供しています。 DAD (Document Access Definition) は、

XML 文書の構造およびマッピング・データを保守するために使用されます。 DAD は XML 文書として定義および保管されるので、操作および理解するのが簡単です。文書を作成または分解するための新しいストアード・プロシージャーが提供されています。

DB2 XML Extender についての詳細は、以下を参照してください。

http://www.ibm.com/software/data/db2/extenders/xmlext/index.html

MQSeries 使用可能性

DB2® Universal Database には、MQSeries® 機能のセットが備えられ、 DB2 アプリケーションによる非同期メッセージング操作との対話を可能にしています。これは、DB2 がサポートするどのプログラミング言語で作成されたアプリケーションにも、 MQSeries サポートが使用可能であることを意味します。

基本的な構成においては、 MQSeries サーバーは DB2 Universal Database™ と一緒にデータベース・サーバー上に配置されます。 MQSeries 機能は DB2 サーバーから利用でき、他の MQSeries アプリケーションへのアクセスを提供します。複数の DB2 クライアントがデータベースを介して MQSeries 機能にアクセスできます。 MQSeries 操作により、 DB2 アプリケーションは非同期で他の MQSeries アプリケーションと通信できます。たとえば、新機能は、DB2 アプリケーションがデータベース・イベントをリモート MQSeries アプリケーションに発行したり、オプションの MQSeries Workflow 製品を介してワークフローを開始したり、オプションの MQSeries Integrator 製品を用いて既存のアプリケーション・パッケージと通信したりするための単純な手段を提供しています。

Net.Data

Net.Data® を使用することにより、 Web アプリケーションを介して、 DB2® データへのインターネットおよびイントラネット・アクセスを行うことができます。 これは、共通ゲートウェイ・インターフェース (CGI) アプリケーションよりも高いパフォーマンスを提供する、 Web サーバー・インターフェース (API) を活用します。 Net.Data は、 Java™、REXX、Perl、および C++ などの言語を使用したサーバー側での処理の他に、クライアント側の処理もサポートしています。 Net.Data は条件付きの論理および豊富なマクロ言語を提供しています。これは XML サポートも提供しているので、ユーザーは XML タグを手動で入力する代わりに、このタグを Net.Data マクロの出力として生成することができます。生成される出力のフォーマット設定と表示に XML スタイル・シート (XSL) を使用することを指定することもできます。 Net.Data は Web ベースのダウンロードとしてのみ利用可能です。詳細については、以下の Web サイトを参照してください。

http://www-4.ibm.com/software/data/net.data/support/index.html

注: Net.Data のサポートは、DB2 Version 7.2 において確立され、今後 Net または Data のサポートを拡張する予定はありません。

関連概念:

- 17 ページの『Web アプリケーションの構築のためのツール』

• 18 ページの『XML Extender』

プログラミング機能

以下のセクションでは、DB2 で使用可能なプログラミング機能について説明しています。

DB2 プログラミング機能

DB2® には、サーバー上で実行するさまざまな機能があります。それらの機能を使用して、アプリケーションを補足したり、拡張したりすることができます。 DB2 機能を使用する場合、同じ作業を行うために独自のコードを作成する必要はありません。また DB2 では、クライアント・アプリケーションに全コードを保持するのではなく、サーバーにコードの一部を保管します。これにより、パフォーマンスが向上し、保守が容易になります。

データの保護、およびデータ間の相互関係を定義する機能があります。さらに、柔軟な拡張アプリケーションを作成する、オブジェクト関連の機能もあります。機能の中には、複数の方法で使用できるものもあります。たとえば、制約はデータの保護と、データ値間のリレーションシップの定義に使用できます。主な DB2 機能を以下にリストします。

- 制約
- ユーザー定義タイプ (UDT) およびラージ・オブジェクト (LOB)
- ユーザー定義関数 (UDF)
- トリガー
- ストアード・プロシージャー

DB2 機能を使用するかどうか決定する場合、以下の点を考慮してください。

アプリケーションの独立性
アプリケーションを、それが処理するデータから独立させることができます。データベース上で実行する DB2 機能を使用すれば、アプリケーションに影響を与えることなくデータを扱うロジックを保守したり、変更したりすることができます。ロジックを変更しなければならない場合、変更を行うのはサーバーだけで、データにアクセスする各アプリケーションでは変更を行う必要はありません。

パフォーマンス
サーバー上にアプリケーションの一部を保管し、実行することにより、アプリケーションのパフォーマンスを向上させることができます。これにより、いくつかの処理を、一般により強力なサーバー・マシンに移し、クライアント・アプリケーションとサーバーの間のネットワーク通信量を減らすことができます。

アプリケーション要件
ご使用のアプリケーションが、他のアプリケーションとは異なる独自のロジックを使用している場合があります。たとえば、ご使用のアプリケーションが、他のアプリケーションでは適切ではない特定の順序でデータ入力エラーを処理する場合、そのような状況を処理するために独自のコードを作成する必要があります。

DB2 機能は複数のアプリケーションで使用できるため、ときには、サーバー上で実行する DB2 機能を使用することにするかもしれません。一方、自分のアプリケーションだけで使用するロジックの場合には、それを自分のアプリケーションに保持してしまう場合もありえます。

関連概念:

- 21 ページの『DB2 ストアード・プロシージャー』
- 22 ページの『DB2 ユーザー定義関数およびメソッド』
- 24 ページの『ユーザー定義タイプ (UDT) およびラージ・オブジェクト (LOB)』
- 26 ページの『DB2 トリガー』

DB2 ストアード・プロシージャー

多くの場合、アプリケーションはネットワークを介してデータベースにアクセスします。このため、たくさんのデータが戻されると、パフォーマンスが低下します。ストアード・プロシージャーはデータベース・サーバーで実行されます。クライアント・アプリケーションはストアード・プロシージャーを呼び出すことができます。ストアード・プロシージャーはデータベースへのアクセスを実行しますが、その際にネットワークを介して必要のないデータを戻すことはありません。クライアント・アプリケーションが必要とする結果だけが、ストアード・プロシージャーによって戻されます。

ストアード・プロシージャーの使用には以下の利点があります。

ネットワーク通信量が削減される

SQL ステートメントをグループ化することにより、ネットワーク通信量を節約できます。典型的なアプリケーションの場合、 SQL ステートメントごとに、ネットワークを介した 2 回のトリップが必要です。 SQL ステートメントをグループ化しておけば、ステートメントのグループごとに、ネットワークを介した 2 回のトリップを行うだけで済みます。これにより、アプリケーションのパフォーマンスが向上します。

サーバー上にしかない機能へのアクセス

ストアード・プロシージャーはサーバー上でしか実行されないコマンド (LIST DATABASE DIRECTORY や LIST NODE DIRECTORY など) にアクセスできます。これには、サーバー・マシンにある豊富なメモリーおよびディスク・スペースを利用できるという利点があります。さらに、サーバーにインストールされている任意の追加ソフトウェアにアクセスすることもできます。

内部規則の施行

ストアード・プロシージャーを使用して、複数のアプリケーションに共通の内部規則を定義できます。これは、制約およびトリガーを使用する方法に加えて、内部規則を定義する別の方法です。

アプリケーションがストアード・プロシージャーを呼び出すと、ストアード・プロシージャーで定義した規則に従って、整合性が確保される方法でデータが処理されます。規則を変更しなければならな

い場合、変更は、ストアード・プロシージャーを呼び出すアプリケーションごとに行うのではなく、ストアード・プロシージャー内で一度行うだけで済みます。

関連概念:

- 22 ページの『Development Center』

DB2 ユーザー定義関数およびメソッド

SQL で提供されている組み込み機能では、アプリケーションの必要すべてを満たせない場合があります。それらの機能を拡張するために、 DB2® はユーザー定義関数 (UDF) およびメソッドをサポートしています。 Visual Basic、C/C++、Java™、または SQL で独自のコードを作成し、単一のスカラー値または表を戻す SQL ステートメントを使用して操作を実行することができます。

UDF およびメソッドにより柔軟性が著しく向上します。これらは式の一部として単一のスカラー値を戻します。さらに関数は、スプレッドシートなどのデータベースではないソースから表全体を戻すことも可能です。

UDF およびメソッドを使用することにより、アプリケーションを標準化することができます。ルーチンの共通セットをインプリメントすることにより、複数のアプリケーションが同じ方法でデータを処理できるので、整合性のある結果を確実に入手できます。

また、ユーザー定義関数およびメソッドは、アプリケーションにおけるオブジェクト指向プログラミングをサポートします。これらは抽象化を提供しており、データ・オブジェクトへの操作を実行するのに使用できる共通インターフェースを定義することができます。さらに、これらのものが提供しているカプセル化を使用すれば、オブジェクトの基本データへのアクセスを制御したり、直接操作や起きるかもしれない破壊からオブジェクトを保護することができます。

Development Center

DB2® Development Center は、ストアード・プロシージャーを作成、インストール、およびテストするための使いやすい開発環境を提供します。これにより、DB2 サーバーでのストアード・プロシージャーの登録、構築、およびインストールに関する詳細にあまり労力をかけることなく、ストアード・プロシージャーのロジックに注意を向けることができます。さらに、Development Center を使用すると、あるオペレーティング・システムで開発したストアード・プロシージャーを、他のサーバー・オペレーティング・システムで構築できます。

Development Center は、迅速な開発をサポートするグラフィカル・アプリケーションです。 Development Center を使用すると以下のような作業を行えます。

- 新規のストアード・プロシージャーを作成する。
- ローカルおよびリモート DB2 サーバーでストアード・プロシージャーを作成する。
- 既存のストアード・プロシージャーを変更して再作成する。
- インストールされたストアード・プロシージャーのテストとデバッグを行う。

Development Center は DB2 Universal Database™ プログラム・グループとは別個のアプリケーションとして立ち上げることもできますし、以下の開発アプリケーションのいずれかから立ち上げることもできます。

- Microsoft® Visual Studio
- Microsoft Visual Basic
- IBM® VisualAge® for Java™

また、Development Center は、DB2 for OS/390® のコントロール・センターからも立ち上げられます。 Development Center は、コントロール・センターの「ツール」メニュー、ツールバー、 または「ストアード・プロシージャー」フォルダーから別個のメニューとして始動することができます。さらに、「Development Center プロジェクト」ウィンドウから、 DB2 for OS/390 サーバーに作成した SQL ストアード・プロシージャーを 1 つ以上選択して、コマンド行プロセッサー (CLP) で実行可能な指定したファイルにエクスポートすることができます。

Development Center は、プロジェクトを使用して作業を管理します。それぞれの Development Center プロジェクトは、 DB2 for OS/390 サーバーなどの特定のデータベースへの接続を保管します。さらに、各データベース上にストアード・プロシージャーのサブセットを表示するためのフィルターを作成することができます。新規または既存の Development Center プロジェクトを開くとき、その名前、スキーマ、言語、またはコレクション ID (DB2 for OS/390 のみ) に基づくストアード・プロシージャーを表示するため、ストアード・プロシージャーにフィルターをかけることができます。

接続情報は、Development Center プロジェクトに保管されます。そのため、既存のプロジェクトを開くと、そのデータベースのユーザー ID とパスワードを入力するプロンプトが自動的に表示されます。「SQL ストアード・プロシージャーの挿入」ウィザードを使用して、 DB2 for OS/390 サーバーで SQL ストアード・プロシージャーを作成することができます。 DB2 for OS/390 サーバーに作成した SQL ストアード・プロシージャーの場合、特定のコンパイル、プリリンク、リンク、バインド、実行時、WLM 環境、および外部セキュリティー・オプションを設定することができます。

さらに、SQL ストアード・プロシージャーに関する SQL コスト情報 (SQL ストアード・プロシージャーが実行中のスレッドに応じた CPU 時間や他の DB2 コスト情報についての情報を含む) を取得することができます。特に、ラッチ/ロックの競合待ち時間、取得したページ数、読み取り I/O の数、および書き込み I/O の数に関するコスト情報を取得することができます。

コスト情報を取得するため、 Development Center は DB2 for OS/390 に接続し、SQL ステートメントを実行して、どの程度の CPU 時間と SQL ストアード・プロシージャーが使われるかを知るため、ストアード・プロシージャー (DSNWSPM) を呼び出します。

関連概念:

- 21 ページの『DB2 ストアード・プロシージャー』
- 25 ページの『OLE オートメーション・ルーチン』

ユーザー定義タイプ (UDT) およびラージ・オブジェクト (LOB)

データベース内の各データ・エレメントは表の列に保管され、各列はデータ・タイプを持つように定義されます。データ・タイプは列に入れることができる値のタイプと、その値に対して実行できる操作を制限します。たとえば、データ・タイプが整数である列に入れることができるのは、決められた範囲内の数字だけです。DB2® には、特性と振る舞いが定義されている、組み込みデータ・タイプ (文字ストリング、数値、日時値、ラージ・オブジェクト、 NULL、GRAPHIC ストリング、バイナリー・ストリング、およびデータ・リンク) のセットが含まれています。

しかし、組み込みデータ・タイプではアプリケーションの必要を満たさない場合もあります。 DB2 が提供しているユーザー定義タイプ (UDT) を使用すれば、アプリケーションが必要とする特殊なデータ・タイプを定義できます。

UDT は組み込みデータ・タイプに基づいています。 UDT を定義する場合、その UDT に有効な操作も定義します。たとえば、 DECIMAL データ・タイプに基づいて、 MONEY というデータ・タイプを定義することができます。しかし、MONEY データ・タイプで行える操作は加算と減算だけで、乗算と除算の操作は行えません。

ラージ・オブジェクト (LOB) を使用すれば、データベース内の大きくて複雑なオブジェクト (音声、ビデオ、画像、および大きい文書など) を保管したり、操作したりすることができます。

UDT と LOB を組み合わせると、非常に便利です。内部データをモデル化したり、そのデータのセマンティクスを取り込んだりする際に、 DB2 が提供している組み込みデータ・タイプを使用するように制限されることがなくなります。 UDT を使用して、拡張アプリケーションで使用する、大きくて複雑なデータ構造を定義することができます。

組み込みデータ・タイプの拡張に加えて、 UDT には他に以下の利点があります。

アプリケーション内のオブジェクト指向プログラミングのサポート
: 類似したオブジェクトを関連データ・タイプにグループ化できます。これらのタイプには、名前、内部表記、および特定の振る舞いを指定できます。 UDT を使用して、新しいタイプの名前、および内部での表記方法を DB2 に知らせることができます。 LOB は新しいタイプの内部表記として使用できるものの 1 つで、大きくて複雑なデータ構造には最も適しています。

強力なタイプ指定およびカプセル化によるデータ保全性
: 強力なタイプ指定により、特殊タイプで定義された関数と操作しかそのタイプには適用されません。カプセル化により、 UDT の振る舞いは UDT に適用できる関数と操作によって確実に制限されます。 DB2 では、 UDT の振る舞いをユーザー定義関数 (UDF) の形式で指定できます。 UDF はユーザーの幅広い要件に合わせて作成できます。

データベース・マネージャーへの組み込みによるパフォーマンス
: UDT は、組み込みデータ・タイプと同じ方法で内部的に表記されるため、組み込みデータ・タイプと同じ有効なコードを共有して、組み込み関数、比

較演算子、索引、および他の関数をインプリメントします。例外は、LOB を使用する UDT です。 LOB は比較演算子および索引と併用することはできません。

関連概念:

- 「*アプリケーション開発ガイド サーバー・アプリケーションのプログラミング*」の『ラージ・オブジェクトの使用法』
- 「*アプリケーション開発ガイド サーバー・アプリケーションのプログラミング*」の『ユーザー定義タイプ (UDT)』

OLE オートメーション・ルーチン

OLE (オブジェクトのリンクと埋め込み) オートメーションは、 Microsoft® Corporation による OLE 2.0 アーキテクチャーの一部です。 OLE を使用すれば、アプリケーションの作成に使用した言語に関係なく、そのアプリケーションで OLE オートメーション・オブジェクトのプロパティーおよびメソッドを公開できます。そうすると、 Lotus® Notes や Microsoft Exchange などの他のアプリケーションが、 OLE オートメーションを介してこれらのプロパティーおよびメソッドを利用することにより、これらのオブジェクトを統合することができます。

Windows® オペレーティング・システム版の DB2® は、UDF、メソッド、およびストアード・プロシージャーを使用する OLE オートメーション・オブジェクトへのアクセスを提供します。OLE オートメーション・オブジェクトにアクセスしそのメソッドを呼び出すためには、オブジェクトのメソッドをルーチン (UDF、メソッド、またはストアード・プロシージャー) として登録しておかなければなりません。登録を行うと、ルーチンを呼び出すことにより DB2 アプリケーションがメソッドを使用できるようになります。

たとえば、 Microsoft Excel などの製品を使用して作成した、スプレッドシート内のデータを照会するアプリケーションを開発できます。このようなアプリケーションを開発する場合、ワークシートからデータを取り出し、DB2 に戻す、 OLE オートメーション表関数を開発します。そうすれば、 DB2 はそのデータを処理し、オンライン分析処理 (OLAP) を実行して、照会結果をアプリケーションに戻すことができます。

関連概念:

- 21 ページの『DB2 ストアード・プロシージャー』
- 22 ページの『Development Center』

OLE DB 表関数

Microsoft® OLE DB は、さまざまな情報ソースに保管されているデータへの同じ方法によるアクセスをアプリケーションに提供する、 OLE/COM インターフェースのセットです。 DB2® Universal Database は、 OLE DB データ・ソースにアクセスする表関数を定義できるようにすることにより、 OLE DB アプリケーションの作成を単純化しています。 GROUP BY、JOIN、および UNION を含む操作を、 OLE DB によってデータを公開しているデータ・ソースに対して実行することができます。たとえば、OLE DB 表関数を定義して、 Microsoft Access データベースまたは

Microsoft Exchange のアドレス帳から表を戻し、それからこの OLE DB 表関数からのデータと DB2 データベース中のデータとをシームレスに結合したレポートを作成することができます。

OLE DB 表関数を使用すると、 OLE DB Provider への組み込みアクセスが提供されるため、アプリケーション開発に費やす労力が軽減されます。データを取り出すためには、C、Java™、および OLE オートメーション表関数では表関数をインプリメントする必要があり、 OLE DB 表関数の場合は OLE DB Provider との間の汎用組み込み OLE DB Consumer インターフェースをインプリメントする必要があります。その際に必要なのは、言語タイプ OLEDB の表関数を登録し、 OLE DB Provider と、関係のある行セットをデータ・ソースとして参照することだけです。OLE DB 表関数を使用するために、なんらかの UDF プログラミングを行う必要はありません。

関連概念:

- 247 ページの『IBM OLE DB Provider for DB2 の目的』
- 251 ページの『IBM OLE DB Provider で自動的に使用可能になる OLE DB サービス』

関連資料:

- 257 ページの『IBM OLE DB Provider での OLE DB コンポーネントおよびインターフェースのサポート』
- 259 ページの『IBM OLE DB Provider での OLE DB プロパティーのサポート』

DB2 トリガー

トリガーは、指定した表に対する INSERT、UPDATE、または DELETE 操作イベントへの応答として実行される、一連のアクションを定義します。そのような SQL 操作が実行されるときに、トリガーが活動化されるように指示が出されます。トリガーの活動化は、 SQL 操作の前と後のどちらでも実行することができます。トリガーは SQL ステートメント CREATE TRIGGER を使用して定義します。

更新または挿入の前に実行するトリガーは、以下のいくつかの用途に使用できます。

- データベース内で実際に値の更新または挿入を行う前に、値のチェックや変更を行う。これは、画面上でユーザーが見ているものから何らかの内部データベース形式に、データを変換しなければならない場合に便利です。
- ユーザー定義関数でコーディングされている、他の非データベース操作を実行する。

同様に、更新または挿入の後に実行するトリガーは、以下のいくつかの用途に使用できます。

- 他の表のデータを更新する。この機能は、データ間のリレーションシップを保守したり、監査証跡情報を保持したりする際に便利です。
- その表または他の表内にある、他のデータに対してチェックを行う。この機能は、参照保全制約が適切でない場合、または表チェック制約によってチェックが現在の表だけに制限されている場合に、データ保全性を確保するのに役立ちます。

- ユーザー定義関数でコーディングされている、非データベース操作を実行する。この機能は、アラートを出す場合や、そのデータベース以外の場所にある情報を更新する場合に役立ちます。

トリガーの使用には以下の利点があります。

アプリケーション開発に費やされる時間が短縮される
トリガーはデータベース内に保管され、すべてのアプリケーションに対して使用できるので、アプリケーションごとに同等の関数をコーディングする必要がなくなります。

内部規則のグローバル制約
トリガーをいったん定義すると、そのトリガーによって管理されるデータを使用するすべてのアプリケーションがそのトリガーを使用します。

保守が容易になる
何らかの変更を行う場合、トリガーを使用するアプリケーションごとにではなく、データベース内で一度変更を行うだけで済みます。

関連概念:

- 「*アプリケーション開発ガイド サーバー・アプリケーションのプログラミング*」の『アプリケーション開発でのトリガー』
- 「*アプリケーション開発ガイド サーバー・アプリケーションのプログラミング*」の『トリガー作成のガイドライン』

第 2 章 DB2 アプリケーションのコーディング

プログラミングの前提条件 29
DB2 アプリケーションのコーディングの概説 . . . 30
スタンドアロン・アプリケーションのプログラミング 30
スタンドアロン・アプリケーションの宣言セクションの作成 31
データベース・マネージャーと対話する変数の宣言 31
SQL オブジェクトを表す変数の宣言 32
db2dclgn 宣言生成プログラムを使用したホスト変数の宣言 34
ホスト変数と SQL ステートメントの関連付け. . 35
エラー処理のための SQLCA の宣言 36
WHENEVER ステートメントを用いたエラー処理 37
アプリケーションへの非実行ステートメントの追加 38
アプリケーションのデータベースへの接続 . . . 38
トランザクションのコーディング 39
COMMIT ステートメントによるトランザクションの終了 40
ROLLBACK ステートメントによるトランザクションの終了 41
アプリケーション・プログラムの終了 42
スタンドアロン・アプリケーションのトランザクションの暗黙的な終了. 43
疑似コードによるアプリケーションの枠組み . . 44
SQL ステートメントのプロトタイプのための機能 45
組み込み SQL または DB2 CLI プログラムにおける管理 API. 46
データ値とリレーションシップの制御 46
データ値の制御 46
データ・タイプを使用したデータ値の制御 . . . 47
ユニーク制約を使用したデータ値の制御 47
表チェック制約を使用したデータ値の制御 . . . 48
参照保全制約を使用したデータ値の制御 48
CHECK OPTION のあるビューを使用したデータ値の制御 48
アプリケーションのロジックとプログラム変数のタイプを使用したデータ値の制御 49
データ・リレーションシップの制御 49
参照保全制約を使用したデータ・リレーションシップの制御 49
トリガーを使用したデータ・リレーションシップの制御 50
BEFORE トリガーを使用したデータ・リレーションシップの制御 50
AFTER トリガーを使用したデータ・リレーションシップの制御 51
アプリケーション・ロジックを使用したデータ・リレーションシップの制御 51
サーバーにおけるアプリケーションのロジック . 51
SQL および API における許可に関する考慮事項. . 53
組み込み SQL における許可に関する考慮事項. . 53
動的 SQL における許可に関する考慮事項 . . . 54
静的 SQL における許可に関する考慮事項 . . . 55
API における許可に関する考慮事項 55
アプリケーションのテスト 56
アプリケーションのテスト環境のセットアップ . 56
テスト環境のセットアップ 56
テスト表とビューの作成 57
テスト・データの生成. 58
アプリケーションのデバッグと最適化 60

プログラミングの前提条件

アプリケーションを開発する前に、適切なオペレーティング環境を構築する必要があります。次のものが適切にインストールされ、構成されている必要があります。

- アプリケーションを開発するための、サポートされているコンパイラーまたはインタープリター
- DB2 Universal Database (ローカルでもリモートでも可)
- DB2 Application Development Client.

アプリケーションの開発は、DB2 Application Development Client をインストールしたサーバーまたは任意のクライアントで行うことができます。アプリケーションの実行は、サーバー、 DB2 Run-Time Client、DB2 Administrative Client のいずれかで行うことができます。 また、クライアントのインストール時に「Java 対応」コンポーネントをインストールした場合は、それらのクライアントのいずれかで Java™ JDBC プログラムを開発することもできます。 これは、それらのクライアント上で DB2 アプリケーションを実行できることを意味します。しかし、クライアントとと

もに DB2 Application Development Client もインストールしていなければ、クライアント上では JDBC アプリケーションしか開発できません。

DB2® は、プリコンパイラーを介して、C、C++、Java (SQLJ)、COBOL、および FORTRAN プログラミング言語をサポートします。さらに、動的に解釈される言語である Perl、 REXX、および Java (JDBC) もサポートしています。

注: FORTRAN および REXX のサポートは DB2 バージョン 5 において確立され、今後 FORTRAN または REXX のサポートを拡張する予定はありません。

DB2 には、サンプル・プログラムを実行する際に必要なサンプル・データベースが備えられています。

関連タスク:

- 「*アプリケーション開発ガイド アプリケーションの構築および実行*」の『アプリケーション開発環境のセットアップ』
- 「*アプリケーション開発ガイド アプリケーションの構築および実行*」の『サンプル・データベースのセットアップ』

DB2 アプリケーションのコーディングの概説

以下のセクションでは、DB2 アプリケーションのコーディングについて概説します。

スタンドアロン・アプリケーションのプログラミング

スタンドアロン・アプリケーションとは、実行時にストアード・プロシージャーなどのデータベース・オブジェクトを呼び出さないアプリケーションのことです。アプリケーションを作成するときには、特定の SQL ステートメントをプログラムの始めと終わりに使用し、ホスト言語から組み込み SQL への遷移を処理するようにしなければなりません。

手順:

スタンドアロン・アプリケーションをプログラミングするには、以下のことを行わなければなりません。

1. 宣言セクションを作成する。
2. データベースへ接続する。
3. 1 つ以上のトランザクションを書き込む。
4. それぞれのトランザクションは、以下のいずれかのメソッドを使用して終了しなければなりません。
   - アプリケーションによりデータベースに行われた変更をコミットする。
   - アプリケーションによりデータベースに行われた変更をロールバックする。
5. プログラムを終了する。

関連概念:

- 29 ページの『プログラミングの前提条件』
- 44 ページの『疑似コードによるアプリケーションの枠組み』

- 45 ページの『SQL ステートメントのプロトタイプのための機能』
- 「*アプリケーション開発ガイド アプリケーションの構築および実行*」の『サンプル・ファイル』

関連タスク:

- 31 ページの『スタンドアロン・アプリケーションの宣言セクションの作成』
- 38 ページの『アプリケーションのデータベースへの接続』
- 39 ページの『トランザクションのコーディング』
- 40 ページの『COMMIT ステートメントによるトランザクションの終了』
- 41 ページの『ROLLBACK ステートメントによるトランザクションの終了』
- 42 ページの『アプリケーション・プログラムの終了』
- 56 ページの『テスト環境のセットアップ』

スタンドアロン・アプリケーションの宣言セクションの作成

プログラムの先頭には、以下のものを含む宣言セクションが必要です。

- データベース・マネージャーがホスト・プログラムと対話するのに使用するすべての変数とデータ構造の宣言。
- SQL 連絡域 (SQLCA) を設定してエラーを処理する SQL ステートメント。

  Java で作成された DB2 アプリケーションは `SQLException` を throw します。これは、SQLCA を使用するのではなく、`catch` ブロックで処理してください。

1 つのプログラムに複数の SQL 宣言セクションが含まれる場合もあります。

手順:

宣言セクションを作成するには、次の手順で行います。

1. セクションを開始するために SQL ステートメント BEGIN DECLARE SECTION を使用する。
2. 宣言をコーディングする
3. セクションを終了するために SQL ステートメント END DECLARE SECTION を使用する。

関連タスク:

- 31 ページの『データベース・マネージャーと対話する変数の宣言』
- 32 ページの『SQL オブジェクトを表す変数の宣言』
- 35 ページの『ホスト変数と SQL ステートメントの関連付け』
- 34 ページの『db2dclgn 宣言生成プログラムを使用したホスト変数の宣言』
- 36 ページの『エラー処理のための SQLCA の宣言』

データベース・マネージャーと対話する変数の宣言

データベース・マネージャーと対話する変数はすべて、 SQL 宣言セクションで宣言しなければなりません。

SQL 宣言セクションで宣言されたホスト・プログラム変数はホスト変数と呼ばれます。 ホスト変数は SQL ステートメント内のホスト変数参照で使用できます。ホスト変数タグを SQL ステートメントの構文図に使用します。

手順:

変数の宣言は、SQL 宣言セクションにコーディングしてください。C/C++ のホスト変数の例は次のようになります。

```
EXEC SQL BEGIN DECLARE SECTION;
  short     dept=38, age=26;
  double    salary;
  char      CH;
  char      name1[9], NAME2[9];
  /* C comment */
  short     nul_ind;
EXEC SQL END DECLARE SECTION;
```

各ホスト変数の属性は、その変数が SQL ステートメントにおいてどのように使用されるかにより異なります。たとえば、DB2 表からデータを受け取る変数、または DB2 表にデータを保管する変数には、アクセスする列と互換性のあるデータ・タイプ属性と長さ属性がなければなりません。各変数のデータ・タイプを決定するには、 DB2 データ・タイプをよく理解していなければなりません。

関連資料:

- 186 ページの『C および C++ においてサポートされている SQL データ・タイプ』
- 215 ページの『COBOL でサポートされている SQL データ・タイプ』
- 233 ページの『FORTRAN でサポートされている SQL データ・タイプ』
- 549 ページの『REXX でサポートされている SQL データ・タイプ』
- 403 ページの『Java、JDBC、および SQL のデータ・タイプ』

SQL オブジェクトを表す変数の宣言

SQL オブジェクトを表す変数を、アプリケーション・プログラムの SQL 宣言セクションに宣言してください。

手順:

アプリケーション・プログラムを書く言語に適切なフォーマットで変数をコーディングしてください。

変数をコーディングする際には、表、別名、ビュー、および相関の名前の長さは、最高で 128 バイトであることに注意してください。列名の長さは、最高で 30 バイトです。スキーマ名の長さは、最高で 30 バイトです。今後のリリースでは、列名および SQL オブジェクトの他の ID の長さが最高 128 バイトまで増える可能性があります。 SQL オブジェクトを表す変数を、128 バイトよりも短い長さで宣言する場合、 SQL オブジェクト ID の長さが今後増えると、アプリケーションの安定度に影響が及ぶ可能性があります。 たとえば、 C++ アプリケーションでスキーマ名を保持するために変数 `char[9]`*`schema_name`* を宣言する場合、そのアプリケーションは DB2 バージョン 6 で許可されているスキーマ名 (最大長が 8 バイト) に対しては正しく機能します。

```
char[9] schema_name; /* holds null-delimited schema name of up to 8 bytes;
 works for DB2 Version 6, but may truncate schema names in future releases */
```

しかし、データベースを、スキーマ名の最大長として 30 バイトを受け入れるバージョンの DB2 に移行する場合、そのアプリケーションはスキーマ名 LONGSCHEMA1 と LONGSCHEMA2 を区別することができません。データベース・マネージャーは LONGSCHE の限度である 8 バイトでスキーマ名を切り捨て、アプリケーション内のステートメントのうち、スキーマ名を区別しなければならないものはすべて失敗します。 アプリケーションを長期間使用するためには、次のように、スキーマ名変数を 128 バイトの長さで宣言してください。

```
char[129] schema_name; /* holds null-delimited schema name of up to 128 bytes
                          good for DB2 Version 7 and beyond */
```

アプリケーションの操作を将来的にさらに向上させるには、アプリケーションで SQL オブジェクト名を表す変数を 128 バイトの長さで宣言することを考慮してください。互換性を向上させることの利点と、変数名を長くすることで必要になるシステム・リソースとを、比較検討する必要があります。

C/C++ アプリケーションの場合、C マクロ展開を使用して SQL オブジェクト ID の長さを宣言することにより、宣言のコーディングを単純化しコードを分かりやすくすることができます。組み込みファイル sql.h は、SQL_MAX_IDENT が 128 になるように宣言しているため、 SQL_MAX_IDENT マクロを使用すれば、SQL オブジェクト ID を簡単に宣言することができます。 たとえば、以下のようにします。

```
#include <sql.h>
    char[SQL_MAX_IDENT+1] schema_name;
    char[SQL_MAX_IDENT+1] table_name;
    char[SQL_MAX_IDENT+1] employee_column;
    char[SQL_MAX_IDENT+1] manager_column;
```

関連概念:

- 158 ページの『C および C++ でのホスト変数』
- 162 ページの『C および C++ における固定および NULL 終了文字ホスト変数の構文』
- 171 ページの『C マクロ展開』
- 205 ページの『COBOL のホスト変数』
- 227 ページの『FORTRAN のホスト変数』
- 543 ページの『REXX のホスト変数』

関連資料:

- 160 ページの『C および C++ における数値ホスト変数の構文』
- 162 ページの『C または C++ における可変長文字ホスト変数の構文』
- 165 ページの『C および C++ における単純グラフィックおよび NULL 終了グラフィック書式のグラフィック宣言の構文』
- 166 ページの『C または C++ における VARGRAPHIC 構造書式のグラフィック宣言の構文』
- 167 ページの『C または C++ におけるラージ・オブジェクト (LOB) ホスト変数の構文』
- 169 ページの『C または C++ におけるラージ・オブジェクト (LOB) ロケーター・ホスト変数の構文』

- 170 ページの『C または C++ におけるファイル参照ホスト変数宣言の構文』
- 206 ページの『COBOL における数値ホスト変数の構文』
- 207 ページの『COBOL における固定長ホスト変数の構文』
- 209 ページの『COBOL における固定長グラフィック・ホスト変数の構文』
- 210 ページの『COBOL における LOB ホスト変数の構文』
- 211 ページの『COBOL における LOB ロケーター・ホスト変数の構文』
- 211 ページの『COBOL におけるファイル参照ホスト変数の構文』
- 229 ページの『FORTRAN における数値ホスト変数の構文』
- 229 ページの『FORTRAN における文字ホスト変数の構文』
- 231 ページの『FORTRAN におけるラージ・オブジェクト (LOB) ホスト変数の構文』
- 232 ページの『FORTRAN におけるラージ・オブジェクト (LOB) ロケーター・ホスト変数の構文』
- 232 ページの『FORTRAN におけるファイル参照ホスト変数の構文』
- 547 ページの『REXX における LOB ロケーター宣言の構文』
- 548 ページの『REXX における LOB ファイル参照宣言の構文』

db2dclgn 宣言生成プログラムを使用したホスト変数の宣言

宣言生成プログラムを使用して、データベース内の指定された表の宣言を生成することができます。これによって、アプリケーションに簡単に挿入できる組み込み SQL 宣言のソース・ファイルを作成します。 db2dclgn は、C/C++、 Java、 COBOL、 FORTRAN の各言語をサポートします。

手順:

宣言ファイルを生成するには、db2dclgn コマンドを次の形式で入力してください。

```
db2dclgn -d database-name -t table-name [options]
```

たとえば、SAMPLE データベース内の STAFF 表の宣言を C 言語で出力ファイル staff.h に生成するには、次のコマンドを入力します。

```
db2dclgn -d sample -t staff -l C
```

生成される staff.h ファイルには以下のものが含まれています。

```
struct
 {
   short id;
   struct
   {
     short length;
     char data[9];
   } name;
   short dept;
   char job[6];
   short years;
   double salary;
   double comm;
 } staff;
```

関連資料:

- 「コマンド・リファレンス」の『db2dclgn - 宣言生成プログラム・コマンド』

ホスト変数と SQL ステートメントの関連付け

ホスト変数を使用して、データベース・マネージャーからデータを受け取ったり、ホスト・プログラムからデータベース・マネージャーにデータを転送します。 データベース・マネージャーからデータを受け取るホスト変数は*出力ホスト変数*、ホスト・プログラムからデータベース・マネージャーにデータを転送するホスト変数は*入力ホスト変数* です。

次の SELECT INTO ステートメントを例にとります。

```
SELECT HIREDATE, EDLEVEL
       INTO :hdate, :lvl
  FROM EMPLOYEE
       WHERE EMPNO = :idno
```

このステートメントには、2 つの出力ホスト変数 hdate と lvl、 1 つの入力ホスト変数 idno が含まれています。データベース・マネージャーは、ホスト変数 idno に保管されているデータを使用して、 EMPLOYEE 表から検索する行の EMPNO を決めます。 データベース・マネージャーは検索基準を満たす行を見つけると、hdate と lvl はそれぞれ HIREDATE 列と EDLEVEL 列に保管されたデータを受け取ります。上記のステートメントは、EMPLOYEE 表の列を使用したホスト・プログラムとデータベース・マネージャー間の対話の例です。

手順:

列で使用するホスト変数の定義は、次のようにしてください。

1. その列の SQL データ・タイプを調べてください。データベース内に作成された表すべてに関する情報を含んだビューの集合である、システム・カタログを照会してこれを調べてください。
2. ホスト言語に基づいた適切な宣言をコーディングしてください。

   表の各列には、CREATE TABLE ステートメントで定義されたデータ・タイプが割り当てられます。このデータ・タイプをホスト言語データ・タイプに関連付ける必要があります。たとえば、INTEGER データ・タイプは 32 ビットの符号付き整数です。これは、各ホスト言語による以下のようなデータ記述項目に対応しています。

   C/C++:

   ```
   sqlint32 variable_name;
   ```

   Java:
   ```
   int variable_name;
   ```

   COBOL:

   ```
   01  variable-name  PICTURE S9(9) COMPUTATIONAL-5.
   ```

   FORTRAN:

   ```
   INTEGER*4 variable_name
   ```

宣言生成プログラム・ユーティリティー (db2dclgn) を使用して、データベース内の指定された表の適切な宣言を生成することもできます。

関連概念:

- 「*SQL リファレンス 第 1 巻*」の『カタログ・ビュー』

関連タスク:

- 31 ページの『データベース・マネージャーと対話する変数の宣言』
- 34 ページの『db2dclgn 宣言生成プログラムを使用したホスト変数の宣言』
- 31 ページの『スタンドアロン・アプリケーションの宣言セクションの作成』

関連資料:

- 186 ページの『C および C++ においてサポートされている SQL データ・タイプ』
- 215 ページの『COBOL でサポートされている SQL データ・タイプ』
- 233 ページの『FORTRAN でサポートされている SQL データ・タイプ』
- 549 ページの『REXX でサポートされている SQL データ・タイプ』
- 403 ページの『Java、JDBC、および SQL のデータ・タイプ』

エラー処理のための SQLCA の宣言

アプリケーション・プログラムで SQLCA を宣言して、データベース・マネージャーからアプリケーションに情報を戻すようにすることができます。プログラムをプリプロセスする際に、データベース・マネージャーは INCLUDE SQLCA ステートメントの代わりにホスト言語変数宣言を挿入します。システムは、警告標識、エラー・コード、および診断情報の変数を使用してプログラムと連絡します。

各 SQL ステートメントを実行すると、システムは SQLCODE および SQLSTATE の両方の戻りコードを戻します。 SQLCODE はステートメントの実行を要約した整数値で、 SQLSTATE は IBM のリレーショナル・データベース製品に共通のエラー・コードを示す文字フィールドです。 SQLSTATE は ISO/ANS SQL92 標準、および FIPS 127-2 標準にも準拠しています。

注: FIPS 127-2 とは、*Federal Information Processing Standards Publication 127-2 for Database Language SQL* のことです。 ISO/ANS SQL92 とは、 *American National Standard Database Language SQL X3.135-1992* および *International Standard ISO/IEC 9075:1992, Database Language SQL* を指します。

0 未満の SQLCODE は、エラーが発生してステートメントが処理されなかったことを示していることに注意してください。 1 以上の SQLCODE は、警告が出されたものの、ステートメントの処理は継続していることを示します。

C または C++ 言語で作成された DB2 アプリケーションの場合、アプリケーションが複数のソース・ファイルで構成されているなら、 SQLCA の多重定義を回避するため、 EXEC SQL INCLUDE SQLCA ステートメントを組み込むのはその中の 1 つのファイルだけにしてください。それ以外のソース・ファイルには、次の行を組み込みます。

```
#include "sqlca.h"
extern struct sqlca sqlca;
```

手順:

SQLCA を宣言するには、プログラムに INCLUDE SQLCA ステートメントを次のようにコーディングします。

- 各言語でのステートメントを以下に示します。 C または C++ アプリケーションの場合、

```
EXEC SQL INCLUDE SQLCA;
```

- Java アプリケーションの場合、明示的に SQLCA を使用することはしません。その代わりに、SQLException インスタンス・メソッドを使用して、 SQLSTATE および SQLCODE 値を入手します。
- COBOL アプリケーションの場合、

```
EXEC SQL INCLUDE SQLCA END-EXEC.
```

- FORTRAN アプリケーションの場合、

```
EXEC SQL INCLUDE SQLCA
```

ご使用のアプリケーションが、 ISO/ANS SQL92 または FIPS 127-2 標準に準拠する必要がある場合は、上記のステートメントや INCLUDE SQLCA ステートメントを使用しないでください。

関連概念:

- 37 ページの『WHENEVER ステートメントを用いたエラー処理』
- 192 ページの『C および C++ における SQLSTATE および SQLCODE 変数』
- 218 ページの『COBOL での SQLSTATE および SQLCODE 変数』
- 235 ページの『FORTRAN の SQLSTATE および SQLCODE 変数』
- 537 ページの『Perl の SQLSTATE および SQLCODE 変数』

関連タスク:

- 31 ページの『スタンドアロン・アプリケーションの宣言セクションの作成』

WHENEVER ステートメントを用いたエラー処理

WHENEVER ステートメントがあると、プリコンパイラーは、エラー、警告、または実行中に行が見つからないなどの状態が起こった場合にアプリケーションに指定のラベルまで進むように指示するソース・コードを生成します。 WHENEVER ステートメントは、別の WHENEVER ステートメントが状況を変更するまで、後続の実行可能 SQL ステートメントに影響を与えます。

WHENEVER ステートメントには以下の 3 つの基本形式があります。

```
EXEC SQL WHENEVER SQLERROR   action
EXEC SQL WHENEVER SQLWARNING action
EXEC SQL WHENEVER NOT FOUND  action
```

上記のステートメントについて説明します。

SQLERROR
SQLCODE < 0 である状態です。

SQLWARNING
SQLWARN(0) = W または SQLCODE > 0 ですが、 100 ではない状態です。

NOT FOUND
SQLCODE = 100 である状態です。

いずれの場合も、*action* は以下のどちらかになります。

CONTINUE
アプリケーションの次の命令を続行するよう指示します。

GO TO *label*
GO TO の後に指定されたラベルの直後のステートメントに進むように指示します。 (GO TO は、2 つの語とすることも、GOTO として 1 つの語とすることもできます。)

WHENEVER ステートメントを使用しない場合、実行中にエラー、警告、または例外状態が起こると、デフォルトのアクションとして処理が継続されることになります。

WHENEVER ステートメントは、影響を及ぼす SQL ステートメントの前に指定しなければなりません。そうしないと、プリコンパイラーは実行可能 SQL ステートメント用に余分のエラー処理コードを生成しなければならないことを認識しません。 3 つの基本形式はいつでも任意に組み合わせてアクティブにすることができます。これら 3 つの形式の宣言順序は重要ではありません。

無限ループ状態を避けるには、SQL ステートメントをハンドラー内部で実行する前に、 WHENEVER 処理が取り消されていることを確認してください。 WHENEVER SQLERROR CONTINUE ステートメントを使用すれば、 SQL ステートメントをハンドラー内部で実行することができます。

関連資料:

- 「*SQL リファレンス 第 2 巻*」の『WHENEVER ステートメント』

アプリケーションへの非実行ステートメントの追加

非実行 SQL ステートメントをアプリケーション・プログラムに組み込むことが必要な場合、普通、それらのステートメントはアプリケーションの宣言セクションに入れられます。非実行ステートメントの例としては、 INCLUDE、INCLUDE SQLDA、および DECLARE CURSOR ステートメントがあります。

手順:

非実行ステートメント INCLUDE をアプリケーションで使用したい場合、以下のようにしてください。

```
INCLUDE text-file-name
```

関連タスク:

- 31 ページの『スタンドアロン・アプリケーションの宣言セクションの作成』

アプリケーションのデータベースへの接続

実行可能な SQL ステートメントを実行するには、その前にターゲット・データベースへの接続を確立しておかなければなりません。 この接続により、プログラムを

実行しているユーザーの許可 ID、およびプログラムが稼働しているデータベース・サーバーの名前を識別します。 一般に、アプリケーション・プロセスが一度に接続できるデータベース・サーバーは 1 つだけです。このサーバーを*現行サーバー* といいます。しかし、マルチサイト更新環境内であれば、複数のデータベース・サーバーに接続することができます。この場合、ただ 1 つのサーバーだけが現行サーバーになります。

制約事項:

以下の制約事項が適用されます。

- 接続は、CONNECT RESET、CONNECT TO、または DISCONNECT ステートメントが発行されるまで設定されたままとなります。
- マルチサイト更新環境では、接続は DB2 RELEASE およびそれに続いて DB2 COMMIT が発行されるまで続きます。マルチサイト更新を使用している場合、CONNECT TO ステートメントでは接続は終了しません。

手順:

プログラムは、以下のいずれかの方法でデータベース・サーバーと接続を確立することができます。

- CONNECT ステートメントを使用して明示的に接続する
- デフォルトのデータベース・サーバーに接続して暗黙的に接続する
- Java アプリケーションの場合は、**接続**インスタンスを使用する

接続状況および CONNECT ステートメントの使用方法については、 CONNECT ステートメントを参照してください。 初期化時に、アプリケーション・リクエスターがデフォルトのデータベース・サーバーを確立します。暗黙接続が使用可能になっている場合は、初期化後に開始されるアプリケーション・プロセスにより、デフォルトのデータベース・サーバーへの接続が暗黙的に行われます。アプリケーション・プログラムが最初に実行する SQL ステートメントを CONNECT ステートメントとして使用することは、良い方法です。明示的な CONNECT により、デフォルトのデータベースに対して不用意に SQL ステートメントを実行しないようにすることができます。

関連概念:

- 688 ページの『マルチサイト更新』

関連資料:

- 「*SQL リファレンス 第 2 巻*」の『CONNECT (タイプ 1) ステートメント』
- 「*SQL リファレンス 第 2 巻*」の『CONNECT (タイプ 2) ステートメント』

トランザクションのコーディング

トランザクションとは、データベース・マネージャーが (可能な場合はホスト言語コードに割り込んで) 一括して処理する一連の SQL ステートメントです。 トランザクションの代わりに、同じ意味として*作業単位* という用語がよく使われます。

前提条件:

トランザクションを実行するデータベースとの接続が確立していなければなりません。

手順:

トランザクションのコーディングは、以下のようにします。

1. *実行可能* SQL ステートメントによりトランザクションを始める。

   データベースへの接続が確立した後、以下の 1 つ以上のことをプログラムで発行することができます。

   - データ操作ステートメント (たとえば SELECT ステートメントなど)
   - データ定義ステートメント (たとえば CREATE ステートメントなど)
   - データ制御ステートメント (たとえば、GRANT ステートメントなど)

   実行可能 SQL ステートメントは常にトランザクション内に現れます。トランザクションが終了した後でプログラムに実行可能 SQL ステートメントが含まれている場合は、新しいトランザクションを自動的に開始します。

   注: 以下の 6 つのステートメントは実行可能ステートメントではないため、トランザクションを開始しません。

   - BEGIN DECLARE SECTION
   - INCLUDE SQLCA
   - END DECLARE SECTION
   - INCLUDE SQLDA
   - DECLARE CURSOR
   - WHENEVER

2. トランザクションを、次にあげるいずれかの方法で終了します。
   - トランザクションを COMMIT する
   - トランザクションを ROLLBACK する

関連タスク:

- 40 ページの『COMMIT ステートメントによるトランザクションの終了』
- 41 ページの『ROLLBACK ステートメントによるトランザクションの終了』

COMMIT ステートメントによるトランザクションの終了

COMMIT ステートメントにより、現行トランザクションは終了し、トランザクションで行われたデータベースの変更が他のプロセスでも認識できるようになります。

手順:

アプリケーションの要件に応じて、できるだけ早く変更がコミットされるようにしてください。特に、端末からの入力を待機している間、コミットされていない変更を保留することがないようなプログラムを作成してください。それが原因で、データベース・リソースが長い間保留にされないようにするためです。これらのリソースを保留してしまうと、それらのリソースを必要とする他のアプリケーションが実行できなくなります。

すべてのトランザクションを明示的に終わらせてから、アプリケーション・プログラムを終了するようにしてください。

トランザクションを明示的に終わらせないと、 DB2 はトランザクションがペンディングになっていた間のすべての変更を、プログラムが正常に終了した時点で自動的にコミットします。ただし、Windows オペレーティング・システムではコミットは行われません。 Windows オペレーティング・システムの場合は、トランザクションを明示的にコミットしないなら、データベース・マネージャーがその変更を必ずロールバックするようにします。

DB2 は以下の条件のときに、変更をロールバックします。

- ログが満杯になったとき。
- データベース・マネージャーの処理を終了させるようなその他のシステム条件。

COMMIT ステートメントは、ホスト変数の内容には影響を与えません。

関連概念:

- 43 ページの『スタンドアロン・アプリケーションのトランザクションの暗黙的な終了』
- 114 ページの『戻りコード』
- 114 ページの『SQLCODE、SQLSTATE、および SQLWARN フィールドのエラー情報』

関連タスク:

- 42 ページの『アプリケーション・プログラムの終了』

関連資料:

- 「*SQL リファレンス 第 2 巻*」の『COMMIT ステートメント』

ROLLBACK ステートメントによるトランザクションの終了

トランザクション・レベルでのデータの一貫性を確保するために、データベース・マネージャーは、トランザクション内の操作がすべて 完了するか、まったく行われなかった かのいずれかの状態を保ちます。たとえば、ある口座から現金を引き出し、それを別の口座に入れるとします。この 2 つの更新が単一のトランザクションで行われる場合、これらの処理中にシステム障害が発生すると、システムの再起動時にデータベース・マネージャーは自動的にクラッシュ・リカバリーを行い、データはトランザクションが開始される前の状態にリストアされます。プログラム・エラーが発生した場合は、データベース・マネージャーがエラーになったステートメントによる変更をすべて元の状態にリストアします。特にロールバックを行わなければ、エラーになったステートメントの実行前にトランザクション内で行った作業をデータベース・マネージャーが取り消すことはありません。

手順:

トランザクションにより影響された変更点がデータベースにコミットされないようにするためには、ROLLBACK ステートメントを発行してトランザクションを終了させてください。 ROLLBACK ステートメントは、データベースをトランザクションが実行される前の状態に戻します。

注: Windows オペレーティング・システムの場合は、トランザクションを明示的にコミットしないなら、データベース・マネージャーがその変更を必ずロールバックするようにします。

エラーまたは警告があったために実行されることになったルーチン内で ROLLBACK ステートメントと SQL WHENEVER ステートメントを使用している場合は、 ROLLBACK ステートメントの前に WHENEVER SQLERROR CONTINUE および WHENEVER SQLWARNING CONTINUE を指定するようにしてください。このようにすれば、エラーまたは警告により ROLLBACK が失敗した場合のプログラム・ループを防げます。

重大エラーの場合には、ROLLBACK ステートメントを発行できないというメッセージが表示されます。クライアントとサーバー・アプリケーション間の通信の損失や、データベースの破損などの重大エラーが発生した場合は、 ROLLBACK ステートメントを発行しないでください。重大エラーが発生した後で発行できるステートメントは CONNECT ステートメントだけです。

ROLLBACK ステートメントは、ホスト変数の内容には影響を与えません。

単一のアプリケーション・プログラム内の 1 つまたは複数のトランザクションをコード化することができます。さらに単一のトランザクション内から複数のデータベースにアクセスすることが可能です。複数のデータベースにアクセスするトランザクションは、マルチサイト更新と呼ばれます。

関連概念:

- 43 ページの『スタンドアロン・アプリケーションのトランザクションの暗黙的な終了』
- 687 ページの『リモート作業単位』
- 688 ページの『マルチサイト更新』

関連資料:

- 「*SQL リファレンス 第 2 巻*」の『CONNECT (タイプ 1) ステートメント』
- 「*SQL リファレンス 第 2 巻*」の『CONNECT (タイプ 2) ステートメント』
- 「*SQL リファレンス 第 2 巻*」の『WHENEVER ステートメント』

アプリケーション・プログラムの終了

アプリケーション・プログラムを終了させ、プログラムが使用したリソースを終結処理してください。

手順:

プログラムを正しく終了させるには、以下のステップに従ってください。

1. COMMIT ステートメントまたは ROLLBACK ステートメントを明示的に発行して、現行トランザクション (処理中のものがある場合) を終了する。
2. CONNECT RESET ステートメントを使用して、データベース・サーバーへの接続を解放する。
3. プログラムが使用したリソースの終結処置を行う。たとえば、使用した一時記憶域またはデータ構造をすべて解放します。

注: プログラム終了時に、現行トランザクションがまだアクティブであると、 DB2 は暗黙にそのトランザクションを終了させます。トランザクションを暗黙に終了させる際の DB2 の動作はプラットフォーム固有なので、プログラム終了前に COMMIT か ROLLBACK ステートメントを発行して、すべてのトランザクションを明示的に終了するようにしてください。

関連概念:

- 43 ページの『スタンドアロン・アプリケーションのトランザクションの暗黙的な終了』

関連資料:

- 「*SQL リファレンス 第 2 巻*」の『CONNECT (タイプ 1) ステートメント』
- 「*SQL リファレンス 第 2 巻*」の『CONNECT (タイプ 2) ステートメント』

スタンドアロン・アプリケーションのトランザクションの暗黙的な終了

現行トランザクションを終了せずにプログラムを終わらせた場合、 DB2® は暗黙に現行トランザクションを終了します。 DB2 はアプリケーション終了時に、 COMMIT または ROLLBACK ステートメントのどちらかを発行することにより、現行トランザクションを暗黙に終了します。 DB2 が COMMIT と ROLLBACK のどちらを発行するかは、以下の要因によって決まります。

- アプリケーションが正常に終了したかどうか

  サポートされているほとんどのオペレーティング・システムでは、正常に終了した場合 DB2 はトランザクションを暗黙にコミットしますが、異常終了の場合はトランザクションを暗黙にロールバックします。プログラムが異常終了と見なすものでも、データベース・マネージャーは異常終了と見なさない場合があることに注意してください。たとえば、アプリケーションが予期しないエラーを察知したときに、アプリケーションを即座に終了するように、 `exit(-16)` をコーディングすることができます。データベース・マネージャーは、この状況を正常終了と見なして、トランザクションをコミットします。一方、データベース・マネージャーは、例外やセグメント化違反などを異常終了と見なします。

- DB2 サーバーが稼働しているプラットフォーム

  Windows® 32 ビット・オペレーティング・システムでは、アプリケーションが正常終了したか異常終了したかに関係なく、常に DB2 はトランザクションをロールバックします。トランザクションをコミットしたい場合には、明示的に COMMIT ステートメントを発行する必要があります。

- アプリケーションがマルチスレッド・データベース・アクセス用の DB2 コンテキスト API を使用しているかどうか

  アプリケーションでこれらのものを使用している場合には、アプリケーションの終了が正常か異常かに関係なく、 DB2 は暗黙にトランザクションをロールバックします。 COMMIT ステートメントを使用して、トランザクションを明示的にコミットしない限り、トランザクションはロールバックされます。

関連概念:

- 193 ページの『マルチスレッド・データベース・アクセスの目的』

関連タスク:

- 42 ページの『アプリケーション・プログラムの終了』

関連資料:

- 「*SQL リファレンス 第 2 巻*」の『COMMIT ステートメント』
- 「*SQL リファレンス 第 2 巻*」の『ROLLBACK ステートメント』

疑似コードによるアプリケーションの枠組み

以下の例は、DB2 アプリケーション・プログラムの一般的な枠組みを疑似コード形式で要約しています。もちろん、実際に使用するプログラムに合うようにこの枠組みを調整することが必要です。

```
プログラム開始
EXEC SQL BEGIN DECLARE SECTION                          |
  DECLARE USERID FIXED CHARACTER (8)                    |
  DECLARE PW FIXED CHARACTER (8)                        |
                                                        | アプリケーションの
  (その他のホスト変数宣言)                              | セットアップ
                                                        |
EXEC SQL END DECLARE SECTION                            |
EXEC SQL INCLUDE SQLCA                                  |
EXEC SQL WHENEVER SQLERROR GOTO ERRCHK                  |

  (プログラムのロジック)

EXEC SQL CONNECT TO database A USER :userid USING :pw   |
EXEC SQL SELECT ...                                     |
EXEC SQL INSERT ...                                     | 最初の
  (SQL ステートメントが続く)                            | 作業単位
EXEC SQL COMMIT                                         |

  (プログラムのロジックが続く)

EXEC SQL CONNECT TO database B USER :userid USING :pw   |
EXEC SQL SELECT ...                                     |
EXEC SQL DELETE ...                                     | 2 番目の
  (SQL ステートメントが続く)                            | 作業単位
EXEC SQL COMMIT                                         |

  (プログラムのロジックが続く)

EXEC SQL CONNECT TO database A                          |
EXEC SQL SELECT ...                                     |
EXEC SQL DELETE ...                                     | 3 番目の
  (SQL ステートメントが続く)                            | 作業単位
EXEC SQL COMMIT                                         |

  (プログラムのロジックが続く)

EXEC SQL CONNECT RESET                                  |
ERRCHK                                                  |
                                                        | アプリケーションの
  (SQLCA のエラー情報をチェックする)                    | 終結処置
                                                        |
プログラム終了
```

関連タスク:

- 30 ページの『スタンドアロン・アプリケーションのプログラミング』

SQL ステートメントのプロトタイプのための機能

アプリケーションを設計およびコード化するにつれて、データベース・マネージャー機能とユーティリティーを利用して、 SQL コード部分をプロトタイプ化したりパフォーマンスを向上することができます。たとえば、次の事柄が行えます。

- 完成したプログラムをコンパイルし、リンクする前に、コントロール・センターまたはコマンド行プロセッサー (CLP) を使用して多数の SQL ステートメントをテストする。

  これにより、データベースの表、索引、またはビューに保管された情報を定義および操作できます。情報を追加、削除、更新するだけでなく、表の内容からレポートを作成できます。組み込み SQL プログラム内のホスト変数を使用するために、最低限いくつかの SQL ステートメントの構文を変更しなければなりません。ホスト変数は、画面に出力されるデータを保管するために使用されます。さらに、組み込み SQL ステートメントの中には、環境に関連していないため、コマンド・センターまたは CLP によってサポートされないもの (BEGIN DECLARE SECTION など) もあります。

  さらに、コマンド行プロセッサー要求の入出力をリダイレクトできます。たとえば、コマンド行プロセッサー要求に入力を行う際に必要となる SQL ステートメントを含んだファイルを 1 つ以上作成できます。そうすれば、ステートメントを再入力する必要がなくなります。
- プログラムでの使用を計画している DELETE、INSERT、UPDATE、または SELECT ステートメントの概算コストを調べるために Explain 機能を使用する。Explain 機能は、対象ステートメントの構造および概算コストについての情報を、ユーザーが提供する表に配置します。この情報は、Visual Explain または `db2exfmt` ユーティリティーを使用して調べることができます。
- システム・カタログ・ビューを使用して、既存のデータベースについての情報を取り出しやすくする。ビューの基準になるシステム・カタログ表は、データベースが作成、変更、更新される通常操作の間、データベース・マネージャーによって作成され、維持されます。これらのビューには、付与された権限、列名、データ・タイプ、索引、パッケージへの依存性、参照制約、表名など、それぞれのデータベースについてのデータが含まれます。システム・カタログ・ビュー内のデータは、通常の SQL 照会機能により使用できます。

  SQL オプティマイザーが使用する統計情報を含むシステム・カタログ・ビューの中には、更新が可能なものもあります。これらのビューの列の中には、オプティマイザーに作用するか、または仮定データベースのパフォーマンスを調査できるように変更が可能なものもあります。この方法は、ユーザーの開発システムまたはテスト・システム上の実動システムのシミュレート、および照会方法の分析に使用できます。

関連概念:

- 「*SQL リファレンス 第 1 巻*」の『カタログ・ビュー』
- 「*管理ガイド: パフォーマンス*」の『カタログ統計の表』
- 「*管理ガイド: パフォーマンス*」の『モデル化および what-if の計画のカタログ統計』
- 「*管理ガイド: パフォーマンス*」の『手動でのカタログ統計更新の一般規則』

- 「*管理ガイド: パフォーマンス*」の『SQL Explain 機能』
- 3 ページの『アプリケーション開発用の DB2 Universal Database ツール』

関連資料:

- 751 ページの『付録 A. サポートされる SQL ステートメント』

組み込み SQL または DB2 CLI プログラムにおける管理 API

アプリケーションは API を使用して、 SQL ステートメントでは使用できないデータベース・マネージャー機能にアクセスすることができます。

DB2® API は、以下の目的のために使用できます。

- データベース・マネージャー環境を操作する。これには、データベースとノードのカタログ作成とアンカタログ、データベースとノード・ディレクトリーのスキャンが含まれます。データベースの作成、削除、および移行を行うための API も使用できます。
- データのエクスポートとインポート、およびデータベースの管理、バックアップ、保管の機能を提供する。
- データベース・マネージャーおよびデータベース構成パラメーターの値を変更する。
- クライアント / サーバー環境固有の操作を提供する。
- プリコンパイル済み SQL ステートメントのランタイム・インターフェースを提供する。この API は通常、プログラマーが直接呼び出すものではありません。その代わり、プリコンパイラーによる処理の後に、修正済みソース・ファイルに挿入されます。

データベース・マネージャーには、独自のプリコンパイラーを作成することを希望する言語ベンダーのための API や、アプリケーションの開発に有効なその他の API があります。

関連概念:

- 55 ページの『API における許可に関する考慮事項』

データ値とリレーションシップの制御

以下のセクションでは、データ値とデータ・リレーションシップの制御方法について説明します。

データ値の制御

データベースで許可する値を制御することによって妥当性検査とデータ保全性保護を行うことは、アプリケーション・ロジックの従来の機能の 1 つです。アプリケーションには、データが有効であるかどうかを確認するために、入力されたデータ値を特別にチェックするロジックがあります。 (たとえば、部門番号が有効な番号であるかどうかと、それが既存の部門の番号であるかをチェックします。) DB2® にはこれらの同一の機能をデータベース内から提供するための方法がいくつかあります。

関連概念:

- 47 ページの『データ・タイプを使用したデータ値の制御』
- 47 ページの『ユニーク制約を使用したデータ値の制御』
- 48 ページの『表チェック制約を使用したデータ値の制御』
- 48 ページの『参照保全制約を使用したデータ値の制御』
- 48 ページの『CHECK OPTION のあるビューを使用したデータ値の制御』
- 49 ページの『アプリケーションのロジックとプログラム変数のタイプを使用したデータ値の制御』

データ・タイプを使用したデータ値の制御

データベース内の各データ・エレメントは表の列に保管されており、各列はデータ・タイプを持つように定義されています。このデータ・タイプにより、列の値のタイプに制限が設けられます。たとえば、整数は、固定された範囲内の数値でなければなりません。 SQL ステートメント内で列を使用する際には、整数は文字ストリングと比較できないなどの、一定の動作に従わなければなりません。 DB2® には、特性および振る舞いを定義した組み込みデータ・タイプのセットが含まれています。 DB2 は、*ユーザー定義特殊タイプ* と呼ばれる、ユーザー固有のデータ・タイプの定義もサポートしています。ユーザー定義特殊タイプは組み込みタイプに基づいていますが、組み込みタイプのすべての振る舞いが自動的にサポートされるわけではありません。データ・タイプをバイナリー・ラージ・オブジェクト (BLOB) と同様に使用して、データ構造など関連する値の集合から成るデータを保管できます。

関連概念:

- 「*アプリケーション開発ガイド サーバー・アプリケーションのプログラミング*」の『ユーザー定義特殊タイプ』

ユニーク制約を使用したデータ値の制御

ユニーク制約により、表の中の 1 つまたは複数の列で同じ値が重複するのを避けられます。ユニーク制約では、ユニーク・キーと主キーがサポートされています。たとえば、DEPARTMENT 表の DEPTNO 列にユニーク制約を定義することにより、同じ部門番号が 2 つの部門で指定されないようにすることができます。

表の中のデータを使用するすべてのアプリケーションにユニーク規則を設ける必要がある場合には、ユニーク制約を使用してください。

関連タスク:

- 「*管理ガイド: インプリメンテーション*」の『ユニーク制約の定義』
- 「*管理ガイド: インプリメンテーション*」の『ユニーク制約の追加』

表チェック制約を使用したデータ値の制御

表チェック制約は、表の列で使用できる値について、データ・タイプによる制限よりもさらに詳しい制限を定義するために使用します。表チェック制約は、範囲のチェック、または同じ表中の同じ行にある他の値との関係に基づくチェックという形式で行うことができます。

データを使用するすべてのアプリケーションに対して規則を適用するには、表チェック制約を使用して表中で使用できるデータを制限してください。表チェック制約により、利用の幅が広がるとともに保守しやすくなります。

関連タスク:

- 「*管理ガイド: インプリメンテーション*」の『表チェック制約の定義』
- 「*管理ガイド: インプリメンテーション*」の『表チェック制約の追加』

参照保全制約を使用したデータ値の制御

参照保全 (RI) 制約は、データを使用するすべてのアプリケーションに対して、値ベースの関係を保持しなければならない場合に使用します。たとえば、RI 制約を使用して、 EMPLOYEE 表の DEPTNO 列の値を DEPARTMENT 表の値と一致させることができます。この制約は、挿入、更新、削除が行われないようにします。制約がない場合には DEPARTMENT に関する情報は失われる可能性があります。 RI 規則をデータベースで集中的に適用すると、一般に適用が確実になり保守しやすくなります。

関連概念:

- 「*SQL リファレンス 第 1 巻*」の『制約』
- 49 ページの『参照保全制約を使用したデータ・リレーションシップの制御』
- 764 ページの『IBM のリレーショナル・データベース・システム間での参照保全の相違点』

CHECK OPTION のあるビューを使用したデータ値の制御

アプリケーションの規則で表チェック制約として定義できないものがある場合や、使用するすべてのデータには当てはまらない規則がある場合には、アプリケーションのロジックへの規則の組み込みに代わる方法があります。データの条件を WHERE 文節または WITH CHECK OPTION 文節の一部として指定した、表のビューを作成することができます。このビュー定義は、アプリケーションに有効なデータ・セットに対して、検索できるデータを制限します。さらに、ビューが更新可能な場合は、 WITH CHECK OPTION 文節がアプリケーションに適用できる行の更新、挿入、および削除を制限します。

関連資料:

- 「*SQL リファレンス 第 2 巻*」の『CREATE VIEW ステートメント』

アプリケーションのロジックとプログラム変数のタイプを使用したデータ値の制御

あるプログラミング言語でアプリケーションのロジックを作成する際には、変数も宣言すると、データ値の制御に関して他のトピックで説明されているのと同じ制限をデータに課すことができます。さらに、データベースではなくアプリケーションで規則を適用させるコードを記述することもできます。以下のような場合に、アプリケーション・サーバーにロジックを組み込んでください。

- 規則が一般的に適用されるものではない。ただし、WITH CHECK OPTION を使用したビューの場合は除く。
- データベース内のデータの定義を制御できない。
- アプリケーションのロジックを使用する方が、規則を効果的に扱えると思われる。

たとえば、データが入力される順序について入力データのエラーを処理する必要があり、データベース内の操作の順序では正しく検査できない、という場合があります。

関連概念:

- 48 ページの『CHECK OPTION のあるビューを使用したデータ値の制御』

データ・リレーションシップの制御

アプリケーションのロジックの主な機能は、システム内の異なる論理エンティティー間のリレーションシップの管理です。たとえば、新しい部門を追加する場合には、新しいアカウント・コードを追加する必要があります。 DB2® は、データベース内の異なるオブジェクト間のリレーションシップを管理するために、参照保全制約とトリガーという 2 つの方法を備えています。

関連概念:

- 49 ページの『参照保全制約を使用したデータ・リレーションシップの制御』
- 50 ページの『トリガーを使用したデータ・リレーションシップの制御』
- 50 ページの『BEFORE トリガーを使用したデータ・リレーションシップの制御』
- 51 ページの『AFTER トリガーを使用したデータ・リレーションシップの制御』
- 51 ページの『アプリケーション・ロジックを使用したデータ・リレーションシップの制御』

参照保全制約を使用したデータ・リレーションシップの制御

データのリレーションシップの制御という点から考えた場合、参照保全 (RI) 制約を使用すると、複数の表のデータ間のリレーションシップを制御できるようになります。関連付けられた主キーに影響を与える操作の振る舞い (DELETE や UPDATE など) について定義するには、 CREATE TABLE または ALTER TABLE ステートメントを使用してください。

RI 制約は、1 つまたは複数の表に対してデータの規則を適用します。データを使用するアプリケーションすべてにその規則が当てはまる場合には、 RI 制約の使用により規則がデータベース内だけで制御できます。これにより、利用の幅が広がるとともに保守しやすくなります。

関連概念:

- 「*SQL リファレンス 第 1 巻*」の『制約』

関連タスク:

- 「*管理ガイド: インプリメンテーション*」の『参照制約の定義』

関連資料:

- 「*SQL リファレンス 第 2 巻*」の『ALTER TABLE ステートメント』
- 「*SQL リファレンス 第 2 巻*」の『CREATE TABLE ステートメント』

トリガーを使用したデータ・リレーションシップの制御

アプリケーションでも実行できるロジックをサポートするために、更新前または更新後にトリガーを使用することができます。トリガーがサポートする規則や操作が、データを使用するすべてのアプリケーションに当てはまる場合は、トリガーの使用により規則や操作がデータベースだけで行えるので、データベースは利用の幅が広がるとともに保守しやすくなります。

関連概念:

- 50 ページの『BEFORE トリガーを使用したデータ・リレーションシップの制御』
- 51 ページの『AFTER トリガーを使用したデータ・リレーションシップの制御』
- 26 ページの『DB2 トリガー』

関連タスク:

- 「*管理ガイド: インプリメンテーション*」の『トリガーの作成』
- 「*アプリケーション開発ガイド サーバー・アプリケーションのプログラミング*」の『トリガーの作成』

関連資料:

- 「*SQL リファレンス 第 2 巻*」の『CREATE TRIGGER ステートメント』

BEFORE トリガーを使用したデータ・リレーションシップの制御

更新または挿入の前に実行されるトリガーを使用すると、修正中または挿入中の値を、データベースが実際に修正される前に修正することができます。これは、アプリケーションからの入力 (ユーザーから見たデータ) を希望の内部データベース形式に変換するために使用できます。また、ユーザー定義の関数により他の非データベース操作を行う際にも、 *BEFORE トリガー* を使用することができます。

関連概念:

- 51 ページの『AFTER トリガーを使用したデータ・リレーションシップの制御』
- 26 ページの『DB2 トリガー』

関連タスク:

- 「*管理ガイド: インプリメンテーション*」の『トリガーの作成』
- 「*アプリケーション開発ガイド サーバー・アプリケーションのプログラミング*」の『トリガーの作成』

関連資料:

- 「*SQL リファレンス 第 2 巻*」の『CREATE TRIGGER ステートメント』

AFTER トリガーを使用したデータ・リレーションシップの制御

更新、挿入、削除の後に実行されるトリガーにはいくつかの使用法があります。

- 同じ表または別の表にあるデータを更新、挿入、削除できます。これは、データ間リレーションシップの保守、または監査証跡情報の保持に役立ちます。
- 表の中の残りのデータ、または別の表のデータの値をチェックできます。これは、RI 制約を利用できない場合や、表の中の他の行または別の表からデータが参照されているために、チェック制約が使用できない場合に役立ちます。
- ユーザー定義関数を使用して、非データベース操作を開始できます。これは、アラート発行やデータベース外部の情報の更新に役立ちます。

関連概念:

- 50 ページの『BEFORE トリガーを使用したデータ・リレーションシップの制御』
- 26 ページの『DB2 トリガー』

関連タスク:

- 「*管理ガイド: インプリメンテーション*」の『トリガーの作成』
- 「*アプリケーション開発ガイド サーバー・アプリケーションのプログラミング*」の『トリガーの作成』

関連資料:

- 「*SQL リファレンス 第 2 巻*」の『CREATE TRIGGER ステートメント』

アプリケーション・ロジックを使用したデータ・リレーションシップの制御

データベースではなくアプリケーションで規則を適用させたり、または関連操作を実行させるコードを作成することができます。これは、その規則がデータベースに一般的に適用されるものではない場合に行ってください。データベース内のデータ定義全体に対する制御ができない場合や、アプリケーションのロジックで規則や操作を取り扱う方がより効果的であると思われる場合にも、ロジックをアプリケーション内に置くことができます。

関連概念:

- 51 ページの『サーバーにおけるアプリケーションのロジック』

サーバーにおけるアプリケーションのロジック

DB2® が追加機能を提供しているアプリケーション設計の最後の面は、アプリケーションのロジックの一部をデータベース・サーバーで実行するというものです。 通

常、この設計はパフォーマンスを向上させるために使用しますが、共通機能をサポートするためにサーバー側でアプリケーションのロジックを実行することもできます。

これを使用して、以下のものを使用することができます。

- ストアード・プロシージャー

  ストアード・プロシージャーはアプリケーションのルーチンで、クライアントのアプリケーションのロジックから呼び出されますが、データベース・サーバーで実行されます。ストアード・プロシージャーを使用する最も一般的な理由は、データベースで集中処理を行い結果データを小さくするためです。 これにより、ストアード・プロシージャーの実行中のネットワーク間の通信量を大幅に減らすことができます。また、複数のアプリケーションで共通する操作において、ストアード・プロシージャーを使用することもできます。この方法では、すべてのアプリケーションが同じロジックを使用して操作を実行します。

- ユーザー定義関数

  ユーザー定義関数 (UDF) は、以下のものを戻す SQL ステートメント内での操作に使用するために作成することができます。

  – 単一のスカラー値 (スカラー関数)
  – 非 DB2 データ・ソースから取り出した表。たとえば、ASCII ファイルまたは Web ページなど (表関数)

  UDF は、データ値の変換、1 つ以上のデータ値に対する計算の実行、値の部分的な抽出 (ラージ・オブジェクトの部分的な抽出) などに役立ちます。

- トリガー

  トリガーを使用してユーザー定義関数を呼び出すことができます。これは、特定のステートメントが現れたとき、またはデータ値が変更されたときには必ず一定の非 SQL 操作が行われるようにしたいという場合に役立ちます。特定の状況において電子メールを出したり、ファイルにアラート・タイプの情報を書き込むなどの操作が例に示されています。

関連概念:

- 50 ページの『BEFORE トリガーを使用したデータ・リレーションシップの制御』
- 51 ページの『AFTER トリガーを使用したデータ・リレーションシップの制御』
- 「*管理ガイド: パフォーマンス*」の『ストアード・プロシージャーのガイドライン』
- 「*アプリケーション開発ガイド サーバー・アプリケーションのプログラミング*」の『参照制約とのトリガーの対話』
- 21 ページの『DB2 ストアード・プロシージャー』
- 22 ページの『DB2 ユーザー定義関数およびメソッド』
- 26 ページの『DB2 トリガー』

関連タスク:

- 「*管理ガイド: インプリメンテーション*」の『トリガーの作成』

- 「アプリケーション開発ガイド サーバー・アプリケーションのプログラミング」の『トリガーの作成』

関連資料:

- 「*SQL リファレンス 第 2 巻*」の『CREATE TRIGGER ステートメント』

SQL および API における許可に関する考慮事項

以下のセクションでは、組み込み SQL における汎用許可に関する考慮事項、および、静的 SQL、動的 SQL、API における許可に関する考慮事項について説明します。

組み込み SQL における許可に関する考慮事項

許可 を与えられたユーザーまたはグループは、データベースへの接続、表の作成、システムの管理などの一般的なタスクを実行できます。*特権* を付与されたユーザーやグループは、特定の方法で特定のデータベースにアクセスできます。 DB2® は、保管された情報を保護するために特権のセットを使用します。

大部分の SQL ステートメントは、ステートメントが使用するデータベース・オブジェクトに対する何らかのタイプの特権を必要とします。ほとんどの API 呼び出しは、通常データベース・オブジェクトに対するいかなる特権も必要としませんが、権限を持っていないと呼び出すことができないものが多くあります。 DB2 API を使用すると、アプリケーション・プログラムから DB2 の管理機能を実行できます。たとえば、データベース内にバインド・ファイルの必要ないパッケージを作成するには、 `sqlarbnd` (または REBIND) API を使用できます。

アプリケーションを設計する際に、ユーザーがアプリケーションを実行するために必要な特権を考慮する必要があります。ユーザーが必要とする特権は、以下によって決まります。

- アプリケーションが動的 SQL (JDBC および DB2 CLI を含む) と静的 SQL のどちらを使用するか。 ステートメントを発行するために必要な特権については、それぞれのステートメントの説明を参照してください。
- アプリケーションがどの API を使用するか。 API 呼び出しに必要な特権と権限については、それぞれの API の説明を参照してください。

STAFF 表に対する照会を行う必要のある 2 名のユーザー、PAYROLL と BUDGET があるとします。 PAYROLL は会社の従業員の給与支払いを管理しており、給与支払い小切手を出す際に、さまざまな SELECT ステートメントを発行する必要があります。 PAYROLL は各従業員の給与にアクセスできなければなりません。
BUDGET は、給与の支払いにいくら必要であるかの決定を管理しています。しかし、BUDGET が個々の従業員の給与を見ることはできないようにしておかなければなりません。

PAYROLL がさまざまな SELECT ステートメントを発行しているため、
PAYROLL 用に設計されたアプリケーションは、動的 SQL を使用することになるでしょう。 動的 SQL では、 PAYROLL には STAFF 表に対する SELECT 特権が必要になります。 PAYROLL は表に対するフルアクセス特権が必要なので、このことは問題になりません。

一方 BUDGET は、個々の従業員の給与にアクセスさせてはなりません。つまり、STAFF 表に対する SELECT 特権を BUDGET に付与してはならないということです。 BUDGET は STAFF 表全体の給与の合計にアクセスする必要があるので、SELECT SUM (SALARY) FROM STAFF を実行するための静的 SQL アプリケーションを作成してバインドし、このアプリケーションのパッケージに対する EXECUTE 特権を BUDGET に付与することができます。 このようにして、見てはならない情報を BUDGET に公開することなく、必要な情報を取得させることができるようになります。

関連概念:

- 54 ページの『動的 SQL における許可に関する考慮事項』
- 55 ページの『静的 SQL における許可に関する考慮事項』
- 55 ページの『API における許可に関する考慮事項』
- 「*管理ガイド: プランニング*」の『許可』

動的 SQL における許可に関する考慮事項

DYNAMICRULES RUN (デフォルト) でバインドされたパッケージで動的 SQL を使用するには、動的 SQL アプリケーションを実行するユーザーが、パッケージに対する EXECUTE 特権だけでなく、実行する各 SQL 要求の発行に必要な特権も持っていなければなりません。 ユーザーの許可 ID、ユーザーがメンバーとなっているグループ、または PUBLIC に対して特権を付与することができます。

DYNAMICRULES BIND オプションを指定してアプリケーションをバインドする場合、 DB2 は許可 ID とアプリケーション・パッケージを関連付けます。これにより、アプリケーションを実行するどのユーザーでも、その許可 ID に関連付けられた特権を継承させることができます。

プログラムに静的 SQL が含まれていなければ、アプリケーション (組み込み動的 SQL アプリケーションの場合) をバインドする場合に必要になるのはデータベースに対する BINDADD 権限だけです。この特権も、ユーザーの許可 ID、ユーザーがメンバーとなっているグループ、または PUBLIC に対して付与することができます。

パッケージがバインドを行ったり定義を行う場合、そのアプリケーションを実行するのに必要なものは実行するパッケージの EXECUTE 特権だけです。ランタイムに、バインドを行うパッケージのバインド・プログラムは、そのパッケージにより生成されるすべての動的ステートメントを実行するのに必要な特権を持っていなければなりません。それは、動的ステートメントに対するすべての許可検査がルーチンを実行する ID ではなくバインド・プログラムの ID を使用して行われるからです。同じように、定義を行うパッケージを持つルーチンを定義する場合も、定義パッケージにより生成されるすべての動的ステートメントを実行するのに必要なすべての特権を持っていなければなりません。 SYSADM または DBADM 権限を持っていてバインドを行うパッケージを作成する場合は、 OWNER BIND オプションを使って異なる許可 ID を指定して作成することを考慮してください。 OWNER BIND オプションを使用すると、動的 SQL ステートメントの SYSADM または DBADM 特権を、パッケージが自動的に継承しないようにすることができます。DYNAMICRULES および OWNER BIND オプションの詳細については、BIND コ

マンドを参照してください。パッケージの振る舞いの詳細については、動的 SQL に対して DYNAMICRULES が及ぼす影響に関する説明を参照してください。

関連概念:

- 53 ページの『組み込み SQL における許可に関する考慮事項』
- 55 ページの『静的 SQL における許可に関する考慮事項』
- 55 ページの『API における許可に関する考慮事項』
- 「*アプリケーション開発ガイド サーバー・アプリケーションのプログラミング*」の『SQL の入ったルーチンの許可およびバインド』

関連資料:

- 「*コマンド・リファレンス*」の『BIND コマンド』

静的 SQL における許可に関する考慮事項

静的 SQL は、アプリケーションを実行しているユーザーがパッケージに対する EXECUTE 特権を持っていれば使用できます。パッケージを構成するそれぞれのステートメントに対する特権は必要ありません。 EXECUTE 特権は、ユーザーの許可 ID、ユーザーがメンバーとなっているグループ、または PUBLIC に対して付与することができます。

アプリケーションをバインドするときに VALIDATE RUN オプションを指定しない場合は、アプリケーションのバインドに使用する許可 ID に、アプリケーション内にあるすべてのステートメントを実行できるだけの権限が必要です。 BIND 時に VALIDATE RUN を指定した場合は、このパッケージ内にある静的 SQL に関連してどのような許可障害が発生しても BIND は成功し、その中にあるステートメントの妥当性はランタイムに再び検査されます。アプリケーションをバインドするためには、BINDADD 権限が必要です。ステートメントを実行するのに必要な特権は、ユーザーの許可 ID または PUBLIC に付与するようにしてください。静的 SQL ステートメントをバインドする際には、グループ特権は使用しません。動的 SQL の場合と同様に、BINDADD はユーザーの許可 ID、ユーザーがメンバーとなっているグループ、または PUBLIC に対して付与することができます。

上述の静的 SQL の特性により、DB2® にある情報のアクセスを正確に制御できます。

関連概念:

- 53 ページの『組み込み SQL における許可に関する考慮事項』
- 54 ページの『動的 SQL における許可に関する考慮事項』
- 55 ページの『API における許可に関する考慮事項』

関連資料:

- 「*コマンド・リファレンス*」の『BIND コマンド』

API における許可に関する考慮事項

DB2® が提供する API の大部分は特権を使用する必要がありませんが、呼び出しに何らかの権限を必要とするものも多くあります。 特権を必要とする API の場合は、アプリケーションを実行するユーザーに特権を付与しなければなりません。特

権は、ユーザーの許可 ID、ユーザーがメンバーとなっているグループ、または PUBLIC に対して付与することもできます。各 API 呼び出しを発行するために必要な特権および権限については、それぞれの API の説明を参照してください。

ストアード・プロシージャーのインターフェースによりアクセスできる API もあります。特定の API がストアード・プロシージャーによりアクセスできるかどうかに関する情報については、その API の説明を参照してください。

関連概念:

- 53 ページの『組み込み SQL における許可に関する考慮事項』
- 54 ページの『動的 SQL における許可に関する考慮事項』
- 55 ページの『静的 SQL における許可に関する考慮事項』

アプリケーションのテスト

以下のセクションでは、テスト環境のセットアップ方法、およびアプリケーションのデバッグと最適化の仕方について説明します。

アプリケーションのテスト環境のセットアップ

以下のセクションでは、アプリケーションのテスト環境のセットアップ方法について説明します。

テスト環境のセットアップ

アプリケーションを検証するには、テスト環境をセットアップしなければなりません。たとえば、アプリケーションの SQL コードをテストするには、データベースが必要です。

手順:

テスト環境のセットアップは、以下のように行います。

1. テスト・データベースを作成する。

   テスト・データベースを作成するには、 CREATE DATABASE API を呼び出す小規模なサーバー・アプリケーションを作成するか、またはコマンド行プロセッサーを使用してください。
2. テスト表またはビューを作成する。

   アプリケーションが表やビューからデータの更新、挿入、または削除を行う場合は、テスト・データを使用してそのアプリケーションが正しく実行されるか検査しなければなりません。アプリケーションが表やビューからデータの取り出しのみを行う場合は、テストの際に実動レベルのデータを使用することを考慮してください。
3. 表のテスト・データを作成する。

   アプリケーションのテストに使用する入力データは、入力される可能性があるものすべてを表す有効なデータにするべきです。アプリケーションが入力データが

有効かどうかを検査する場合は、有効なデータと無効なデータの両方を含めるようにして、有効なデータは処理され、無効なデータにはフラグが付けられることを確認してください。

4. アプリケーションのデバッグと最適化を行う。

関連タスク:

- 57 ページの『テスト表とビューの作成』
- 58 ページの『テスト・データの生成』
- 60 ページの『アプリケーションのデバッグと最適化』

関連資料:

- 「*管理 API リファレンス*」の『sqlecrea - データベースの作成』
- 「*コマンド・リファレンス*」の『CREATE DATABASE コマンド』

テスト表とビューの作成

必要なテスト表またビューを設計するには、まずアプリケーションのデータ要求を分析してください。表を作成するには、スキーマで CREATETAB 権限と CREATEIN 特権が必要です。代替権限については、CREATE TABLE ステートメントを参照してください。

手順:

テスト表を作成するには、次の手順で行います。

1. アプリケーションがアクセスするデータをリストし、それぞれのデータ項目がどのようにアクセスされるかを記述してください。たとえば、開発中のアプリケーションが TEST.TEMPL、TEST.TDEPT、および TEST.TPROJ 表にアクセスするとします。アクセスのタイプは、次のように記録されることになります。

表 1. アプリケーション・データの説明

表またはビューの名	行の挿入	行の削除	列名	データ・タイプ	更新アクセス
TEST.TEMPL	いいえ	いいえ	EMPNO	CHAR(6)	はい
			LASTNAME	VARCHAR(15)	はい
			WORKDEPT	CHAR(3)	はい
			PHONENO	CHAR(4)	
			JOBCODE	DECIMAL(3)	
TEST.TDEPT	いいえ	いいえ	DEPTNO	CHAR(3)	
			MGRNO	CHAR(6)	
TEST.TPROJ	はい	はい	PROJNO	CHAR(6)	はい
			DEPTNO	CHAR(3)	はい
			RESPEMP	CHAR(6)	はい
			PRSTAFF	DECIMAL(5,2)	はい
			PRSTDATE	DECIMAL(6)	はい
			PRENDATE	DECIMAL(6)	

2. アプリケーション・データ・アクセスの記述が完成したら、そのアプリケーションに必要なテスト表およびビューを構成してください。

- アプリケーションが表またはビューのデータを修正する場合、テスト表を作成する。 CREATE TABLE SQL ステートメントを用いて以下のテスト表を作成します。
  - TEMPL
  - TPROJ
- アプリケーションが実動データベースのデータを修正しない場合にテスト・ビューを作成する。

  この例では、CREATE VIEW SQL ステートメントを用いて TDEPT 表のテスト・ビューを作成します。

3. 表のテスト・データを作成する。

アプリケーションと一緒にデータベース・スキーマを開発している場合は、テスト表の定義が開発プロセス中に繰り返し詳細化されていきます。 1 次アプリケーションは普通、表を作成することもそれにアクセスすることもできません。それはデータベース・マネージャーが実際にはない表やビューを参照するステートメントをバインドできないからです。時間をかけずに表の作成および変更を処理するには、表を作成するために別個のアプリケーションを開発することを考慮してください。コマンド行プロセッサー (CLP) を使用して対話的にテスト表を作成することもできます。

すべての手順が終わったら、この作業のために関連するトピックで説明されていることを行う必要があります。

関連タスク:

- 58 ページの『テスト・データの生成』

関連資料:

- 「*SQL リファレンス 第 2 巻*」の『CREATE TABLE ステートメント』

テスト・データの生成

テスト表を作成したら、データを入れてアプリケーションのデータ処理の検査を行えます。

手順:

データを表に挿入するには、以下の方法のいずれかを使用してください。

- INSERT...VALUES (SQL ステートメント) は、コマンドが発行されるたびに表に 1 行以上の行を挿入する。
- INSERT...SELECT は、既存表からデータを入手し (SELECT 文節に基づく)、そのデータを INSERT ステートメントで識別された表に挿入する。
- IMPORT または LOAD ユーティリティーは、定義されたソースから大量の新規データまたは既存データを挿入する。
- RESTORE ユーティリティーは、元のデータベースの BACKUP コピーを使用して、既存のデータベースの内容を同一のテスト・データベースにコピーする。
- DB2MOVE ユーティリティーは、複数のワークステーションにある複数の DB2 データベース間で多くの表を移動するためのものです。

ランダムに生成されたテスト・データを表に挿入するための技法は、次の SQL ステートメントに具体的に示されています。以下の CREATE TABLE ステートメントにあるように、 EMP 表には 4 つの列、ENO (従業員番号)、LASTNAME (名字)、HIREDATE (入社日付)、そして SALARY (従業員の給料) があるものとします。

```
CREATE TABLE EMP (ENO INTEGER, LASTNAME VARCHAR(30),
                  HIREDATE DATE, SALARY INTEGER);
```

この表で、従業員番号には 1 からある数値、たとえば 100 までを、残りの列にはランダム・データを挿入するとします。このことを実行するには、次の SQL ステートメントを使用します。

```
INSERT INTO EMP
-- 100 個のレコードを生成する。
WITH DT(ENO) AS (VALUES(1) UNION ALL
SELECT ENO+1 FROM DT WHERE ENO < 100 ) 1

-- 次に、DT 内に生成されたレコードを使用して、従業員レコードの
-- 他の列を作成します。
SELECT ENO, 2
    TRANSLATE(CHAR(INTEGER(RAND()*1000000)), 3
              CASE MOD(ENO,4) WHEN 0 THEN 'aeiou' || 'bcdfg'
                              WHEN 1 THEN 'aeiou' || 'hjklm'
                              WHEN 2 THEN 'aeiou' || 'npqrs'
                                     ELSE 'aeiou' || 'twxyz' END,
                                          '1234567890') AS LASTNAME,
    CURRENT DATE - (RAND()*10957) DAYS AS HIREDATE, 4
    INTEGER(10000+RAND()*200000) AS SALARY 5
  FROM DT;

SELECT * FROM EMP;
```

上記のステートメントについて説明します。

1. INSERT ステートメントの最初の部分では、再帰副照会を用いて従業員番号を生成し、最初の 100 人の従業員用の 100 個のレコードを生成します。各レコードには、従業員番号が入ります。従業員数を変更するには、100 ではない数値を入力してください。
2. SELECT ステートメントで LASTNAME 列を生成します。まず RAND 関数を使用して、最大 6 桁の長さのランダム整数を生成します。次に CHAR 関数を使用して、その整数を数字形式に変換します。
3. 数字を英字に変換するために、TRANSLATE 関数を使用して、 10 の異なる数字 (0 から 9) をそれぞれの英字に変換します。 11 個以上の英字があるので、ステートメントは 5 つの異なる方式から変換方式を選択します。その結果、発音できる十分なランダム母音を持つ名前が生成され、その母音はそれぞれの変換で含められます。
4. ステートメントは、ランダムな HIREDATE 値を生成します。 HIREDATE の値は、現在日付から 30 年前までに及んでいますが、現在日付から 0 と 10 957 間のランダム日数を減算して計算されます。 (10 957 は 30 年間の日数です。)
5. 最後に、ステートメントは SALARY をランダムに生成します。給与の最小値は 10 000 で、この値に 0 ～ 200 000 のランダムな数値が加えられます。

開発中のすべてのユーザー定義関数 (UDF) を、テスト・データのプロトタイプとしたい場合もあります。

関連概念:

- 「*データ移動ユーティリティー ガイドおよびリファレンス*」の『インポートの概要』
- 「*データ移動ユーティリティー ガイドおよびリファレンス*」の『ロードの概要』
- 22 ページの『DB2 ユーザー定義関数およびメソッド』

関連タスク:

- 60 ページの『アプリケーションのデバッグと最適化』

関連資料:

- 「*SQL リファレンス 第 1 巻*」の『INSERT スカラー関数』
- 「コマンド・リファレンス」の『RESTORE DATABASE コマンド』

アプリケーションのデバッグと最適化

開発中にも、アプリケーションのデバッグと最適化を行えます。

手順:

アプリケーションのデバッグと最適化を行うには、以下のようにしてください。

- SQL ステートメントをプロトタイプ化する。コマンド行プロセッサー、Explain 機能の使用、プログラムが操作する表およびデータベースに関する情報のシステム・カタログ・ビューの分析、および実動条件をシミュレートするためのシステム・カタログ統計の更新を行うことができます。
- flagger を使用して、 DB2 Universal Database for z/OS and OS/390 用に開発されているアプリケーション内の SQL ステートメントの構文をチェックしたり、SQL92 基本レベル標準に準拠させる。この機能はプリコンパイル中に呼び出されます。
- エラー処理 API をすべて利用する。たとえば、エラー処理 API を使用して、テスト段階中のメッセージをすべて印刷できます。
- データベース・システム・モニターを使用して、分析のための最適化情報を取得する。

関連概念:

- 「*管理ガイド: パフォーマンス*」の『モデル化および what-if の計画のカタログ統計』
- 45 ページの『SQL ステートメントのプロトタイプのための機能』
- 「*管理ガイド: パフォーマンス*」の『データベース・システム・モニター情報』
- 71 ページの『組み込み SQL アプリケーション用のソース・ファイル要件』

第 2 部 組み込み SQL

第 3 章 組み込み SQL の概説

ホスト言語における組み込み SQL ステートメントの使用 63
ソース・ファイルの作成と準備 65
パッケージ、バインディング、および組み込み SQL 67
組み込み SQL 用のパッケージの作成. 67
組み込み SQL を含むソース・ファイルのプリコンパイル 69
組み込み SQL アプリケーション用のソース・ファイル要件 71
組み込み SQL を含むソース・ファイルのコンパイルとリンケージ 72
BIND コマンドを使用したパッケージの作成 . . 73
パッケージにバージョンを付ける 74
バインドされる動的 SQL における特殊レジスターの影響 75
パッケージ・スキーマ用 CURRENT PACKAGE PATH 特殊レジスター. 76
非修飾表名の解決 79
バインディング時のその他の考慮事項 80
実行据え置きバインドの利点 81
ファイル内容のバインド 81
アプリケーション、バインド・ファイル、およびパッケージの関係 82
プリコンパイラー生成タイム・スタンプ. . . . 82
パッケージの再バインド 84

ホスト言語における組み込み SQL ステートメントの使用

ホスト言語に組み込まれた SQL ステートメントを使用して、アプリケーションを作成することができます。 SQL ステートメントはデータベース・インターフェースを提供し、一方、ホスト言語はアプリケーションの実行に必要なサポートの残りの部分を提供します。

手順:

ホスト言語アプリケーションに SQL ステートメントを組み込む方法のガイドとして、以下の表の例を使用してください。この例では、更新が正常に行われたかどうかを判断するために、 SQLCA 構造の SQLCODE フィールドをアプリケーションがチェックします。

表 2. ホスト言語における組み込み SQL ステートメントの使用

言語	サンプル・ソース・コード

C/C++	`EXEC SQL UPDATE staff SET job = 'Clerk' WHERE job = 'Mgr';`
	`if ( SQLCODE < 0 )`
	`  printf( "Update Error:  SQLCODE = %ld ¥n", SQLCODE );`

Java (SQLJ)	`try {`
	`  #sql { UPDATE staff SET job = 'Clerk' WHERE job = 'Mgr' };`
	`}`
	`catch (SQLException e) {`
	`  println( "Update Error:  SQLCODE = " + e.getErrorCode() );`
	`}`

COBOL	`EXEC SQL UPDATE staff SET job = 'Clerk' WHERE job = 'Mgr' END_EXEC.`
	`IF SQLCODE LESS THAN 0`
	`  DISPLAY 'UPDATE ERROR:  SQLCODE = ', SQLCODE.`

FORTRAN	`EXEC SQL UPDATE staff SET job = 'Clerk' WHERE job = 'Mgr'`
	`if ( sqlcode .lt. 0 ) THEN`
	`  write(*,*) 'Update error:  sqlcode = ', sqlcode`

アプリケーションに組み込まれる SQL ステートメントは、ホスト言語固有のものではありません。データベース・マネージャーには、SQL 構文をホスト言語が処理できるように変換する機能があります。

- C、C++、COBOL または FORTRAN 言語では、 DB2 プリコンパイラーによって変換されます。 DB2 プリコンパイラーは、PREP コマンドで呼び出します。プリコンパイラーは、組み込み SQL ステートメントを DB2 ランタイム・サービス API 呼び出しに直接変換します。
- Java 言語では、SQLJ プログラムは SQLJ 文節を JDBC ステートメントに変換します。 SQLJ 変換プログラムは、**sqlj** コマンドで呼び出します。

プリコンパイラーはソース・ファイルを処理する際に、特に SQL ステートメントを探して処理し、非 SQL ホスト言語は無視します。 SQL ステートメントは特殊な区切り文字で囲まれているので、プリコンパイラーはこれを発見することができます。次の表にある例は、区切り文字とコメントを使用して、サポートされているコンパイル済みホスト言語で有効な組み込み SQL ステートメントを作成する方法を示しています。

表 3. ホスト言語における組み込み SQL ステートメントの使用

言語	サンプル・ソース・コード

C/C++	`/* Only C or C++ comments allowed here */`
	`EXEC SQL`
	`-- SQL comments or`
	`/* C comments or */`
	`// C++ comments allowed here`
	`DECLARE C1 CURSOR FOR sname;`
	`/* Only C or C++ comments allowed here */`

SQLJ	`/* Only Java comments allowed here */`
	`#sql c1 = {`
	`-- SQL comments or`
	`/* Java comments or */`
	`// Java comments allowed here`
	`SELECT name FROM employee`
	`};`
	`/* Only Java comments allowed here */`

COBOL	`* See COBOL documentation for comment rules`
	`* Only COBOL comments are allowed here`
	`EXEC SQL`
	`-- SQL comments or`
	`* full-line COBOL comments are allowed here`
	`DECLARE C1 CURSOR FOR sname END-EXEC.`
	`* Only COBOL comments are allowed here`

FORTRAN	`C Only FORTRAN comments are allowed here`
	`EXEC SQL`
	`+ -- SQL comments, and`
	`C full-line FORTRAN comment are allowed here`
	`+ DECLARE C1 CURSOR FOR sname`
	`I=7 ! End of line FORTRAN comments allowed here`
	`C Only FORTRAN comments are allowed here`

関連概念:

- 541 ページの『REXX アプリケーションでの組み込み SQL』
- 156 ページの『C および C++ での組み込み SQL ステートメント』
- 203 ページの『COBOL における組み込み SQL』

• 226 ページの『FORTRAN における組み込み SQL ステートメント』

ソース・ファイルの作成と準備

ソース・コードは、テキスト・エディターを使用して、ソース・ファイルと呼ばれる ASCII ファイル内に作成されます。ソース・ファイルには、コーディングに使用しているホスト言語に適した拡張子が付いていなければなりません。

注: すべてのプラットフォームがすべてのホスト言語をサポートするわけではありません。

ここでは、すでにソース・コードが記述されていることを前提として説明しています。

コンパイルされるホスト言語を使用してアプリケーションを記述した場合は、さらに追加の手順に従ってアプリケーションを作成する必要があります。プログラムのコンパイルとリンクに加えて、*プリコンパイル* および*バインド* を行わなくてはなりません。

簡単に言えば、プリコンパイルによって、組み込み SQL ステートメントをホスト・コンパイラーが処理できる DB2 ランタイム API 呼び出しに変換し、バインド・ファイルを作成します。バインド・ファイルには、アプリケーション・プログラム内の SQL ステートメントに関する情報が入ります。 `BIND` コマンドを実行すると、データベース内にパッケージ が作成されます。任意で、プリコンパイラーがプリコンパイル時にバインドを行うようにさせることができます。

バインドとは、バインド・ファイルからパッケージ を作成し、それをデータベースに保管する処理です。アプリケーションが複数のデータベースにアクセスする場合、パッケージはデータベースごとに作成する必要があります。

次の図は、上記のステップの順序とともに、一般的なコンパイルされる DB2 アプリケーションのさまざまなモジュールを示しています。

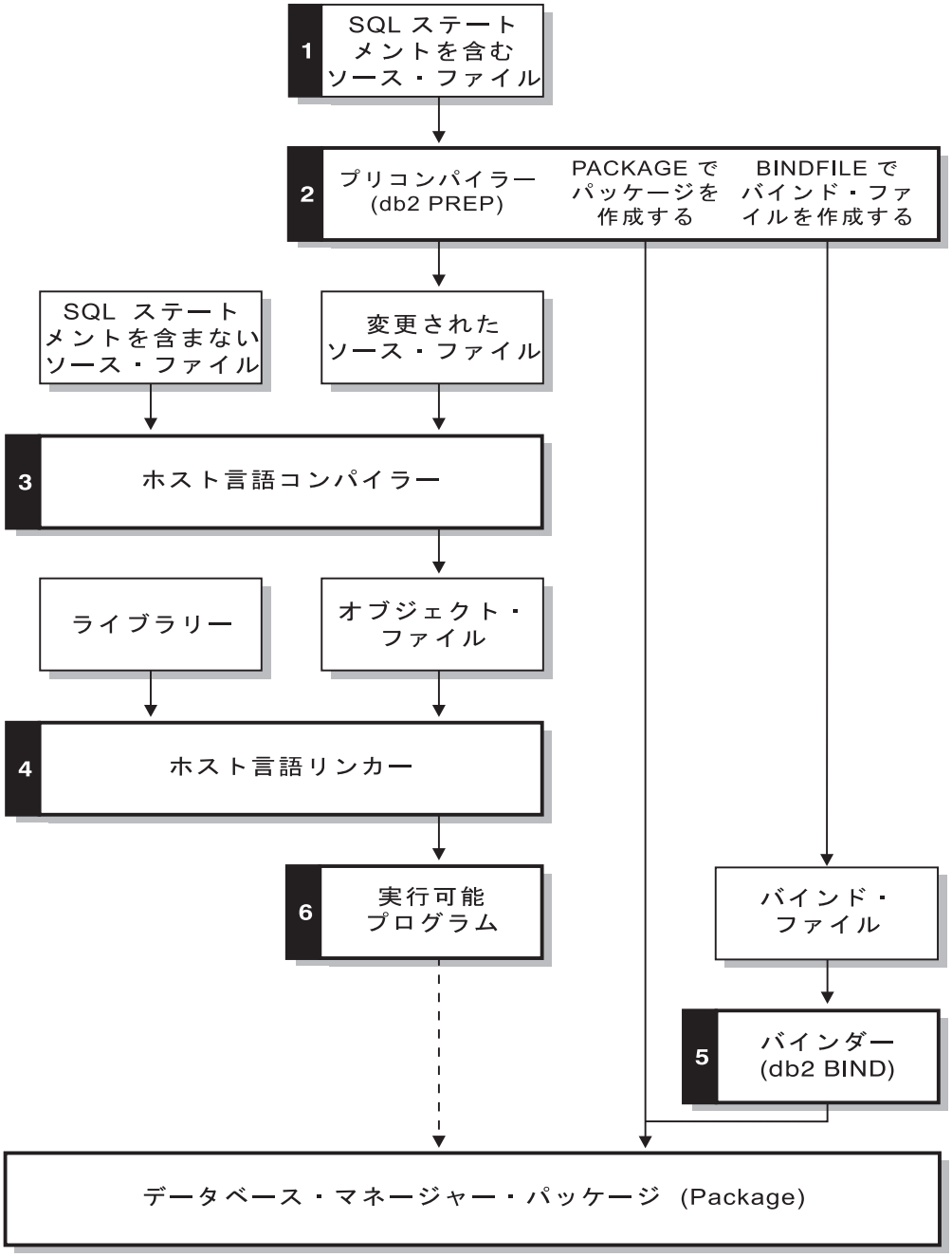


図 1. コンパイルされるホスト言語で記述されたプログラムの作成

関連概念:

- 69 ページの『組み込み SQL を含むソース・ファイルのプリコンパイル』
- 71 ページの『組み込み SQL アプリケーション用のソース・ファイル要件』
- 72 ページの『組み込み SQL を含むソース・ファイルのコンパイルとリンケージ』
- 8 ページの『組み込み SQL』

関連資料:

- 「コマンド・リファレンス」の『BIND コマンド』

パッケージ、バインディング、および組み込み SQL

以下のセクションでは、組み込み SQL アプリケーション用のパッケージの作成方法について説明し、さらに、据え置きバインディング、アプリケーション間、バインド・ファイル間、およびパッケージ間のリレーションシップについても説明します。

組み込み SQL 用のパッケージの作成

コンパイルされるホスト言語で記述されたアプリケーションを実行するには、データベース・マネージャーが実行時に必要とするパッケージを作成しなければなりません。これには、次の図で示されているようなステップが含まれます。

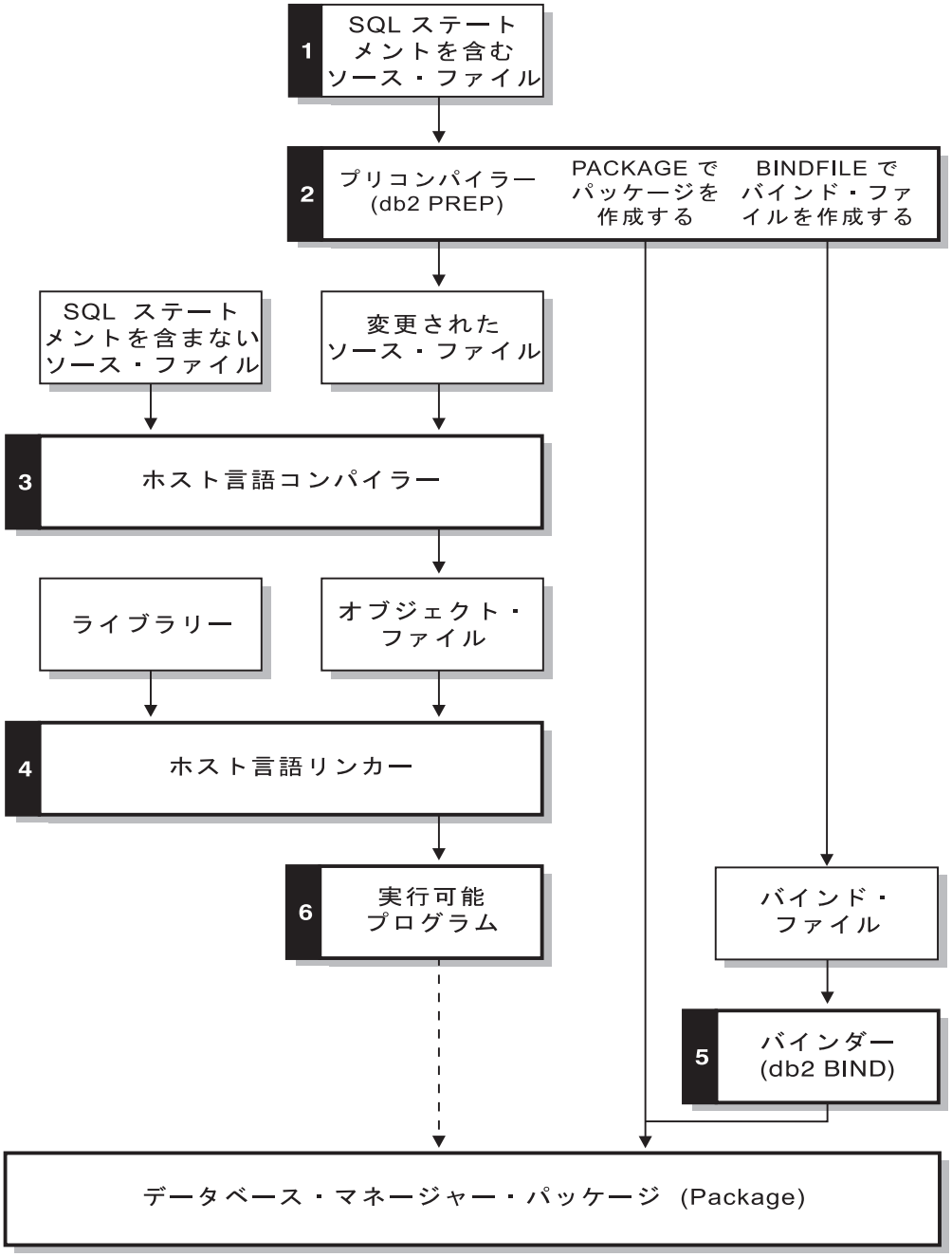


図 2. コンパイルされるホスト言語で記述されたプログラムの作成

- プリコンパイル (ステップ 2)。組み込み SQL のソース・ステートメントをデータベース・マネージャーが使用できる形式に変換する作業です。
- コンパイルとリンク (ステップ 3 および 4)。必要なオブジェクト・モジュールを作成する作業です。
- バインディング (ステップ 5)。プログラムの実行時にデータベース・マネージャーが使用するパッケージを作成する作業です。

関連概念:

- 69 ページの『組み込み SQL を含むソース・ファイルのプリコンパイル』
- 71 ページの『組み込み SQL アプリケーション用のソース・ファイル要件』

- 72 ページの『組み込み SQL を含むソース・ファイルのコンパイルとリンケージ』
- 73 ページの『BIND コマンドを使用したパッケージの作成』
- 74 ページの『パッケージにバージョンを付ける』
- 75 ページの『バインドされる動的 SQL における特殊レジスターの影響』
- 79 ページの『非修飾表名の解決』
- 80 ページの『バインディング時のその他の考慮事項』
- 81 ページの『実行据え置きバインドの利点』
- 82 ページの『アプリケーション、バインド・ファイル、およびパッケージの関係』
- 82 ページの『プリコンパイラー生成タイム・スタンプ』
- 84 ページの『パッケージの再バインド』

関連資料:

- 「コマンド・リファレンス」の『db2bfd - バインド・ファイル記述ツール・コマンド』

組み込み SQL を含むソース・ファイルのプリコンパイル

ソース・ファイルを作成してから、 SQL ステートメントが入っているそれぞれのホスト言語ファイルを、ホスト言語ソース・ファイル用の PREP コマンドを使ってプリコンパイルする必要があります。 プリコンパイラーはソース・ファイルに含まれている SQL ステートメントをコメントに変換し、そのステートメントについて DB2 ランタイム API 呼び出しを生成します。

アプリケーションをプリコンパイルする前に、明示的または暗黙に、サーバーに接続する必要があります。アプリケーション・プログラムをクライアント・ワークステーションでプリコンパイルして、プリコンパイラーが変更されたソースとメッセージをクライアント上で生成しても、プリコンパイラーはサーバー接続を使用していくらかの妥当性検査を実行します。

さらに、プリコンパイラーは、データベース・マネージャーがデータベースに対する SQL ステートメントを処理する上で必要な情報も作成します。この情報は、選択したプリコンパイラー・オプションによって、パッケージ、バインド・ファイル、またはその両方に保管されます。

プリコンパイラーの使用の一般的な例を以下に示します。 *filename.sqc* という C 組み込み SQL ソース・ファイルをプリコンパイルするために、以下のコマンドを発行して、デフォルト名 `filename.c` の C ソース・ファイルと、デフォルト名 `filename.bnd` のバインド・ファイルを作成することができます。

```
DB2® PREP filename.sqc BINDFILE
```

プリコンパイラーは、最大で次の 4 つのタイプの出力を生成します。

修正済みソース

このファイルは、プリコンパイラーが SQL ステートメントを DB2 ランタイム API 呼び出しに変換した後の、元のソース・ファイルの新バージョンです。適切なホスト言語拡張子が与えられます。

パッケージ

PACKAGE オプション (デフォルト) が使用されている場合、または BINDFILE、SYNTAX、SQLFLAG オプションのいずれも指定されていない場合は、パッケージは接続しているデータベースに保管されます。このパッケージには、このデータベースだけに対して特定のソース・ファイルの静的 SQL ステートメントを実行するために必要なすべての情報が入ります。 PACKAGE USING オプションを使って別の名前を指定した場合以外は、プリコンパイラーはパッケージ名を、ソース・ファイル名の最初の 8 文字から作成します。

PACKAGE オプションを SQLERROR CONTINUE を指定せずに使用する場合、プリコンパイル処理中に使用するデータベースには、ソース・ファイル内の静的 SQL が参照するデータベース・オブジェクトがすべて含まれていなければなりません。たとえば、SELECT ステートメントは、参照する表がデータベースに入っていない限り、プリコンパイルすることはできません。

VERSION を指定すると、バインド・ファイル (BINDFILE オプションが使用された場合) とパッケージ (PREP 実行時にバインドされるか個別にバインドされる場合) には特定のバージョン ID が付けられます。同じ名前と作成者を持つ複数のバージョンを同時に存在させることができます。

バインド・ファイル

BINDFILE オプションを使用すると、プリコンパイラーはパッケージの作成に必要なデータを含むバインド・ファイル (拡張子は .bnd) を作成します。このファイルは後に BIND コマンドとともに使用して、 1 つまたは複数のデータベースにバインドすることができます。 BINDFILE を指定していても PACKAGE オプションを指定していない場合、 BIND コマンドを呼び出さない限りバインドは行われません。 コマンド行プロセッサー (CLP) の場合は、 PREP のデフォルトは BINDFILE オプションを指定しません。したがって、CLP の使用中にバインドを延期したいときは、 BINDFILE オプションを指定する必要があります。

SQLERROR CONTINUE を指定すると、SQL ステートメントをバインドするときにエラーが発生してもパッケージが作成されます。VALIDATE RUN も指定されている場合、許可やファイルの存在の問題でバインドが失敗したそれらのステートメントを、実行時に追加バインドすることができます。ランタイムにそれらのステートメントを実行しようとするとエラーが発生します。

メッセージ・ファイル

MESSAGES オプションを使用している場合、プリコンパイラーはメッセージを指定のファイルに転送します。このメッセージには、警告およびプリコンパイル中に発生した問題を示しているエラー・メッセージが含まれます。 ソース・ファイルが正常にプリコンパイルされない場合は、警告およびエラー・メッセージを使用して問題を判別し、ソース・ファイルを訂正してから、再度プリコンパイルしてみてください。 MESSAGES オプションを使用していない場合は、プリコンパイルのメッセージは標準出力に書き込まれます。

関連概念:

- 74 ページの『パッケージにバージョンを付ける』

関連資料:

- 「コマンド・リファレンス」の『PRECOMPILE コマンド』

組み込み SQL アプリケーション用のソース・ファイル要件

ソース・ファイルは必ず、特定のデータベースに対してプリコンパイルしなければなりません。そのデータベースは、最終的にそのアプリケーションとともに使用されることのないデータベースであってもかまいません。実際に、テスト・データベースを開発用に使用することができます。そして、アプリケーションを十分にテストしてから、バインド・ファイルを 1 つ以上の実動データベースにバインドすることができます。これは、*据え置きバインディング* として知られています。

アプリケーションが使用しているコード・ページがデータベースのコード・ページと異なる場合、プリコンパイル時にどのコード・ページを使用するのかを考慮する必要があります。

アプリケーションでユーザー定義関数 (UDF) またはユーザー定義特殊タイプ (UDT) を使用している場合は、アプリケーションをコンパイルする際に FUNCPATH オプションを使用する必要があります。このオプションは、静的 SQL を含むアプリケーションで UDF と UDT を使用できるようにするための関数パスを指定します。 FUNCPATH を指定しない場合、デフォルトの関数パスは *SYSIBM*、*SYSFUN*、*USER* になります (*"USER"* は現行のユーザー ID のことです)。

複数のサーバーにアクセスするアプリケーション・プログラムをプリコンパイルするには、以下のいずれかを行ってください。

- データベースごとに SQL ステートメントを個別のソース・ファイルに分割する。異なるデータベースのための SQL ステートメントを同じファイルに混合しないでください。それぞれのソース・ファイルを、適切なデータベースに対してプリコンパイルすることができます。この方法で行うことをお勧めします。
- 動的 SQL のみを使用してアプリケーションをコーディングし、プログラムがアクセスするそれぞれのデータベースにバインドする。
- すべてのデータベースが同じであると思われる場合、つまり定義が同じである場合は、 SQL ステートメントを 1 つのソース・ファイルにグループ化することができる。

アプリケーションが DB2 Connect を介して AS/400® または iSeries アプリケーション・サーバーへアクセスするとしても、同じプロシージャーが適用されます。サーバーで使用可能な PREP オプションを使用して、接続する予定のサーバーに対してアプリケーションをプリコンパイルしてください。

DB2 Universal Database for z/OS and OS/390 で稼働するアプリケーションをプリコンパイルしている場合、 SQL ステートメントの構文のチェックに flagger を使用することを考慮してください。 flagger は、DB2 Universal Database でサポートされていても、 DB2 Universal Database for z/OS and OS/390 ではサポートされていない SQL 構文を示します。また flagger を使用して、作成した SQL 構文が SQL92 Entry Level 構文に従っているかもチェックできます。 PREP コマンドの SQLFLAG

オプションを使用して、標識機能を呼び出し、比較する DB2 Universal Database for z/OS and OS/390 の SQL 構文のバージョンを指定できます。 flagger を使用しても、必ずしも SQL の使用を変更する必要はありません。これは、構文の不適合に関する通知または警告メッセージを出すだけで、プリプロセスが異常終了することはありません。

関連概念:

- 81 ページの『実行据え置きバインドの利点』
- 666 ページの『異なるコード・ページ間での文字変換』
- 667 ページの『コード・ページ変換はいつ行われるか』
- 668 ページの『コード・ページ変換時の文字置換』
- 668 ページの『サポートされるコード・ページ変換』
- 669 ページの『コード・ページ変換の拡張係数』

関連資料:

- 「コマンド・リファレンス」の『PRECOMPILE コマンド』

組み込み SQL を含むソース・ファイルのコンパイルとリンケージ

修正済みソース・ファイルと、SQL ステートメントが含まれていない追加ソース・ファイルを、適切なホスト言語コンパイラーでコンパイルしてください。言語コンパイラーは、それぞれの修正済みソース・ファイルをオブジェクト・モジュール に変換します。

デフォルトのコンパイラー・オプションに対する例外については、ご使用のオペレーティング・プラットフォーム用のプログラミング資料を参照してください。使用可能なコンパイラー・オプションの完全な説明については、コンパイラーの資料を参照してください。

ホスト言語リンカーは実行可能アプリケーションを作成します。たとえば、次のようになります。

- Windows® オペレーティング・システム上では、アプリケーションは実行可能ファイルまたはダイナミック・リンク・ライブラリー (DLL) にすることができます。
- UNIX® ベースのシステム上では、アプリケーションは実行可能ロード・モジュールまたは共用ライブラリーにすることができます。

注: Windows オペレーティング・システム上ではアプリケーションを DLL にすることはできますが、 DLL は DB2® データベース・マネージャーではなく、アプリケーションにより直接ロードされます。 Windows オペレーティング・システム上で、データベース・マネージャーが DLL をロードすることも可能です。 ストアード・プロシージャーは、通常は DLL または共用ライブラリーとして作成されます。

実行可能ファイルを作成するには、以下のものにリンクします。

- ユーザー・オブジェクト・モジュール。修正済みソース・ファイル、および SQL ステートメントが入っていないその他のファイルから、言語コンパイラーによって生成されます。

- ホスト言語ライブラリー API。言語コンパイラーによって提供されます。
- データベース・マネージャー・ライブラリー。ご使用のオペレーティング環境用のデータベース・マネージャー API が入っています。使用するデータベース・マネージャー API に必要なデータベース・マネージャー・ライブラリーの特定の名前については、ご使用のオペレーティング・プラットフォーム用のプログラミング資料を参照してください。

関連概念:

- 21 ページの『DB2 ストアード・プロシージャー』

関連タスク:

- 551 ページの『REXX アプリケーションの構築と実行』
- 「*アプリケーション開発ガイド アプリケーションの構築および実行*」の『JDBC アプレットの作成』
- 「*アプリケーション開発ガイド アプリケーションの構築および実行*」の『JDBC アプリケーションの作成』
- 「*アプリケーション開発ガイド アプリケーションの構築および実行*」の『SQLJ アプレットの作成』
- 「*アプリケーション開発ガイド アプリケーションの構築および実行*」の『SQLJ アプリケーションの作成』
- 「*アプリケーション開発ガイド アプリケーションの構築および実行*」の『UNIX C アプリケーションの構築』
- 「*アプリケーション開発ガイド アプリケーションの構築および実行*」の『UNIX C++ アプリケーションの構築』
- 「*アプリケーション開発ガイド アプリケーションの構築および実行*」の『Building IBM COBOL applications on AIX』
- 「*アプリケーション開発ガイド アプリケーションの構築および実行*」の『UNIX Micro Focus COBOL アプリケーションの構築』

BIND コマンドを使用したパッケージの作成

バインドとは、データベース・マネージャーがアプリケーションの実行時にデータベースをアクセスするために必要とするパッケージを作成する処理です。バインドは、プリコンパイル中に PACKAGE オプションを指定して暗黙的に行うことも、プリコンパイル中に作成されたバインド・ファイルに対して BIND コマンドを使用することにより明示的に行うこともできます。

BIND コマンドの使用の一般的な例を以下に示します。 *filename.bnd* という名前のバインド・ファイルをデータベースにバインドするには、以下のコマンドを発行します。

```
DB2® BIND filename.bnd
```

個別にプリコンパイルされたソース・コード・モジュールごとに 1 つのパッケージが作成されます。アプリケーションに 5 つのソース・ファイルがあって、そのうちの 3 ファイルにプリコンパイルが必要な場合、 3 つのパッケージまたはバインド・ファイルが作成されます。デフォルトの解釈では、それぞれのパッケージには .bnd ファイルの作成元となったソース・モジュールの名前と同じ名前が付けられま

すが、8 文字で切り捨てられます。異なるパッケージ名を明示的に指定するには、PREP コマンドの PACKAGE USING オプションを使用しなければなりません。パッケージのバージョンは、VERSION プリコンパイル・オプションを指定すれば与えられますが、デフォルトでは空ストリングになっています。新しく作成されたパッケージの名前とスキーマが、ターゲット・データベースに現在存在しているパッケージの名前と同じであるもののバージョン ID が異なる場合、新規パッケージが作成されて既存パッケージに代わって新規のパッケージが使用されます。しかしながら、バインドされるパッケージの名前とスキーマとバージョンが同じパッケージが存在する場合、そのパッケージはドロップされバインドされる新規パッケージと置き換えられます (バインドに ACTION ADD が指定されていれば、こうしたことは起こらず、代わりにエラー (SQL0719) が戻されます)。

関連資料:

- 「コマンド・リファレンス」の『BIND コマンド』
- 「コマンド・リファレンス」の『PRECOMPILE コマンド』

パッケージにバージョンを付ける

アプリケーションの複数のバージョンを作成する必要がある場合は、PRECOMPILE コマンドの VERSION オプションを使用できます。このオプションにより、同じパッケージ名 (つまり、パッケージ名と作成者名) の複数のバージョンを同時に存在させることができます。たとえば、foo と呼ばれるアプリケーションがあり、foo.sqc からコンパイルされたとしましょう。パッケージ foo をプリコンパイルしてデータベースにバインドし、アプリケーションをユーザーに配布したとします。ユーザーはアプリケーションを実行させることができます。その後アプリケーションに変更を加え、foo.sqc を更新し、再コンパイル、バインディング、およびユーザーへのアプリケーションの配布を繰り返したとします。 foo.sqc の最初のプリコンパイルまたは 2 番目のプリコンパイルのどちらかで VERSION オプションを指定していなかったとすると、最初のパッケージは 2 番目のパッケージで置き換えられてしまいます。古いバージョンのアプリケーションを実行しようとすると、タイム・スタンプがミスマッチしていることを示す SQLCODE -818 を受け取ります。

タイム・スタンプのミスマッチのエラーを避け、同時に両方のバージョンのアプリケーションを実行できるようにするためには、パッケージにバージョンを付けるようにしてください。たとえば、foo の最初のバージョンの作成時に、次のように VERSION オプションを使用してプリコンパイルします。

```
DB2® PREP FOO.SQC VERSION V1.1
```

最初のバージョンが実行されているとします。foo の新しいバージョンの作成時に、次のコマンドを使用してプリコンパイルします。

```
DB2 PREP FOO.SQC VERSION V1.2
```

この時点で、アプリケーションの新規バージョンも実行できますが、最初のアプリケーションのインスタンスも実行されたままです。最初のパッケージのパッケージ・バージョンは V1.1 で 2 番目のパッケージ・バージョンは V1.2 なので、名前の衝突は発生しません。両方のパッケージがデータベースに存在し両方のバージョンのアプリケーションが使用できます。

PRECOMPILE または BIND コマンドで、PRECOMPILE コマンドの VERSION オプションといっしょに ACTION オプションを使用できます。ACTION オプションを使用して、異なるバージョンのパッケージを追加したり置き換える方法を制御できます。

パッケージ特権は、バージョン・レベルではきめ細かに制御できません。つまり、パッケージ特権の GRANT または REVOKE は、名前と作成者を共用するすべてのパッケージのバージョンに適用されるということです。それで、パッケージ `foo` の特権がバージョン V1.1 が作成された後ユーザーまたはグループに付与された場合、バージョン V1.2 が配布されるときにはユーザーまたはグループは同じ特権をバージョン V1.2 に対しても持つことになります。一般的に、同じユーザーおよびグループはパッケージのすべてのバージョンに対して同じ特権を持つので、この振る舞いは普通必要とされます。アプリケーションのすべてのバージョンが同じパッケージ特権になることを望まない場合は、アプリケーションにバージョンを付ける際に PRECOMPILE VERSION オプションを使用するべきではありません。その代わり、(更新済みソース・ファイルをリネームするか、または PACKAGE USING オプションを使用して明示的にパッケージをリネームして) 異なるパッケージ名を使用するようにすべきです。

関連概念:

- 82 ページの『プリコンパイラー生成タイム・スタンプ』

関連資料:

- 「コマンド・リファレンス」の『BIND コマンド』
- 「コマンド・リファレンス」の『PRECOMPILE コマンド』

バインドされる動的 SQL における特殊レジスターの影響

動的に作成されたステートメントについては、特殊レジスターの数によってステートメントのコンパイル環境が決定されます。

- CURRENT QUERY OPTIMIZATION 特殊レジスターは、使用される最適化クラスを決定します。
- CURRENT PATH 特殊レジスターは、 UDF および UDT の解決に使用される関数パスを決定します。
- CURRENT EXPLAIN SNAPSHOT レジスターは、 Explain スナップショット情報を収集するかどうかを決定します。
- CURRENT EXPLAIN MODE レジスターは、適格な動的 SQL ステートメントについての Explain 表情報を収集するかどうかを決定します。 これらの特殊レジスターのデフォルトは、関連する BIND オプションで使用されるデフォルトと同じです。

関連資料:

- 「*SQL リファレンス 第 1 巻*」の『CURRENT EXPLAIN MODE 特殊レジスター』
- 「*SQL リファレンス 第 1 巻*」の『CURRENT EXPLAIN SNAPSHOT 特殊レジスター』
- 「*SQL リファレンス 第 1 巻*」の『CURRENT PATH 特殊レジスター』

- 「*SQL リファレンス 第 1 巻*」の『CURRENT QUERY OPTIMIZATION 特殊レジスター』

パッケージ・スキーマ用 CURRENT PACKAGE PATH 特殊レジスター

パッケージ・スキーマは、パッケージを論理的にグループ化する手段を提供します。パッケージをスキーマにグループ化するための他の方法もあります。一部のインプリメンテーションでは、環境ごとに 1 つのスキーマを使用します (たとえば production と test スキーマ)。他のインプリメンテーションでは、ビジネス領域ごとに 1 つのスキーマを使用するか (たとえば stocktrd と onlinebnk スキーマ)、またはアプリケーションごとに 1 つのスキーマを使用します (たとえば stocktrdAddUser と onlinebnkAddUser)。一般の管理目的で、またはパッケージのバリエーションを提供するために (たとえば、アプリケーションのバックアップ・バリエーションの保守、またはアプリケーションの新規バリエーションのテスト)、パッケージをグループ化することもできます 。

複数のスキーマがパッケージに使用される場合、データベース・マネージャーは、どのスキーマからパッケージを探すのかを決定する必要があります。このタスクを実行するために、データベース・マネージャーは CURRENT PACKAGESET 特殊レジスターの値を使用します。この特殊レジスターを単一のスキーマ名に設定して、呼び出されるどのパッケージもそのスキーマに属していることを示すことができます。アプリケーションが異なるスキーマのパッケージを使用する場合、パッケージのスキーマが前のパッケージのスキーマと異なるなら、各パッケージが呼び出される前に SET CURRENT PACKAGESET ステートメントを発行する必要があります。

注: DB2® Universal Database for OS/390® and z/OS™ だけが CURRENT PACKAGESET 特殊レジスターを持っており、これによって値 (単一のスキーマ名) を、SET CURRENT PACKAGESET ステートメントに対応させて明示的に設定することができます。 DB2 Universal Database™ for Linux、UNIX®、および Windows® は SET CURRENT PACKAGESET ステートメントを持っていますが、CURRENT PACKAGESET 特殊レジスターは持っていません。これは、DB2 Universal Database Linux、UNIX、および Windows の他のコンテキスト (SELECT ステートメント内など) では CURRENT PACKAGESET を参照できないことを意味します。 DB2 Universal Database for AS/00 は CURRENT PACKAGESET のサポートを提供していません。

DB2 は、パッケージ解決中にスキーマのリストを考慮できるときは、さらに柔軟性が増します。スキーマのリストは、CURRENT PATH 特殊レジスターによって提供される SQL パスに似ています。スキーマ・リストは、ユーザー定義関数、プロシージャー、メソッド、および特殊タイプに使用されます。

注: SQL パスとは、非修飾関数、プロシージャー、メソッド、または特殊タイプ名のスキーマを決定する場合に、DB2 が考慮するスキーマ名のリストです。

パッケージの複数のバリエーション (パッケージ用の BIND オプションの複数セット) を、単一のコンパイル済みプログラムに関連付けたい場合には、SQL オブジェクトに使用されるスキーマのパスを、パッケージに使用されるスキーマのパスから分離することを考慮してください。

CURRENT PACKAGE PATH 特殊レジスターによって、パッケージ・スキーマのリストを指定することができます。他の DB2 ファミリー製品は、CURRENT PATH および CURRENT PACKAGESET などの特殊レジスターと同様の機能を提供します。それらは、ネストされたプロシージャーおよびユーザー定義関数に対して、アプリケーションを呼び出すランタイム環境を破壊することなく、プッシュまたはポップ されます。 CURRENT PACKAGE PATH 特殊レジスターは、パッケージ・スキーマ解決用に、この機能を提供しています。

多くのインストールでは、パッケージ用に複数のスキーマを使用します。パッケージ・スキーマのリストを指定しない場合、異なるスキーマからパッケージを要求するごとに、SET CURRENT PACKAGESET ステートメント (多くても 1 つのスキーマ名を含むことができる) を発行する必要があります。ただし、スキーマ名のリストを指定するためにアプリケーションの開始時点で SET CURRENT PACKAGE PATH ステートメントを発行する場合、異なるスキーマでパッケージが必要となるごとに、SET CURRENT PACKAGESET ステートメントを発行する必要はありません。この機能は、アプリケーションが SQLJ とJDBC を切り替えるごとに SET CURRENT PACKAGESET ステートメントを発行する必要なしにパッケージ・スキーマのリストを検索できるので、SQLJ アプリケーションを構築している場合は特に役立ちます。

たとえば、以下のパッケージが存在し、以下のリストを使用して、サーバー上に存在している最初のものを呼び出したいと想定します。
SCHEMA1.PKG1、SCHEMA2.PKG2、SCHEMA3.PKG3、SCHEMA.PKG、および SCHEMA5.PKG5。 DB2 Universal Database for Linux、UNIX、および Windows (単一のスキーマ名を受け入れる) での SET CURRENT PACKAGESET ステートメントに対する現行のサポートを想定すると、特定のスキーマを指定する各パッケージの呼び出しを試行する前に、SET CURRENT PACKAGESET ステートメントを発行する必要があります。たとえば、5 つの SET CURRENT PACKAGESET ステートメントが発行される必要があります。ただし、CURRENT PACKAGE PATH 特殊レジスターを使用すると、単一の SET ステートメントで十分です。たとえば、次のようにします。

```
SET CURRENT PACKAGE PATH = SCHEMA1, SCHEMA2, SCHEMA3, SCHEMA, SCHEMA5;
```

注: DB2 Universal Database for Linux、UNIX、Windows では、SQLE-CLIENT-INFO 構造内の SQLSetConnectAttr API を使用することによって、および組み込み SQL プログラム内の SET CURRENT PACKAGE PATH ステートメントを組み込むことによって、 db2cli.ini ファイル内に CURRENT PACKAGE PATH 特殊レジスターを設定することができます。 DB2 Universal Database for OS/390 and z/OS のみが、バージョン 8 またはそれ以降で、SET CURRENT PACKAGE PATH ステートメントをサポートしています。 DB2 Universal Database for Linux、UNIX、Windows サーバーまたは DB2 Universal Database for AS/00 に対してこのステートメントを発行した場合、-30005 が戻されます。

複数のスキーマを使用して、パッケージのいくつかのバリエーションを保守することができます。これらのバリエーションは、実稼働環境で加えられた変更を制御するために非常に役立ちます。パッケージの異なるバリエーションを使用して、パッケージのバックアップ・バージョン、またはパッケージのテスト・バージョン (たとえば、新規索引の影響を評価するための) を保持することもできます。パッケー

ジの前のバージョンは、バックアップ・アプリケーション (ロード・モジュールまたは実行可能ファイル) と同じ方法で使用され、特に前のバージョンに復帰する機能を備えることができます。

たとえば、PROD スキーマが実働アプリケーションによって使用される現行のパッケージを組み込み、BACKUP スキーマがそれらのパッケージのバックアップ・コピーを保管すると想定します。アプリケーションの新規バージョン (つまりパッケージ) は、 PROD スキーマを使用してバインディングすることによって、実動にプロモートされます。バックアップ・スキーマ (BACKUP) を使用するアプリケーションの現行バージョンをバインディングすることによって、パッケージのバックアップ・コピーが作成されます。それから、ランタイムに SET CURRENT PACKAGE PATH ステートメントを使用して、スキーマがパッケージについてチェックされる順序を指定することができます。 MYAPPL アプリケーションのバックアップ・コピーは、BACKUP スキーマを使用してバインドされ、現在実動中のアプリケーションの現行バージョンは、 PROD.MYAPPL パッケージを作成する PROD スキーマにバインドされていると想定します。使用可能な場合には PROD スキーマのパッケージのバリエーションを使用することを指定するには (不可能な場合には BACKUP スキーマのバリエーションが使用される)、特殊レジスターのために次の SET ステートメントを発行します。

```
SET CURRENT PACKAGE PATH = PROD, BACKUP;
```

パッケージの前のバージョンに復帰することが必要な場合、アプリケーションの実働バージョンは DROP PACKAGE ステートメントでドロップすることができます。これは代わりに、BACKUP スキーマを使用してバインドされたアプリケーション (ロード・モジュールまたは実行可能ファイル) の旧バージョンを呼び出します (各オペレーティング・システム・プラットフォームに固有の、アプリケーション・パス技法がここで使用できます)。

注: この例では、パッケージのバージョン間の相違は、パッケージを作成するために使用された BIND オプションだけであると想定しています (つまり、実行可能コードには相違はないということです)。

アプリケーションは、使用したいスキーマを選択するために SET CURRENT PACKAGESET ステートメントを使用しません。その代わりこれによって、DB2 は CURRENT PACKAGE PATH 特殊レジスターにリストされたスキーマからチェックして、パッケージを選ぶことができます。

注: DB2 Universal Database for OS/390 and z/OS のプリコンパイル・プロセスは、DBRM (LEVEL オプションを使用して設定できる) に整合性トークンを保管し、パッケージ解決中に、プログラム内の整合性トークンがパッケージと一致することを確認するためにチェックが行われます。同様に、DB2 Universal Database for Linux、UNIX、Windows のバインド・プロセスでは、バインド・ファイルにタイム・スタンプが保管されます。 DB2 Universal Database for Linux、UNIX、Windows は、LEVEL オプションもサポートしています。

異なるスキーマでパッケージの複数のバージョンを作成する別の理由は、さまざまな BIND オプションを有効にできるということです。たとえば、パッケージ内の非修飾名のリファレンスにはさまざまな修飾子を使用できます。

アプリケーションは、非修飾テーブル名で作成されることもよくあります。これは同一のテーブル名および構造を持つが、さまざまなインスタンスを識別するためのさまざまな修飾子を持つ、複数の表をサポートしています。たとえば、テスト・システムと実動システムはそれぞれの中で同一のオブジェクトを作成できますが、それらは異なる修飾子を持つことがあります (たとえば、PROD および TEST)。別の例は、異なる DB2 システム間のさまざまな表を水平にパーティション化してデータを入れるアプリケーションですが、それぞれは異なる修飾子を持っています (たとえば、EAST、WEST、NORTH、SOUTH や、COMPANYA、COMPANYB や、Y1999、Y2000、Y2001 など)。 DB2 Universal Database for OS/390 and z/OS では、BIND コマンドの QUALIFIER オプションを使用して表の修飾子を指定します。 QUALIFIER オプションを使用する場合には、複数のプログラムを保守する必要はなく、それぞれは非修飾表にアクセスする必要がある完全修飾名を指定します。その代わりに、アプリケーションからの SET CURRENT PACKAGESET ステートメントを発行し、単一のスキーマ名を指定することによって、訂正したパッケージに実行時にアクセスすることができます。ただし、SET CURRENT PACKAGESET を使用する場合、複数のアプリケーションは依然として保持されて変更される必要があります。要求されたパッケージにアクセスするために、それぞれのアプリケーションが独自の SET CURRENT PACKAGESET ステートメントを持ちます。代わりに SET CURRENT PACKAGE PATH ステートメントを発行する場合、すべてのスキーマをリストすることができます。実行時に、DB2 は訂正したパッケージを選択することができます。

注: DB2 Universal Database for Linux、UNIX、Windows は、 QUALIFIER BIND オプションもサポートしています。ただし、QUALIFIER BIND オプションは、BIND コマンドの DYNAMICRULES オプションを使用する静的 SQL またはパッケージだけに影響を与えます。

非修飾表名の解決

以下の方法の一つを使用すると、アプリケーション内で非修飾表名を処理することができます。

- 各ユーザーごとに、以下のコマンドを用いて異なる許可 ID からのさまざまな COLLECTION パラメーターでパッケージをバインドします。

```
CONNECT TO db_name USER user_name
BIND file_name COLLECTION schema_name
```

  上記の例では、*db_name* はデータベース名、*user_name* はユーザー名、そして *file_name* はバインドされるアプリケーション名を表しています。 *user_name* と *schema_name* は普通は同じ値です。その後、SET CURRENT PACKAGESET ステートメントを使用して、使用するパッケージ、すなわち使用する修飾子を指定します。そのデフォルト修飾子が、パッケージをバインドするときに使用する許可 ID になります。
- 各ユーザーに非修飾表名と同じ名前のビューを作成して、非修飾表名が正しく解決されるようにします。
- ユーザーごとに一つの別名を作成し、希望する表を指すようにします。

関連資料:

- 「*SQL リファレンス 第 2 巻*」の『SET CURRENT PACKAGESET ステートメント』

- 「コマンド・リファレンス」の『BIND コマンド』

バインディング時のその他の考慮事項

アプリケーションのコード・ページがデータベースのコード・ページと異なる場合、バインド時にどちらのコード・ページを使用するかを考慮する必要があります。

アプリケーションが IMPORT または EXPORT のようなデータベース・マネージャー・ユーティリティー API に呼び出しを発行する場合、提供されるユーティリティー・バインド・ファイルをデータベースにバインドしておくことが必要です。

BIND オプションを使用して、バインド中に発生する一定の操作を制御することができます。

- QUERYOPT BIND オプションは、バインド時に特定の最適化クラスを利用します。
- EXPLSNAP BIND オプションは、 Explain 表内の適格な SQL ステートメント用の Explain スナップショット情報を保管します。
- FUNCPATH BIND オプションは、静的 SQL のユーザー定義特殊タイプおよびユーザー定義関数を正しく解決します。

バインド処理を開始しても応答がない場合、データベースに接続している他のアプリケーションが、必要なロックを保持している可能性があります。この場合には、どのアプリケーションもデータベースに接続されていないことを確認してください。接続されていることがわかったなら、サーバー上のすべてのアプリケーションを切り離して、バインド処理を継続します。

アプリケーションが DB2 Connect を使用してサーバーにアクセスする場合、そのサーバーで使用可能な BIND オプションを使用できます。

バインド・ファイルは、以前の DB2 Universal Database のバージョンとの下位互換性はありません。レベルが混合した環境では、 DB2® は最下位レベルのデータベース環境で使用できる機能しか使用できません。 たとえば、V8 クライアントが V7.2 サーバーと接続している場合、そのクライアントは V7.2 の機能しか使用できません。 バインド・ファイルはデータベースの機能を伝えるので、それらは混合レベルの制限に従います。

下位レベルのシステムで、上位レベルのバインド・ファイルを再バインドする必要がある場合は、以下のようにすることができます。

- 上位レベルのサーバーに接続するために下位レベルの DB2 Application Development Client を使用して、下位レベルの DB2 Universal Database 環境に搬出してバインドできるバインド・ファイルを作成します。
- 上位レベルの DB2 クライアントを下位レベルの実稼働環境で使用して、テスト環境で作成された上位レベルのバインド・ファイルをバインドします。上位レベルのクライアントは、下位レベルのサーバーに適用されるオプションだけを渡します。

関連概念:

- 「*管理ガイド: インプリメンテーション*」の『データベースへのユーティリティーのバインド』
- 666 ページの『異なるコード・ページ間での文字変換』
- 668 ページの『コード・ページ変換時の文字置換』
- 669 ページの『コード・ページ変換の拡張係数』

関連資料:

- 「*コマンド・リファレンス*」の『BIND コマンド』

実行据え置きバインドの利点

バインドを実行可能にしてプリコンパイルを行うと、アプリケーションはプリコンパイル処理中に使用されたデータベースだけにアクセスできます。バインドを実行据え置きにしてプリコンパイルすると、 BIND ファイルを各データベースにバインドできるので、多数のデータベースにアクセスできます。このアプリケーション開発方法は、アプリケーションを 1 回だけプリコンパイルするという点で本来柔軟性のあるものですが、アプリケーションはデータベースにいつでもバインドすることができます。

実行中に BIND API を使用すると、アプリケーションはそれ自体をバインドすることができます。これはおそらくインストール手順の一部として、または関連モジュールが実行される前に行われます。たとえば、アプリケーションは複数のタスクを実行することができますが、そのうちで SQL ステートメントを使用する必要があるのは 1 つだけです。アプリケーションは、SQL ステートメントを必要とするタスクが呼び出されるとき、および関連パッケージが存在しない場合にだけ、それ自体をデータベースにバインドできるように設計することができます。

実行据え置きバインドのもう 1 つの利点は、エンド・ユーザーにソース・コードを提供しなくてもパッケージを作成できるという点です。関連したバインド・ファイルをアプリケーションと共に出荷することができます。

関連資料:

- 「*管理 API リファレンス*」の『sqlabndx - バインド』

ファイル内容のバインド

DB2® Bind File Description (`db2bfd`) ユーティリティーを使用することにより、バインド・ファイルを作成するのに使用するプリコンパイル・オプションを表示するだけでなく、バインド・ファイルの内容を簡単に表示して、ファイル内部の SQL ステートメントを調査および検証することができます。 これは、アプリケーションのバインド・ファイルに関連する問題の判別に役立ちます。

関連資料:

- 「*コマンド・リファレンス*」の『db2bfd - バインド・ファイル記述ツール・コマンド』

アプリケーション、バインド・ファイル、およびパッケージの関係

パッケージはデータベースに保管されたオブジェクトで、単一のソース・ファイル内の特定の SQL ステートメントを実行するために必要な情報を含んでいます。データベース・アプリケーションは、そのアプリケーションを作成するのに使用するプリコンパイルされたすべてのソース・ファイルに対して、 1 つのパッケージを用います。それぞれのパッケージは個別のエンティティーであり、同じアプリケーションまたは他のアプリケーションで使用される別のパッケージとは関係ありません。パッケージを作成するには、バインドを実行可能にしてソース・ファイルに対してプリコンパイラーを実行するか、 1 つまたは複数のバインド・ファイルで後からバインド・プログラムを実行します。

データベース・アプリケーションはパフォーマンスの向上とサイズを小さくするためにパッケージを使用しますが、これはアプリケーションをコンパイルする理由と同じものです。 SQL ステートメントをプリコンパイルすることによって、このステートメントはアプリケーションのランタイムではなく作成時にコンパイルされてパッケージとなります。それぞれのステートメントが構文解析され、さらに効率的に解釈されたオペランド・ストリングがパッケージに保管されます。ランタイムに、プリコンパイラーにより生成されるコードにより、入出力データに必要な変数情報を指定したランタイム・サービス・データベース・マネージャー API が呼び出され、パッケージに保管されている情報が実行されます。

プリコンパイルの利点は静的 SQL ステートメントにだけ当てはまります。動的に実行される SQL ステートメント (PREPARE と、EXECUTE か EXECUTE IMMEDIATE を用いる) はプリコンパイルされません。したがって、一連のステップの処理全体をランタイムに行わなければなりません。

注: SQL ステートメントの静的 SQL は、動的に処理される同じステートメントよりも自動的に速く実行されるとは考えないでください。動的ステートメントの準備にはオーバーヘッドが必要なため、静的 SQL の方が速く実行されます。それ以外の場合は、オプティマイザーがバインド時以前に使用可能なデータベース統計ではなく、現行のデータベース統計を使用できるので、動的に作成された同じステートメントのほうが速く実行されます。トランザクションの完了にかかる時間が 2 秒未満である場合は、一般に静的 SQL の方が速くなります。使用する方式を選択するために、両方のバインド方式をプロトタイプ化してください。

関連概念:

- 121 ページの『動的 SQL と静的 SQL』

プリコンパイラー生成タイム・スタンプ

パッケージまたはバインド・ファイルを生成すると、プリコンパイラーはタイム・スタンプを生成します。タイム・スタンプはバインド・ファイルまたはパッケージ、および修正済みソース・ファイルに保管されます。タイム・スタンプは、*整合性トークン* としても知られています。

バインドを実行可能にしてアプリケーションをプリコンパイルすると、タイム・スタンプが一致するパッケージと修正済みソース・ファイルが生成されます。複数の

バージョンのパッケージが (PRECOMPILE VERSION オプションを使用したために) 存在する場合、それぞれのバージョンが関連したタイム・スタンプを持ちます。アプリケーションが実行されるとき、パッケージ名、作成者およびタイム・スタンプがデータベース・マネージャーに送信され、管理しているパッケージの名前、作成者およびタイム・スタンプと一致するかどうかチェックされます。一致するものがない場合、次の 2 つの SQL エラー・コードのうちのどちらかがアプリケーションに戻されます。

- SQL0818N (タイム・スタンプの矛盾)。名前と作成者が一致する (しかし整合性トークンは一致しない) パッケージは 1 つしか見つからなかったものの、パッケージのバージョンが空ストリング (″″) だった場合、このエラーが戻されます。
- SQL0805N (パッケージが見つからない)。このエラーは、上記以外のすべてのエラーで戻されます。

アプリケーションをデータベースにバインドする場合、 *PREP* コマンドの *PACKAGE USING* オプションを使用してデフォルトを指定変更しない限り、アプリケーション名の最初の 8 文字がパッケージ名として使用されることを覚えておいてください。 PREP コマンドの VERSION オプションを指定しないかぎり、バージョン ID は空ストリング (″″) になります。つまり、同じ名前の 2 つのプログラムをバージョン ID を変えずにプリコンパイルしてバインドすると、 2 番目のプログラムが最初のパッケージを置き換えます。最初のプログラムを実行すると、そのプログラムでは修正済みソース・ファイルのタイム・スタンプとデータベースのパッケージのタイム・スタンプとが一致していないため、タイム・スタンプ・エラーまたはパッケージが見つからないというエラーになります。パッケージが見つからないというエラーは、以下の例のようにプリコンパイルまたはバインドの ACTION REPLACE REPLVER オプションを使用した場合にも発生します。

1. パッケージ SCHEMA1.PKG を VERSION VER1 を指定してプリコンパイルする。そして、関連したアプリケーション A1 を生成する。
2. パッケージ SCHEMA1.PKG を VERSION VER2 ACTION REPLACE REPLVER VER1 を指定してプリコンパイルおよびバインドする。そして、関連したアプリケーション A2 を生成する。

   2 番目のプリコンパイルとバインドにより VER2 という VERSION を持つパッケージ SCHEMA1.PKG が生成され、ACTION REPLACE REPLVER VER1 が指定されているので VER1 という VERSION を持っていた SCHEMA1.PKG パッケージは削除されます。

   最初のアプリケーションを実行しようとすると、パッケージのミスマッチが発生し失敗します。

同じような症状が次の例でも発生します。

1. パッケージ SCHEMA1.PKG を VERSION VER1 を指定してプリコンパイルする。そして、関連したアプリケーション A1 を生成する。
2. パッケージ SCHEMA1.PKG を VERSION VER2 を指定してプリコンパイルする。そして、関連したアプリケーション A2 を生成する。

   この時点では、アプリケーション A1 も A2 も実行可能であり、パッケージ SCHEMA1.PKG からバージョン VER1 と VER2 をそれぞれ実行します。たとえば、最初のパッケージが (DROP PACKAGE SCHEMA1.PKG VERSION VER1

SQL ステートメントを使用して) ドロップされている場合、アプリケーション A1 を実行しようとすると、パッケージが見つからないというエラーになります。

ソース・ファイルがプリコンパイルされているものの対応するパッケージが作成されていない場合、合致するタイム・スタンプを持ったバインド・ファイルと修正済みソースを生成します。アプリケーションを実行するためには、パッケージを作成するためにバインド・ファイルを個別に BIND ステップでバインドし、修正済みソース・ファイルをコンパイルしてリンクします。複数のソース・モジュールを必要とするアプリケーションでは、バインディング・プロセスをそれぞれのバインド・ファイルごとに行わなければなりません。

この実行据え置きバインディングのシナリオでは、バインド・ファイルにはプリコンパイル中に修正済みソース・ファイルに保管されたタイム・スタンプと同じものが入るため、アプリケーションとパッケージのタイム・スタンプは一致することになります。

関連概念:

- 73 ページの『BIND コマンドを使用したパッケージの作成』

パッケージの再バインド

再バインド は、以前にバインドされたアプリケーション・プログラムのパッケージを再作成する処理です。パッケージが無効、または作動不能と示されている場合は、再バインドしなければなりません。しかし、パッケージが有効であっても再バインドが必要な場合もあります。たとえば、新規に作成された索引を利用する場合、または RUNSTATS コマンドの実行後の更新統計を利用する場合です。

パッケージは、表、ビュー、別名、索引、トリガー、参照制約、および表チェック制約など、データベース・オブジェクトの一定のタイプに従属させることができます。パッケージがデータベース・オブジェクト (表、ビュー、トリガーなど) に従属している場合にそのオブジェクトがドロップされると、パッケージは*無効* な状態になります。ドロップされたオブジェクトが UDF である場合、パッケージは*作動不能* 状態になります。

無効なパッケージは、実行される際にデータベース・マネージャーによって暗黙に (つまり自動的に) 再バインドされます。作動不能パッケージは、 BIND コマンドまたは REBIND コマンドのいずれかを実行して、明示的に再バインドしなければなりません。暗黙の再バインドに失敗すると、予期しないエラーが生じる場合があることに気を付けてください。つまり、暗黙の再バインドのエラーは、実際にエラーのあるステートメントではなく実行中のステートメントに戻される場合もあります。作動不能パッケージを実行しようとすると、エラーが発生します。無効なパッケージをシステムで自動的に再バインドするのではなく、これらを明示的に再バインドすることができます。これにより、いつ再バインドを行うかを制御できるようになります。

パッケージを明示的にバインドするために使用するコマンドは、状況により異なります。 SQL ステートメントの数を変更するか、または変更された SQL ステートメントを含めるために修正されたプログラムのパッケージを再バインドするには、BIND コマンドを使用しなければなりません。パッケージが最初にバインドされた変

数から BIND オプションを変更する必要がある場合にも、 BIND コマンドを使用します。その他の場合には、 BIND コマンドと REBIND コマンドのいずれかを使用してください。パフォーマンスの面では BIND よりも REBIND の方が優れているので、特に BIND を使用する必要がないときは REBIND を使用してください。

同じパッケージ名の複数のバージョンが同時にカタログに存在する場合、1 回に 1 つのバージョンしか再バインドできません。

関連概念:

- 「*管理ガイド: インプリメンテーション*」の『オブジェクトを変更するときのステートメント従属関係』

関連資料:

- 「*コマンド・リファレンス*」の『BIND コマンド』
- 「*コマンド・リファレンス*」の『REBIND コマンド』

第 4 章 静的 SQL プログラムの作成

静的 SQL を使用する場合の特性とそれを使用する理由 87
静的 SQL の利点 88
静的 SQL プログラムの例 89
静的 SQL プログラムにおけるデータ検索 90
| 静的 SQL における REOPT の影響 91
静的 SQL プログラムにおけるホスト変数 91
静的 SQL におけるホスト変数 91
静的 SQL プログラムにおけるホスト変数の宣言 93
静的 SQL プログラムにおけるホスト変数の参照 94
静的 SQL プログラムにおける標識変数 94
静的 SQL プログラムにおける標識変数の組み込み 95
静的 SQL プログラムにおける標識変数のデータ・タイプ 97
静的 SQL プログラムにおける標識変数の例 . . 99
カーソルを用いた複数行の選択 100
カーソルを用いた複数行の選択 100
静的 SQL プログラムにおけるカーソルの宣言と使用 101
カーソル・タイプおよび作業単位に関する考慮事項 102
静的 SQL プログラムにおけるカーソルの例 . . 104
検索されたデータの取り扱い 105
静的 SQL プログラムにおける検索データの更新と削除. 105
カーソル・タイプ 106
静的 SQL プログラムにおける取り出しの例 . . 107
検索されたデータのスクロールと取り扱い. . . . 108
以前に検索されたデータのスクロール 108
データのコピーを保持する方法 109
データを 2 度検索する方法 109
最初の結果表と 2 番目の結果表の行の順序の違い 110
カーソルの位置を表の最後にする. 111
以前に検索されたデータの更新 112
静的 SQL プログラムにおける挿入、更新、および削除の例 112
診断情報 114
戻りコード 114
SQLCODE、SQLSTATE、および SQLWARN フィールドのエラー情報 114
SQLCA 構造体におけるトークンの切り捨て . . 115
例外、シグナル、および割り込みハンドラーについての考慮事項. 115
出口リスト・ルーチンに関する考慮事項 . . . 116
アプリケーション中でのエラー・メッセージの検索 116

静的 SQL を使用する場合の特性とそれを使用する理由

組み込み SQL ステートメントの構文がプリコンパイル時に完全に認識されている場合、そのステートメントは*静的* SQL と呼ばれます。静的 SQL は、ランタイムまで構文が認識されない*動的* SQL ステートメントとは対照的です。

注: 静的 SQL は、REXX などのインタープリター言語ではサポートされません。

SQL ステートメントの構造は、静的と考えられるステートメントとして完全に指定されていなければなりません。たとえば、ステートメントで参照される列または表の名前は、プリコンパイル時に完全に認識されている必要があります。ランタイムに指定できる唯一の情報は、ステートメントが参照するホスト変数の値だけです。ただし、データ・タイプなどのホスト変数情報は、プリコンパイルしなければなりません。

静的 SQL ステートメントを準備する際に、ステートメントの実行可能形式が作成され、データベースのパッケージに保管されます。実行可能形式はプリコンパイル時か、後のバインド時に構成することができます。どちらの場合も、実行*前*に 準備が行われます。アプリケーションのバインドを行うユーザーの権限が使用されて、データベース統計および構成パラメーター (アプリケーション実行時に最新ではない場合がある) に基づいて最適化が行われます。

静的 SQL の利点

静的 SQL を用いたプログラミングは、組み込み動的 SQL を用いたプログラミングよりも簡単です。静的 SQL ステートメントはホスト言語ソース・ファイルに簡単に組み込まれ、プリコンパイラーは必要に応じて、ホスト言語コンパイラーが処理できるデータベース・マネージャー・ランタイム・サービス API 呼び出しへの変換を処理します。

アプリケーションのバインドを行うユーザーの許可が使用されているため、エンド・ユーザーはパッケージでステートメントを実行するための直接的な特権は不要です。たとえば、表全体についての更新特権がないユーザーでも、表の一部を更新できるようにすることが可能です。これを行うには、特定の列または値の範囲だけを更新できるように静的 SQL ステートメントを制限します。

静的 SQL ステートメントには*持続性* があります。つまりこれは、ステートメントはパッケージが存在する限り続くという意味です。

動的 SQL ステートメントは、スペース管理の理由で無効にされるか解放されるまで、またはデータベースが遮断されるまでキャッシュされます。必要であれば、動的 SQL ステートメントはキャッシュされたステートメントが無効になった時点で、 DB2® SQL コンパイラーにより暗黙に再コンパイルされます。

持続性に関する静的 SQL の重要な利点は、静的ステートメントはある特定のデータベースが遮断した後でも存在するという点です。一方、動的 SQL ステートメントはデータベースが遮断すると失われてしまいます。さらに、静的 SQL はランタイムに DB2 SQL コンパイラーでコンパイルする必要はありませんが、動的 SQL はランタイムに明示的に (PREPARE ステートメントなどを使用して) コンパイルしなければなりません。 DB2 は動的 SQL ステートメントをキャッシュするため、動的ステートメントを頻繁に DB2 でコンパイルする必要はありませんが、アプリケーション実行時に少なくとも 1 回はコンパイルしなければなりません。

静的 SQL にはパフォーマンス上の利点もあります。簡単に言うと、SQL プログラムの実行時間が短いという点です。すなわち、ステートメントの実行可能形式を準備するオーバーヘッドは、ランタイムではなくプリコンパイル時に行われるため、静的 SQL ステートメントは動的に処理された同じステートメントよりも迅速に実行されます。

注: 静的 SQL のパフォーマンスは、アプリケーションが最後にバインドされた時のデータベースの統計によって決まります。一方、この統計が変化すると、対応する動的 SQL のパフォーマンスは非常に異なってくる可能性があります。たとえば、後でデータベースに索引を追加する場合、静的 SQL を用いたアプリケーションは、データベースにバインドし直さなければその索引を利用することはできません。 また、静的 SQL ステートメントでホスト変数を使用する場合、オプティマイザーは表の分散統計を活用することができません。

関連資料:

- 「*SQL リファレンス 第 2 巻*」の『EXECUTE ステートメント』

静的 SQL プログラムの例

このサンプル・プログラムでは、静的 SQL ステートメントおよび C/C++、Java™、COBOL 言語でのデータベース・マネージャー API 呼び出しの例を説明しています。

C/C++ および Java の例では、サンプル・データベースにある **org** 表から New York にある部門の部門名と部門番号を照会し、部門名と部門番号をホスト変数に代入しています。

COBOL の例では、サンプル・データベースにある **employee** 表から Johnson という姓の従業員の名前を照会し、ホスト変数に代入しています。

注: REXX 言語は静的 SQL をサポートしないため、サンプルはありません。

- C/C++ (**tbread**)

  ```
  SELECT deptnumb, deptname INTO :deptnumb, :deptname
  FROM org
  WHERE location = 'New York'
  ```

  この照会は、サンプルの `TbRowSubselect()` 関数にあります。詳しくは、以下の関連サンプルを参照してください。

- Java (TbRead.sqlj)

  ```
  #sql cur2 = {SELECT deptnumb, deptname
   FROM org
   WHERE location = 'New York'};

   // fetch the cursor
   #sql {FETCH :cur2 INTO :deptnumb, :deptname};
  ```

  この照会は、**TbRead.sqlj** サンプルの `rowSubselect()` 関数にあります。詳しくは、以下の関連サンプルを参照してください。

- COBOL (**static.sqb**)

  サンプル **static** には、静的 SQL ステートメントおよび COBOL 言語でのデータベース・マネージャー API 呼び出しの例があります。SELECT INTO ステートメントはデータベース内の複数の表から一行のデータを選択し、そしてその値をステートメントで指定したホスト変数に割り当てます。たとえば、次のステートメントは、JOHNSON という姓の従業員の名前をホスト変数 `firstname` に代入します。

  ```
  SELECT FIRSTNME
   INTO :firstname
   FROM EMPLOYEE
   WHERE LASTNAME = 'JOHNSON'
  ```

関連概念:

- 90 ページの『静的 SQL プログラムにおけるデータ検索』
- 116 ページの『アプリケーション中でのエラー・メッセージの検索』

関連タスク:

- 93 ページの『静的 SQL プログラムにおけるホスト変数の宣言』
- 100 ページの『カーソルを用いた複数行の選択』

- 「*アプリケーション開発ガイド アプリケーションの構築および実行*」の『サンプル・データベースのセットアップ』

関連資料:

- 「*SQL リファレンス 第 2 巻*」の『SELECT INTO ステートメント』

関連サンプル:

- 『dtlob.out -- HOW TO USE THE LOB DATA TYPE (C)』
- 『tbinfo.out -- HOW TO GET INFORMATION AT THE TABLE LEVEL (C)』
- 『tbread.out -- HOW TO READ TABLES (C)』
- 『tbread.sqc -- How to read tables (C)』
- 『dtlob.sqC -- How to use the LOB data type (C++)』
- 『tbinfo.sqC -- How to get information at the table level (C++)』
- 『tbread.out -- HOW TO READ TABLES (C++)』
- 『tbread.sqC -- How to read tables (C++)』
- 『static.sqb -- Get table data using static SQL statement (IBM COBOL)』
- 『static.sqb -- Get table data using static SQL statement (MF COBOL)』
- 『TbRead.out -- HOW TO READ TABLE DATA. Connect to 'sample' database using JDBC type 2 driver (JDBC)』
- 『TbRead.sqlj -- How to read table data (SQLj)』

静的 SQL プログラムにおけるデータ検索

SQL アプリケーション・プログラムの最も一般的なタスクの 1 つはデータの検索です。このタスクは、*SELECT* ステートメント を使用して行います。 SELECT ステートメントは、データベース内の表の行のうち特定の探索条件を満たすものを探索する照会の形式です。 探索条件に該当する行が存在すると、そのデータは検索されてホスト・プログラムの特定の変数に入れられ、どのような使用目的にも応えることができます。

選択ステートメントのコーディングが終わったら、アプリケーションに渡される情報を定義する SQL ステートメントをコーディングします。

選択ステートメントの結果は、行と列を持つデータベース内の表であると見なすことができます。 1 行だけが戻された場合、その結果は SELECT INTO ステートメントで指定したホスト変数に直接渡されます。

複数行が戻された場合は、*カーソル* を使ってそれらの行を一度に 1 行ずつ取り出さなければなりません。 カーソルは、順序付きの行ブロック内の特定の行を指し示す、アプリケーション・プログラムで用いられる名前付き制御構造です。

関連概念:

- 91 ページの『静的 SQL におけるホスト変数』
- 104 ページの『静的 SQL プログラムにおけるカーソルの例』

関連タスク:

- 93 ページの『静的 SQL プログラムにおけるホスト変数の宣言』

- 94 ページの『静的 SQL プログラムにおけるホスト変数の参照』
- 95 ページの『静的 SQL プログラムにおける標識変数の組み込み』
- 100 ページの『カーソルを用いた複数行の選択』
- 101 ページの『静的 SQL プログラムにおけるカーソルの宣言と使用』

静的 SQL における REOPT の影響

BIND オプション REOPT は、ホスト変数や特殊レジスターを含む静的 SQL ステートメントを、増分バインド・ステートメントのように動作させることができます。これは、これらのステートメントが、バインド時ではなく、 EXECUTE または OPEN の時にコンパイルされることを意味します。このコンパイル時に、これらの変数の実際の値に基づいて、アクセス・プランが選択されます。

REOPT ONCE の場合は、最初の OPEN または EXECUTE 要求の後にアクセス・プランがキャッシュされ、このステートメントの後の実行で使用されます。 REOPT ALWAYS の場合は、OPEN および EXECUTE 要求のたびにアクセス・プランが再生成され、現行のホスト変数、パラメーター・マーカー、および特殊レジスターの値のセットでこのプランが作成されます。

静的 SQL プログラムにおけるホスト変数

以下のセクションでは、静的 SQL プログラムにおけるホスト変数の使用方法について説明します。

静的 SQL におけるホスト変数

ホスト変数 とは、組み込み SQL ステートメントが参照する変数です。それによりデータベース・マネージャーとアプリケーション・プログラム間でデータをやり取りします。 SQL ステートメントでホスト変数を使用するときは、名前の接頭部にコロン (:) を付けてください。 ホスト言語ステートメントでホスト変数を使用するときは、コロンは省略してください。

ホスト変数はコンパイル・ホスト言語で宣言され、 BEGIN DECLARE SECTION および END DECLARE SECTION ステートメントで区切られます。プリコンパイラーは、これらのステートメントによって宣言を検出することができます。

注: Java™ JDBC および SQLJ プログラムは、宣言セクションを使用しません。 Java のホスト変数は、通常の Java 変数宣言構文に従います。

ホスト変数は、ホスト言語のサブセットを用いて宣言されます。

以下の規則は、ホスト変数宣言セクションに当てはまります。

- ホスト変数はすべてソース・ファイルで宣言してから参照しなければならない。ただし、SQLDA 構造を参照するホスト変数は例外です。
- 複数の宣言セクションを 1 つのソース・ファイルで使用する場合もある。
- プリコンパイラーはホスト言語変数の効力範囲の規則に従わない。

SQL ステートメントについては、ホスト変数がどの単一ソース・ファイルで実際に宣言されているかにかかわらず、すべてのホスト変数の有効範囲はグローバルです。したがって、ホスト変数の名前はソース・ファイル内でユニークでなければなりません。

これは、DB2® プリコンパイラーが、定義されている有効範囲外でもアクセスできるようにするためにホスト変数の有効範囲をグローバルに変更するということではありません。次の例を考えてください。

```
foo1(){
  .
  .
  .
  BEGIN SQL DECLARE SECTION;
  int x;
  END SQL DECLARE SECTION;
x=10;
  .
  .
  .
}


foo2(){
  .
  .
  .
  y=x;
  .
  .
  .
}
```

言語によっては、変数 x が foo2() 関数内で宣言されていなかったり、 x の値が foo2() 内で 10 に設定されていないため、どちらの場合も上記の例ではコンパイルが失敗します。 こうした問題を避けるには、グローバル変数として x を宣言するか、 x をパラメーターとして foo2() 関数に渡すかのいずれかにする必要があります。以下のようにします。

```
  foo1(){
  .
  .
  .
    BEGIN SQL DECLARE SECTION;
    int x;
    END SQL DECLARE SECTION;
    x=10;
    foo2(x);
  .
  .
  .
  }


  foo2(int x){
  .
  .
  .
    y=x;
  .
  .
  .
  }
```

関連概念:

- 158 ページの『C および C++ でのホスト変数』
- 205 ページの『COBOL のホスト変数』
- 227 ページの『FORTRAN のホスト変数』
- 543 ページの『REXX のホスト変数』

関連タスク:

- 34 ページの『db2dclgn 宣言生成プログラムを使用したホスト変数の宣言』
- 93 ページの『静的 SQL プログラムにおけるホスト変数の宣言』
- 94 ページの『静的 SQL プログラムにおけるホスト変数の参照』

静的 SQL プログラムにおけるホスト変数の宣言

プログラムでホスト変数を宣言して、データベース・マネージャーとアプリケーション間でデータをやり取りするために使用できます。

手順:

使用するホスト言語の構文を使用してホスト変数を宣言する。次の表に例示します。

表 4. ホスト言語によるホスト変数宣言

言語	ソース・コード例

C/C++	`EXEC SQL BEGIN DECLARE SECTION;`
	`  short  dept=38, age=26;`
	`  double  salary;`
	`  char  CH;`
	`  char  name1[9], NAME2[9];`
	`  /* C comment */`
	`  short  nul_ind;`
	`EXEC SQL END DECLARE SECTION;`

Java	`// Note that Java host variable declarations follow`
	`// normal Java variable declaration rules, and have`
	`// no equivalent of a DECLARE SECTION`
	`  short  dept=38, age=26;`
	`  double  salary;`
	`  char  CH;`
	`  String  name1[9], NAME2[9];`
	`  /* Java comment */`
	`  short  nul_ind;`

COBOL	`EXEC SQL BEGIN DECLARE SECTION END-EXEC.`
	`  01 age  PIC S9(4) COMP-5 VALUE 26.`
	`  01 DEPT  PIC S9(9) COMP-5 VALUE 38.`
	`  01 salary  PIC S9(6)V9(3) COMP-3.`
	`  01 CH  PIC X(1).`
	`  01 name1  PIC X(8).`
	`  01 NAME2  PIC X(8).`
	`*  COBOL comment`
	`  01 nul-ind  PIC S9(4) COMP-5.`
	`EXEC SQL END DECLARE SECTION END-EXEC.`

表 4. ホスト言語によるホスト変数宣言 (続き)

言語	ソース・コード例
FORTRAN	EXEC SQL BEGIN DECLARE SECTION integer*2 age /26/ integer*4 dept /38/ real*8 salary character ch character*8 name1,NAME2 C FORTRAN comment integer*2 nul_ind EXEC SQL END DECLARE SECTION

関連タスク:

- 34 ページの『db2dclgn 宣言生成プログラムを使用したホスト変数の宣言』
- 94 ページの『静的 SQL プログラムにおけるホスト変数の参照』

静的 SQL プログラムにおけるホスト変数の参照

ホスト変数を宣言すれば、アプリケーション・プログラムでその変数を参照できます。SQL ステートメントでホスト変数を使用するときは、名前の接頭部にコロン (:) を付けてください。 ホスト言語ステートメントでホスト変数を使用するときは、コロンは省略してください。

手順:

使用するホスト言語の構文を使用してホスト変数を参照する。次の表に例示します。

表 5. ホスト言語によるホスト変数参照

言語	ソース・コード例
C/C++	EXEC SQL FETCH C1 INTO :cm; printf("Commission = %f¥n", cm);

JAVA (SQLJ)	#SQL { FETCH :c1 INTO :cm };
	System.out.println("Commission = " + cm);

COBOL	EXEC SQL FETCH C1 INTO :cm END-EXEC DISPLAY 'Commission = ' cm
FORTRAN	EXEC SQL FETCH C1 INTO :cm WRITE(*,*) 'Commission = ', cm

関連タスク:

- 34 ページの『db2dclgn 宣言生成プログラムを使用したホスト変数の宣言』
- 93 ページの『静的 SQL プログラムにおけるホスト変数の宣言』

静的 SQL プログラムにおける標識変数

以下のセクションでは、静的 SQL プログラムにおける標識変数の使用方法について説明します。

静的 SQL プログラムにおける標識変数の組み込み

Java 以外の言語で作成されたアプリケーションは、 NULL 値を受け取ることができるホスト変数と*標識変数* とを関連付けることによって、 NULL 値を受け取る準備をしておく必要があります。 Java アプリケーションは、ホスト変数の値を Java の NULL 値と比較して、受け取った値が NULL 値かどうかを判別します。標識変数は、データベース・マネージャーとホスト・アプリケーションにより共用されます。したがって、標識変数はアプリケーション内ではホスト変数として宣言しておくことが必要です。このホスト変数は、SQL データのタイプ SMALLINT に対応します。

標識変数は SQL ステートメントでホスト変数の直後に置かれ、接頭部としてコロンが付けられます。スペースを用いると標識変数とホスト変数を分けることができますが、このスペースは必須ではありません。ただし、ホスト変数と標識変数の間にコンマを使用しないでください。また、オプションの INDICATOR キーワードをホスト変数とその標識の間に置いて標識変数を指定することもできます。

手順:

INDICATOR キーワードを使用して標識変数を書く。次の表にサポートされるホスト言語を例示します。

表 6. ホスト言語による標識変数

言語	ソース・コード例

C/C++	`EXEC SQL FETCH C1 INTO :cm INDICATOR :cmind;`
	`if ( cmind < 0 )`
	`  printf( "Commission is NULL¥n" );`

JAVA (SQLJ)	`#SQL { FETCH :c1 INTO :cm };`
	`if ( cm == null )`
	`  System.out.println( "Commission is NULL¥n" );`

COBOL	`EXEC SQL FETCH C1 INTO :cm INDICATOR :cmind END-EXEC`
	`IF cmind LESS THAN 0`
	`  DISPLAY 'Commission is NULL'`

FORTRAN	`EXEC SQL FETCH C1 INTO :cm INDICATOR :cmind`
	`IF ( cmind .LT. 0 ) THEN`
	`  WRITE(*,*) 'Commission is NULL'`
	`ENDIF`

上の例では、*cmind* が負の値かどうかを検査します。変数が負の値ではない場合、アプリケーションは *cm* の戻り値を使用することができます。変数が負の値の場合、取り出される値は NULL なので、*cm* は使用すべきではありません。この場合、データベース・マネージャーはホスト変数の値を変更しません。

注: データベースの構成パラメーター *dft_sqlmathwarn* を 'YES' に設定した場合、*cmind* の値は -2 になることがあります。この値は、算術計算エラーのある式を評価したため、または結果数値をホスト変数に変換しようとした時にオーバーフローが起きたために、 NULL になったことを示しています。

データ・タイプが NULL を処理できる場合、アプリケーションは NULL 標識を指定しなければなりません。そうでない場合は、エラーになります。 NULL 標識を使用しない場合、SQLCODE -305 (SQLSTATE 22002) が戻されます。

SQLCA 構造体が切り捨て警告を示す場合、標識変数を検査して切り捨てが行われているかどうか調べます。標識変数が正の値の場合は、切り捨てが行われています。

- TIME データ・タイプの秒の部分が切り捨てられると、標識値には切り捨てられたデータの秒の部分が含まれます。
- 他のすべてのストリングのデータ・タイプ (ラージ・オブジェクト (LOB) を除く) の場合、標識値は戻されたデータの実際の長さを表します。ユーザー定義特殊タイプ (UDT) は、それらの基本タイプと同じように扱われます。

INSERT または UPDATE ステートメントを処理中に、データベース・マネージャーは標識変数をチェックします (標識変数がある場合)。標識変数が負の値の場合、データベース・マネージャーはターゲットの列値を NULL に設定します (NULL が使用できる場合)。

標識変数がゼロ以上の場合、データベース・マネージャーは関連付けられたホスト変数の値を使用します。

ホスト変数に割り当てられる際にストリング列の値が切り捨てられた場合、 SQLCA 構造体の SQLWARN1 フィールドに X または W が入れられる場合があります。 NULL 終止符が切り捨てられた場合、そのフィールドには 'N' が入ります。

下記の条件のすべてに適合した場合にのみ、データベース・マネージャーから X の値が戻されます。

- データベースのコード・ページからアプリケーションのコード・ページに文字ストリング・データを変換するとデータの長さが変わる場合に、混合コード・ページ接続が行われる。
- カーソルがブロック化されている。
- 標識変数がアプリケーションにより指定されている。

標識変数に戻される値は、アプリケーションのコード・ページでの文字ストリングの長さになります。

それ以外の場合でデータ切り捨てが行われると、 (NULL 終止符の切り捨てとは対照的に) データベース・マネージャーは必ず W を戻します。 この場合、データベース・マネージャーは、選択リスト項目のコード・ページ (アプリケーションのコード・ページであれ、データベースのコード・ページであれ、あるいは何もない場合であれ) にある結果文字ストリングの長さを指す標識変数の値をアプリケーションに戻します。

関連タスク:

- 34 ページの『db2dclgn 宣言生成プログラムを使用したホスト変数の宣言』
- 93 ページの『静的 SQL プログラムにおけるホスト変数の宣言』
- 94 ページの『静的 SQL プログラムにおけるホスト変数の参照』

関連資料:

- 97 ページの『静的 SQL プログラムにおける標識変数のデータ・タイプ』

静的 SQL プログラムにおける標識変数のデータ・タイプ

DB2 表の各列には、列の作成時に *SQL データ・タイプ* が付与されます。列にデータ・タイプを割り当てる方法については、CREATE TABLE ステートメントを参照してください。データベース・マネージャーは、以下の列データ・タイプをサポートしています。

SMALLINT
16 ビットの符号付き整数。

INTEGER
32 ビットの符号付き整数。このタイプの同義語として **INT** を使用できます。

BIGINT
64 ビットの符号付き整数。

DOUBLE
倍精度浮動小数点。 **DOUBLE PRECISION** および **FLOAT(*n*)** (*n* は 24 より大きい) はこのタイプの同義語です。

REAL 単精度浮動小数点 **FLOAT(*n*)** (*n* は 24 より小さい) はこのタイプの同義語です。

DECIMAL
パック 10 進数。このタイプの同義語として **DEC**、**NUMERIC**、および **NUM** を使用できます。

CHAR 1〜254 バイトの長さの固定長文字ストリング。このタイプの同義語として **CHARACTER** を使用できます。

VARCHAR
1 バイトから 32 672 バイトの長さの可変長文字ストリング。 このタイプの同義語として **CHARACTER VARYING** および **CHAR VARYING** を使用できます。

LONG VARCHAR
1 バイトから 32 700 バイトの長さのロング可変長文字ストリング

CLOB 1 バイトから 2 ギガバイトの長さのラージ・オブジェクト可変長文字ストリング

BLOB 1 バイトから 2 ギガバイトの長さのラージ・オブジェクト可変長バイナリー・ストリング

DATE タイム・スタンプを表す長さ 10 バイトの文字ストリング

TIME タイムを表す長さ 8 バイトの文字ストリング

TIMESTAMP
タイム・スタンプを表す長さ 26 バイトの文字ストリング

以下のデータ・タイプは、 2 バイト文字セット (DBCS) および拡張 UNIX コード (EUC) 文字環境でしか使用できません。

GRAPHIC
長さ 1〜127 の 2 バイト文字のラージ・オブジェクト固定長 GRAPHIC ストリング

VARGRAPHIC
長さ 1〜16 336 の 2 バイト文字の固定長 GRAPHIC ストリング

LONG VARGRAPHIC
長さ 1〜16 350 の 2 バイト文字のロング可変長 GRAPHIC ストリング

DBCLOB
長さ 1〜1 073 741 823 の 2 バイト文字のラージ・オブジェクト可変長 GRAPHIC ストリング

注:

1. サポートされているすべてのデータ・タイプに、NOT NULL 属性を指定できます。これは別のタイプとして扱われます。
2. 上記の一連のデータ・タイプは、ユーザー定義特殊タイプ (UDT) を定義することにより拡張することができます。 UDT は、組み込み SQL タイプの 1 つの表現を使用する、別個のデータ・タイプです。

サポートされるホスト言語のデータ・タイプは、データベース・マネージャーのデータ・タイプの大多数に対応しています。ホスト変数宣言には、これらのホスト言語のデータ・タイプだけが使用できます。プリコンパイラーは、ホスト変数宣言を検出すると、適切な SQL データ・タイプの値を判別します。データベース・マネージャーはこの値を使用して、アプリケーションとの間でやり取りするデータを変換します。

データベース・マネージャーが異なるデータ・タイプ間の比較と割り当てをどのように処理するかを理解することは、アプリケーション・プログラマーにとって重要です。つまり、データベース・マネージャーが 2 つの SQL 列データ・タイプと 2 つのホスト言語データ・タイプ (あるいはそのいずれか) で処理を行っているとしても、データ・タイプは割り当ておよび比較の操作中は互いに互換性を持たなければなりません。

データ・タイプの互換性に関する*一般的な* 規則とは、サポートされるホスト言語の数値データ・タイプはすべてデータベース・マネージャーの数値データ・タイプと比較して割り当てることができ、ホスト言語の文字タイプはすべてデータベース・マネージャーの文字タイプと互換性があるということです。数値タイプには文字タイプとの互換性がありません。ただし、この一般的な規則には、ラージ・オブジェクトでの処理時に課される特徴および制限に基づいた例外もあります。

SQL ステートメント内であれば、DB2 では互換性のあるデータ・タイプ同士での変換が可能です。たとえば、以下の SELECT ステートメントでは、SALARY と BONUS は DECIMAL 列ですが、各従業員の合計支給額は DOUBLE データとして戻されます。

```
SELECT EMPNO, DOUBLE(SALARY+BONUS) FROM EMPLOYEE
```

上記のステートメントの実行では、 DECIMAL と DOUBLE のデータ・タイプ間で変換が行われることに注目してください。

画面に表示される照会結果をもっと読みやすくするには、次の SELECT ステートメントを使用することができます。

```
SELECT EMPNO, DIGIT(SALARY+BONUS) FROM EMPLOYEE
```

アプリケーション内でデータを変換するには、この変換をサポートするその他のルーチン、クラス、組み込みタイプ、または API があるかどうかコンパイラーのベンダーにお問い合わせください。

アプリケーションのコード・ページがデータベースのコード・ページと異なる場合は、文字データ・タイプは文字変換にも従います。

関連概念:

- 「*アプリケーション開発ガイド サーバー・アプリケーションのプログラミング*」の『データ変換についての考慮事項』
- 666 ページの『異なるコード・ページ間での文字変換』

関連資料:

- 「*SQL リファレンス 第 2 巻*」の『CREATE TABLE ステートメント』
- 186 ページの『C および C++ においてサポートされている SQL データ・タイプ』
- 215 ページの『COBOL でサポートされている SQL データ・タイプ』
- 233 ページの『FORTRAN でサポートされている SQL データ・タイプ』
- 549 ページの『REXX でサポートされている SQL データ・タイプ』
- 403 ページの『Java、JDBC、および SQL のデータ・タイプ』

静的 SQL プログラムにおける標識変数の例

次の例は、静的 SQL を使用する C/C++ プログラムにおける標識変数の使用の例です。

- 例 1

  次の例は、NULL 可能なデータ列での標識変数のインプリメンテーションを示しています。この例では、列 FIRSTNAME は NULL 可能ではありませんが、列 WORKDEPT は NULL 値であってもかまいません。

```
EXEC SQL BEGIN DECLARE SECTION;
  char wd[3];
  short wd_ind;
  char firstname[13];
EXEC SQL END DECLARE SECTION;

  /* connect to sample database */

EXEC SQL SELECT FIRSTNME, WORKDEPT
  INTO :firstname, :wd:wdind
  FROM EMPLOYEE
  WHERE LASTNAME = 'JOHNSON';
```

  列 WORKDEPT が NULL 値である場合があるため、使用する前に標識変数をホスト変数として宣言しなければなりません。

- 例 2 (**dtlob**)

  サンプル **dtlob** には `BlobFileUse()` と呼ばれる関数があります。関数 `BlobFileUse()` には、SELECT INTO ステートメントを使用してファイル中の BLOB データを読み取る照会が含まれています。

```
EXEC SQL BEGIN DECLARE SECTION;
  SQL TYPE IS BLOB_FILE blobFilePhoto;
  char photoFormat[10];
  char empno[7];
  short lobind;
EXEC SQL END DECLARE SECTION;

  /* Connect to the sample database */

SELECT picture INTO :blobFilePhoto:lobind
  FROM emp_photo
  WHERE photo_format = :photoFormat AND empno = '000130'
```

列 BLOBFILEPHOTO が NULL 値である場合があるため、使用する前に標識変数 LOBIND をホスト変数として宣言しなければなりません。サンプル **dtlob** には、LOB ではどうなるかが示されています。LOB の使用についての詳細は、サンプルを参照してください。

関連概念:

- 89 ページの『静的 SQL プログラムの例』

関連タスク:

- 95 ページの『静的 SQL プログラムにおける標識変数の組み込み』

関連資料:

- 97 ページの『静的 SQL プログラムにおける標識変数のデータ・タイプ』

関連サンプル:

- 『dtlob.out -- HOW TO USE THE LOB DATA TYPE (C)』
- 『dtlob.sqc -- How to use the LOB data type (C)』
- 『dtlob.out -- HOW TO USE THE LOB DATA TYPE (C++)』
- 『dtlob.sqC -- How to use the LOB data type (C++)』

カーソルを用いた複数行の選択

以下のセクションでは、カーソルを用いた行の選択方法について説明します。サンプル・プログラムでは、カーソルの宣言、カーソルのオープン、表からの行の取り出し、およびカーソルのクローズの仕方も簡単に説明されています。

カーソルを用いた複数行の選択

SQL では、アプリケーションが行のセットを取り出すことができるようにするため、*カーソル* という手法を用います。

カーソルの概念を理解しやすくするために、データベース・マネージャーが*結果表*を作成し、そこに SELECT ステートメントを実行して検索されたすべての行を保持する場合を考えてみてください。カーソルを用いて結果表の*現在行* を識別して指示することにより、その表からの行をアプリケーションで使用できるようにします。カーソルを使用するとアプリケーションは、データの終わり状態、すなわち NOT FOUND 状態、SQLCODE +100 (SQLSTATE 02000) になるまで結果表から各行を順次取り出すことができます。 SELECT ステートメントを実行した結果取り出された行のセットは、0 行、または 1 行以上で構成されます。これは探索条件を満たす行数によって決まります。

手順:

カーソル処理に必要な手順は以下のとおりです。

1. DECLARE CURSOR ステートメントを用いてカーソルを指定する。
2. OPEN ステートメントを用いて照会を実行し、結果表を作成する。
3. FETCH ステートメントを用いて行を一度に 1 行ずつ取り出す。
4. DELETE または UPDATE ステートメントを用いて行を処理する (必要な場合)。
5. CLOSE ステートメントを用いて行を終了する。

アプリケーションは同時に複数のカーソルを使用することができます。各カーソルには DECLARE CURSOR、OPEN、CLOSE、および FETCH ステートメントのセットが必要です。

関連概念:

- 104 ページの『静的 SQL プログラムにおけるカーソルの例』

静的 SQL プログラムにおけるカーソルの宣言と使用

DECLARE CURSOR ステートメントによりカーソルを定義、命名し、 SELECT ステートメントを用いて取り出した行のセットを識別します。

アプリケーションはカーソルに名前を割り当てます。この名前は、その後に続く OPEN、FETCH、および CLOSE ステートメントで参照されます。照会とは、任意の有効な SELECT ステートメントです。

制約事項:

DECLARE ステートメントの位置は自由ですが、最初に使用するカーソルの位置よりも上に置く必要があります。

手順:

DECLARE ステートメントを使用してカーソルを定義します。次の表にサポートされるホスト言語を例示します。

表 7. ホスト言語によるカーソル宣言

言語	ソース・コード例

C/C++	`EXEC SQL DECLARE C1 CURSOR FOR`
	`  SELECT PNAME, DEPT FROM STAFF`
	`  WHERE JOB=:host_var;`

JAVA (SQLJ)	`#sql iterator cursor1(host_var data type);`
	`#sql cursor1 = { SELECT PNAME, DEPT FROM STAFF`
	`  WHERE JOB=:host_var };`

COBOL	`EXEC SQL DECLARE C1 CURSOR FOR`
	`  SELECT NAME, DEPT FROM STAFF`
	`  WHERE JOB=:host-var END-EXEC.`

FORTRAN	`EXEC SQL DECLARE C1 CURSOR FOR`
	`+  SELECT NAME, DEPT FROM STAFF`
	`+  WHERE JOB=:host_var`

関連概念:

- 102 ページの『カーソル・タイプおよび作業単位に関する考慮事項』

関連タスク:

- 100 ページの『カーソルを用いた複数行の選択』

関連資料:

- 106 ページの『カーソル・タイプ』

カーソル・タイプおよび作業単位に関する考慮事項

COMMIT または ROLLBACK 操作のカーソルのアクションは、カーソルがどのように定義されているかによって異なります。

読み取り専用カーソル

カーソルが読み取り専用として定義され、かつ反復可能読み取り分離レベルを使用する場合であっても、反復可能読み取りロックは作業単位に必要なシステム表上に集められ保持されます。そのため、アプリケーションは読み取り専用カーソルの場合であっても定期的に COMMIT ステートメントを発行することが重要です。

WITH HOLD オプション

アプリケーションが COMMIT ステートメントを発行してある作業単位を完了すると、すべての*オープン・カーソル* (WITH HOLD オプションを使用して宣言されるものを除く) は、データベース・マネージャーにより自動的にクローズされます。

WITH HOLD で宣言されたカーソルは、アクセスするリソースを複数の作業単位間で保持します。カーソルを WITH HOLD で宣言した場合の影響は、作業単位がどのように終了するかによって決まります。

- 作業単位が COMMIT ステートメントで終了する場合、 WITH HOLD で定義されたオープン・カーソルは OPEN のままです。カーソルは結果表の次の論理行の前に置かれます。さらに、WITH HOLD で定義された OPEN カーソルを参照する準備済みのステートメントも保存されます。COMMIT の直前にある、特定のカーソルに関連する FETCH および CLOSE 要求だけが有効です。 UPDATE WHERE CURRENT OF および DELETE WHERE CURRENT OF ステートメントは、同じ作業単位内で取り出された行の場合に限り有効です。

  注: 作業単位中にパッケージが再バインドされると、保留されたカーソルはすべてクローズします。

- 作業単位が ROLLBACK ステートメントで終了する場合、オープン・カーソルはすべてクローズされ、作業単位の間に獲得したロックはすべて解放され、その作業単位での処理に依存する準備済みステートメントはすべて消去されます。

たとえば、TEMPL 表に 1 000 項目が入っており、 すべての従業員の給与列を更新するとします。 100 行更新するたびに COMMIT ステートメントを発行するようにします。

1. WITH HOLD オプションを用いてカーソルを宣言する。

```
EXEC SQL DECLARE EMPLUPDT CURSOR WITH HOLD FOR
  SELECT EMPNO, LASTNAME, PHONENO, JOBCODE, SALARY
  FROM TEMPL FOR UPDATE OF SALARY
```

2. カーソルをオープンし、一度に 1 行ずつ結果表からデータを取り出す。

```
EXEC SQL OPEN EMPLUPDT
  .
  .
  .

EXEC SQL FETCH EMPLUPDT
  INTO :upd_emp, :upd_lname, :upd_tele, :upd_jobcd, :upd_wage,
```

3. 行を更新または削除する場合は、 WHERE CURRENT OF オプションを用いて UPDATE または DELETE ステートメントを使用する。たとえば、現在行を更新するには、プログラムは以下のようにすることができます。

```
EXEC SQL UPDATE TEMPL SET SALARY = :newsalary
  WHERE CURRENT OF EMPLUPDT
```

4. COMMIT を発行した後、別の行を更新する前に FETCH を発行しなければならない。

アプリケーションに、SQLCODE -501 (SQLSTATE 24501) を検出し処理するコードを組み込むようにしてください。この戻り値は、アプリケーションが次のいずれかである場合に、 FETCH または CLOSE ステートメントで戻されることがあります。

- WITH HOLD 宣言されたカーソルを使用している場合
- 2 つ以上の作業単位を実行し、かつ WITH HOLD カーソルが作業単位の境界を超えてオープンしている場合

アプリケーションがある表を消去したために、その表に依存しているパッケージが無効になると、そのパッケージは動的に再バインドされます。このような場合、データベース・マネージャーがカーソルをクローズするため、FETCH または CLOSE ステートメントに SQLCODE -501 (SQLSTATE 24501) が戻されます。こうした状況で SQLCODE -501 (SQLSTATE 24501) を処理する方法は、カーソルから行を取り出したいかどうかによって決まります。

- カーソルから行を取り出したい場合は、カーソルをオープンしてから、FETCH ステートメントを実行します。ただし、OPEN ステートメントによってカーソルの開始位置が移動するので注意してください。 COMMIT WORK ステートメントで保留されていた直前の位置は失われます。
- カーソルから行を取り出すことを望まない場合は、カーソルに対してどんな SQL 要求も発行しません。

WITH RELEASE オプション

アプリケーションが WITH RELEASE オプションを使ってカーソルをクローズすると、 DB2® はカーソルが保持している読み取りロックをすべて解放しようとします。 そうすると、カーソルは書き込みロックしか保持できなくなります。アプリケーションが RELEASE オプションを使わないでカーソルをクローズすると、作業単位の完了時に読み取りおよび書き込みロックは解放されます。

関連タスク:

- 100 ページの『カーソルを用いた複数行の選択』
- 101 ページの『静的 SQL プログラムにおけるカーソルの宣言と使用』

静的 SQL プログラムにおけるカーソルの例

サンプル **tut_read.sqc** (C の場合)、**tut_read.sqC/sqx** (C++ の場合)、**TutRead.sqlj** (SQLJ の場合)、および **cursor.sqb** (COBOL の場合) は、カーソルの宣言、カーソルのオープン、表からの行の取り出し、およびカーソルのクローズの仕方を示しています。

REXX 言語は、静的 SQL をサポートしないため、サンプルはありません。

- C/C++

  サンプル tut_read は、カーソルを使用した表からの基本的な選択方法を示します。たとえば、以下のようにします。

```
 /* declare cursor */
  EXEC SQL DECLARE c1 CURSOR FOR
    SELECT deptnumb, deptname FROM org WHERE deptnumb < 40;

/* open cursor */
  EXEC SQL OPEN c1;

/* fetch cursor */
  EXEC SQL FETCH c1 INTO :deptnumb, :deptname;

while (sqlca.sqlcode != 100)
  {
    printf("    %8d %-14s¥n", deptnumb, deptname);
    EXEC SQL FETCH c1 INTO :deptnumb, :deptname;
  }

  /* close cursor */
  EXEC SQL CLOSE c1;
```

- Java™

  サンプル **TutRead** は、カーソルを使用した簡単な選択で、表データを読み取る方法を示します。たとえば、以下のようにします。

```
// cursor definition
  #sql iterator TutRead_Cursor(int, String);

  // declare cursor
  TutRead_Cursor cur2;
  #sql cur2 = {SELECT deptnumb, deptname FROM org WHERE deptnumb < 40};

  // fetch cursor
  #sql {FETCH :cur2 INTO :deptnumb, :deptname};

  // retrieve and display the result from the SELECT statement
  while (!cur2.endFetch())
  {
    System.out.println(deptnumb + ", " + deptname);
    #sql {FETCH :cur2 INTO :deptnumb, :deptname};
  }

  // close cursor
  cur2.close();
```

- COBOL

サンプル **cursor** は、静的 SQL ステートメントでカーソルを使用した表データの検索方法を示します。たとえば、以下のようにします。

```
* Declare a cursor
     EXEC SQL DECLARE c1 CURSOR FOR
              SELECT name, dept FROM staff
              WHERE job='Mgr' END-EXEC.

* Open the cursor
     EXEC SQL OPEN c1 END-EXEC.

* Fetch rows from the 'staff' table
     perform Fetch-Loop thru End-Fetch-Loop
      until SQLCODE not equal 0.

* Close the cursor
     EXEC SQL CLOSE c1 END-EXEC.
     move "CLOSE CURSOR" to errloc.
```

関連概念:

- 102 ページの『カーソル・タイプおよび作業単位に関する考慮事項』
- 116 ページの『アプリケーション中でのエラー・メッセージの検索』

関連タスク:

- 100 ページの『カーソルを用いた複数行の選択』
- 101 ページの『静的 SQL プログラムにおけるカーソルの宣言と使用』

関連資料:

- 106 ページの『カーソル・タイプ』

関連サンプル:

- 『cursor.sqb -- How to update table data with cursor statically (IBM COBOL)』
- 『tut_read.out -- HOW TO READ TABLES (C)』
- 『tut_read.sqc -- How to read tables (C)』
- 『tut_read.out -- HOW TO READ TABLES (C++)』
- 『tut_read.sqC -- How to read tables (C++)』
- 『TutRead.out -- HOW TO READ TABLE DATA. Connect to 'sample' database using JDBC type 2 driver (SQLJ)』
- 『TutRead.sqlj -- Read data in a table (SQLj)』

検索されたデータの取り扱い

以下のセクションでは、検索されたデータの更新方法と削除方法について説明します。サンプル・プログラムでは、データの取り扱い方法も簡単に説明されています。

静的 SQL プログラムにおける検索データの更新と削除

カーソルによって参照された行は、更新したり削除したりできます。更新可能な行の場合、カーソルに対応する照会は読み取り専用であってはなりません。

手順:

カーソルを用いて更新を行うためには、 UPDATE ステートメントで WHERE CURRENT OF 文節を使用してください。結果表の列のうちどれを更新したいのかをシステムに指示するには、FOR UPDATE 文節を使用します。 FOR UPDATE での列の指定は全部を選択しなくてもよいため、カーソルで明確に検索されない列でも更新することができます。 FOR UPDATE 文節を列名を使わずに指定すると、表の中のすべての列や、外部で全選択された最初の FROM 文節で識別されたビューは更新可能であると見なされます。 FOR UPDATE 文節では、必要以上の列を指定しないでください。 FOR UPDATE 文節に余分な列の名前を指定すると、 DB2 がデータにアクセスする能率を低下させる場合もあります。

カーソルを用いた削除は、DELETE ステートメントで WHERE CURRENT OF 文節を使用して行います。一般に、FOR UPDATE 文節はカーソルの現在行の削除には必要ありません。 SAA1 に設定された LANGLEVEL でプリコンパイルされ、BLOCKING ALL でバインドされたアプリケーション内の SELECT ステートメントまたは DELETE ステートメントのいずれかに対して動的 SQL を使った場合だけは例外です。この場合、SELECT ステートメントで FOR UPDATE 文節を指定する必要があります。

DELETE ステートメントを使用すると、カーソルで参照される行を削除することができます。削除では、カーソルは次の 行の前に置かれたままになるため、カーソルに対して WHERE CURRENT OF 操作をさらに実行する前に、 FETCH ステートメントを発行する必要があります。

関連資料:

- 「*コマンド・リファレンス*」の『PRECOMPILE コマンド』
- 「*SQL リファレンス 第 1 巻*」の『SQL 照会』

カーソル・タイプ

カーソルは以下の 3 種類に分類されます。

読み取り専用
このカーソルの行は読み取り専用で、更新することはできません。読み取り専用カーソルは、アプリケーションがデータを読み取る場合にだけ使用され、データの修正には使用されません。カーソルは、読み取り専用の選択ステートメントに基づいている場合に限り、読み取り専用と見なされます。更新可能ではない結果表を定義している SELECT ステートメントの規則については、データの更新と検索を行う方法に関する説明を参照してください。

読み取り専用カーソルにはパフォーマンス上の利点もあります。

更新可能
このカーソルの行は更新することができます。アプリケーションがカーソルの行の取り出しに伴ってデータを修正する場合に、更新可能カーソルを使用します。指定した照会は、表またはビューを 1 つだけ参照することができます。また、照会には FOR UPDATE 文節を組み込んで、更新されるそれぞれの列に名前を付ける必要があります (LANGLEVEL MIA プリコンパイル・オプションを使用しない場合)。

未確定 定義またはコンテキストからはカーソルが更新可能か読み取り専用かを判別

することができません。こうした状況は、本来なら読み取り専用のはずのカーソルが動的 SQL ステートメントによって変更される場合に生じます。

未確定カーソルは、プリコンパイル時またはバインド時に BLOCKING ALL オプションが指定されると読み取り専用と見なされます。そうでない場合、カーソルは更新可能と見なされます。

注: 動的に処理されるカーソルは常に未確定です。

関連概念:

- 252 ページの『IBM OLE DB Provider でサポートされているカーソル・モード』

関連タスク:

- 105 ページの『静的 SQL プログラムにおける検索データの更新と削除』

静的 SQL プログラムにおける取り出しの例

次の例では、カーソルを使用して表から選択し、カーソルをオープンし、その表から行を取り出します。そして、取り出したそれぞれの行に対して、プログラムは単純な基準に基づいて削除すべきか更新すべきかを判別します。

REXX 言語は静的 SQL をサポートしないため、サンプルはありません。

- C/C++ (**tut_mod.sqc/tut_mod.sqC**)

  次の例は、サンプル **tut_mod** からの例です。この例では、カーソルを使用して表から選択し、カーソルをオープンし、その表から行を取り出し、取り出した行の更新または削除を行い、その後カーソルをクローズします。

```
EXEC SQL DECLARE c1 CURSOR FOR SELECT * FROM staff WHERE id >= 310;
  EXEC SQL OPEN c1;
  EXEC SQL FETCH c1 INTO :id, :name, :dept, :job:jobInd, :years:yearsInd, :salary,
     :comm:commInd;
```

  サンプル **tbmod** は、サンプル **tut_mod** の長いもので、表データの変更に関するほとんどすべてのケースを示しています。

- Java™ (**TutMod.sqlj**)

  次の例は、サンプル **TutMod** からの例です。この例では、カーソルを使用して表から選択し、カーソルをオープンし、その表から行を取り出し、取り出した行の更新または削除を行い、その後カーソルをクローズします。

```
#sql cur = {SELECT * FROM staff WHERE id >= 310};
#sql {FETCH :cur INTO :id, :name, :dept, :job, :years, :salary, :comm};
```

  サンプル **TbMod** は、サンプル **TutMod** の長いもので、表データの変更に関するほとんどすべてのケースを示しています。

- COBOL (**openftch.sqb**)

  次の例は、サンプル **openftch** からの例です。この例では、カーソルを使用して表から選択し、カーソルをオープンし、その表から行を取り出します。

```
 EXEC SQL DECLARE c1 CURSOR FOR
   SELECT name, dept FROM staff
   WHERE job='Mgr'
   FOR UPDATE OF job END-EXEC.
```

```
EXEC SQL OPEN c1 END-EXEC


* call the FETCH and UPDATE/DELETE loop.
  perform Fetch-Loop thru End-Fetch-Loop
  until SQLCODE not equal 0.

EXEC SQL CLOSE c1 END-EXEC.
```

関連概念:

- 116 ページの『アプリケーション中でのエラー・メッセージの検索』

関連サンプル:

- 『openftch.sqb -- How to modify table data using cursor statically (IBM COBOL)』
- 『tbmod.sqc -- How to modify table data (C)』
- 『tut_mod.out -- HOW TO MODIFY TABLE DATA (C)』
- 『tut_mod.sqc -- How to modify table data (C)』
- 『tbmod.sqC -- How to modify table data (C++)』
- 『tut_mod.out -- HOW TO MODIFY TABLE DATA (C++)』
- 『tut_mod.sqC -- How to modify table data (C++)』
- 『TbMod.sqlj -- How to modify table data (SQLj)』
- 『TutMod.out -- HOW TO MODIFY TABLE DATA. Connect to 'sample' database using JDBC type 2 driver (SQLJ)』
- 『TutMod.sqlj -- Modify data in a table (SQLj)』

検索されたデータのスクロールと取り扱い

以下のセクションでは、検索されたデータをスクロールする方法について説明します。サンプル・プログラムでは、データの取り扱い方法も簡単に説明されています。

以前に検索されたデータのスクロール

アプリケーションがデータベースからデータを検索するとき、 FETCH ステートメントを使うとデータを順方向へスクロールすることができます。しかし、データベース・マネージャーの組み込み SQL ステートメントにはデータを逆方向へスクロールする機能 (逆方向 FETCH 機能に相当) がありません。一方、DB2 CLI と Java では読み取り専用の両方向スクロール・カーソルによる逆方向 FETCH 機能をサポートしています。

手順:

組み込み SQL アプリケーションの場合、すでに検索されたデータをスクロールするには以下の技法を使うことができます。

- 取り出されたデータのコピーを保管しておき、何らかのプログラム技法を用いてそれをスクロールする方法。
- SQL を用いて (一般的には 2 番目の SELECT ステートメントを使用して) データを再び検索する方法。

関連タスク:

- 109 ページの『データのコピーを保持する方法』
- 109 ページの『データを 2 度検索する方法』

関連資料:

- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLFetchScroll 関数 (CLI) - すべてのバインド列の行セットの取り出しとデータの戻り』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLFetchScroll() (CLI) のカーソル位置決めの規則』

データのコピーを保持する方法

ある状況では、アプリケーションにより取り出されたデータのコピーを保持しておくと便利な場合があります。

手順:

データのコピーを保持するには、アプリケーションで次のようにします。

- 取り出されたデータを仮想記憶域に保管します。
- データを一時ファイルに書き込みます (仮想記憶域にデータを入れられない場合)。この方法の利点は、データベース内のデータがトランザクションによって一時的に変更された場合でさえも、ユーザーは、取り出されたデータとまったく同じものを、逆方向スクロールによって常に見ることができるという点です。
- 反復可能読み取りの分離レベルを使用すると、カーソルをクローズしたりオープンすることにより、トランザクションから検索したデータをもう一度検索することができます。検索結果のデータは、他のアプリケーションにより更新されることはありません。データの更新方法は、分離レベルおよびロックにより左右されます。

関連概念:

- 110 ページの『最初の結果表と 2 番目の結果表の行の順序の違い』

関連タスク:

- 109 ページの『データを 2 度検索する方法』

データを 2 度検索する方法

データを 2 度検索するために使用する技法は、データをもう一度見ようとする順序により異なります。

手順:

次のいずれかのメソッドを使用して、データを 2 度検索できます。

- 先頭からデータを検索する

  結果表の先頭からデータを再び検索するには、アクティブ・カーソルをクローズしてそれを再オープンします。このアクションによってカーソルは結果表の先頭に置かれます。ただし、アプリケーションがその表に対してロックを保持してい

ない限り、他のユーザーがその表に変更を加える可能性があるので、以前に結果表の最初の行であったものが、最初の行でなくなるということもあり得ます。

- 途中からデータを検索する

  結果表の中ほどから 2 度目のデータ検索を行うには、 2 度目の SELECT ステートメントを実行し、そのステートメント上で 2 つ目のカーソルを宣言してください。たとえば、最初の SELECT ステートメントが次のものであるとします。

  ```
  SELECT * FROM DEPARTMENT
    WHERE LOCATION = 'CALIFORNIA'
    ORDER BY DEPTNO
  ```

  今度は DEPTNO = 'M95' から始まる行に戻って、その場所から順番に行を取り出すとします。この場合は、次のようにコーディングしてください。

  ```
  SELECT * FROM DEPARTMENT
    WHERE LOCATION = 'CALIFORNIA'
    AND DEPTNO >= 'M95'
    ORDER BY DEPTNO
  ```

  このステートメントによって、カーソルは希望する場所に置かれます。

- 逆順にデータを検索する

  行の昇順がデフォルトの設定です。 DEPTNO のおのおのの値に対する行が 1 つしかない場合、次のステートメントは行をユニークな昇順に指定します。

  ```
  SELECT * FROM DEPARTMENT
    WHERE LOCATION = 'CALIFORNIA'
    ORDER BY DEPTNO
  ```

  同じ行を逆順に検索するには、次のステートメントのように順序を降順として指定してください。

  ```
  SELECT * FROM DEPARTMENT
    WHERE LOCATION = 'CALIFORNIA'
    ORDER BY DEPTNO DESC
  ```

  2 番目のステートメント上のカーソルは、最初のステートメント上のカーソルからの順番とはまったく逆の順番に行を検索します。検索の順序は、最初のステートメントがユニークな順序を指定している場合にのみ保証されます。

  行を逆順で検索する場合、DEPTNO 列に、 1 つは昇順で、もう 1 つは降順の 2 つの索引を持つと便利です。

関連概念:

- 110 ページの『最初の結果表と 2 番目の結果表の行の順序の違い』

最初の結果表と 2 番目の結果表の行の順序の違い

2 番目の結果表の行は、最初の結果表と同じ順序で表示されるとは限りません。データベース・マネージャーは、SELECT ステートメントが ORDER BY 機能を使用していない場合、行の順序を重要視しません。そのため、同じ DEPTNO 値を持つ行がいくつかある場合には、 2 番目の SELECT ステートメントが最初のものとは違う順序で行を検索する場合があります。保証されているのは、ORDER BY DEPTNO 文節での要求に従って、それらすべてが部門番号の順に並べられるということだけです。

同じ SQL ステートメントを同じホスト変数を指定して 2 度実行したとしても、順序付けが異なる場合があります。たとえば、2 度目の実行がなされるまでの間にカタログの統計が更新されたり、索引が作成されるかドロップされる場合もあります。その後で SELECT ステートメントをもう一度実行することも考えられます。

最初の SELECT が持っていなかった述部を 2 番目の SELECT が持っている場合、配列が変更することがあります。それはデータベース・マネージャーが新しい述部に対して索引を使用するということがあり得るからです。たとえば、この例で、データベース・マネージャーが最初のステートメントに対しては LOCATION 上の索引を選び、 2 番目のものに対しては DEPTNO 上の索引を選ぶ場合があります。行は索引キーの順に従って取り出されるため、 2 番目の順序は最初の順序と同じとは限りません。

また、2 つの同様な SELECT ステートメントを実行したときに、統計が変更されず、索引の作成もドロップも行われなかったにもかかわらず、行の順序が異なる場合があります。例では、LOCATION の異なる値が多数ある場合、データベース・マネージャーは両方のステートメント用に LOCATION 上で 1 つの索引を選択することができます。 しかし、2 番目のステートメントの DEPTNO の値を次のように変えると、データベース・マネージャーは DEPTNO 上の索引を選ぶことがあります。

```
SELECT * FROM DEPARTMENT
  WHERE LOCATION = 'CALIFORNIA'
  AND DEPTNO >= 'Z98'
  ORDER BY DEPTNO
```

SQL ステートメントの形式とこのステートメントの値との間にはわずかな関係しかないため、順序が ORDER BY 文節で固有のものとして定められているのでない限り、 2 つの異なった SQL ステートメントが同じ順序で行を戻してくるとは考えないでください。

関連タスク:

- 109 ページの『データを 2 度検索する方法』

カーソルの位置を表の最後にする

カーソルの位置を表の最後にする必要がある場合、SQL ステートメントを使用して置くことができます。

手順:

カーソルを配置するメソッドとして、次のいずれかの例を使用してください。

- データベース・マネージャーは表内に保管されているデータを配列することは行いません。そのため、表の末尾は定義されていません。しかし、SQL ステートメントの結果としては順序が定義されます。

  ```
  SELECT * FROM DEPARTMENT
    ORDER BY DEPTNO DESC
  ```

- 次のステートメントは DEPTNO の値が最も高い行にカーソルを位置付けします。

  ```
  SELECT * FROM DEPARTMENT
    WHERE DEPTNO =
    (SELECT MAX(DEPTNO) FROM DEPARTMENT)
  ```

ただし、同じ値を持つ行がいくつかある場合には、カーソルはそれらのうちの最初の行に置かれることに注意してください。

以前に検索されたデータの更新

逆方向にスクロールして以前に検索されたデータを更新するには、以前に検索されたデータをスクロールする技法と検索されたデータを更新する技法を組み合わせて使用することができます。

手順:

以前に検索されたデータを更新するには、以下の 2 つの技法のいずれかを行うことができます。

- 更新するデータ上に 2 番目のカーソルがあり、 SELECT ステートメントが制限されたエレメントをまったく使用していない場合には、カーソル制御 UPDATE ステートメントを使用できる。 2 番目のカーソルを、WHERE CURRENT OF 文節の中で指名してください。
- それ以外の場合は、行の中のすべての値を指名するか、あるいは表の主キーを指定する WHERE 文節を伴った UPDATE を使用する。 1 つのステートメントを、変数の異なった値について何度でも実行することができます。

関連タスク:

- 105 ページの『静的 SQL プログラムにおける検索データの更新と削除』
- 108 ページの『以前に検索されたデータのスクロール』

静的 SQL プログラムにおける挿入、更新、および削除の例

次の例は、静的 SQL を使用してデータの挿入、更新、および削除を行う方法を示しています。

- C/C++ (**tut_mod.sqc/tut_mod.sqC**)

  次の 3 つの例は、サンプル **tut_mod** からの例です。C または C++ において表データを変更する方法を示している完全なプログラムについては、このサンプルを参照してください。

  以下に、表にデータを挿入する方法の例を示します。

```
EXEC SQL INSERT INTO staff(id, name, dept, job, salary)
        VALUES(380, 'Pearce', 38, 'Clerk', 13217.50),
              (390, 'Hachey', 38, 'Mgr', 21270.00),
              (400, 'Wagland', 38, 'Clerk', 14575.00);
```

  以下に、表のデータを更新する方法の例を示します。

```
EXEC SQL UPDATE staff
  SET salary = salary + 10000
  WHERE id >= 310 AND dept = 84;
```

  以下に、表からデータを削除する方法の例を示します。

```
EXEC SQL DELETE
  FROM staff
  WHERE id >= 310 AND salary > 20000;
```

- Java™ (**TutMod.sqlj**)

次の 3 つの例は、サンプル **TutMod** からの例です。SQLJ において表データを変更する方法を示している完全なプログラムについては、このサンプルを参照してください。

以下に、表にデータを挿入する方法の例を示します。

```
#sql {INSERT INTO staff(id, name, dept, job, salary)
        VALUES(380, 'Pearce', 38, 'Clerk', 13217.50),
              (390, 'Hachey', 38, 'Mgr', 21270.00),
              (400, 'Wagland', 38, 'Clerk', 14575.00)};
```

以下に、表のデータを更新する方法の例を示します。

```
#sql {UPDATE staff
        SET salary = salary + 1000
        WHERE id >= 310 AND dept = 84};
```

以下に、表からデータを削除する方法の例を示します。

```
#sql {DELETE FROM staff
        WHERE id >= 310 AND salary > 20000};
```

- COBOL (**updat.sqb**)

次の 3 つの例は、サンプル **updat** からの例です。COBOL において表データを変更する方法を示している完全なプログラムについては、このサンプルを参照してください。

以下に、表にデータを挿入する方法の例を示します。

```
EXEC SQL INSERT INTO staff
    VALUES (999, 'Testing', 99, :job-update, 0, 0, 0)
    END-EXEC.
```

以下に、表のデータを更新する方法の例を示します。

```
EXEC SQL UPDATE staff
    SET job=:job-update
    WHERE job='Mgr'
    END-EXEC.
```

以下に、表からデータを削除する方法の例を示します。

```
EXEC SQL DELETE
    FROM staff
    WHERE job=:job-update
    END-EXEC.
```

関連概念:

- 116 ページの『アプリケーション中でのエラー・メッセージの検索』

関連サンプル:

- 『tbinfo.out -- HOW TO GET INFORMATION AT THE TABLE LEVEL (C++)』
- 『tbmod.out -- HOW TO MODIFY TABLE DATA (C++)』
- 『tbmod.sqC -- How to modify table data (C++)』
- 『tut_mod.out -- HOW TO MODIFY TABLE DATA (C++)』
- 『tut_mod.sqC -- How to modify table data (C++)』
- 『tbmod.out -- HOW TO MODIFY TABLE DATA (C)』
- 『tbmod.sqc -- How to modify table data (C)』

- 『tut_mod.out -- HOW TO MODIFY TABLE DATA (C)』
- 『tut_mod.sqc -- How to modify table data (C)』
- 『TbMod.out -- HOW TO MODIFY TABLE DATA. Connect to 'sample' database using JDBC type 2 driver (SQLJ)』
- 『TbMod.sqlj -- How to modify table data (SQLj)』
- 『TutMod.out -- HOW TO MODIFY TABLE DATA. Connect to 'sample' database using JDBC type 2 driver (SQLJ)』
- 『TutMod.sqlj -- Modify data in a table (SQLj)』

診断情報

以下のセクションでは、戻りコードやアプリケーションがエラー・メッセージを検索する方法などの、静的 SQL プログラムで使用可能な診断情報について説明します。

戻りコード

ほとんどのデータベース・マネージャーは処理が正常であれば、ゼロの戻りコードを戻します。一般に、ゼロ以外の戻りコードは、 2 次エラー処理機構の SQLCA 構造体が破壊された可能性があることを示します。この場合、呼び出された API は実行されません。 SQLCA 構造体が破壊される原因として、構造体に対して無効なアドレスを渡したことが考えられます。

関連資料:

- 「*管理 API リファレンス*」の『SQLCA』

SQLCODE、SQLSTATE、および SQLWARN フィールドのエラー情報

エラー情報は、SQLCA 構造体の SQLCODE と SQLSTATE のフィールドに戻されます。SQLCA 構造体は、すべての実行可能 SQL ステートメントとほとんどのデータベース・マネージャー API 呼び出しの実行後に更新されます。

実行可能 SQL ステートメントが入っているソース・ファイルには、 `sqlca` という名前を持つ SQLCA 構造体が少なくとも 1 つあります。 SQLCA 構造体は SQLCA 組み込みファイルで定義されます。組み込み SQL ステートメントはないがデータベース・マネージャー API を呼び出すソース・ファイルには、 1 つまたは複数の SQLCA 構造体を組み込むことができますが、各構造体の名前は任意に付けられます。

ご使用のアプリケーションが FIPS 127-2 標準に準拠している場合、 SQLCA 構造体の代わりに SQLSTATE および SQLCODE を C、C++、COBOL、および FORTRAN アプリケーション用のホスト変数として宣言することができます。

SQLCODE 値が 0 である場合は、正常に実行されたことを示します (SQLWARN 警告状態を伴うこともあります)。正の値は、ステートメントは正常に実行されたが、ホスト変数の切り捨てなどの警告を伴うことを意味します。負の値は、エラー状態が起こったことを意味します。

追加のフィールド SQLSTATE には、他の IBM® データベース製品および SQL92 準拠のデータベース・マネージャーと一貫性がある標準化エラー・コードが含まれています。実際には、SQLSTATE 値は多くのデータベース・マネージャーと共通であるため、移植性を考慮する場合は SQLSTATE 値を使用します。

SQLWARN フィールドには、SQLCODE がゼロの場合でも警告標識の配列が含まれます。 SQLWARN 配列の第 1 エレメントである SQLWARN0 には、その他のエレメントがすべてブランクである場合はブランクが入ります。その他のエレメントの少なくとも 1 つに警告文字が含まれている場合は、 SQLWARN0 には `W` が入ります。

注: さまざまな IBM RDBMS サーバーにアクセスするアプリケーションを開発する場合は、以下のようにしてください。

- 可能な時点で、アプリケーションが SQLCODE ではなく SQLSTATE をチェックするようにする。
- アプリケーションが DB2 Connect を使用する場合は、 DB2 Connect に付属しているマッピング機能を用いて、異なるデータベース間の SQLCODE 変換をマップする。

関連概念:

- 114 ページの『戻りコード』
- 192 ページの『C および C++ における SQLSTATE および SQLCODE 変数』
- 218 ページの『COBOL での SQLSTATE および SQLCODE 変数』
- 235 ページの『FORTRAN の SQLSTATE および SQLCODE 変数』
- 537 ページの『Perl の SQLSTATE および SQLCODE 変数』

関連資料:

- 「*管理 API リファレンス*」の『SQLCA』

SQLCA 構造体におけるトークンの切り捨て

SQLCA 構造体ではトークンが切り捨てられることがあるため、診断目的ではトークン情報を使用しないでください。 表および列の名前は最高 128 バイトの長さで定義できますが、 SQLCA トークンは、17 バイトと切り捨て終止符 (>) を加えた長さで切り捨てられます。アプリケーションの論理は、 `sqlerrmc` フィールドの実際の値によって決めるべきではありません。

関連資料:

- 「*管理 API リファレンス*」の『SQLCA』

例外、シグナル、および割り込みハンドラーについての考慮事項

例外、シグナル、または割り込みハンドラーは、例外が起きたときに制御を獲得するルーチンです。ここで適用されるハンドラーのタイプは、オペレーティング環境によって異なります。以下のとおりです。

Windows® オペレーティング・システム
`Ctrl-C` または `Ctrl-Break` を押すことにより、割り込みが生成されます。

UNIX® ベースのシステム

通常、Ctrl-C を押すと SIGINT 割り込みシグナルが生成されます。キーボードは簡単に再定義できるため、 SIGINT はマシン上のさまざまなキー・シーケンスで生成される場合があることに注意してください。

例外、シグナル、および割り込みハンドラーの中には SQL ステートメントを置かないでください (COMMIT と ROLLBACK は例外)。これらのエラー状態が起きた場合には、データの矛盾を避けるために、 ROLLBACK するのが普通です。

例外/シグナル/割り込みハンドラーでの COMMIT および ROLLBACK のコーディングは、慎重に行ってください。これらのステートメントのいずれかをそれだけで呼び出す場合、 COMMIT または ROLLBACK は現行の SQL ステートメントが完了するまで実行されません (SQL ステートメントが実行中の場合)。これは、Ctrl-C ハンドラーから実行するには望ましい動作ではありません。

ROLLBACK を発行する前に、 INTERRUPT API (sqleintr/sqlgintr) を呼び出すことで解決できます。 この API は、現行の SQL 照会を中断させ (アプリケーションが SQL 照会を実行中の場合)、 ROLLBACK が即時に開始されるようにします。 ROLLBACK よりも COMMIT を実行する場合、現行のコマンドを中断する必要はありません。

APPC を使用してリモート・データベース・サーバー (DB2 (AIX 版) または DB2 Connect を使用したホスト・データベース・システム) にアクセスする場合に、アプリケーションは SIGUSR1 シグナルを受信することがあります。このシグナルは、リカバリー不可能エラーが発生したり SNA 接続が停止したときに、 SNA サービス/6000 により生成されます。 SIGUSR1 を処理するには、シグナル・ハンドラーをアプリケーションにインストールする必要があります。

さまざまなハンドラーに特有の詳細な考慮事項については、ご使用のプラットフォームの資料を参照してください。

関連概念:

- 761 ページの『割り込み要求の処理』

出口リスト・ルーチンに関する考慮事項

出口リスト・ルーチンでは、SQL や DB2 API 呼び出しを使用しないでください。出口ルーチンの中ではデータベースから切断することはできないことに注意してください。

アプリケーション中でのエラー・メッセージの検索

アプリケーションが書かれている言語により、エラー情報を検索するメソッドが異なります。

- C、C++、および COBOL アプリケーションでは渡される SQLCA に関する情報を GET ERROR MESSAGE API を使って得られます。
- JDBC および SQLJ アプリケーションは、 SQL 処理中にエラーが発生すると、SQLException を throw します。使用中のアプリケーションは SQLException を受け取り、以下のコードでそれを表示します。

```
try {
   Statement stmt = connection.createStatement();
   int rowsDeleted = stmt.executeUpdate(
      "DELETE FROM employee WHERE empno = '000010'");
   System.out.println( rowsDeleted + " rows were deleted");
}

catch (SQLException sqle) {
   System.out.println(sqle);
}
```

- REXX アプリケーションは CHECKERR プロシージャーを使用します。

関連概念:

- 192 ページの『C および C++ における SQLSTATE および SQLCODE 変数』
- 218 ページの『COBOL での SQLSTATE および SQLCODE 変数』
- 235 ページの『FORTRAN の SQLSTATE および SQLCODE 変数』
- 537 ページの『Perl の SQLSTATE および SQLCODE 変数』

関連資料:

- 「*管理 API リファレンス*」の『sqlaintp - エラー・メッセージの入手』

第 5 章 動的 SQL プログラムの作成

動的 SQL を使用する場合の特性とそれを使用する理由 119
動的 SQL を使用する理由 119
動的 SQL サポート・ステートメント 120
動的 SQL と静的 SQL 121
動的 SQL におけるカーソル 123
動的 SQL プログラムにおけるカーソルの宣言と使用 123
動的 SQL プログラムにおけるカーソルの例 . . 124
| 動的 SQL における REOPT の影響 126
動的 SQL における DYNAMICRULES BIND オプションの影響 126
動的 SQL プログラムにおける SQLDA. 128
動的 SQL におけるホスト変数と SQLDA . . . 128
動的 SQL プログラムにおける SQLDA 構造体の宣言. 129
最小の SQLDA 構造体を用いた動的 SQL におけるステートメントの準備 131
十分な数の SQLVAR 項目を指定した動的 SQL プログラム用の SQLDA の割り振り. 133
動的 SQL プログラムにおける SELECT ステートメントの記述. 134
行を保持するためのストレージの獲得 134
動的 SQL プログラムにおけるカーソルの処理 135
動的 SQL プログラムにおける SQLDA 構造体の割り振り 136
SQLDA 構造体を使用した動的 SQL プログラムにおけるデータ転送 139
動的 SQL プログラムにおける対話式 SQL ステートメントの処理 140
動的 SQL プログラムにおけるステートメント・タイプの判別 140
動的 SELECT プログラムにおける可変リスト SQL ステートメントの処理. 141
エンド・ユーザーからの SQL 要求の保管. . . . 142
動的 SQL プログラムにおけるパラメーター・マーカー 142
パラメーター・マーカーを使用する動的 SQL への変数入力の提供 142
動的 SQL プログラムにおけるパラメーター・マーカーの例 143
DB2 コール・レベル・インターフェース (CLI) と動的 SQL との比較 145
DB2 コール・レベル・インターフェース (CLI) と組み込み動的 SQL の比較 145
組み込み SQL と比較した DB2 CLI の利点 . . 146
DB2 CLI または組み込み SQL をいつ使用するか 148

動的 SQL を使用する場合の特性とそれを使用する理由

以下のセクションでは、静的 SQL と比較して動的 SQL を使用するべき場合とその理由について説明します。

動的 SQL を使用する理由

動的 SQL は以下の場合に使用します。

- アプリケーションの実行時に SQL ステートメントの全体または一部を生成する必要がある。
- SQL ステートメントが参照するオブジェクトが、プリコンパイル時には存在しない。
- 現在のデータベース統計に基づき、常に最適なアクセス・パスをステートメントで使用したい。
- ステートメントのコンパイル環境を変更する (つまり、特殊レジスターを試してみる)。

関連概念:

- 120 ページの『動的 SQL サポート・ステートメント』
- 121 ページの『動的 SQL と静的 SQL』

動的 SQL サポート・ステートメント

動的 SQL サポート・ステートメントは、文字ストリング・ホスト変数とステートメント名を引き数として受け入れます。ホスト変数には、テキスト形式で動的に処理される SQL ステートメントが入っています。ステートメント・テキストは、アプリケーションのプリコンパイル時には処理されません。実際、ステートメント・テキストは、アプリケーションのプリコンパイル時には必要ありません。その代わりに、SQL ステートメントはプリコンパイルの段階でホスト変数と見なされ、その変数はアプリケーションを実行するときに参照されます。このような SQL ステートメントを*動的* SQL と呼びます。

動的 SQL サポート・ステートメントでは SQL テキストが入っているホスト変数を実行可能形式に変換し、ステートメント名を参照して操作しなければなりません。それらのステートメントは以下のとおりです。

EXECUTE IMMEDIATE
ホスト変数を使用しないステートメントを準備し実行します。アプリケーション内のすべての EXECUTE IMMEDIATE ステートメントは、ランタイムには同じ場所にキャッシュされるため、最後のステートメントのみが認識されます。このステートメントは PREPARE および EXECUTE ステートメントの代用として使います。

PREPARE
SQL ステートメントの文字ストリング式をステートメントの実行可能書式に変換し、ステートメント名を割り当て、オプションでそのステートメントに関する情報を SQLDA 構造体に入れます。

EXECUTE
前に準備した SQL ステートメントを実行します。このステートメントは、接続内で繰り返し実行することができます。

DESCRIBE
準備済みステートメントに関する情報を SQLDA に入れます。

アプリケーションは、ほとんどのサポートされている SQL ステートメントを動的に実行することができます。

注: 動的 SQL ステートメントの内容は、静的 SQL ステートメントの場合と同じ構文に従っていますが、以下の点が異なります。

- コメントは使用できない。
- ステートメントの先頭に EXEC SQL を使用してはならない。
- ステートメントをステートメント終止符で終了してはならない。これに対する例外は CREATE TRIGGER ステートメントで、このステートメントの場合はセミコロン (;) を入れることができます。

関連資料:

• 751 ページの『付録 A. サポートされる SQL ステートメント』

動的 SQL と静的 SQL

パフォーマンスのために静的 SQL または動的 SQL のどちらを使用するかは、プログラマーにとって常に関心の高い問題です。この問題の答えは状況によります。

静的 SQL と動的 SQL のどちらを使用するかを決めるときには、次の表を使用してください。セキュリティーなどの考慮事項では静的 SQL が必要になりますし、一方、環境 (たとえば DB2 CLI または CLP を使用する場合) を考慮すると、動的 SQL が必要になります。決定する際には、ある特定の状況下で静的 SQL または動的 SQL のどちらを選んだらよいかに関して、以下の提案を考慮してください。 次の表に「両方」とある場合は、静的 SQL と動的 SQL のどちらでも変わりがないことを示します。

注: この情報はあくまでも一般的な提案に過ぎません。どちらを選ぶにしても、ご使用のアプリケーションの本来の用途や作業環境を考慮に入れる必要があります。はっきりしない場合は、まずステートメントを静的 SQL としてプロトタイプ化してから、次に動的 SQL としてプロトタイプ化し、その違いを比較することが最善の方法です。

表 8. 静的 SQL と動的 SQL の比較

考慮事項	推奨 SQL
SQL ステートメントを実行する時間	
• 2 秒未満	• 静的
• 2〜10 秒	• 両方
• 11 秒以上	• 動的
データの均一性	
• データ分散が均一	• 静的
• やや不均一	• 両方
• 非常に不均一な分散	• 動的
範囲 (<、>、BETWEEN、LIKE) 述部	
• 非常に少ない	• 静的
• 随時	• 両方
• 頻繁	• 動的
繰り返し実行	
• 何回も実行 (10 回以上)	• 両方
• 少数回実行 (10 回未満)	• 両方
• 1 回だけ実行	• 静的
照会の種類	
• ランダム	• 動的
• 永久的	• 両方
ランタイム環境 (DML/DDL)	
• トランザクション処理 (DML のみ)	• 両方
• 混合 (DML および DDL - DDL はパッケージに影響を及ぼす)	• 動的
• 混合 (DML および DDL - DDL はパッケージに影響を及ぼさない)	• 両方
RUNSTATS の使用頻度	
• 非常にまれ	• 静的
• 普通	• 両方
• 頻繁	• 動的

一般に、動的 SQL を使用したアプリケーションは、使用前に SQL ステートメントをコンパイルする必要があるため、 SQL ステートメント当たりの始動 (初期) コストが大きくなります。コンパイルを行うと、動的 SQL の実行時間は静的 SQL の場合と同じですが、オプティマイザーがより優れたアクセス・プランを選択することによりもっと速くなることがあります。初期のコンパイル・コストは、動的ステートメントを実行するたびに低減されます。複数のユーザーが同じステートメントを用いて同じアプリケーションを実行しているなら、ステートメントを発行した最初のアプリケーションだけにステートメントのコンパイル・コストが生じます。

DML と DDL が混在している環境では、アプリケーションの実行中にシステムがステートメントを暗黙のうちに再コンパイルすることがあるため、動的 SQL ステートメントのコンパイル・コストが変わる可能性があります。混合環境では、静的 SQL と動的 SQL との間の選択はパッケージが無効にされる頻度にも影響してきます。 DDL によってパッケージが無効にされる場合は、動的 SQL のほうが便利でしょう。実際に実行する照会だけが次回の使用時に再コンパイルされ、その他の照会は再コンパイルされないからです。静的 SQL の場合は、いったん無効にされるとパッケージ全体が再バインドされます。

特定のアプリケーションが上記の特性の混合したものであり、一方の特性には静的 SQL が適しており、他方の特性には動的 SQL が適していることがあります。この場合、明確な決定方法はありませんので、最も慣れていて使用しやすい手法を用いてください。前の表の考慮事項は、重要性の高い順におおまかに掲載してあることに注意してください。

注: 静的および動的 SQL はそれぞれ、DB2 オプティマイザーにとって重要な 2 種類に分けられます。それらは以下のとおりです。

1. ホスト変数を含まない静的 SQL

   これは、以下の場合にしか見られない、まれな状態です。

   - *初期化* コード
   - 初心者トレーニング用の例

   実際これは、ランタイムのパフォーマンスのオーバーヘッドがなく、しかも DB2 オプティマイザーの機能を十分に活用しているため、パフォーマンスの観点からは最善の組み合わせです。

2. ホスト変数を含む静的 SQL

   これは DB2® アプリケーションの従来の*継承* スタイルです。 ステートメントのコンパイル中に獲得する PREPARE およびカタログ・ロックのランタイム・オーバーヘッドがなくなります。オプティマイザーは SQL ステートメント全体を認識しないため、残念ながら、オプティマイザーの全性能を活用することはできません。高度に均一化されていないデータ分散の場合には、特殊な問題があります。

3. パラメーター・マーカーを含まない動的 SQL

   これはランダム照会インターフェース (CLP など) 用の標準的なスタイルで、オプティマイザーに好まれる*タイプの* SQL です。複雑な照会の場合は、 PREPARE ステートメントによるオーバーヘッドは通常、実行時間の短縮により相殺されます。

4. パラメーター・マーカーを含む動的 SQL

   これは CLI アプリケーション用の最も一般的なタイプの SQL です。主な利点は、パラメーター・マーカーがあるために、 PREPARE ステートメント (主に選択または挿入) を繰り返し実行するうちに、 PREPARE ステートメントのコストを償却できるという点です。この償却は、すべての反復性のある動的 SQL アプリケーションに当てはまります。残念なことに、ホスト変数を含む静的 SQL と同様に、 DB2 オプティマイザーの一部は、情報のすべてを利用することができないために作動しません。 最も効率的な方法としては、*ホスト変数を指定して静的 SQL を使用すること* または*パラメーター・マーカーなしで動的 SQL を使用すること* をお勧めします。

関連概念:

- 143 ページの『動的 SQL プログラムにおけるパラメーター・マーカーの例』

関連タスク:

- 142 ページの『パラメーター・マーカーを使用する動的 SQL への変数入力の提供』

動的 SQL におけるカーソル

以下のセクションでは、動的 SQL におけるカーソルの宣言方法と使用方法について説明し、カーソルを使用するサンプル・プログラムについても簡単に説明します。

動的 SQL プログラムにおけるカーソルの宣言と使用

カーソルを動的に処理することは、静的 SQL を用いてカーソルを処理することとほとんど同じです。カーソルを宣言する際に、カーソルは照会と関連付けられます。

静的 SQL では照会はテキストの SELECT ステートメントですが、動的 SQL の場合は照会は PREPARE ステートメントで割り当てられたステートメント名と関連付けられます。参照したホスト変数はパラメーター・マーカーで表されます。

静的カーソルと動的カーソルの主な相違点は、静的カーソルがプリコンパイル時に準備されるのに対して、動的カーソルはランタイムに準備されることです。さらに、照会で参照されるホスト変数はパラメーター・マーカーで表され、カーソルをオープンする際にランタイム・ホスト変数で置き換えられます。

手順:

動的 SQL プログラム用のカーソルをコーディングするときには、次の表にある例を使用してください。

表 9. 動的 SELECT に関連付けられた DECLARE ステートメント

言語	ソース・コード例

C/C++	`strcpy( prep_string, "SELECT tabname FROM syscat.tables"`
	`"WHERE tabschema = ?" );`
	`EXEC SQL PREPARE s1 FROM :prep_string;`
	`EXEC SQL DECLARE c1 CURSOR FOR s1;`
	`EXEC SQL OPEN c1 USING :host_var;`

Java (JDBC)	`PreparedStatement prep_string = ("SELECT tabname FROM syscat.tables`
	`WHERE tabschema = ?" );`
	`prep_string.setCursor("c1");`
	`prep_string.setString(1, host_var);`
	`ResultSet rs = prep_string.executeQuery();`

COBOL	`MOVE "SELECT TABNAME FROM SYSCAT.TABLES WHERE TABSCHEMA = ?"`
	`TO PREP-STRING.`
	`EXEC SQL PREPARE S1 FROM :PREP-STRING END-EXEC.`
	`EXEC SQL DECLARE C1 CURSOR FOR S1 END-EXEC.`
	`EXEC SQL OPEN C1 USING :host-var END-EXEC.`

FORTRAN	`prep_string = 'SELECT tabname FROM syscat.tables WHERE tabschema = ?'`
	`EXEC SQL PREPARE s1 FROM :prep_string`
	`EXEC SQL DECLARE c1 CURSOR FOR s1`
	`EXEC SQL OPEN c1 USING :host_var`

関連概念:

- 124 ページの『動的 SQL プログラムにおけるカーソルの例』
- 549 ページの『REXX でのカーソル』

関連タスク:

- 100 ページの『カーソルを用いた複数行の選択』

動的 SQL プログラムにおけるカーソルの例

動的 SQL ステートメントは、PREPARE ステートメントにより実行用に準備できますし、EXECUTE ステートメントまたは DECLARE CURSOR ステートメントで実行できます。

EXECUTE のある PREPARE

次の例はどのようにして、動的 SQL ステートメントを PREPARE ステートメントにより実行用に準備し、EXECUTE ステートメントで実行できるようにするのかを示しています。

- C/C++ (**dbuse.sqc/dbuse.sqC**):

  次の例は、サンプル **dbuse** からの例です。

```
EXEC SQL BEGIN DECLARE SECTION;
  char hostVarStmt[50];
EXEC SQL END DECLARE SECTION;

strcpy(hostVarStmt, "DELETE FROM org WHERE deptnumb = 15");
EXEC SQL PREPARE Stmt FROM :hostVarStmt;
EXEC SQL EXECUTE Stmt;
```

DECLARE CURSOR のある PREPARE

次の例はどのようにして、動的 SQL ステートメントを PREPARE ステートメントにより実行用に準備し、DECLARE CURSOR ステートメントで実行できるようにするのかを示しています。

- C

```
EXEC SQL BEGIN DECLARE SECTION;
  char  st[80];
  char  parm_var[19};
EXEC SQL END DECLARE SECTION;

strcpy( st, "SELECT tabname FROM syscat.tables" );
strcat( st, " WHERE tabname <> ? ORDER BY 1" );
EXEC SQL PREPARE s1 FROM :st;
EXEC SQL DECLARE c1 CURSOR FOR s1;
strcpy( parm_var, "STAFF" );
EXEC SQL OPEN c1 USING :parm_var;
```

- Java™

```
PreparedStatement pstmt1 = con.prepareStatement(
  "SELECT tabname FROM syscat.tables " +
  "WHERE tabname <> ? ORDER BY 1");

// set cursor name for the positioned update statement
pstmt1.setCursorName("c1");
pstmt1.setString(1, "STAFF");
ResultSet rs = pstmt1.executeQuery();
```

- COBOL **(dynamic.sqb)**

  次の例は、サンプル **dynamic.sqb** からの例です。

```
EXEC SQL BEGIN DECLARE SECTION END-EXEC.
     01 st              pic x(80).
     01 parm-var        pic x(18).
EXEC SQL END DECLARE SECTION END-EXEC.

move "SELECT TABNAME FROM SYSCAT.TABLES ORDER BY 1 WHERE TABNAME <> ?" to st.
EXEC SQL PREPARE s1 FROM :st END-EXEC.

EXEC SQL DECLARE c1 CURSOR FOR s1 END-EXEC.

move "STAFF" to parm-var.
EXEC SQL OPEN c1 USING :parm-var END-EXEC.
```

EXECUTE IMMEDIATE

また、動的 SQL を EXECUTE IMMEDIATE ステートメントにより準備し実行することもできます (ただし、 SELECT ステートメントは複数の行を戻すので例外です)。

- C/C++ (**dbuse.sqc/dbuse.sqC**)

  次の例は、サンプル **dbuse** 中の関数 DynamicStmtEXECUTE_IMMEDIATE() からの例です。

```
EXEC SQL BEGIN DECLARE SECTION;
  char stmt1[50];
EXEC SQL END DECLARE SECTION;

strcpy(stmt1, "CREATE TABLE table1(col1 INTEGER)");
EXEC SQL EXECUTE IMMEDIATE :stmt1;
```

関連概念:

- 116 ページの『アプリケーション中でのエラー・メッセージの検索』

関連サンプル:

- 『dbuse.out -- HOW TO USE A DATABASE (C)』
- 『dbuse.sqc -- How to use a database (C)』
- 『dbuse.out -- HOW TO USE A DATABASE (C++)』
- 『dbuse.sqC -- How to use a database (C++)』

動的 SQL における REOPT の影響

REOPT ALWAYS オプションを指定している場合、 DB2® は、OPEN または EXECUTE ステートメントを検出するまで、つまり、これらの変数の値が分かるときまで、ホスト変数、パラメーター・マーカー、または特殊レジスターを含むステートメントの準備を延期します。その時点で、これらの値を使用してアクセス・プランが生成されます。同じステートメントに対するその後の OPEN または EXECUTE 要求では、ステートメントが再コンパイルされ、変数の現行の値のセットを使用して照会プランが再最適化され、新しく生成された照会プランが実行されます。

REOPT ONCE オプションにも同様の効果がありますが、異なる点として、プランがホスト変数、パラメーター・マーカー、特殊レジスターの値で 1 回だけ最適化されます。このプランはキャッシュされ、その後の要求で使用されます。

動的 SQL における DYNAMICRULES BIND オプションの影響

PRECOMPILE および BIND の DYNAMICRULES オプションにより、次の動的 SQL 属性にランタイムに適用される値を決定できます。

- 許可検査中に使用される許可 ID。
- 非修飾オブジェクトの修飾に使用される修飾子。
- GRANT、 REVOKE、 ALTER、 CREATE、 DROP、 COMMENT ON、 RENAME、 SET INTEGRITY および SET EVENT MONITOR STATE ステートメントを動的に準備するためにパッケージを使用できるかどうか。

DYNAMICRULES 値に加えてパッケージのランタイム環境が、動的 SQL ステートメントがランタイムにどのように振る舞うかを制御します。次の 2 つのランタイム環境が考えられます。

- パッケージはスタンドアロン・プログラムの一部として実行される。
- パッケージはルーチン・コンテキスト内で実行される。

DYNAMICRULES 値とランタイム環境の組み合わせで動的 SQL 属性の値が決まります。その属性値の集合が、動的 SQL ステートメントの振る舞いと呼ばれます。次のような 4 つの振る舞いがあります。

実行振る舞い 動的 SQL ステートメントの許可検査に使用する値および動的 SQL ステートメント内の非修飾オブジェクト参照の暗黙修飾に使用される初期値として、DB2® はパッケージを実行しているユーザーの許可 ID (最初に DB2 へ接続した ID) を使用します。

バインド振る舞い

ランタイムに DB2 は、静的 SQL に適用されるすべての規則を許可と修飾に使用します。つまり、パッケージ所有者の許可 ID を動的 SQL ステートメントの許可検査で使用される値とし、動的 SQL ステートメント内の非修飾オブジェクト参照の暗黙修飾に使用されるパッケージ・デフォルト修飾子にするということです。

定義振る舞い 定義振る舞いは、動的 SQL ステートメントがルーチン・コンテキスト内で実行されるパッケージに存在し、DYNAMICRULES DEFINEBIND または DYNAMICRULES DEFINERUN でバインドされている場合にのみ適用されます。 DB2 は、 (ルーチンのパッケージ・バインド・プログラムではなく) ルーチンの定義者の許可 ID を、動的 SQL ステートメントの許可検査で使用される値として使用し、動的 SQL ステートメント内の非修飾オブジェクト参照の暗黙修飾に使用されるパッケージ・デフォルト修飾として使用します。

呼び出し振る舞い

呼び出し振る舞いは、動的 SQL ステートメントがルーチン・コンテキスト内で実行されるパッケージに存在し、DYNAMICRULES INVOKEBIND または DYNAMICRULES INVOKERUN でバインドされている場合にのみ適用されます。 DB2 は、ルーチンが呼び出されたときに有効であった現行ステートメント許可 ID を、動的 SQL の許可検査で使用される値として使用し、そのルーチン内の動的 SQL 内で非修飾オブジェクト参照の暗黙修飾として使用します。これらのことを次の表にまとめます。

呼び出し環境	使用される ID
任意の静的 SQL	ルーチンを呼び出した SQL の入っているパッケージの OWNER の暗黙または明示的な値。
ビューまたはトリガー定義で使用	ビューまたはトリガーの定義者。
実行振る舞いパッケージからの動的 SQL	DB2 への初期接続を行うために使用された ID。
定義振る舞いパッケージからの動的 SQL	ルーチンを呼び出した SQL の入っているパッケージを使用しているルーチンの定義者。
呼び出し振る舞いパッケージからの動的 SQL	ルーチンを呼び出すカレント許可 ID。

次の表は、それぞれの動的 SQL 振る舞いを決定する DYNAMICRULES 値とランタイム環境の組み合わせを示します。

表 10. 動的 SQL ステートメントの振る舞いを決定する DYNAMICRULES とランタイム環境

DYNAMICRULES 値	スタンドアロン・プログラム環境における動的 SQL ステートメントの振る舞い	ルーチン環境における動的 SQL ステートメントの振る舞い
BIND	バインド振る舞い	バインド振る舞い
RUN	実行振る舞い	実行振る舞い
DEFINEBIND	バインド振る舞い	定義振る舞い

表 10. 動的 SQL ステートメントの振る舞いを決定する DYNAMICRULES とランタイム環境 (続き)

DYNAMICRULES 値	スタンドアロン・プログラム環境における動的 SQL ステートメントの振る舞い	ルーチン環境における動的 SQL ステートメントの振る舞い
DEFINERUN	実行振る舞い	定義振る舞い
INVOKEBIND	バインド振る舞い	呼び出し振る舞い
INVOKERUN	実行振る舞い	呼び出し振る舞い

次の表に、動的 SQL 振る舞いの各タイプ用の動的 SQL 属性値を示します。

表 11. 動的 SQL ステートメント振る舞いの定義

動的 SQL 属性	バインド振る舞いの動的 SQL 属性の設定	実行振る舞いの動的 SQL 属性の設定	定義振る舞いの動的 SQL 属性の設定	呼び出し振る舞いの動的 SQL 属性の設定
許可 ID	OWNER BIND オプションの暗黙または明示的な値	パッケージを実行するユーザーの ID	ルーチン定義者 (ルーチンのパッケージ所有者ではない)	ルーチンが呼び出されたときの現行ステートメント許可 ID
非修飾オブジェクトのデフォルト修飾子	QUALIFIER BIND オプションの暗黙または明示的な値	CURRENT SCHEMA 特殊レジスター	ルーチン定義者 (ルーチンのパッケージ所有者ではない)	ルーチンが呼び出されたときの現行ステートメント許可 ID
GRANT、REVOKE、ALTER、CREATE、DROP、COMMENT ON、RENAME、 SET INTEGRITY および SET EVENT MONITOR STATE の実行	不可	可	不可	不可

関連概念:

- 54 ページの『動的 SQL における許可に関する考慮事項』
- 「*アプリケーション開発ガイド サーバー・アプリケーションのプログラミング*」の『SQL の入ったルーチンの許可およびバインド』

動的 SQL プログラムにおける SQLDA

以下のセクションでは、動的 SQL プログラム用の SQLDA を宣言するときに適用される異なる考慮事項について説明します。

動的 SQL におけるホスト変数と SQLDA

静的 SQL を使用すると、組み込み SQL ステートメントで使用されているホスト変数は、アプリケーションのコンパイル時に認識されます。動的 SQL を使用した場合には、組み込み SQL ステートメントとその結果としてのホスト変数は、アプリ

ケーションを実行するまで認識されません。このように、動的 SQL アプリケーションの場合は、アプリケーションで使用するホスト変数のリストを扱う必要があります。 (PREPARE を使用して) 準備された SELECT ステートメントのホスト変数情報を得るためには、 DESCRIBE ステートメントを使用し、その情報を SQL 記述子域 (SQLDA) に保管することができます。

注: Java™ アプリケーションは SQLDA 構造を使用しないので、 PREPARE または DESCRIBE ステートメントも使用しません。 JDBC アプリケーションでは、`PreparedStatement` オブジェクトと `executeQuery()` メソッドを使って `ResultSet` オブジェクトを生成することができますが、これはホスト言語カーソルと同等です。 SQLJ アプリケーションでは、SQLJ イテレーター・オブジェクトを `CursorByPos` または `CursorByName` カーソルと共に宣言し、 FETCH ステートメントからデータを戻すこともできます。

アプリケーション内で DESCRIBE ステートメントを実行すると、データベース・マネージャーはホスト変数を SQLDA に定義します。ホスト変数を SQLDA に定義すると、 FETCH ステートメントにより、カーソルを使用してホスト変数に値を割り当てることができます。

関連概念:

- 124 ページの『動的 SQL プログラムにおけるカーソルの例』

関連資料:

- 「*SQL リファレンス 第 2 巻*」の『DESCRIBE ステートメント』
- 「*SQL リファレンス 第 2 巻*」の『FETCH ステートメント』
- 「*SQL リファレンス 第 2 巻*」の『PREPARE ステートメント』
- 「*管理 API リファレンス*」の『SQLDA』

動的 SQL プログラムにおける SQLDA 構造体の宣言

SQLDA には不定数の SQLVAR 項目のオカレンスがあり、次の図に示されているように、各 SQLVAR 項目は 1 データ行に 1 つの列を記述するフィールドの集まりで構成されます。 SQLVAR 項目には、基本 SQLVAR 項目と 2 次 SQLVAR 項目という 2 つのタイプがあります。

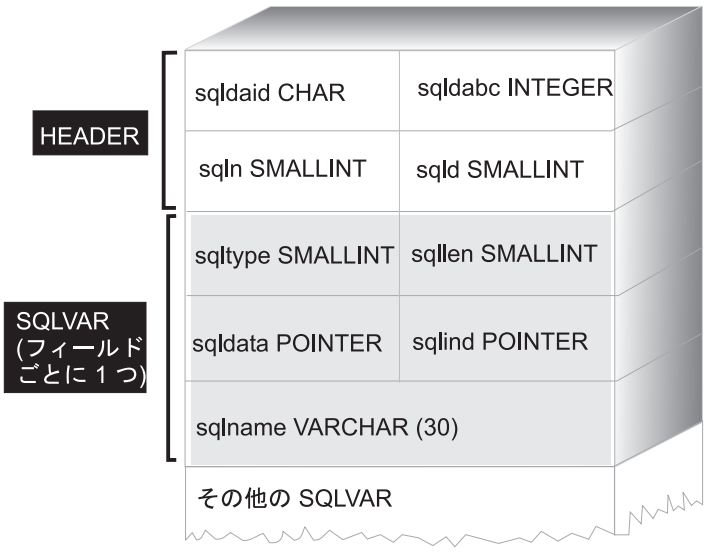


図 3. SQL 記述子域 (SQLDA)

手順:

必要とされる SQLVAR 項目の数は結果表の列の数によって決まるため、アプリケーションは必要に応じて適切な数の SQLVAR エレメントを割り振ることができなければなりません。 次のいずれかの方式を使用してください。

- 必要とされる最大の SQLDA (つまり、SQLVAR 項目の最大数を指定した SQLDA) を与える。結果表に戻すことができる列は最大 255 です。戻される列のいずれかが LOB タイプか独特なタイプの列である場合、SQLN の値は 2 倍になり、情報を入れるために必要な SQLVAR 項目の数も 2 倍の 510 になります。しかし、255 列も戻す SELECT ステートメントはほとんどないので、割り当てられたスペースの大部分が未使用となります。
- SQLVAR 項目を少なめに指定して、小さめの SQLDA を与える。この場合、結果表に SQLDA で使用できる SQLVAR 項目より多い列が入っていると、記述は戻されません。代わりに、データベース・マネージャーは SELECT ステートメントで検出される選択リスト項目の数を戻します。アプリケーションは SQLDA に SQLVAR の必要数を割り当ててから、 DESCRIBE ステートメントを使用して列記述を入手します。

上記のいずれの方式の場合でも、最初にいくつの SQLVAR 項目を割り当てればよいかという疑問が生じます。各 SQLVAR エレメントは、最高 44 バイトのストレージを使用します (SQLDATA および SQLIND フィールドに割り振られるストレージはカウントしていません)。メモリーに十分余裕があれば、 SQLDA の最大サイズを割り振るという最初の方法を実行するのが簡単です。

より小さい SQLDA を割り当てるという 2 番目の方法は、メモリーの動的割り振りをサポートする C および C++ のようなプログラミング言語にしか適用できません。メモリーの動的割り振りをサポートしない COBOL や FORTRAN のような言語の場合、最初の方法を使用することが必要です。

関連タスク:

- 131 ページの『最小の SQLDA 構造体を用いた動的 SQL におけるステートメントの準備』
- 133 ページの『十分な数の SQLVAR 項目を指定した動的 SQL プログラム用の SQLDA の割り振り』
- 136 ページの『動的 SQL プログラムにおける SQLDA 構造体の割り振り』

関連資料:

- 「*管理 API リファレンス*」の『SQLDA』

最小の SQLDA 構造体を用いた動的 SQL におけるステートメントの準備

ここにある情報を、最小の SQLDA 構造体をステートメント用に割り振る例として用いてください。

制約事項:

メモリーの動的割り振りをサポートする C および C++ のようなプログラミング言語でしか、より小さい SQLDA 構造体を割り当てることはできません。

手順:

アプリケーションが、 SQLVAR 項目の入っていない `minsqlda` という名前の SQLDA 構造体を宣言する場面を例にとって考えてみましょう。 SQLDA の SQLN フィールドは割り振られる SQLVAR 項目の数を記述します。この場合、SQLN は 0 にセットされていなければなりません。次に、文字ストリング `dstring` から 1 つのステートメントを準備し、そしてその記述を `minsqlda` の中に入力するには、次の SQL ステートメントを発行します。 (C 構文を使用し、 `minsqlda` が SQLDA 構造体へのポインターとして宣言されているものとします。)

```
EXEC SQL
  PREPARE STMT INTO :*minsqlda FROM :dstring;
```

`dstring` に含まれるステートメントが、各行に 20 列を戻す SELECT ステートメントであるとします。 PREPARE ステートメント (または DESCRIBE ステートメント) の後の SQLDA の SQLD フィールドには、準備済み SELECT ステートメントの結果行の列数が入っています。

SQLDA の SQLVAR 項目は、以下の場合に設定されます。

- SQLN >= SQLD であり、かつどの列も LOB または特殊タイプではない場合

  最初の SQLD SQLVAR 項目が設定され、SQLDOUBLED はブランクに設定されます。

- SQLN >= 2*SQLD であり、かつ少なくとも 1 つの列が LOB または特殊タイプである場合

  2* SQLD SQLVAR 項目が設定され、SQLDOUBLED は 2 に設定されます。

- SQLD <= SQLN < 2*SQLD であり、少なくとも 1 つの列が特殊タイプであり、かつ LOB の列はない場合

最初の SQLD SQLVAR 項目が設定され、SQLDOUBLED はブランクに設定されます。 SQLWARN BIND オプションが YES の場合は、警告 SQLCODE +237 (SQLSTATE 01594) が発行されます。

以下の場合には、SQLDA 内の SQLVAR 項目は設定されません (追加スペースを割り振り、 DESCRIBE をもう一度指定するよう要求されます)。

- SQLN < SQLD であり、どの列も LOB または特殊タイプではない場合

  SQLVAR 項目は設定されず、SQLDOUBLED はブランクに設定されます。 SQLWARN BIND オプションが YES の場合は、警告 SQLCODE +236 (SQLSTATE 01005) が発行されます。

  DESCRIBE が正常に実行される場合には、SQLD 個の SQLVAR 項目が割り振られます。
- SQLN < SQLD であり、少なくとも 1 つの列が特殊タイプであり、かつ LOB の列はない場合

  SQLVAR 項目は設定されず、SQLDOUBLED はブランクに設定されます。 SQLWARN BIND オプションが YES の場合は、警告 SQLCODE +239 (SQLSTATE 01005) が発行されます。

  特殊タイプの名前を含む DESCRIBE を正常に実行するためには、 2*SQLD SQLVAR 項目を割り振ってください。
- SQLN < 2*SQLD であり、かつ少なくとも 1 つの列が LOB である場合

  SQLVAR 項目は設定されず、SQLDOUBLED はブランクに設定されます。警告 SQLCODE +238 (SQLSTATE 01005) が発行されます (SQLWARN BIND オプションの設定に関係なく)。

  DESCRIBE を正常に実行するためには、2*SQLD SQLVAR 項目を割り振ってください。

BIND コマンドの SQLWARN オプションは、 DESCRIBE (または PREPARE...INTO) が以下の警告を戻すかどうかを制御します。

- SQLCODE +236 (SQLSTATE 01005)
- SQLCODE +237 (SQLSTATE 01594)
- SQLCODE +239 (SQLSTATE 01005)

アプリケーション・コードでは、これらの SQLCODE 値が戻される可能性のあることを常に考慮に入れておいてください。警告 SQLCODE +238 (SQLSTATE 01005) は、選択リストに LOB 列があり SQLDA に不適当な SQLVAR 項目がある場合に、必ず戻されます。これは、結果セット内に LOB 列があるために SQLVAR 項目の数が 2 倍になっていなければならないということをアプリケーションが認識できる唯一の方法です。

関連タスク:

- 129 ページの『動的 SQL プログラムにおける SQLDA 構造体の宣言』
- 133 ページの『十分な数の SQLVAR 項目を指定した動的 SQL プログラム用の SQLDA の割り振り』

- 136 ページの『動的 SQL プログラムにおける SQLDA 構造体の割り振り』

十分な数の SQLVAR 項目を指定した動的 SQL プログラム用の SQLDA の割り振り

結果表の列数が決まったら、ストレージを 2 番目の、フルサイズの SQLDA に割り振ってください。

手順:

結果表の列が 20 あるとします (すべて LOB 列ではないとします)。この場合、 2 番目の SQLDA 構造体である `fulsqlda` には、少なくとも 20 の SQLVAR エレメント (または結果表に LOB または特殊タイプがある場合には 40 エレメント) を割り振らなければなりません。この例のその他の部分では、LOB または特殊タイプは結果表に含まれないものとします。

SQLDA 構造体のストレージ要件を計算するときには、以下のものを含めてください。

- 長さ 16 バイトの固定長ヘッダー (SQLN および SQLD などのフィールドを含む)。
- SQLVAR 項目の可変長配列、それぞれのエレメントは長さが 44 バイト (32 ビット・プラットフォームの場合) または 56 バイト (64 ビット・プラットフォームの場合)。

`fulsqlda` に必要な SQLVAR 項目の数は、 `minsqlda` という SQLD フィールドに指定されます。 その値が 20 であるとします。そのため、`fulsqlda` に必要なストレージ割り振りは、以下のようになります。

```
16 + (20 * sizeof(struct sqlvar))
```

注: 64 ビット・プラットフォームの場合、 `sizeof(struct sqlvar)` および `sizeof(struct sqlvar2)` は `56` を戻します。 32 ビット・プラットフォームの場合、 `sizeof(struct sqlvar)` および `sizeof(struct sqlvar2)` は `44` を戻します。

この値は、ヘッダーのサイズに各 SQLVAR 項目のサイズの 20 倍を加えて、合計 896 バイトであることを表しています。

SQLDASIZE マクロを使用することにより、自分で計算をしないようにすれば、バージョン間の違いをすべて回避することができます。

関連タスク:

- 129 ページの『動的 SQL プログラムにおける SQLDA 構造体の宣言』
- 131 ページの『最小の SQLDA 構造体を用いた動的 SQL におけるステートメントの準備』
- 136 ページの『動的 SQL プログラムにおける SQLDA 構造体の割り振り』

動的 SQL プログラムにおける SELECT ステートメントの記述

十分な量のスペースを 2 番目の SQLDA (この例の場合 fulsqlda と呼ばれる) に割り振ってから、アプリケーションをコーディングして SELECT ステートメントを記述しなければなりません。

手順:

次のステップを実行するようにアプリケーションをコーディングしてください。

1. fulsqlda の SQLN フィールドの値を 20 にする (この例では結果表の列は 20 であり、すべての列は LOB 列ではないものとしています)。
2. 2 番目の SQLDA 構造体 fulsqlda を用いて SELECT ステートメントに関する情報を入手する。 これには、次の 2 とおりの方法があります。
   - minsqlda の代わりに fulsqlda を指定する別の PREPARE ステートメントを使用する。
   - fulsqlda を指定する DESCRIBE ステートメントを使用する。

DESCRIBE ステートメントを使用するとステートメントを 2 回準備するコストが省けるため、この方法のほうが好んで使用されます。 DESCRIBE ステートメントは、準備操作中に入手した前の情報を再度使用するだけで、その情報を新規の SQLDA 構造体に入れます。次のステートメントを発行することができます。

```
EXEC SQL DESCRIBE STMT INTO :fulsqlda
```

このステートメントを実行すると、それぞれの SQLVAR エレメントには結果表の 1 つの列の記述が含まれるようになります。

関連タスク:

- 134 ページの『行を保持するためのストレージの獲得』

行を保持するためのストレージの獲得

先にアプリケーションが行のためのストレージを割り振ってからでなければ、SQLDA 構造体を使用して結果表からアプリケーション行を取り出すことはできません。

手順:

次のステップを実行するようにアプリケーションをコーディングしてください。

1. それぞれの SQLVAR 記述を分析して、その列の値に必要なスペースの量を判別する。

   SELECT が記述されている場合、LOB の値について SQLVAR に指定されるデータ・タイプは SQL_TYP_*x*LOB であることに注意してください。このデータ・タイプは一般的な LOB ホスト変数と同じであり、すべての LOB は 1 回でメモリーに保管されます。これは (数 MB までの) 小さい LOB の場合に作動しますが、このデータ・タイプを大きい LOB (1 GB など) に使用することはできません。 SQLVAR 内のアプリケーションの列定義を変更して、SQL_TYP_*x*LOB_LOCATOR または SQL_TYPE_*x*LOB_FILE のいずれかにすることが必要になります。 (SQLVAR の SQLTYPE フィールドを変更する場合に

は、 SQLLEN フィールドも変更する必要があるので注意してください。)
SQLVAR 内の列定義を変更すると、アプリケーションではその新しいタイプに対して正しい容量のストレージを割り振ることができます。

2. その列の値にストレージを割り振る。
3. 割り振ったストレージのアドレスを SQLDA 構造体の SQLDATA フィールドに保管する。

これらのステップは、各列の記述を分析し、それぞれの SQLDATA フィールドの内容をその列の値を保持するだけの大きさをもつストレージと置き換えることによって行われます。長さ属性は、LOB タイプでないデータ項目に対する各 SQLVAR 項目の SQLLEN フィールドから判別されます。タイプが BLOB、CLOB、または DBCLOB の項目の場合、長さ属性は 2 番目の SQLVAR 項目の SQLLONGLEN フィールドから判別されます。

さらに、指定した列に NULL を使用できる場合、アプリケーションは SQLIND フィールドの内容を列の標識変数のアドレスと置き換えなければなりません。

関連概念:

- 「*アプリケーション開発ガイド サーバー・アプリケーションのプログラミング*」の『ラージ・オブジェクトの使用法』

関連タスク:

- 135 ページの『動的 SQL プログラムにおけるカーソルの処理』

動的 SQL プログラムにおけるカーソルの処理

SQLDA 構造の割り振りが適切に行われると、 SELECT ステートメントに関連するカーソルをオープンし、行を取り出すことができます。

手順:

SELECT ステートメントに関連したカーソルを処理するには、最初にカーソルをオープンし、次に FETCH ステートメントの USING DESCRIPTOR 文節を指定して行を取り出します。たとえば、C アプリケーションでは次のようにします。

```
EXEC SQL OPEN pcurs
EMB_SQL_CHECK( "OPEN" ) ;
EXEC SQL FETCH pcurs USING DESCRIPTOR :*sqldaPointer
EMB_SQL_CHECK( "FETCH" ) ;
```

FETCH が成功したならば SQLDA からデータを獲得し列見出しを表示するようなアプリケーションを書けるでしょう。たとえば、以下のようにします。

```
display_col_titles( sqldaPointer ) ;
```

データを表示したなら、カーソルをクローズし動的に割り振ったメモリーを解放しなければなりません。たとえば、以下のようにします。

```
EXEC SQL CLOSE pcurs ;
EMB_SQL_CHECK( "CLOSE CURSOR" ) ;
```

動的 SQL プログラムにおける SQLDA 構造体の割り振り

データのやり取りで使用できるように、アプリケーションで使用する SQLDA 構造体を割り振ってください。

手順:

C 言語で SQLDA 構造体を作成するには、ホスト言語で INCLUDE SQLDA ステートメントを組み込むか、または SQLDA 組み込みファイルを組み込んで、構造体定義を入手してください。次に、SQLDA のサイズは固定されていないため、アプリケーションは SQLDA へのポインターを宣言し、それにストレージを割り振らなければなりません。 SQLDA 構造体の実際のサイズは、SQLDA を用いて渡される個別データ項目の数によって決まります。

C/C++ プログラム言語では、SQLDA の割り振りを簡単に行うためにマクロが提供されています。このマクロの形式は以下のとおりです (例外として、HP-UX プラットフォームの場合は形式が異なります)。

```
#define SQLDASIZE(n) (offsetof(struct sqlda, sqlvar) ¥
        + (n) × sizeof(struct sqlvar))
```

HP-UX プラットフォームの場合、このマクロの形式は以下のとおりです。

```
#define SQLDASIZE(n) (sizeof(struct sqlda) ¥
        + (n-1) × sizeof(struct sqlvar))
```

このマクロを使用することによって、 n 個の SQLVAR エレメントに必要なストレージを計算することができます。

COBOL で SQLDA 構造体を作成するには、 INCLUDE SQLDA ステートメントを組み込むか、または COPY ステートメントを使用します。 SQLVAR 項目の最大数を制御して SQLDA が使用するストレージの容量を制御したい場合は、 COPY ステートメントを使用してください。たとえば、SQLVAR 項目のデフォルトの数を 1489 から 1 に変更するには、以下の COPY ステートメントを使用します。

```
COPY "sqlda.cbl"
  replacing --1489--
  by --1--.
```

FORTRAN 言語では、自己定義データ構造または動的割り振りは直接にはサポートされていません。 SQLDA 組み込みファイルは FORTRAN では使用できません。これは、FORTRAN では SQLDA をデータ構造としてサポートできないためです。FORTRAN プログラムでは、プリコンパイラーは INCLUDE SQLDA ステートメントを無視します。

ただし、FORTRAN プログラムで静的 SQLDA 構造体に似た構造体を作成し、これを SQLDA を使用できる任意の場所で使用することができます。 sqldact.f ファイルには、FORTRAN で SQLDA 構造体を宣言するのに役立つ定数が含まれています。

ポインター値を必要とする SQLDA エレメントに値を割り当てるには、SQLGADDR の呼び出しを実行してください。

次の表は、SQLVAR エレメントを 1 つ持つ SQLDA 構造体の宣言および使用方法を示しています。

言語	ソース・コード例

C/C++	`#include <sqlda.h>`
	`struct sqlda *outda = (struct sqlda *)malloc(SQLDASIZE(1));`
	
	`/* DECLARE LOCAL VARIABLES FOR HOLDING ACTUAL DATA */`
	`double sal;`
	`double sal = 0;`
	`short salind;`
	`short salind = 0;`
	
	`/* INITIALIZE ONE ELEMENT OF SQLDA */`
	`memcpy( outda->sqldaid,"SQLDA   ",sizeof(outda->sqldaid));`
	`outda->sqln = outda->sqld = 1;`
	`outda->sqlvar[0].sqltype = SQL_TYP_NFLOAT;`
	`outda->sqlvar[0].sqllen  = sizeof( double );.`
	`outda->sqlvar[0].sqldata = (unsigned char *)&sal;`
	`outda->sqlvar[0].sqlind  = (short *)&salind;`

COBOL	`WORKING-STORAGE SECTION.`
	`77 SALARY          PIC S99999V99 COMP-3.`
	`77 SAL-IND         PIC S9(4)     COMP-5.`
	
	`EXEC SQL INCLUDE SQLDA END-EXEC`
	
	`* Or code a useful way to save unused SQLVAR entries.`
	`* COPY "sqlda.cbl" REPLACING --1489-- BY --1--.`
	
	`01 decimal-sqllen pic s9(4) comp-5.`
	`01 decimal-parts redefines decimal-sqllen.`
	`05 precision pic x.`
	`05 scale pic x.`
	
	`* Initialize one element of output SQLDA`
	`MOVE 1 TO SQLN`
	`MOVE 1 TO SQLD`
	`MOVE SQL-TYP-NDECIMAL TO SQLTYPE(1)`
	
	`* Length = 7 digits precision and 2 digits scale`
	
	`MOVE x"07" TO PRECISION.`
	`MOVE x"02" TO SCALE.`
	`MOVE DECIMAL-SQLLEN TO O-SQLLEN(1).`
	`SET SQLDATA(1) TO ADDRESS OF SALARY`
	`SET SQLIND(1)  TO ADDRESS OF SAL-IND`

言語	ソース・コード例

FORTRAN

```
      include 'sqldact.f'

      integer*2  sqlvar1
      parameter ( sqlvar1 = sqlda_header_sz + 0*sqlvar_struct_sz )

C     Declare an Output SQLDA -- 1 Variable
      character    out_sqlda(sqlda_header_sz + 1*sqlvar_struct_sz)

      character*8  out_sqldaid     ! Header
      integer*4    out_sqldabc
      integer*2    out_sqln
      integer*2    out_sqld

      integer*2    out_sqltype1    ! First Variable
      integer*2    out_sqllen1
      integer*4    out_sqldata1
      integer*4    out_sqlind1
      integer*2    out_sqlnamel1
      character*30 out_sqlnamec1

      equivalence( out_sqlda(sqlda_sqldaid_ofs), out_sqldaid )
      equivalence( out_sqlda(sqlda_sqldabc_ofs), out_sqldabc )
      equivalence( out_sqlda(sqlda_sqln_ofs), out_sqln       )
      equivalence( out_sqlda(sqlda_sqld_ofs), out_sqld       )
      equivalence( out_sqlda(sqlvar1+sqlvar_type_ofs), out_sqltype1 )
      equivalence( out_sqlda(sqlvar1+sqlvar_len_ofs), out_sqllen1   )
      equivalence( out_sqlda(sqlvar1+sqlvar_data_ofs), out_sqldata1 )
      equivalence( out_sqlda(sqlvar1+sqlvar_ind_ofs), out_sqlind1   )
      equivalence( out_sqlda(sqlvar1+sqlvar_name_length_ofs),
     +             out_sqlnamel1                                   )
      equivalence( out_sqlda(sqlvar1+sqlvar_name_data_ofs),
     +             out_sqlnamec1                                   )

C     Declare Local Variables for Holding Returned Data.
      real*8       salary
      integer*2    sal_ind

C     Initialize the Output SQLDA (Header)
      out_sqldaid  = 'OUT_SQLDA'
      out_sqldabc  = sqlda_header_sz + 1*sqlvar_struct_sz
      out_sqln     = 1
      out_sqld     = 1
C     Initialize VAR1
      out_sqltype1 = SQL_TYP_NFLOAT
      out_sqllen1  = 8
      rc = sqlgaddr( %ref(salary), %ref(out_sqldata1) )
      rc = sqlgaddr( %ref(sal_ind), %ref(out_sqlind1) )
```

動的なメモリー割り振りをサポートしない言語では、 SQLVAR エレメントの希望数を指定した SQLDA をホスト言語で明示的に宣言しなければなりません。SQLVAR のエレメントには、アプリケーションの必要に応じて決定されたとおりの十分な数を必ず宣言してください。

関連タスク:

- 131 ページの『最小の SQLDA 構造体を用いた動的 SQL におけるステートメントの準備』
- 133 ページの『十分な数の SQLVAR 項目を指定した動的 SQL プログラム用の SQLDA の割り振り』

• 139 ページの『SQLDA 構造体を使用した動的 SQL プログラムにおけるデータ転送』

SQLDA 構造体を使用した動的 SQL プログラムにおけるデータ転送

ホスト変数のリストを使用してデータを転送するよりも、 SQLDA を使用してデータを転送するほうが、より高い柔軟性が得られます。たとえば、SQLDA を用いて、ホスト言語に対応するものがないデータ (C 言語における DECIMAL データなど) でも転送することができます。

手順:

次の表を、数値とシンボル名がどのように関連付けられるかを示す相互参照リストとして使用してください。

表 12. DB2 SQLDA SQL タイプ: 数値および対応するシンボル名

SQL 列名	SQLTYPE 数値	SQLTYPE シンボル名[1]
DATE	384/385	SQL_TYP_DATE / SQL_TYP_NDATE
TIME	388/389	SQL_TYP_TIME / SQL_TYP_NTIME
TIMESTAMP	392/393	SQL_TYP_STAMP / SQL_TYP_NSTAMP
n/a[2]	400/401	SQL_TYP_CGSTR / SQL_TYP_NCGSTR
BLOB	404/405	SQL_TYP_BLOB / SQL_TYP_NBLOB
CLOB	408/409	SQL_TYP_CLOB / SQL_TYP_NCLOB
DBCLOB	412/413	SQL_TYP_DBCLOB / SQL_TYP_NDBCLOB
VARCHAR	448/449	SQL_TYP_VARCHAR / SQL_TYP_NVARCHAR
CHAR	452/453	SQL_TYP_CHAR / SQL_TYP_NCHAR
LONG VARCHAR	456/457	SQL_TYP_LONG / SQL_TYP_NLONG
n/a[3]	460/461	SQL_TYP_CSTR / SQL_TYP_NCSTR
VARGRAPHIC	464/465	SQL_TYP_VARGRAPH / SQL_TYP_NVARGRAPH
GRAPHIC	468/469	SQL_TYP_GRAPHIC / SQL_TYP_NGRAPHIC
LONG VARGRAPHIC	472/473	SQL_TYP_LONGRAPH / SQL_TYP_NLONGRAPH
FLOAT	480/481	SQL_TYP_FLOAT / SQL_TYP_NFLOAT
REAL[4]	480/481	SQL_TYP_FLOAT / SQL_TYP_NFLOAT
DECIMAL[5]	484/485	SQL_TYP_DECIMAL / SQL_TYP_DECIMAL
INTEGER	496/497	SQL_TYP_INTEGER / SQL_TYP_NINTEGER
SMALLINT	500/501	SQL_TYP_SMALL / SQL_TYP_NSMALL
n/a	804/805	SQL_TYP_BLOB_FILE / SQL_TYPE_NBLOB_FILE
n/a	808/809	SQL_TYP_CLOB_FILE / SQL_TYPE_NCLOB_FILE
n/a	812/813	SQL_TYP_DBCLOB_FILE / SQL_TYPE_NDBCLOB_FILE
n/a	960/961	SQL_TYP_BLOB_LOCATOR / SQL_TYP_NBLOB_LOCATOR
n/a	964/965	SQL_TYP_CLOB_LOCATOR / SQL_TYP_NCLOB_LOCATOR
n/a	968/969	SQL_TYP_DBCLOB_LOCATOR / SQL_TYP_NDBCLOB_LOCATOR

表 12. DB2 SQLDA SQL タイプ (続き): 数値および対応するシンボル名

SQL 列名	SQLTYPE 数値	SQLTYPE シンボル名[1]

注: これらの定義タイプは sql.h 組み込みファイルにあり、組み込みファイル自体は、 sqllib ディレクトリーの include サブディレクトリーにあります。 (たとえば、C プログラミング言語の場合は sqllib/include/sql.h となります。)

1. COBOL プログラミング言語の場合、SQLTYPE には下線 (_) を使用しませんが、その代わりにハイフン (-) を使用します。
2. これは NULL 終了 GRAPHIC ストリングです。
3. これは NULL 終了文字ストリングです。
4. SQLDA での REAL と DOUBLE の違いは長さの値です (4 または 8)。
5. 精度は最初のバイトにあります。位取りは 2 番目のバイトにあります。

関連タスク:

- 134 ページの『動的 SQL プログラムにおける SELECT ステートメントの記述』
- 134 ページの『行を保持するためのストレージの獲得』
- 135 ページの『動的 SQL プログラムにおけるカーソルの処理』

動的 SQL プログラムにおける対話式 SQL ステートメントの処理

動的 SQL を使用するアプリケーションを作成し、任意の SQL ステートメントを処理することができます。たとえば、アプリケーションがユーザーから SQL ステートメントを受け入れる場合、アプリケーションはステートメントについて事前にわかっていなくても、そのステートメントを実行できなければなりません。

手順:

PREPARE および DESCRIBE ステートメントを SQLDA 構造体で使用して、アプリケーションは実行される SQL ステートメントのタイプを判別し、それに応じて処理することができるようにしてください。

関連概念:

- 140 ページの『動的 SQL プログラムにおけるステートメント・タイプの判別』

動的 SQL プログラムにおけるステートメント・タイプの判別

SQL ステートメントを準備する場合、ステートメントのタイプに関する情報は SQLDA 構造体を調べて判別することができます。この情報はステートメントの準備時に INTO 文節を指定して SQLDA 構造体に入れるか、または事前に準備されたステートメントに対して DESCRIBE ステートメントを発行することによって、SQLDA 構造体に入れることができます。

いずれの場合でも、データベース・マネージャーは SQLDA 構造体の SQLD フィールドに SQL ステートメントにより生成された結果表の列数を示す値を入れます。 SQLD フィールドにゼロ (0) が入っている場合、このステートメントは SELECT ステートメントではありません。ステートメントはすでに準備されているため、 EXECUTE ステートメントを使用してただちに実行することができます。

ステートメントにパラメーター・マーカーが含まれている場合、USING 文節を指定しなければなりません。 USING 文節は、ホスト変数のリストか SQLDA 構造体のどちらかを指定することができます。

SQLD フィールドが 0 より大きい場合、ステートメントは SELECT ステートメントであるため、次の節での説明に従って処理しなければなりません。

関連資料:

- 「*SQL リファレンス 第 2 巻*」の『EXECUTE ステートメント』

動的 SELECT プログラムにおける可変リスト SQL ステートメントの処理

可変リスト SELECT ステートメントとは、戻される列の数およびタイプがプリコンパイル時にはわからないステートメントのことです。この場合、アプリケーションには、結果表の行を保持するために宣言しなければならない正確なホスト変数がわかりません。

手順:

可変リスト SELECT ステートメントを処理するには、次のステップを実行するようにアプリケーションをコーディングしてください。

1. SQLDA を宣言する。

   可変リスト SELECT ステートメントを処理するには、 SQLDA 構造体を必ず使用します。

2. INTO 文節を使用してステートメントを PREPARE (準備) する。

   アプリケーションは、宣言した SQLDA 構造体に十分な SQLVAR エレメントがあるかどうかを判別します。十分なエレメントがない場合、アプリケーションは必要な数の SQLVAR エレメントを持つ別の SQLDA 構造体を割り振り、新規の SQLDA を用いて追加の DESCRIBE ステートメントを発行します。

3. SQLVAR エレメントを割り振る。

   各 SQLVAR に必要なホスト変数および標識に、ストレージを割り振ります。このステップでは、それぞれの SQLVAR エレメントにデータの割り振りアドレスおよび標識変数を入れます。

4. SELECT ステートメントを処理する。

   カーソルは準備済みステートメントに関連付けられ、オープンされます。行は適切に割り振られた SQLDA 構造体を用いて取り出されます。

関連タスク:

- 129 ページの『動的 SQL プログラムにおける SQLDA 構造体の宣言』
- 131 ページの『最小の SQLDA 構造体を用いた動的 SQL におけるステートメントの準備』
- 133 ページの『十分な数の SQLVAR 項目を指定した動的 SQL プログラム用の SQLDA の割り振り』
- 134 ページの『動的 SQL プログラムにおける SELECT ステートメントの記述』

- 134 ページの『行を保持するためのストレージの獲得』
- 135 ページの『動的 SQL プログラムにおけるカーソルの処理』

エンド・ユーザーからの SQL 要求の保管

アプリケーションのユーザーがアプリケーションから SQL 要求を発行することができる場合、これらの要求を保管しておきたいかもしれません。

手順:

アプリケーションで任意の SQL ステートメントを保管できる場合、これらをデータ・タイプが VARCHAR、LONG VARCHAR、CLOB、VARGRAPHIC、 LONG VARGRAPHIC、または DBCLOB の列を持つ表に保管することができます。VARGRAPHIC、LONG VARGRAPHIC、および DBCLOB データ・タイプは、 2 バイト文字セット (DBCS) および拡張 UNIX コード (EUC) 環境でしか使用できないことに注意してください。

ユーザーは、準備済みのバージョンの SQL ステートメントではなく、そのソースを保管しなければなりません。これは、表に保管されているバージョンを実行する前に、各ステートメントを検索して準備しなければならないことを意味します。つまり、アプリケーションは、文字ストリングから SQL ステートメントを準備し、このステートメントを動的に実行します。

動的 SQL プログラムにおけるパラメーター・マーカー

以下のセクションでは、パラメーター・マーカーを使用して動的 SQL プログラムへ入力変数を提供する方法について説明し、カーソルを使用するサンプル・プログラムについても簡単に説明します。

パラメーター・マーカーを使用する動的 SQL への変数入力の提供

動的 SQL ステートメントにはホスト変数を入れることができません。それは、ホスト変数情報 (データ・タイプおよび長さ) がアプリケーションのプリコンパイルの間しか使用できないためです。実行時には、ホスト変数情報は使用できません。

動的 SQL では、ホスト変数の代わりにパラメーター・マーカーを使用します。ホスト変数は疑問符 (?) で示されます。そして、ホスト変数が SQL ステートメントの内部で置換される位置を示します。

手順:

アプリケーションで動的 SQL を使用していて、DELETE を実行できるようにしたいとしましょう。パラメーター・マーカーが入っている文字ストリングは、次のような形となります。

```
DELETE FROM TEMPL WHERE EMPNO = ?
```

このステートメントが実行されると、 EXECUTE ステートメントの USING 文節によってホスト変数つまり SQLDA 構造体が指定されます。ステートメントを実行する際に、ホスト変数の内容が使用されます。

パラメーター・マーカーは、 SQL ステートメント内部で使用するコンテキストによって想定されたデータ・タイプおよび長さを持っています。パラメーター・マーカーのデータ・タイプが、これを使用しているステートメントの内容からはっきり判別できない場合は、 CAST を使用してタイプを指定することができます。このようなパラメーター・マーカーは、*タイプ付きパラメーター・マーカー* と見なされます。タイプ付きパラメーター・マーカーは、指定されたタイプのホスト変数と同様に扱われます。たとえば、ステートメント `SELECT ? FROM SYSCAT.TABLES` は、結果列のタイプが DB2 には認識されないため無効です。ただし、`SELECT CAST(? AS INTEGER) FROM SYSCAT.TABLES` は、パラメーター・マーカーが INTEGER を表すことがキャストによって示されているため、 DB2 で結果列のタイプが認識されます。

SQL ステートメントにパラメーター・マーカーが 1 つ以上あると、 EXECUTE ステートメントの USING 文節は、ホスト変数 (各パラメーター・マーカーに 1 つずつ) のリストを指定するか、または各パラメーター・マーカーの SQLVAR 項目を持つ SQLDA を識別しなければなりません。 (LOB の場合は、各パラメーター・マーカーに SQLVAR 項目が 2 つずつあることに注意してください。) ホスト変数リストまたは SQLVAR 項目は、ステートメント内のパラメーター・マーカーの順序に従って突き合わせが行われます。また、これらのデータ・タイプには互換性がなければなりません。

注: 動的 SQL でのパラメーター・マーカーの使用は、静的 SQL でのホスト変数の使用に似ています。いずれの場合も、オプティマイザーは分散統計を使用せず、最適なアクセス・プランを選択することはありえません。

パラメーター・マーカーに適用される規則は、 PREPARE ステートメントで説明されています。

関連資料:

- 「*SQL リファレンス 第 2 巻*」の『PREPARE ステートメント』

動的 SQL プログラムにおけるパラメーター・マーカーの例

次の例は、動的 SQL プログラムにおけるパラメーター・マーカーの使用方法を示しています。

- C/C++ (**dbuse.sqc/dbuse.sqC**)

  C 言語サンプル **dbuse.sqc** 中の関数 `DynamicStmtWithMarkersEXECUTEusingHostVars()` は、パラメーター・マーカーとホスト変数を使用して削除を実行する仕方を示しています。

```
EXEC SQL BEGIN DECLARE SECTION;
  char hostVarStmt1[50];
  short hostVarDeptnumb;
EXEC SQL END DECLARE SECTION;

/* prepare the statement with a parameter marker */
strcpy(hostVarStmt1, "DELETE FROM org WHERE deptnumb = ?");
EXEC SQL PREPARE Stmt1 FROM :hostVarStmt1;
```

```
/* execute the statement for hostVarDeptnumb = 15 */
hostVarDeptnumb = 15;
EXEC SQL EXECUTE Stmt1 USING :hostVarDeptnumb;
```

- JDBC (**DbUse.java**)

  JDBC サンプル **DbUse.java** 中の関数 execPreparedStatementWithParam() は、パラメーター・マーカーを使用して削除を実行する方法について示しています。

```
// prepare the statement with parameter markers
PreparedStatement prepStmt = con.prepareStatement(
  " DELETE FROM org WHERE deptnumb <= ? AND division = ? ");

// execute the statement
prepStmt.setInt(1, 70);
prepStmt.setString(2, "Eastern");
prepStmt.execute();

// close the statement
prepStmt.close();
```

- COBOL (**varinp.sqb**)

  次の例は COBOL サンプル **varinp.sqb** からのもので、検索および更新条件におけるパラメーター・マーカーの使用方法を示しています。

```
   EXEC SQL BEGIN DECLARE SECTION END-EXEC.
01 pname             pic x(10).
01 dept              pic s9(4) comp-5.
01 st                pic x(127).
01 parm-var          pic x(5).
    EXEC SQL END DECLARE SECTION END-EXEC.

   move "SELECT name, dept FROM staff
-         "   WHERE job = ? FOR UPDATE OF job" to st.
   EXEC SQL PREPARE s1 FROM :st END-EXEC.

   EXEC SQL DECLARE c1 CURSOR FOR s1 END-EXEC.

   move "Mgr" to parm-var.
   EXEC SQL OPEN c1 USING :parm-var END-EXEC

   move "Clerk" to parm-var.
   move "UPDATE staff SET job = ? WHERE CURRENT OF c1" to st.
   EXEC SQL PREPARE s2 from :st END-EXEC.

* call the FETCH and UPDATE loop.
   perform Fetch-Loop thru End-Fetch-Loop
      until SQLCODE not equal 0.

   EXEC SQL CLOSE c1 END-EXEC.
```

関連概念:

- 116 ページの『アプリケーション中でのエラー・メッセージの検索』

関連サンプル:

- 『dbuse.out -- HOW TO USE A DATABASE (C)』
- 『dbuse.sqc -- How to use a database (C)』
- 『dbuse.out -- HOW TO USE A DATABASE (C++)』
- 『dbuse.sqC -- How to use a database (C++)』
- 『DbUse.java -- How to use a database (JDBC)』

- 『DbUse.out -- HOW TO USE A DATABASE. Connect to 'sample' database using JDBC type 2 driver (JDBC)』

DB2 コール・レベル・インターフェース (CLI) と動的 SQL との比較

以下のセクションでは、DB2 CLI と動的 SQL の違い、動的 SQL よりも DB2 CLI の方が有利な点、およびどのようなときに DB2 CLI または動的 SQL を使用するべきかについて説明します。

DB2 コール・レベル・インターフェース (CLI) と組み込み動的 SQL の比較

組み込み SQL インターフェースを使用するアプリケーションには、 SQL ステートメントをコードに変換するプリコンパイラーが必要で、変換後そのコードはコンパイルされ、データベースにバインドされ、実行されます。対照的に、DB2 CLI アプリケーションは、プリコンパイルまたはバインドする必要がなく、代わりに関数の標準セットを使用してランタイムに SQL ステートメントおよび関連サービスを実行します。

この違いは重要です。というのはこれまでプリコンパイラーは、個々のデータベース製品に特定のものであったからです。これは効果的にアプリケーションをその製品に結び付けるものでした。 DB2 CLI を使用すると、どの特定のデータベース製品からも独立した移植可能なアプリケーションを作成することが可能になります。この独立性によって、 DB2 CLI アプリケーションはさまざまな DB2® データベース (ホスト・システム・データベースを含む) にアクセスするために再コンパイルまたは再バインドする必要がなくなります。ただ該当するデータベースにランタイムに接続するだけで済むようになります。

DB2 CLI と組み込み SQL の相違点と類似点を次に示します。

- DB2 CLI では、カーソルの明示宣言は必要ありません。必要に応じて DB2 CLI で生成されます。そして、アプリケーションはその生成されたカーソルを通常のカーソル取り出しモデルとして、複数行の `SELECT` ステートメント、および位置指定 `UPDATE` と `DELETE` ステートメント用に使用します。
- `OPEN` ステートメントは DB2 CLI では使用しません。その代わりに、`SELECT` の実行によって自動的にカーソルがオープンされます。
- 組み込み SQL とは違って、DB2 CLI では、 `EXECUTE IMMEDIATE` ステートメントに相当するステートメント (`SQLExecDirect()` 関数) でパラメーター・マーカーの使用が可能です。
- DB2 CLI の `COMMIT` または `ROLLBACK` は、普通、 SQL ステートメントとして実行されるのではなく、 `SQLEndTran()` 関数呼び出しによって発行されますが、その実行は許可されています。
- DB2 CLI はステートメント関連情報をアプリケーションのために管理し、ステートメント・ハンドル と呼ばれる情報を表す抽象オブジェクトを提供します。このハンドルによって、アプリケーションが製品特有のデータ構造を使用する必要がなくなります。

- ステートメント・ハンドルと同様に、*環境ハンドル* および*接続ハンドル* は、グローバル変数および接続特有の情報を参照する方法を提供します。*記述子ハンドル* は、SQL ステートメントのパラメーターか、結果セットの列のどちらかの状況を記述します。
- DB2 CLI アプリケーションは、 CLI および組み込み SQL アプリケーションが結果セットを記述するのと同じように、 SQL ステートメントにパラメーターを動的に記述できます。これによって、CLI アプリケーションは、あらかじめパラメーター・マーカーのデータ・タイプを知らなくても、パラメーター・マーカーを含む SQL ステートメントを動的に処理することが可能になります。 SQL ステートメントが準備されると、記述子情報がパラメーターのデータ・タイプの詳細と共に返されます。
- DB2 CLI は、X/Open SQL CAE 仕様で定義された SQLSTATE 値を使用します。この形式および値のほとんどは、 IBM® リレーショナル・データベース製品で使用する値と一貫性がありますが、違いもあります。 (ODBC SQLSTATES と X/Open 定義の SQLSTATES の間にも違いがあります。)

上記の違いは別にして、組み込み SQL と DB2 CLI には次の重要な共通の概念があります。 *DB2 CLI は組み込み SQL で動的に作成できる SQL ステートメントを実行することができます。*

注: さらに、DB2 CLI はコンパウンド SQL ステートメントのような、動的に準備できない一部の SQL ステートメントも受け入れることができます。

各 DBMS には動的に作成できるステートメントがさらにある場合もありますが、この場合には DB2 CLI がステートメントを直接 DBMS に渡します。 1 つの例外があります。一部の DBMS では COMMIT および ROLLBACK ステートメントを動的に準備できますが、DB2 CLI により代行受信されて、適切な `SQLEndTran()` 要求として処理されます。しかし、`SQLEndTran()` 関数を使用して、 COMMIT または ROLLBACK ステートメントのいずれかを指定することが推奨されています。

関連資料:

- 751 ページの『付録 A. サポートされる SQL ステートメント』

組み込み SQL と比較した DB2 CLI の利点

DB2 CLI インターフェースには、組み込み SQL よりも優れている点がいくつかあります。

- これはクライアント・サーバー環境に理想的です。この環境では、アプリケーションの作成時にはターゲット・データベースは不明です。アプリケーションがどのデータベース・サーバーに接続するかに関係なく、 SQL ステートメント実行用の一貫性のあるインターフェースが得られます。
- プリコンパイラーへの従属性が排除されているので、アプリケーションの移植性が高まります。アプリケーションは、データベース製品ごとにプリプロセスする必要がある組み込み SQL ソース・コードとしてではなく、コンパイル済みアプリケーションまたはランタイム・ライブラリーとして配布されます。
- 個々の DB2 CLI アプリケーションを各データベースにバインドする必要はなく、すべての DB2 CLI アプリケーションについて、 DB2 CLI に付いているバ

インド・ファイルを一度バインドする必要があるだけです。いったんこれを汎用にすると、アプリケーションに必要な管理の量を著しく減らすことができます。

- DB2 CLI アプリケーションを複数のデータベースに接続することができます。同一アプリケーションから同一データベースに複数接続することもできます。各接続には、それぞれのコミット範囲があります。アプリケーションでマルチスレッド化の使用により同一結果になる組み込み SQL を使用するよりは CLI を使用する方がずっと簡単です。
- 一般に、組み込み SQL アプリケーションには、アプリケーションが制御する複合データ域 (SQLDA および SQLCA など) が関連付けられており、それらはしばしば複雑になることがあります。しかし、DB2 CLI にはデータ域が必要ではありません。その代わりに、 DB2 CLI が、必要なデータ構造を割り当てて制御し、それらを参照するためのアプリケーションの*ハンドル* を提供します。
- DB2 CLI では、マルチスレッドのスレッド保護アプリケーションの開発が可能になります。この場合、各スレッドは、独自の接続および他のスレッドとは別のコミット有効範囲を持つことができます。 DB2 CLI は、上記のデータ域を除去し、特定のハンドルでアプリケーションにアクセス可能なデータ構造のすべてを関連付けることにより、このことを成し遂げます。組み込み SQL とは違って、マルチスレッドの CLI アプリケーションではコンテキスト管理の DB2® API のいずれかを呼び出す必要はありません。これは、DB2 CLI ドライバーで自動的に操作されます。
- DB2 CLI では、拡張パラメーターの入力と取り出しの機能が備えられており、データの配列が入力時に指定され、結果セットの複数行を直接配列に取り出し、複数の結果セットを生成するステートメントを実行します。
- DB2 CLI では、スキーマ (表、列、外部キー、主キーなど) 情報を照会するための一貫性のあるインターフェースが与えられます。この情報はさまざまな DBMS カタログ表に入っています。返される結果セットは DBMS 間で一貫性があります。したがって、アプリケーションは、データベース・サーバーのリリース間のカタログ変更や、さまざまなデータベース・サーバー間のカタログの違いの影響を受けません。その結果、アプリケーションでは、バージョン固有およびサーバー固有のカタログ照会は書き込まれません。
- 拡張データ変換も DB2 CLI に備えられており、さまざまな SQL と C データ・タイプ間での情報の変換時にアプリケーション・コードが少なくて済みます。
- DB2 CLI には、ODBC と X/Open CLI の両方の関数が組み込まれており、両方とも業界仕様として受け入れられています。 DB2 CLI は、ISO CLI 標準にも合わせています。アプリケーション開発者がこれらの仕様で得た知識は、 DB2 CLI の開発に直接応用することができ、その逆も可能です。このインターフェースは、関数ライブラリーについての知識はあるが、ホスト言語に SQL ステートメントを組み込む、製品特定の方法についてあまり知らないプログラマーにとって、直感的に理解できるものです。
- DB2 CLI には、 DB2 Universal Database (つまり DB2 Universal Database for z/OS and OS/390 バージョン 5 またはそれ以降) のサーバーにあるストアード・プロシージャーから生成される複数行と結果セットを取り出す機能が備えられています。しかし、この機能は DataJoiner® のバージョン 2 サーバーによりアクセス可能なサーバーにストアード・プロシージャーがある場合に、組み込み SQL を使用しているバージョン 5 の DB2 Universal Database のクライアントのために用意されているものです。

- DB2 CLI はスクロール可能カーソルをさらに強力にサポートします。スクロール可能カーソルを使用すると、カーソルによって次のようなスクロールが可能になります。
  - 1 行または複数行ごとに順方向へ
  - 1 行または複数行ごとに逆方向へ
  - 最初の行から 1 行または複数行ごとに
  - 最後の行から 1 行または複数行ごとに

  スクロール可能カーソルは、配列出力と組み合わせて使用できます。更新可能カーソルをスクロール可能と宣言すれば、 1 行または複数行ごとに結果セット内を順方向または逆方向に移動できます。下記において、オフセットを指定することにより、複数の行を取り出すことも可能です。
  - 現在行
  - 結果セットの開始または終了位置
  - 以前にブックマークで設定した特定の行

DB2 CLI または組み込み SQL をいつ使用するか

どのインターフェースを選択するかは、使用するアプリケーションによって決まります。

DB2 CLI は、移植性が要求される、照会ベースのグラフィカル・ユーザー・インターフェース (GUI) アプリケーションに理想的です。前述の利点のため、 DB2 CLI の方が明らかにどのようなアプリケーションにもふさわしいように見えるかもしれません。しかし、考慮しなければならない要素が 1 つあります。静的 SQL と動的 SQL の比較です。組み込みアプリケーションでは静的 SQL を使用する方がはるかに簡単です。

静的 SQL には、次のようないくつかの利点があります。

- パフォーマンス

  動的 SQL はランタイムに準備され、静的 SQL はプリコンパイル時に準備されます。より多くの処理が必要になると同時に、準備ステップのためにランタイムに追加のネットワーク通信量が生じることがあります。 DB2 CLI アプリケーションが (デフォルト振る舞いの) 据え置き準備を使用する場合、追加ネットワーク・トラフィックを避けられます。

  静的 SQL が動的 SQL よりパフォーマンスが常に高いわけではない、というのも重要なことです。動的 SQL は実行時に準備され、その時点で使用可能なデータベース統計を使用しますが、静的 SQL は BIND 時に使用可能なデータベース統計を使用します。動的 SQL では、新規の索引などのデータベースに対する変更事項を利用して最適のアクセス・プランを選択することができるので、同じ SQL を静的 SQL として実行するよりも潜在的に良好なパフォーマンスが達成されます。さらに、キャッシュされていれば、動的 SQL ステートメントのプリコンパイルを避けることができます。
- カプセル化およびセキュリティー

静的 SQL では、オブジェクト (表やビューなど) に対するアクセス権限はパッケージに関連付けられ、パッケージのバインド時に妥当性検査されます。このため、データベース管理者は、特定のパッケージに関する実行権を 1 つのユーザーの集まりに付与する (つまり、特権をパッケージにカプセル化する) だけで済み、各データベース・オブジェクトへの明示アクセス権を付与する必要はありません。動的 SQL では、権限はランタイムにステートメント単位で妥当性検査されます。したがって、ユーザーは各データベース・オブジェクトへの明示アクセス権を付与してもらわなければなりません。これによってこれらのユーザーは、アクセスする必要のないオブジェクトの部分にアクセスすることが認可されます。

- 組み込み SQL は、C または C++ 以外の言語でサポートされます。
- 固定照会の選択の場合、組み込み SQL はより簡単です。

アプリケーションで両方のインターフェースの利点が必要な場合は、静的 SQL を含むストアード・プロシージャーを作成して、 DB2 CLI アプリケーションで静的 SQL を利用できます。ストアード・プロシージャーは、DB2 CLI アプリケーション内から呼び出され、サーバーで実行されます。ストアード・プロシージャーを作成すると、どの DB2 CLI または ODBC アプリケーションでもこれを呼び出すことができます。

DB2 CLI と組み込み SQL の両方を使う混合アプリケーションを作成して、それぞれの利点を活用することもできます。この場合、DB2 CLI を使用して基本のアプリケーションを作成し、パフォーマンスまたはセキュリティー上の理由のために静的 SQL を使用してキー・モジュールを作成します。このためアプリケーション設計が複雑になるので、ストアード・プロシージャーがアプリケーション要件に合わない場合に限りこの方法を使用してください。

結局、それぞれのインターフェースをいつ使用するかの判断は、 1 つの要因によるというのではなく、個々の必要性と以前の経験によって決まります。

関連概念:

- 504 ページの『CLI/ODBC/JDBC トレース機能』

関連タスク:

- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 1 巻*」の『CLI アプリケーションでの SQL ステートメントの準備と実行』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 1 巻*」の『CLI アプリケーションでの SQL ステートメントの発行』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 1 巻*」の『CLI/ODBC/JDBC 静的プロファイル作成による静的 SQL の作成』

第 6 章 C および C++ でのプログラミング

C/C++ に関するプログラミング考慮事項 151
C および C++ での 3 文字表記 151
C および C++ の入出力ファイル. 152
組み込みファイル 152
C および C++ の組み込みファイル 152
C および C++ における組み込みファイル. . . 155
C および C++ での組み込み SQL ステートメント 156
C および C++ でのホスト変数 158
C および C++ でのホスト変数 158
C および C++ でのホスト変数名. 159
C および C++ でのホスト変数の宣言 160
C および C++ における数値ホスト変数の構文 160
C および C++ における固定および NULL 終了文字ホスト変数の構文 162
C または C++ における可変長文字ホスト変数の構文 162
C および C++ での標識変数 164
C および C++ でのグラフィック・ホスト変数 164
C および C++ における単純グラフィックおよび NULL 終了グラフィック書式のグラフィック宣言の構文 165
C または C++ における VARGRAPHIC 構造書式のグラフィック宣言の構文 166
C または C++ におけるラージ・オブジェクト (LOB) ホスト変数の構文 167
C または C++ におけるラージ・オブジェクト (LOB) ロケーター・ホスト変数の構文 169
C または C++ におけるファイル参照ホスト変数宣言の構文 170
C および C++ におけるホスト変数の初期化 . . 171
C マクロ展開 171
C および C++ でのホスト構造体サポート. . . 172
C および C++ での標識表 174
C および C++ でのヌル終了ストリング . . . 175
C および C++ でポインター・データ・タイプとして使用されるホスト変数 176
C および C++ でホスト変数として使用されるクラス・データ・メンバー. 177
C および C++ での修飾およびメンバー演算子 178
C および C++ でのマルチバイト文字のエンコード 179
C および C++ での wchar_t および sqldbchar データ・タイプ. 179
C および C++ での WCHARTYPE プリコンパイラー・オプション 180
C および C++ での日本語または中国語 (繁体字) EUC、および UCS-2 に関する考慮事項 . . 183
C および C++ 用のホスト変数がある SQL 宣言セクション 184
C および C++ におけるデータ・タイプに関する考慮事項. 186
C および C++ においてサポートされている SQL データ・タイプ 186
C および C++ における FOR BIT DATA . . . 190
プロシージャー、関数、およびメソッド用の C/C++ データ・タイプ 190
C および C++ における SQLSTATE および SQLCODE 変数 192

C/C++ に関するプログラミング考慮事項

次のトピックで、特定のホスト言語によるプログラミングの考慮事項が記述されています。言語制限、ホスト言語別の組み込みファイル、組み込み SQL ステートメント、ホスト変数、およびサポートされるホスト変数のデータ・タイプについての情報が含まれています。

関連資料:

- 「*アプリケーション開発ガイド アプリケーションの構築および実行*」の『C サンプル』

C および C++ での 3 文字表記

C または C++ 文字セットの文字の中には、すべてのキーボードで使用できないものもあります。これらの文字は、*3 文字表記* と呼ばれる一続きの 3 つの文字を使用して C または C++ のソース・プログラムに入力することができます。 3 文字表記は SQL ステートメントでは認識されません。プリコンパイラーは、ホスト変数宣言内で以下の 3 文字表記を認識します。

3 文字表記	定義
??(	左大括弧 '['
??)	右大括弧 ']'
??<	左中括弧 '{'
??>	右中括弧 '}'

以下に示すその他の 3 文字表記は、C または C++ ソース・プログラムの別の場所で使用されることがあります。

3 文字表記	定義
??=	ハッシュ・マーク '#'
??/	円記号 '¥'
??'	脱字記号 '^'
??!	縦線 '|'
??-	波形記号 '~'

C および C++ の入出力ファイル

デフォルトでは、入力ファイルに次のような拡張子を付けることができます。

.sqc サポートされているすべてのプラットフォーム上の C ファイル

.sqC UNIX® プラットフォーム上の C++ ファイル

.sqx Windows® オペレーティング・システム上の C++ ファイル

デフォルトでは、対応するプリコンパイラーの出力ファイルは、以下の拡張子が付けられます。

.c サポートされているすべてのプラットフォーム上の C ファイル

.C UNIX プラットフォーム上の C++ ファイル

.cxx Windows オペレーティング・システム上の C++ ファイル

OUTPUT プリコンパイル・オプションを使用することにより、出力修正後のソース・ファイルの名前とパスを無効にできます。 TARGET C または TARGET CPLUSPLUS プリコンパイル・オプションを使用する場合、入力ファイルに拡張子は必要ありません。

組み込みファイル

以下のセクションでは、C および C++ の組み込みファイルについて説明します。

C および C++ の組み込みファイル

C および C++ でのホスト言語固有の組み込みファイル (ヘッダー・ファイル) には、ファイル拡張子の `.h` が付いています。アプリケーションで使用するためのこれらの組み込みファイルについて以下で説明します。

SQL (sql.h)
このファイルには、バインド・プログラム、プリコンパイラー、およびエラー・メッセージ検索 API 用の言語固有プロトタイプが含まれています。また、システム定数も定義されています。

SQLADEF (sqladef.h)
プリコンパイル済みの C および C++ アプリケーションが使用する関数プロトタイプが含まれています。

SQLAPREP (sqlaprep.h)
このファイルには、独自のプリコンパイラーの作成に必要な定義が入っています。

SQLCA (sqlca.h)
このファイルは SQL 連絡域 (SQLCA) 構造体を定義します。 SQLCA には、データベース・マネージャーが SQL ステートメントおよび API 呼び出しの実行に関するエラー情報をアプリケーションに提供するために使用する変数が含まれています。

SQLCLI (sqlcli.h)
コール・レベル・インターフェース (DB2 CLI) アプリケーションを作成するために必要な関数プロトタイプと定数が含まれています。 このファイル内の関数は、X/Open コール・レベル・インターフェースおよび ODBC コア・レベルの両方に共通です。

SQLCLI1 (sqlcli1.h)
DB2 CLI でより拡張された機能を使用するコール・レベル・インターフェース (DB2 CLI) を作成するために必要な関数プロトタイプおよび定数が含まれています。このファイル内の関数の大部分は、 X/Open コール・レベル・インターフェースおよび ODBC レベル 1 に共通です。さらに、このファイルには X/Open 専用の関数および DB2 固有の関数も含まれています。

このファイルには、`sqlcli.h` と `sqlext.h` (ODBC レベル 2 API 定義) の両方が含まれます。

SQLCODES (sqlcodes.h)
このファイルは、SQLCA 構造体の SQLCODE フィールドで使用する定数を定義します。

SQLDA (sqlda.h)
このファイルは SQL 記述子域 (SQLDA) 構造体を定義します。 SQLDA はアプリケーションとデータベース・マネージャーとの間でデータをやりとりするために使用されます。

SQLENV (sqlenv.h)
このファイルは、データベース環境 API に対する言語固有の呼び出し、およびそれらのインターフェースの構造、定数、戻りコードを定義します。

SQLEXT (sqlext.h)
X/Open コール・レベル・インターフェースの仕様の一部ではないこれらの ODBC レベル 1 およびレベル 2 API の関数プロトタイプと定数が含まれており、 Microsoft 社の許可を得て使用します。

SQLE819A (sqle819a.h)
データベースのコード・ページが 819 (ISO ラテン 1) の場合、このシーケンスは、ホスト CCSID 500 (EBCDIC 各国対応) バイナリー照合に従って、 FOR BIT DATA ではない文字ストリングをソートします。このファイルは CREATE DATABASE API が使用します。

SQLE819B (sqle819b.h)
データベースのコード・ページが 819 (ISO ラテン 1) の場合、このシーケンスは、ホスト CCSID 037 (EBCDIC 米国英語) バイナリー照合に従って、 FOR BIT DATA ではない文字ストリングをソートします。このファイルは CREATE DATABASE API が使用します。

SQLE850A (sqle850a.h)
データベースのコード・ページが 850 (ASCII ラテン 1) の場合、このシーケンスは、ホスト CCSID 500 (EBCDIC 各国対応) バイナリー照合に従って、 FOR BIT DATA ではない文字ストリングをソートします。このファイルは CREATE DATABASE API が使用します。

SQLE850B (sqle850b.h)
データベースのコード・ページが 850 (ASCII ラテン 1) の場合、このシーケンスは、ホスト CCSID 037 (EBCDIC 米国英語) バイナリー照合に従って、 FOR BIT DATA ではない文字ストリングをソートします。このファイルは CREATE DATABASE API が使用します。

SQLE932A (sqle932a.h)
データベースのコード・ページが 932 (ASCII 日本語) の場合、このシーケンスは、ホスト CCSID 5035 (EBCDIC 日本語) バイナリー照合に従って、FOR BIT DATA ではない文字ストリングをソートします。このファイルは CREATE DATABASE API が使用します。

SQLE932B (sqle932b.h)
データベースのコード・ページが 932 (ASCII 日本語) の場合、このシーケンスは、ホスト CCSID 5026 (EBCDIC 日本語) バイナリー照合に従って、FOR BIT DATA ではない文字ストリングをソートします。このファイルは CREATE DATABASE API が使用します。

SQLJACB (sqljacb.h)
DB2 Connect インターフェースの定数、構造、および制御ブロックを定義します。

SQLMON (sqlmon.h)
このファイルは、データベース・システム・モニター API に対する言語固有の呼び出し、およびそれらのインターフェースの構造、定数、戻りコードを定義します。

SQLSTATE (sqlstate.h)
このファイルは、SQLCA 構造体の SQLSTATE フィールドで使用する定数を定義します。

SQLSYSTM (sqlsystm.h)
このファイルには、データベース・マネージャー API およびデータ構造が用いるプラットフォーム固有の定義が含まれています。

SQLUDF (sqludf.h)
ユーザー定義関数 (UDF) を作成するための定数およびインターフェース構造を定義します。

SQLUTIL (sqlutil.h)
このファイルは、ユーティリティー API に対する言語固有の呼び出し、およびそれらのインターフェースに必要な構造、定数、コードを定義します。

SQLUV (sqluv.h)
非同期ログ読み取り API および表のロード/アンロードを行うベンダーが使用する API の構造、定数、プロトタイプなどを定義します。

SQLUVEND (sqluvend.h)
記憶管理ベンダーが使用する API の構造、定数およびプロトタイプを定義します。

SQLXA (sqlxa.h)
X/Open XA インターフェースを使用するアプリケーションが使用する関数プロトタイプと定数が含まれます。

関連概念:

• 155 ページの『C および C++ における組み込みファイル』

C および C++ における組み込みファイル

ファイルの組み込みには、EXEC SQL INCLUDE ステートメントを使用する方法と、#include マクロを使用する方法の 2 つがあります。プリコンパイラーは #include を無視し、 EXEC SQL INCLUDE ステートメントを使用して組み込まれたファイルのみを処理します。

EXEC SQL INCLUDE を使用して組み込んだファイルを探索する際に、 DB2® C プリコンパイラーはまず最初に現行ディレクトリーを検索し、次いで DB2INCLUDE 環境変数に指定されたディレクトリーを検索します。 次の例を考えてみてください。

- `EXEC SQL INCLUDE payroll;`

  INCLUDE ステートメントに指定されたファイルが、上のように、単引用符 (') で囲まれていない場合、 C プリコンパイラーは各ディレクトリーで最初に payroll.sqc を検索し、次に payroll.h を検索します。 UNIX® オペレーティング・システムの場合、C プリコンパイラーは、 payroll.sqC、payroll.sqx、payroll.hpp、payroll.h の順で各ディレクトリーを検索します。 Windows® 32 ビット・オペレーティング・システムの場合、 C++ プリコンパイラーは、payroll.sqx、payroll.hpp、payroll.h の順で検索し、最後にそれがロックする各ディレクトリーを検索します。

- `EXEC SQL INCLUDE 'pay/payroll.h';`

  ファイル名が単引用符 (') で囲まれている場合、すでに説明したとおり、名前に拡張子は追加されません。

  単引用符 (') 内のファイル名に完全パスが含まれていない場合、 DB2INCLUDE の中身によってそのファイルの検索が行われ、何らかのパスが INCLUDE ファイル名に指定されます。たとえば、UNIX ベースのシステムでは、 DB2INCLUDE

は ‘/disk2:myfiles/c’ に設定され、 C/C++ プリコンパイラーは ‘./pay/payroll.h’、 ‘/disk2/pay/payroll.h’、 ‘./myfiles/c/pay/payroll.h’ の順に探索します。 プリコンパイラー・メッセージには、実際にファイルが見つかったパスが表示されます。 Windows ベースのプラットフォームでは、上の例のスラッシュを円記号 (¥) に置き換えます。

注: DB2INCLUDE の設定は、コマンド行プロセッサーによってキャッシュされます。 何らかの CLP コマンドを発行した後に DB2INCLUDE の設定を変更する場合は、 TERMINATE コマンドを入力してから、データベースに接続し直し、あとは通常どおりプリコンパイルしてください。

プリコンパイラーは、コンパイラー・エラーを元のソースに戻して関連付けるために、出力ファイルに ANSI #line マクロを生成します。これにより、コンパイラーはプリコンパイラー出力ではなく、ソースまたは組み込まれたソース・ファイルのファイル名と行番号を使用してエラーを報告することができます。

PREPROCESSOR オプションを指定する場合は、プリコンパイラーにより生成されるすべての #line マクロは、外部 C プリプロセッサーからプリプロセスされたファイルを参照します。

ソース・コードをオブジェクト・コードに関連付けるデバッガーやその他のツールの中には、#line マクロを使用して常に正常に作動するわけではないものもあります。使用したいツールが期待どおりに動作しない場合は、プリコンパイル時に (DB2 PREP と共に使用される) NOLINEMACRO オプションを使用してください。 このオプションにより、#line マクロは生成されなくなります。

関連概念:

- 171 ページの『C マクロ展開』

関連資料:

- 「*SQL リファレンス 第 2 巻*」の『PREPARE ステートメント』
- 152 ページの『C および C++ の組み込みファイル』

C および C++ での組み込み SQL ステートメント

組み込み SQL ステートメントは次の 3 つのエレメントからなります。

エレメント	正しい構文
ステートメント初期化指定子	EXEC SQL
ステートメント・ストリング	任意の有効な SQL ステートメント
ステートメント終止符	セミコロン (;)

たとえば、次のようになります。

```
EXEC SQL SELECT col INTO :hostvar FROM table;
```

組み込み SQL ステートメントには、以下の規則が適用されます。

- SQL ステートメント・ストリングは、キーワード対または別の行として同じ行で開始することができる。ステートメント・ストリングは、複数行にわたる長さであってもかまいません。ただし、対になった EXEC SQL キーワードを複数行に分割してはなりません。
- SQL ステートメント終止符を使用しなければならない。使用しない場合、プリコンパイラーはアプリケーション内の次の終止符まで処理を継続します。これにより、不確定のエラーが起こる恐れがあります。

  C/C++ コメントは、ステートメント初期化指定子の前、またはステートメント終止符の後に入れることができます。
- 複数の SQL ステートメントと C/C++ ステートメントは同じ行に置くことができる。たとえば、次のようになります。

  ```
  EXEC SQL OPEN c1; if (SQLCODE >= 0) EXEC SQL FETCH c1 INTO :hv;
  ```
- SQL プリコンパイラーは、復帰 (CR)、改行 (LF)、 および TAB を引用符付きストリングにそのまま残す。
- SQL コメントは、組み込み SQL ステートメントの一部となっている行であればどこでも使用することができる。このコメントは、動的に実行するステートメントでは使用できません。 SQL コメントの形式は、ダブル・ダッシュ (--) の後に 0 個以上の文字ストリングが続き、行末で終了します。 SQL コメントは SQL ステートメント終止符の後に置かないでください。終止符の後に置かれたコメントは、C/C++ 言語の一部のように見えるため、コンパイル・エラーの原因となるからです。

  コメントは、ブランクを使用できる静的ステートメント・ストリング内であればどこでも使用することができます。 C/C++ コメント区切り文字の /* */、または SQL コメント記号の (--) を使用してください。 // スタイルの C++ コメントは静的 SQL ステートメントでは使用できませんが、プログラム内の他の場所では使用できます。プリコンパイラーは、SQL ステートメントを処理する前にコメントを削除します。動的 SQL ステートメントでは、C および C++ のコメント区切り文字である /* */ または // を使用することは**できません**。 ただし、プログラム内の他の場所では使用できます。
- C および C++ アプリケーションでは、 SQL ストリング・リテラルや区切り ID は次の行にわたって続けることができる。このことを行うには、次の行に続けたい行の末尾に円記号 (¥) を使用します。たとえば、次のようになります。

  ```
  EXEC SQL SELECT "NA¥
  ME" INTO :n FROM staff WHERE name='Sa¥
  nders';
  ```

  改行文字 (復帰や改行など) はいずれも、ストリングまたは区切り ID には含まれません。
- 行末文字およびタブ文字などの空白文字の置換は、次のように行われる。
  - 引用符の外 (ただし、SQL ステートメント内) では、行末文字または TAB 文字は単一スペースと置き換えられる。
  - 引用符内では、ストリングが C プログラムに適合した形で継続されていれば、行末文字は消去される。 TAB は修正されません。

  行末および TAB に使用される実際の文字は、プラットフォームごとに異なります。たとえば、UNIX® ベースのシステムでは改行 (LF) が使用されます。

関連資料:

- 751 ページの『付録 A. サポートされる SQL ステートメント』

C および C++ でのホスト変数

以下のセクションでは、C および C++ プログラムにおけるホスト変数の宣言と使用について説明します。

C および C++ でのホスト変数

ホスト変数は、SQL ステートメント内で参照される C または C++ の言語変数です。これにより、アプリケーションは入力データをデータベース・マネージャーに渡し、またデータベース・マネージャーから出力データを受け取ることができます。アプリケーションのプリコンパイルが行われると、コンパイラーはホスト変数をその他の C/C++ 変数と同様に使用します。ホスト変数の命名、宣言、および使用は、以下の節で述べる規則に従って行ってください。

手作業で SQLDA を構成するアプリケーションでは、sqlvar::sqltype==SQL_TYP_INTEGER のときは long 変数を使用できません。その代わり、sqlint32 タイプを使用しなければなりません。この問題は、ホスト変数宣言で long 変数を使用する場合と同様で、手動で SQLDA を構成しなければ、プリコンパイラーはこのエラーをカバーできずランタイムにエラーが発生します。

sqlvar::sqldata 情報にアクセスするために使用されるどんな long および unsigned long によるキャストも、sqlint32 および sqluint32 に変更しなければなりません。sqloptions および sqla_option 構造体の val メンバーは sqluintptr として宣言されます。それゆえ、ポインター・メンバーを sqla_option::val または sqloptions::val メンバーに割り当てる場合には、unsigned long でキャストするのではなく sqluintptr でキャストしなければなりません。この変更点は、64 ビット UNIX® プラットフォームでは問題を引き起こしませんが、long タイプが 32 ビットでしかない 64 ビット Windows® NT アプリケーションでは問題になります。

関連概念:

- 159 ページの『C および C++ でのホスト変数名』
- 160 ページの『C および C++ でのホスト変数の宣言』
- 162 ページの『C および C++ における固定および NULL 終了文字ホスト変数の構文』
- 164 ページの『C および C++ での標識変数』
- 164 ページの『C および C++ でのグラフィック・ホスト変数』
- 171 ページの『C および C++ におけるホスト変数の初期化』
- 172 ページの『C および C++ でのホスト構造体サポート』
- 184 ページの『C および C++ 用のホスト変数がある SQL 宣言セクション』

関連資料:

- 160 ページの『C および C++ における数値ホスト変数の構文』
- 162 ページの『C または C++ における可変長文字ホスト変数の構文』

- 167 ページの『C または C++ におけるラージ・オブジェクト (LOB) ホスト変数の構文』
- 169 ページの『C または C++ におけるラージ・オブジェクト (LOB) ロケーター・ホスト変数の構文』
- 170 ページの『C または C++ におけるファイル参照ホスト変数宣言の構文』

C および C++ でのホスト変数名

SQL プリコンパイラーは、宣言された名前によってホスト変数を識別します。以下の規則が適用されます。

- 変数名は 255 文字以内の長さで指定する。
- ホスト変数名は、システムの予約語である SQL、sql、DB2®、および db2 以外の接頭部で開始する。 たとえば、次のようになります。

```
EXEC SQL BEGIN DECLARE SECTION;
  char  varsql;     /* allowed */
  char  sqlvar;     /* not allowed */
  char  SQL_VAR;    /* not allowed */
EXEC SQL END DECLARE SECTION;
```

- プリコンパイラーは、ホスト変数名をモジュールに対してグローバルであると見なす。ただし、これは、ホスト変数をグローバル変数として宣言しなければならないという意味ではありません。ホスト変数を関数内でローカル変数として宣言しても全く問題ありません。たとえば、次ページのコーディングは正しいものとして処理されます。

```
void f1(int i)
{
EXEC SQL BEGIN DECLARE SECTION;
  short host_var_1;
EXEC SQL END DECLARE SECTION;
EXEC SQL SELECT COL1 INTO :host_var_1 from TBL1;
}
void f2(int i)
{
EXEC SQL BEGIN DECLARE SECTION;
  short host_var_2;
EXEC SQL END DECLARE SECTION;
EXEC SQL INSERT INTO TBL1 VALUES (:host_var_2);
}
```

複数のローカル・ホスト変数に同じ名前を付けることも可能です。ただしこの場合、それらの変数がすべて同じタイプかつ同じサイズであることが条件です。 このことを行うには、そのようなホスト変数の最初の出現箇所を BEGIN DECLARE SECTION ステートメントと END DECLARE SECTION ステートメントの間でプリコンパイラーに宣言し、残りの変数の宣言については宣言セクション外でそのままにしておきます。以下のコーディング例は、このことを示したものです。

```
void f3(int i)
{
EXEC SQL BEGIN DECLARE SECTION;
char host_var_3[25];
EXEC SQL END DECLARE SECTION;
EXEC SQL SELECT COL2 INTO :host_var_3 FROM TBL2;
}
void f4(int i)
```

```
{
char host_var_3[25];
EXEC SQL INSERT INTO TBL2 VALUES (:host_var_3);
}
```

f3 と f4 は同じモジュールにあり、 host_var_3 はどちらの関数でも同じタイプかつ長さなので、プリコンパイラーへの宣言は 1 回で済みます。

関連概念:

- 160 ページの『C および C++ でのホスト変数の宣言』

C および C++ でのホスト変数の宣言

ホスト変数宣言の識別には、SQL の宣言セクションを使用しなければなりません。このようにして、それ以降の SQL ステートメントで参照が可能なホスト変数をプリコンパイラーに知らせます。

C/C++ プリコンパイラーは、有効な C または C++ 宣言のサブセットのみを有効なホスト変数宣言として認識します。これらの宣言は、数値変数または文字変数のいずれかを宣言します。ホスト変数のタイプとして typedef は使用できません。 ホスト変数は、グループ化して単一のホスト構造体にすることができます。 C++ クラスのデータ・メンバーは、ホスト変数として宣言できます。

数値ホスト変数は、数値の SQL 入出力値に対する入出力値として使用することができます。文字ホスト変数は任意の文字、日付、時間またはタイム・スタンプの SQL 入出力値に対する入出力値として使用できます。 アプリケーションでは、出力変数が受け取る値を入れることのできる長さを持つようにしなければなりません。

関連概念:

- 162 ページの『C および C++ における固定および NULL 終了文字ホスト変数の構文』
- 164 ページの『C および C++ でのグラフィック・ホスト変数』
- 172 ページの『C および C++ でのホスト構造体サポート』
- 177 ページの『C および C++ でホスト変数として使用されるクラス・データ・メンバー』

関連タスク:

- 34 ページの『db2dclgn 宣言生成プログラムを使用したホスト変数の宣言』
- 「*アプリケーション開発ガイド サーバー・アプリケーションのプログラミング*」の『構造化型ホスト変数の宣言』

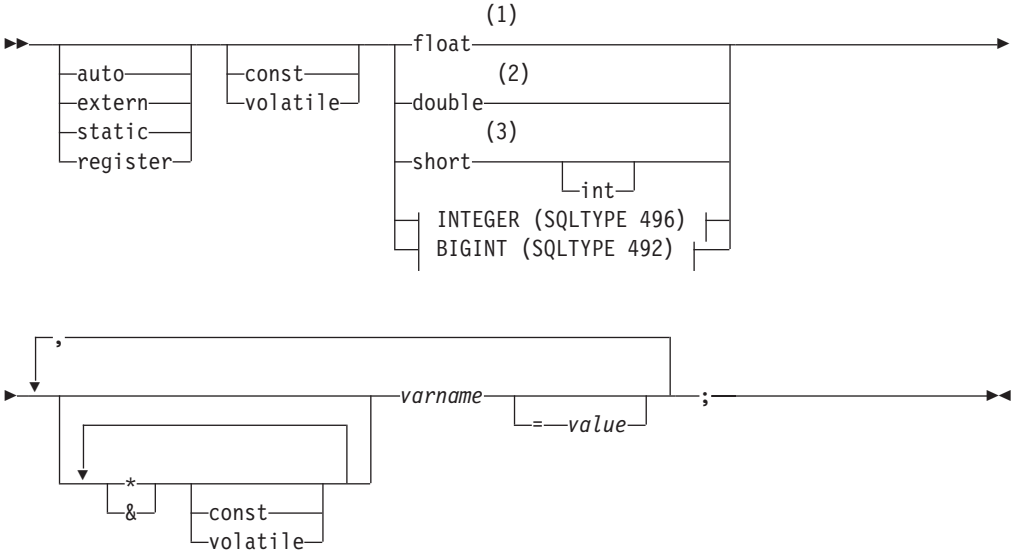
関連資料:

- 160 ページの『C および C++ における数値ホスト変数の構文』
- 162 ページの『C または C++ における可変長文字ホスト変数の構文』

C および C++ における数値ホスト変数の構文

C または C++ における数値ホスト変数の宣言構文を次に示します。

C または C++ における数値ホスト変数の構文



INTEGER (SQLTYPE 496)

BIGINT (SQLTYPE 492)

注:

1 REAL (SQLTYPE 480)、長さ 4

2 DOUBLE (SQLTYPE 480)、長さ 8

3 SMALLINT (SQLTYPE 500)

4 アプリケーションの移植性を最大限にするには、 INTEGER および BIGINT ホスト変数でそれぞれ sqlint32 および sqlint64 を使用してください。デフォルトでは、 long のホスト変数を使用すると、 long が 64 ビットである 64 BIT UNIX などでプリコンパイル・エラー SQL0402 が起きます。 PREP オプション LONGERROR NO を使用して、 DB2 が long 変数を受け入れ可能なホスト変数型として認めるようにしてください。そして、それらを BIGINT 変数として扱ってください。

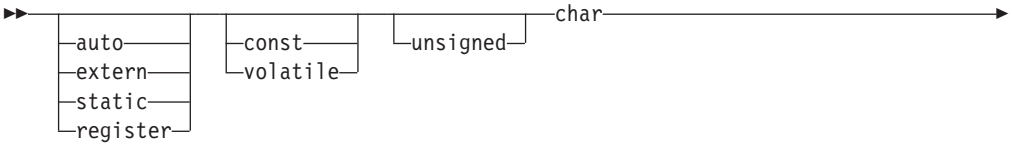
5 アプリケーションの移植性を最大限にするには、 INTEGER および BIGINT ホスト変数でそれぞれ sqlint32 および sqlint64 を使用してください。 BIGINT データ・タイプを使用するには、プラットフォームで 64 ビットの整数値がサ

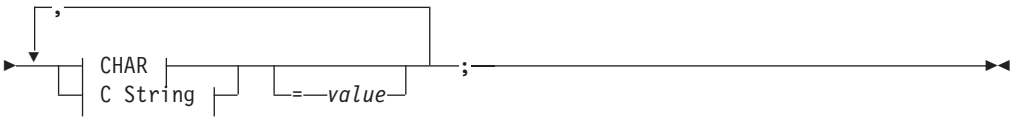
ポートされていなければなりません。デフォルトでは、 long のホスト変数を使用すると、 long が 64 ビットである 64 BIT UNIX などでプリコンパイル・エラー SQL0402 が起きます。 PREP オプション LONGERROR NO を使用して、 DB2 が long 変数を受け入れ可能なホスト変数型として認めるようにしてください。そして、それらを BIGINT 変数として扱ってください。

C および C++ における固定および NULL 終了文字ホスト変数の構文

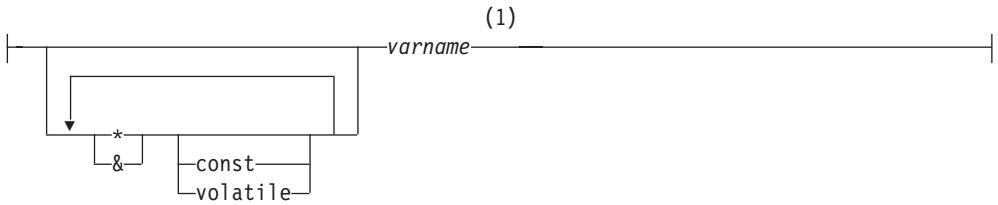
C または C++ における固定および NULL 終了文字ホスト変数の宣言構文を次に示します。

C および C++ における固定および NULL 終了文字ホスト変数の構文 - 書式 1

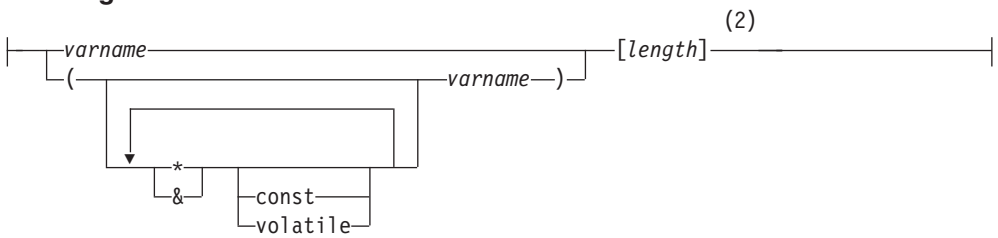




CHAR



C String



注:

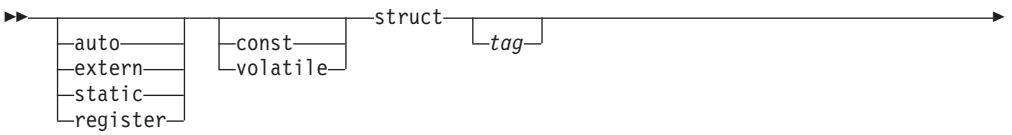
1 CHAR (SQLTYPE 452)、長さ 1

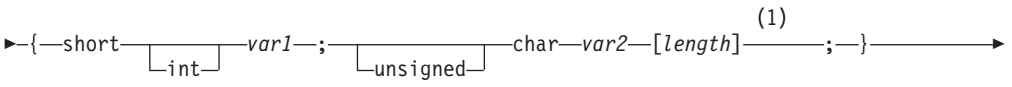
2 NULL 終了 C ストリング (SQLTYPE 460); 長さは任意の有効な定数式

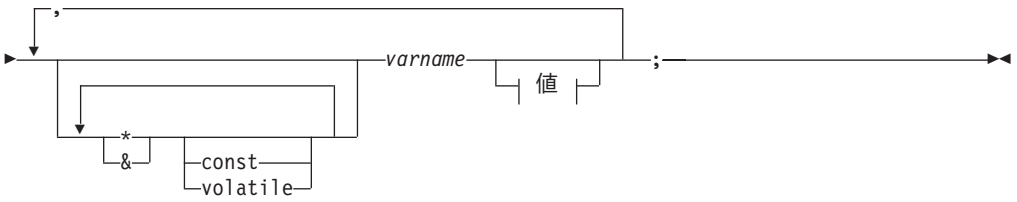
C または C++ における可変長文字ホスト変数の構文

C または C++ における可変長文字ホスト変数の宣言構文を次に示します。

C または C++ における可変長文字ホスト変数の構文







値

注:

1 書式 2 では、length は任意の有効な定数式。評価後の値で、ホスト変数が VARCHAR (SQLTYPE 448) または LONG VARCHAR (SQLTYPE 456) のどちらであるかが判別されます。

可変長文字ホスト変数に関する考慮事項:

1. データベース・マネージャーは、可能な場合には必ず文字データを**書式 1** または**書式 2** に変換するが、**書式 1** が列タイプ CHAR または VARCHAR に対応するのに対し、**書式 2** は列タイプ VARCHAR および LONG VARCHAR に対応する。
2. **書式 1** を長さ指定子の *[n]* と共に使用した場合、評価後の長さ指定子の値は 32 672 より大きくてはならず、変数に含めるストリングは NULL で終わらなければならない。
3. **書式 2** を使用する場合は、評価後の長さ指定子の値は 32 700 より小さくなければならない。
4. **書式 2** では、 *var1* および *var2* は単純変数参照 (演算子ではない) でなければならず、ホスト変数として使用することはできない (*varname* がホスト変数)。
5. *varname* を単純変数とすることもできるし、 **varname* などの演算子を含むこともできる。詳しくは、C および C++ におけるポインター・データ・タイプの説明を参照してください。
6. プリコンパイラーはすべてのホスト変数の SQLTYPE および SQLLEN を判別する。ホスト変数が SQL ステートメント内に標識変数とともにある場合、そのステートメントの持続期間中は、 SQLTYPE は基本の SQLTYPE プラス 1 となるように割り当てられます。
7. プリコンパイラーは、C または C++ において構文的に無効であるものも宣言できる場合がある。特定の宣言構文に疑問がある場合には、ご使用のコンパイラーに関する資料をご覧ください。

関連概念:

- 176 ページの『C および C++ でポインター・データ・タイプとして使用されるホスト変数』

C および C++ での標識変数

標識変数のデータ・タイプは short と宣言してください。

関連概念:

- 174 ページの『C および C++ での標識表』

C および C++ でのグラフィック・ホスト変数

C または C++ で作成されたアプリケーションでグラフィック・データを処理するには、 wchar_t C/C++ データ・タイプまたは DB2® 提供の sqldbchar データ・タイプに基づくホスト変数を使用してください。 これら 2 つのホスト変数は、GRAPHIC、 VARGRAPHIC、または DBCLOB などの表の列に割り当てることができます。たとえば、表の GRAPHIC または VARGRAPHIC 列から、DBCS データを更新したり選択したりすることができます。

グラフィック・ホスト変数には、以下のような 3 つの有効な書式があります。

- 単純グラフィック書式

  単純グラフィック・ホスト変数は、GRAPHIC(1) SQL データ・タイプに相当する 468/469 の SQLTYPE を持っています。

- NULL 終了グラフィック書式

  NULL 終了とは、GRAPHIC ストリングの最後の文字のバイトがすべてバイナリー・ゼロ (¥0') である状態を言います。これらは SQLTYPE が 400®/401 となります。

- VARGRAPHIC 構造書式

  VARGRAPHIC 構造ホスト変数は、長さが 1 から 16 336 バイトの間の場合は SQLTYPE が 464/465 となります。この変数の長さが 2 000 から 16 350 バイトの間の場合は、SQLTYPE が 472/473 となります。

関連概念:

- 159 ページの『C および C++ でのホスト変数名』
- 160 ページの『C および C++ でのホスト変数の宣言』
- 171 ページの『C および C++ におけるホスト変数の初期化』
- 172 ページの『C および C++ でのホスト構造体サポート』
- 174 ページの『C および C++ での標識表』
- 179 ページの『C および C++ でのマルチバイト文字のエンコード』
- 179 ページの『C および C++ での wchar_t および sqldbchar データ・タイプ』
- 180 ページの『C および C++ での WCHARTYPE プリコンパイラー・オプション』

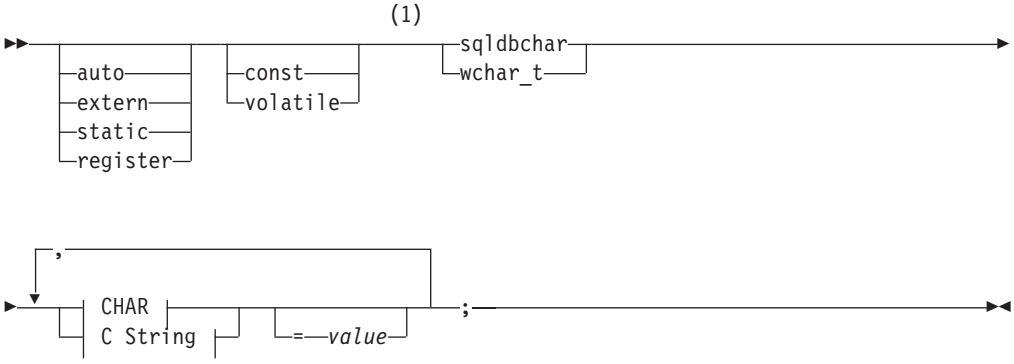
関連資料:

- 165 ページの『C および C++ における単純グラフィックおよび NULL 終了グラフィック書式のグラフィック宣言の構文』
- 166 ページの『C または C++ における VARGRAPHIC 構造書式のグラフィック宣言の構文』
- 167 ページの『C または C++ におけるラージ・オブジェクト (LOB) ホスト変数の構文』
- 169 ページの『C または C++ におけるラージ・オブジェクト (LOB) ロケーター・ホスト変数の構文』
- 170 ページの『C または C++ におけるファイル参照ホスト変数宣言の構文』

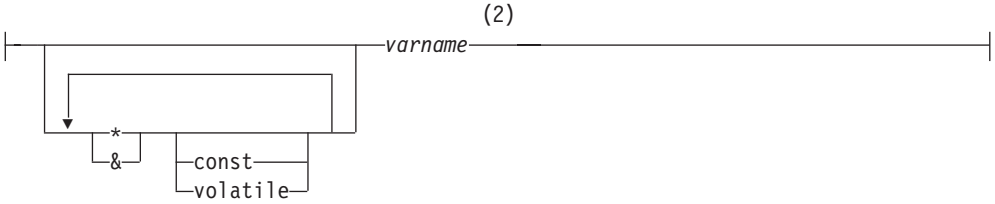
C および C++ における単純グラフィックおよび NULL 終了グラフィック書式のグラフィック宣言の構文

単純グラフィック書式および NULL 終了グラフィック書式を用いるグラフィック・ホスト変数の宣言構文を、次に示します。

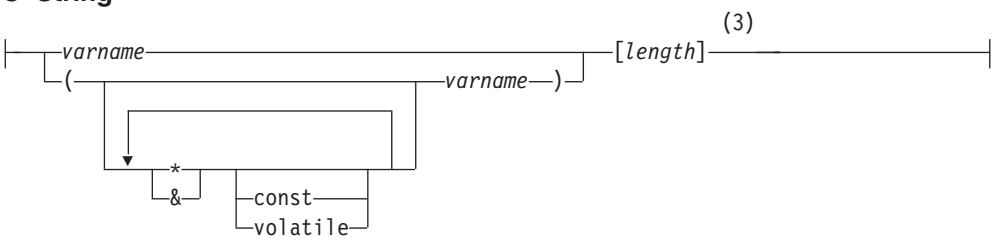
単純グラフィックおよび NULL 終了グラフィック書式のグラフィック宣言の構文



CHAR



C String



注:

1 2 つのグラフィック・タイプのどちらを使用するかを判別するための基準については、C および C++ における wchar_t および sqldbchar データ・タイプの説明を参照してください。

2 GRAPHIC (SQLTYPE 468)、長さ 1

3 NULL 終了 GRAPHIC ストリング (SQLTYPE 400)

グラフィック・ホスト変数に関する考慮事項:

1. 単純グラフィック書式では、SQLTYPE が 468 または 469 で長さ 1 の固定長の GRAPHIC ストリング・ホスト変数が宣言される。
2. *value* は初期化指定子である。 WCHARTYPE CONVERT プリコンパイラー・オプションを使用している場合は、ワイド・キャラクターのストリング・リテラル (L-リテラル) を使用してください。
3. *length* は任意の有効な定数式にすることができる。評価後の値は、1 〜 16 336 (VARGRAPHIC の最大長) の範囲でなければなりません。
4. NULL 終了 GRAPHIC ストリングは、標準レベルのプリコンパイラー・オプションの値に基づいてさまざまに処理されます。

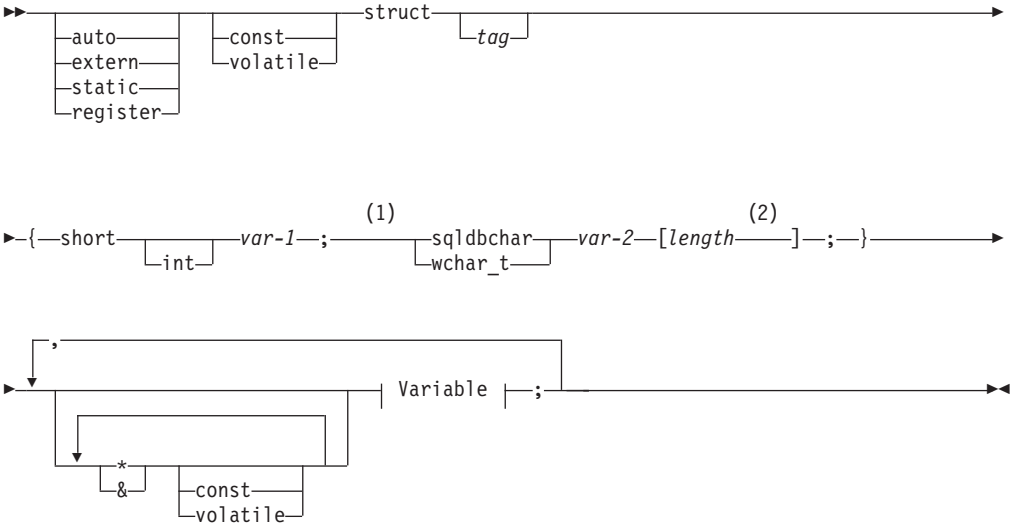
関連概念:

- 175 ページの『C および C++ でのヌル終了ストリング』
- 179 ページの『C および C++ での wchar_t および sqldbchar データ・タイプ』

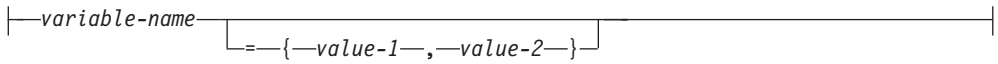
C または C++ における VARGRAPHIC 構造書式のグラフィック宣言の構文

VARGRAPHIC 構造書式を用いるグラフィック・ホスト変数の宣言構文を次に示します。

VARGRAPHIC 構造書式のグラフィック宣言の構文



変数:



注:

1 2 つのグラフィック・タイプのどちらを使用するかを判別するための基準については、C および C++ における wchar_t および sqldbchar データ・タイプの説明を参照してください。

2 *length* は、任意の有効な定数式。評価後の値で、ホスト変数が VARGRAPHIC (SQLTYPE 464) または LONG VARGRAPHIC (SQLTYPE 472) のどちらであるかが判別されます。 length の値は 1 以上でなければならず、かつ LONG VARGRAPHIC の最大長である 16 350 以下でなければなりません。

グラフィック宣言 (VARGRAPHIC 構造書式) に関する考慮事項:

1. *var-1* および *var-2* は単純変数参照 (演算子ではない) でなければならず、ホスト変数として使用することはできない。
2. *value-1* および *value-2* は、 *var-1* と *var-2* に対する初期化指定子である。 WCHARTYPE CONVERT プリコンパイラー・オプションを使用している場合、 *value-1* は整数でなければならず、 *value-2* はワイド・キャラクター・ストリング・リテラル (L-リテラル) を使用してください。
3. struct *tag* は他のデータ域を定義するために使用できるが、それ自体はホスト変数としては使用できない。

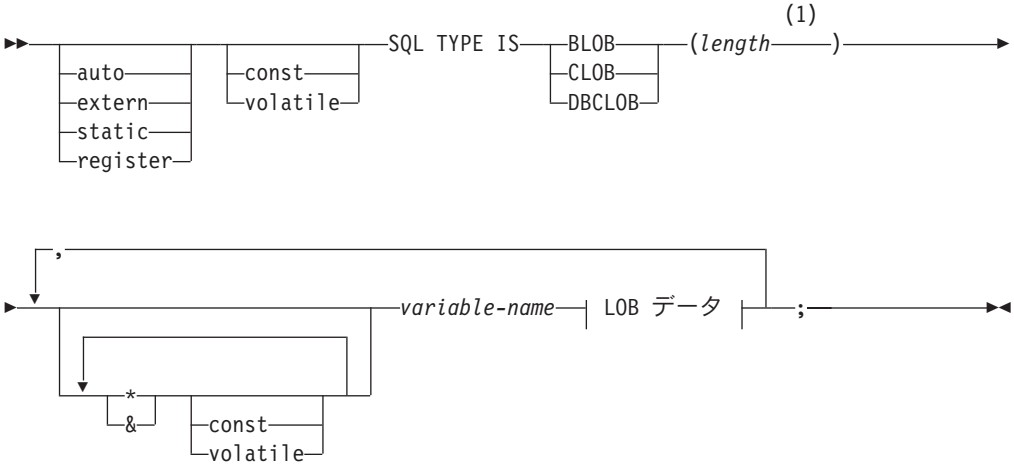
関連概念:

• 179 ページの『C および C++ での wchar_t および sqldbchar データ・タイプ』

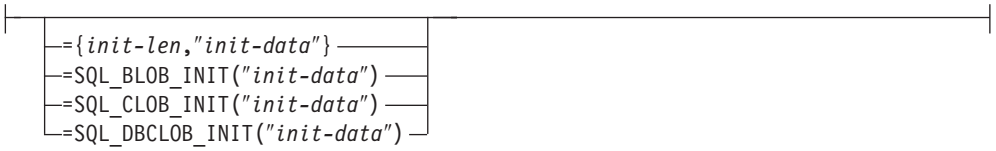
C または C++ におけるラージ・オブジェクト (LOB) ホスト変数の構文

C または C++ におけるラージ・オブジェクト (LOB) ホスト変数の宣言構文を次に示します。

C または C++ におけるラージ・オブジェクト (LOB) ホスト変数の構文



LOB データ



注:

1 *length* は、任意の有効な定数式。これには定数 K、M、または G を使用できる。 BLOB および CLOB の評価後の length 値は、 1 <= length <= 2 147 483 647 でなければなりません。 DBCLOB の評価後の *length* の値は、1 <= length <= 1 073 741 823 でなければなりません。

LOB ホスト変数に関する考慮事項:

1. 関数に渡される LOB タイプのホスト変数に対してタイプ検査と関数分解を実行できるように、 3 タイプの LOB を区別するための SQL TYPE IS 文節が必要である。
2. SQL TYPE IS、BLOB、CLOB、DBCLOB、K、M、G は、大文字と小文字が混在してもかまわない。
3. 初期化ストリング *"init-data"* に許可される最大長は、ストリング区切り文字を含めて 32 702 バイトである (プリコンパイラー内の C/C++ ストリングの限界と同じ)。
4. 初期化長である *init-len* は、数値の定数でなければならない (すなわち、K、M、G は使用できない)。
5. LOB の長さを指定しなければならない。すなわち、次の宣言は無効です。

   ```
   SQL TYPE IS BLOB my_blob;
   ```

6. LOB を宣言内で初期化しないと、プリコンパイラーで生成されたコード内での初期化は行われない。
7. DBCLOB を初期化する場合、ユーザーは、ストリングに 'L' (ワイド・キャラクター・ストリングを表す) という接頭部を付けること。

   注: ワイド・キャラクター・リテラル、たとえば `L"Hello"` は、 WCHARTYPE CONVERT プリコンパイル・オプションを選択した場合に、プリコンパイル済みプログラムでのみ使用すべきである。
8. プリコンパイラーは、ホスト変数のタイプをキャストするために使用できる構造体タグを生成する。

BLOB の例:

宣言:

```
static Sql Type is Blob(2M) my_blob=SQL_BLOB_INIT("mydata");
```

この結果、以下の構造が生成されます。

```
static struct my_blob_t {
      sqluint32        length;
      char             data[2097152];
 } my_blob=SQL_BLOB_INIT("mydata");
```

CLOB の例:

宣言:

```
volatile sql type is clob(125m) *var1, var2 = {10, "data5data5"};
```

この結果、以下の構造が生成されます。

```
volatile struct var1_t {
     sqluint32        length;
     char             data[131072000];
} * var1, var2 = {10, "data5data5"};
```

DBCLOB の例:

宣言:

```
SQL TYPE IS DBCLOB(30000) my_dbclob1;
```

WCHARTYPE NOCONVERT オプション指定でプリコンパイルされ、その結果、以下の構造体が生成されます。

```
struct my_dbclob1_t {
     sqluint32        length;
     sqldbchar        data[30000];
} my_dbclob1;
```

宣言:

```
SQL TYPE IS DBCLOB(30000) my_dbclob2 = SQL_DBCLOB_INIT(L"mydbdata");
```

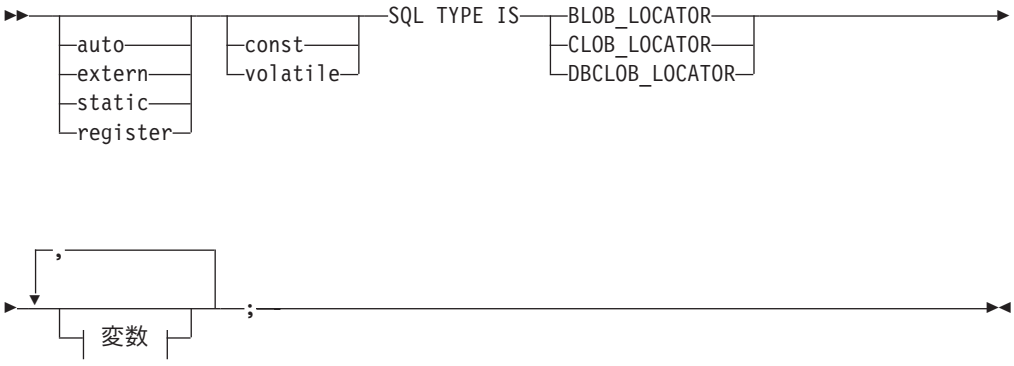
WCHARTYPE CONVERT オプション指定でプリコンパイルされ、その結果、以下の構造体が生成されます。

```
struct my_dbclob2_t {
     sqluint32        length;
     wchar_t          data[30000];
} my_dbclob2 = SQL_DBCLOB_INIT(L"mydbdata");
```

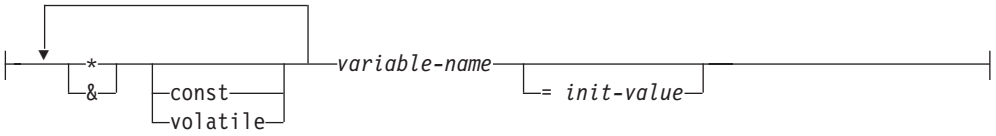
C または C++ におけるラージ・オブジェクト (LOB) ロケーター・ホスト変数の構文

C または C++ におけるラージ・オブジェクト (LOB) ロケーター・ホスト変数の宣言構文を次に示します。

C または C++ におけるラージ・オブジェクト (LOB) ロケーター・ホスト変数の構文



変数



LOB ロケーター・ホスト変数に関する考慮事項:

1. SQL TYPE IS、BLOB-LOCATOR、CLOB-LOCATOR、DBCLOB-LOCATOR は、大文字小文字混合のいずれでもかまわない。
2. *init-value* により、ポインターの初期化およびロケーター変数の参照ができる。他のタイプの初期化は、無意味となる。

CLOB ロケーターの例 (他のタイプの LOB ロケーターの場合も同様):

宣言:

```
SQL TYPE IS CLOB_LOCATOR my_locator;
```

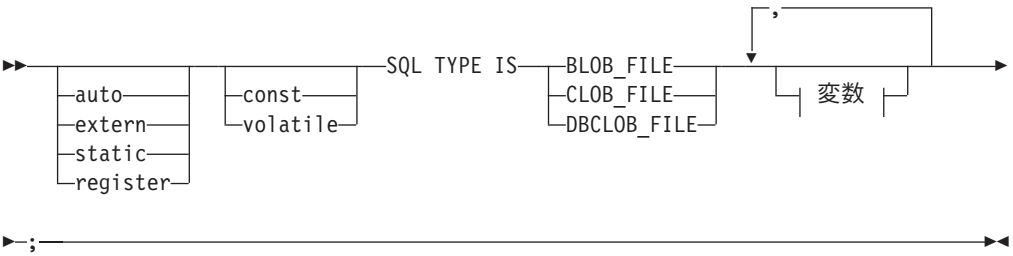
この結果、以下の宣言が生成されます。

```
sqluint32 my_locator;
```

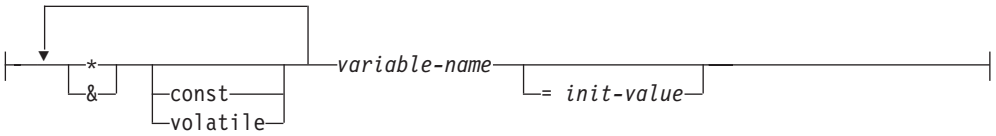
C または C++ におけるファイル参照ホスト変数宣言の構文

C または C++ におけるファイル参照ホスト変数の宣言構文を次に示します。

C または C++ におけるファイル参照ホスト変数の構文



変数



注: SQL TYPE IS、BLOB-FILE、CLOB-FILE、DBCLOB-FILE は、大文字小文字混合のいずれでもかまわない。

CLOB ファイル参照の例 (その他の LOB ファイル参照の型宣言も同様):

宣言:

```
static volatile SQL TYPE IS BLOB_FILE my_file;
```

この結果、以下の構造が生成されます。

```
static volatile struct {
     sqluint32        name_length;
     sqluint32        data_length;
     sqluint32        file_options;
              char    name[255];
} my_file;
```

C および C++ におけるホスト変数の初期化

C++ の宣言セクションでは、括弧を使用してホスト変数を初期化することはできません。次に、宣言セクション内での初期化の正しい方法と誤った方法の例を示します。

```
EXEC SQL BEGIN DECLARE SECTION;
  short my_short_2 = 5;       /* correct   */
  short my_short_1(5);        /* incorrect */
EXEC SQL END DECLARE SECTION;
```

C マクロ展開

C/C++ プリコンパイラーは、宣言セクション内の宣言で使用された C マクロを直接処理できません。代わりに、まず、外部 C プリプロセッサーでソース・ファイルをプリプロセスしなければなりません。これを実行するには、PREPROCESSOR オプションを使って、C プリプロセッサーを起動するためのコマンドをプリコンパイラーに指定します。

PREPROCESSOR オプションを指定すると、プリコンパイラーはまず、 SQL INCLUDE ステートメントで参照されているすべてのファイルの内容をソース・ファイルに結合させることによって、すべての SQL INCLUDE ステートメントを処理します。次にプリコンパイラーは、修正したソース・ファイルを入力として指定するコマンドを使用して、外部 C プリプロセッサーを起動します。プリプロセス済みのファイル (拡張子 .i によってプリコンパイラーは識別) は、プリコンパイルの残りのプロセスでの新しいソース・ファイルとして使用されます。

プリコンパイラーにより生成された任意の #line マクロは、元のソース・ファイルを参照することはありません。代わりに、プリプロセスされたファイルを参照します。コンパイラー・エラーを元のソース・ファイルに関連付けるには、プリプロセスされたファイルにコメントを含めるようにします。これにより、ヘッダー・ファイルを含むオリジナル・ソース・ファイルのあらゆるセクションを位置指定できます。通常、コメントを含めるようにするオプションは C プリプロセッサーで使用でき、 PREPROCESSOR オプションにより、指定するコマンドにこのオプションを含めることができます。 C プリプロセッサーには、#line マクロ自体を出力させてはなりません。これには、プリコンパイラーにより生成されたものが誤って混在している可能性があるためです。

マクロ展開の使用上の注意:

1. PREPROCESSOR オプションを使用して指定するコマンドには、すべての望むオプションを含めることができますが、入力ファイルの名前を含めることはできません。 たとえば、AIX® 上の IBM® C には、次のオプションを使用できます。

   ```
   xlC -P -DMYMACRO=1
   ```

2. プリコンパイラーは、このコマンドによって、拡張子 .i の付いたプリプロセス済みのファイルが生成されることを予期します。ただし、プリプロセス済みのファイルを生成するためにリダイレクトを使用することはできません。たとえば、以下のオプションを使用してプリプロセス済みファイルを生成することは**できません**。

   ```
   xlC -E > x.i
   ```

3. 外部 C プリプロセッサーが検出したエラーは、元のソース・ファイルに対応する名前に拡張子 .err を付けたファイルにレポートされます。

たとえば、以下のようにソース・コード内でマクロ展開を使用することができます。

```
#define SIZE 3

EXEC SQL BEGIN DECLARE SECTION;
  char a[SIZE+1];
  char b[(SIZE+1)*3];
  struct
  {
    short length;
    char  data[SIZE*6];
  } m;
  SQL TYPE IS BLOB(SIZE+1) x;
  SQL TYPE IS CLOB((SIZE+2)*3) y;
  SQL TYPE IS DBCLOB(SIZE*2K) z;
EXEC SQL END DECLARE SECTION;
```

PREPROCESSOR オプションを使用した後は、上記の宣言は以下のように解決します。

```
EXEC SQL BEGIN DECLARE SECTION;
  char a[4];
  char b[12];
  struct
  {
    short length;
    char  data[18];
  } m;
  SQL TYPE IS BLOB(4) x;
  SQL TYPE IS CLOB(15) y;
  SQL TYPE IS DBCLOB(6144) z;
EXEC SQL END DECLARE SECTION;
```

C および C++ でのホスト構造体サポート

ホスト構造体サポートを使用すると、C/C++ プリコンパイラーは、複数のホスト変数を単一のホスト構造体にグループ化することができます。この機能により、SQL ステートメントで同じセットのホスト変数を参照するのが簡単になります。たとえば、以下のホスト構造体は、SAMPLE データベース内の STAFF 表のいくつかの列へのアクセスに使用できます。

```
struct tag
   {
     short id;
     struct
     {
       short length;
       char  data[10];
     } name;
     struct
```

```
    {
      short   years;
      double salary;
    } info;
  } staff_record;
```

ホスト構造体のフィールドは、有効な任意のホスト変数タイプにすることができます。有効なタイプには、すべての数字、文字、およびラージ・オブジェクト・タイプが含まれます。ネストされるホスト構造体は、25 レベルまでサポートされます。上の例では、フィールド info は下位の構造体であるのに対し、フィールド name は下位の構造体ではなく、VARCHAR フィールドを示しています。 同じ原則は、LONG VARCHAR、 VARGRAPHIC および LONG VARGRAPHIC にも当てはまります。ホスト構造体へのポインターもサポートされます。

SQL ステートメントでホスト構造体にグループ化されるホスト変数を参照するには、以下の 2 つの方法があります。

- SQL ステートメントでホスト構造体名を参照する。

  ```
  EXEC SQL SELECT id, name, years, salary
       INTO :staff_record
       FROM staff
       WHERE id = 10;
  ```

  プリコンパイラーは staff_record の参照を、ホスト構造体で宣言されたすべてのフィールドをコンマで区切ったリストに変換します。 他のホスト変数またはフィールドとの重複を避けるために、それぞれのフィールドは、すべてのレベルのホスト構造体で修飾されます。これは以下の使用法と同じです。

- SQL ステートメントで完全修飾ホスト変数名を参照する。

  ```
  EXEC SQL SELECT id, name, years, salary
       INTO :staff_record.id, :staff_record.name,
            :staff_record.info.years, :staff_record.info.salary
       FROM staff
       WHERE id = 10;
  ```

  同じ名前のホスト変数がない場合でも、フィールド名を参照する際には完全修飾しなければなりません。 修飾された下位の構造体も参照できます。上の例では、:staff_record.info.years, :staff_record.info.salary を、:staff_record.info に置換することができます。

ホスト構造体への参照 (1 番目の例) は、コンマで区切ったフィールドのリストと等しいため、このタイプの参照はエラーとなる場合があります。 たとえば、次のようになります。

```
EXEC SQL DELETE FROM :staff_record;
```

ここでの DELETE ステートメントは、1 バイト文字ベースの変数を予期しています。代わりにホスト構造体を指定すると、ステートメントはプリコンパイル時にエラーになる可能性があります。

```
SQL0087N  Host variable "staff_record" is a structure used where structure
references are not permitted.
```

SQL0087N エラーの原因となる可能性があるホスト構造体のこの他の使用には、PREPARE、EXECUTE IMMEDIATE、CALL、標識変数、および SQLDA 参照などがあります。このような状況では、個々のフィールドへの参照と同じように (2 番目の例)、フィールドを 1 つしか持たないホスト構造体なら許可されます。

関連概念:

- 174 ページの『C および C++ での標識表』

C および C++ での標識表

標識表は、ホスト構造体で使用される標識変数の集合です。これは、短整数の配列として宣言しなければなりません。たとえば、次のようになります。

```
short ind_tab[10];
```

上の例は、エレメントが 10 個の標識表を宣言します。以下に、これを SQL ステートメントで使用する方法を示します。

```
EXEC SQL SELECT id, name, years, salary
     INTO :staff_record INDICATOR :ind_tab
     FROM staff
     WHERE id = 10;
```

以下の表では、それぞれのホスト構造体フィールドとそれに対応する標識変数をリストしています。

staff_record.id	ind_tab[0]
staff_record.name	ind_tab[1]
staff_record.info.years	ind_tab[2]
staff_record.info.salary	ind_tab[3]

注: 標識表エレメント、たとえば `ind_tab[1]` は、 SQL ステートメントで個々に参照することはできません。キーワード INDICATOR はオプションです。構造化フィールドと標識の数が一致している必要はありません。余分の標識が未使用だったり、標識が割り当てられていない余分のフィールドがあってもかまいません。

標識表の代わりにスカラー標識変数を使用して、ホスト構造体の最初のフィールドに標識を提供することもできます。これは、1 つのエレメントだけの標識表を持つことと同じです。 たとえば、次のようになります。

```
short scalar_ind;

EXEC SQL SELECT id, name, years, salary
          INTO :staff_record INDICATOR :scalar_ind
          FROM staff
          WHERE id = 10;
```

ホスト構造体の代わりにホスト変数を指定して標識表を指定すると、標識表の最初のエレメント、たとえば `ind_tab[0]` しか使用されません。

```
EXEC SQL SELECT id
          INTO :staff_record.id INDICATOR :ind_tab
          FROM staff
          WHERE id = 10;
```

短整数の配列がホスト構造体内で宣言される場合、以下のようになります。

```
struct tag
{
  short i[2];
} test_record;
```

SQL ステートメントで test_record が参照されるときに配列がエレメントに展開されると、 :test_record は、:test_record.i[0], :test_record.i[1] と同等になります。

関連概念:

- 172 ページの『C および C++ でのホスト構造体サポート』

C および C++ でのヌル終了ストリング

C/C++ のヌル終了ストリングは、独自の SQLTYPE (文字の場合は 460/461 で、グラフィックの場合は 468/469) を持ちます。

C/C++ のヌル終了ストリングは、 LANGLEVEL プリコンパイラー・オプションの値に基づいてさまざまに処理されます。 これらの SQLTYPE 値のうちの 1 つのホスト変数と宣言された長さ *n* を SQL ステートメント内に指定し、さらにデータのバイト数 (文字タイプの場合) または 2 バイト文字 (グラフィック・タイプの場合) が *k* である場合を以下に説明します。

- PREP コマンドの LANGLEVEL オプションが SAA1 (デフォルト) の場合:

  出力の場合:

k と n の関係 ...	**結果 ...**
k **>** *n*	*n* 文字はターゲットのホスト変数に移動され、SQLWARN1 は 'W'、および SQLCODE 0 (SQLSTATE 01004) に設定される。 NULL 終止符はストリング内では使用されません。標識変数をホスト変数で指定した場合には、標識変数の値は *k* に設定されます。
k **=** *n*	*k* 文字はターゲットのホスト変数に移動され、SQLWARN1 は 'N'、および SQLCODE 0 (SQLSTATE 01004) に設定される。 NULL 終止符はストリング内では使用されません。標識変数をホスト変数で指定した場合には、標識変数の値は 0 に設定されます。
k **<** *n*	*k* 文字はターゲットのホスト変数に移動され、NULL 文字は文字 *k + 1* に置かれる。 標識変数をホスト変数で指定した場合には、標識変数の値は 0 に設定されます。

  入力の場合: データベース・マネージャーは、最後が NULL 終止符ではないこれらの SQLTYPE 値のうちの 1 つの入力ホスト変数を発見すると、文字 *n+1* に NULL 終止符文字が含まれると想定します。

- PREP コマンドの LANGLEVEL オプションが MIA である場合:

  出力の場合:

k と n の関係 ...	**結果 ...**
k **>=** *n*	*n - 1* 文字はターゲットのホスト変数に移動さ

れ、 SQLWARN1 は 'W'、および SQLCODE 0 (SQLSTATE 01501) に設定される。 *n* 番目の文字は NULL 終止符に設定されます。標識変数をホスト変数で指定した場合には、標識変数の値は *k* に設定されます。

k + 1 = n *k* 文字はターゲットのホスト変数に移動され、NULL 終止符は文字 *n* に置かれる。標識変数をホスト変数で指定した場合には、標識変数の値は 0 に設定されます。

k + 1 < n *k* 文字はターゲットのホスト変数に移動され、文字 *k + 1* で開始している右側に *n - k -1* 個のブランクが追加される。その後、NULL 終止符が文字 *n* に置かれます。標識変数をホスト変数で指定した場合には、標識変数の値は 0 に設定されます。

入力の場合: データベース・マネージャーが、最後が NULL 文字ではないこれらの SQLTYPE 値のうちの 1 つの入力ホスト変数を発見すると、 SQLCODE -302 (SQLSTATE 22501) が戻されます。

長さが *n* の SQLTYPE 460 のホスト変数を他の SQL コンテキスト内に指定すると、上で定義したように、長さ *n* の VARCHAR データ・タイプとして処理されます。長さが *n* の SQLTYPE 468 のホスト変数をその他の SQL コンテキスト内に指定した場合には、上での定義のように、長さ *n* の VARGRAPHIC データ・タイプとして処理されます。

C および C++ でポインター・データ・タイプとして使用されるホスト変数

ホスト変数は、特定のデータ・タイプへのポインターとして、以下のような制限付きで宣言されることがあります。

- ホスト変数をポインターとして宣言する場合、その他のホスト変数を、同じソース・ファイル内で同じ名前で宣言することはできない。 以下の例は無効です。

```
char mystring[20];
char (*mystring)[20];
```

- NULL 終了文字配列へのポインターを宣言する場合は、括弧を使用する。その他のすべての場合は、括弧を使用することはできません。たとえば、次のようになります。

```
EXEC SQL BEGIN DECLARE SECTION;
  char  (*arr)[10];  /* correct   */
  char  *(arr);      /* incorrect */
  char  *arr[10];    /* incorrect */
EXEC SQL END DECLARE SECTION;
```

この例で、最初の宣言は 10 バイトの文字配列へのポインターです。これは有効なホスト変数となっています。 2 番目は無効な宣言です。文字へのポインターでは括弧は使用できません。 3 番目の宣言はポインターの配列です。このデータ・タイプはサポートされていません。

以下のようなホスト変数の宣言があるとします。

```
char *ptr
```

この宣言は受け入れられますが、*長さが未指定の NULL 終了文字ストリング* を意味するわけではありません。その代わり、これは*固定長の単一文字のホスト変数へのポインター* であることを表します。 これは意図された宣言ではないかもしれません。別の文字ストリングを指示することができるポインター・ホスト変数を定義するには、上記の最初の宣言書式を用いてください。

- SQL ステートメントでポインター・ホスト変数を使用する場合は、以下の例のように、宣言されているのと同じ数のアスタリスクを前に付ける。

```
EXEC SQL BEGIN DECLARE SECTION;
  char (*mychar)[20];    /* Pointer to character array of 20 bytes */
EXEC SQL END DECLARE SECTION;

EXEC SQL SELECT column INTO :*mychar FROM table;   /* Correct */
```

- ホスト変数名には、アスタリスクだけが演算子として使用できる。
- アスタリスクは名前の一部と見なされないため、ホスト変数名の最大長は指定されたアスタリスクの数の影響を受けない。
- SQL ステートメント内でポインター変数を使用する場合は必ず、最適化レベルのプリコンパイル・オプション (OPTLEVEL) をデフォルト設定の 0 (最適化を行わない) のままにしておく。これは、データベース・マネージャーが SQLDA の最適化を行わないということを意味します。

C および C++ でホスト変数として使用されるクラス・データ・メンバー

クラス・データ・メンバーは、ホスト変数として宣言できます (クラスまたはオブジェクト自身ではなく)。以下は、使用方法を説明する例です。

```
class STAFF
{
    private:
         EXEC SQL BEGIN DECLARE SECTION;
           char        staff_name[20];
           short int   staff_id;
           double      staff_salary;
         EXEC SQL END DECLARE SECTION;
         short       staff_in_db;
    .
    .
};
```

データ・メンバーへは、クラス・メンバー関数内の C++ コンパイラーにより提供される暗黙の *this* ポインターを介して、 SQL ステートメント内で直接アクセスできるだけです。 SQL ステートメント内でオブジェクト・インスタンス (`SELECT name INTO :my_obj.staff_name ...` など) を明示的に修飾することは**できません**。

SQL ステートメント内でクラス・データ・メンバーを直接参照する場合は、データベース・マネージャーが *this* ポインターを使用して参照を解決します。こうした理由から、最適化レベルのプリコンパイル・オプション (OPTLEVEL) は、デフォルト設定の 0 (最適化を行わない) のままにしておいてください。これは、データベー

ス・マネージャーが SQLDA の最適化を行わないということを意味します。 (これは、ポインター・ホスト変数が SQL ステートメントに含まれていれば常に当てはまります。)

次の例は、SQL ステートメント内でホスト変数として宣言したクラス・データ・メンバーを、直接使用する方法を示しています。

```
class STAFF
{

⋮
    public:


⋮
       short int hire( void )
       {
         EXEC SQL INSERT INTO staff ( name,id,salary )
           VALUES ( :staff_name, :staff_id, :staff_salary );
         staff_in_db = (sqlca.sqlcode == 0);
         return sqlca.sqlcode;
       }
};
```

この例では、クラス・データ・メンバーである staff_name、staff_id、および staff_salary が INSERT ステートメント内で直接使用されています。これらはホスト変数として宣言されているため (このセクションの最初の例を参照)、 *this* ポインターを用いて、現行対象に対して暗黙のうちに修飾されています。 SQL ステートメントでは、 *this* ポインターを介してアクセスすることができないデータ・メンバーも参照することができます。 これは、ポインターまたは参照ホスト変数を使用してこれらを間接的に参照することにより行うことができます。

次の例は、2 番目のオブジェクトである *otherGuy* を獲得する、 *asWellPaidAs* という新しい方法を示しています。 この方法では、SQL ステートメント内でメンバーを直接参照できないため、ローカル・ポインターまたは参照ホスト変数を介して間接的にメンバーを参照します。

```
short int STAFF::asWellPaidAs( STAFF otherGuy )
{
    EXEC SQL BEGIN DECLARE SECTION;
      short &otherID = otherGuy.staff_id
      double otherSalary;
    EXEC SQL END DECLARE SECTION;
    EXEC SQL SELECT SALARY INTO :otherSalary
      FROM STAFF WHERE id = :otherID;
      if( sqlca.sqlcode == 0 )
         return staff_salary >= otherSalary;
      else
         return 0;
}
```

C および C++ での修飾およびメンバー演算子

組み込み SQL ステートメント内で、C++ 有効範囲解決演算子 '::' を使用したり、C/C++ メンバー演算子 '.' または '->' を使用したりすることはできません。 これと同じことは、ローカル・ポインターまたは参照変数を使用することにより、簡単に行うことができます。ローカル・ポインターや参照変数は SQL ステートメントの

外部に設定して使用したい効力範囲内の変数を指定し、その後は SQL ステートメント内でこれを参照するために使用されます。以下に、正しい使用方法の例を示します。

```
EXEC SQL BEGIN DECLARE SECTION;
  char (& localName)[20] = ::name;
EXEC SQL END DECLARE SECTION;
EXEC SQL
  SELECT name INTO :localName FROM STAFF
  WHERE name = 'Sanders';
```

C および C++ でのマルチバイト文字のエンコード

文字のエンコード・スキーマの中には、特に東アジアの国々の文字には 1 つの文字を表すのにマルチバイトを必要とするものがあります。 このデータの外部表現は文字の *マルチバイト文字コード* 表現と呼ばれ、 2 バイト文字 (2 バイトで表される文字) を含みます。 DB2® のグラフィック・データは、2 バイト文字からなります。

2 バイト文字で文字ストリングを扱うためには、アプリケーションでデータの内部表現を使用するのが便利です。この内部表現は、2 バイト文字の *ワイド・キャラクター・コード* 表現と呼ばれており、通常 wchar_t C/C++ データ・タイプで使用される形式です。 ワイド・キャラクター・データの処理やワイド・キャラクター形式データのマルチバイト形式との変換を行うためには、 ANSI C および X/OPEN Portability Guide 4 (XPG4) に準拠するサブルーチンを使用することができます。

アプリケーションでは、文字データをマルチバイト形式またはワイド・キャラクター形式のどちらかで処理できますが、データベース・マネージャーとの対話は、DBCS (マルチバイト) 文字コードでしか行うことができないことに注意してください。 つまり、データの GRAPHIC 列への保管や GRAPHIC 列からの検索は、DBCS 形式で行われます。 WCHARTYPE プリコンパイラー・オプションは、ワイド・キャラクター形式のアプリケーション・データがデータベース・エンジンで交換される際に、これをマルチバイト形式に変換したり元に戻したりするために使用されます。

関連概念:

- 164 ページの『C および C++ でのグラフィック・ホスト変数』
- 179 ページの『C および C++ での wchar_t および sqldbchar データ・タイプ』

C および C++ での wchar_t および sqldbchar データ・タイプ

DB2® グラフィック・データのサイズおよびエンコードは、特定のコード・ページではどのプラットフォームでも同じですが、 ANSI C または C++ wchar_t データ・タイプのサイズおよび内部書式は、使用するコンパイラーとプラットフォームによって異なります。 しかしながら、sqldbchar データ・タイプは、DB2 によってサイズが 2 バイトと定義されており、データベース内で保管されるのと同じ形式で DBCS および UCS-2 データを操作する、移植可能な方法が使用されています。

DB2 C グラフィック・ホスト変数タイプはすべて、wchar_t か sqldbchar によって定義できます。 WCHARTYPE CONVERT プリコンパイル・オプションを使用してアプリケーションを構築する場合には、必ず wchar_t の方を使用してください。

注: Windows® プラットフォーム上で WCHARTYPE CONVERT オプションを指定する際には、 Windows プラットフォーム上の `wchar_t` は Unicode であることに注意してください。 したがって、ご使用の C/C++ コンパイラーの `wchar_t` が Unicode でない場合には、 `wcstombs()` 関数呼び出しは SQLCODE -1421 (SQLSTATE=22504) を出して失敗します。 この場合、WCHARTYPE NOCONVERT オプションを指定したり、ご使用のプログラムから `wcstombs()` および `mbstowcs()` 関数を明示的に呼び出したりすることができます。

WCHARTYPE NOCONVERT プリコンパイル・オプションを使用してアプリケーションを構築する場合には、異なる DB2 クライアントとサーバー・プラットフォーム間でも最大限の移植性を得られるよう、 `sqldbchar` の方を使用してください。 WCHARTYPE NOCONVERT を使用する場合でも `wchar_t` は使えますが、 `wchar_t` が 2 バイトで定義されているプラットフォームだけに限ります。

ホスト変数宣言で `wchar_t` か `sqldbchar` を誤って使用すると、プリコンパイル時に SQLCODE 15 (SQLSTATE ではない) が返されます。

関連概念:

- 180 ページの『C および C++ での WCHARTYPE プリコンパイラー・オプション』
- 673 ページの『日本語および中国語 (繁体字) EUC および UCS-2 コード・セットに関する考慮事項』

C および C++ での WCHARTYPE プリコンパイラー・オプション

WCHARTYPE プリコンパイラー・オプションを使用すると、 C/C++ アプリケーションでどのグラフィック文字形式を使用するかを指定できます。このオプションにより、グラフィック・データをマルチバイト形式またはワイド・キャラクター形式のどちらにするかを柔軟に選択することができます。 WCHARTYPE オプションには、次の 2 つの値があります。

CONVERT

WCHARTYPE CONVERT オプションを選択した場合、文字コードはグラフィック・ホスト変数とデータベース・マネージャーとの間で変換されます。グラフィック入力ホスト変数の場合、ワイド・キャラクター形式からマルチバイト DBCS 文字形式への文字コード変換は、データがデータベース・マネージャーに送信される前に ANSI C 関数の `wcstombs()` を使用して行われます。 グラフィック出力ホスト変数の場合には、マルチバイト DBCS 文字形式からワイド・キャラクター形式への文字コード変換は、データベース・マネージャーから受け取られたデータがホスト変数に保管される前に、ANSI C 関数の `mbstowcs()` を使用して実行されます。

WCHARTYPE CONVERT を使用する利点は、それによってアプリケーションが、データベース・マネージャーと通信する前に、データをマルチバイト形式に明示的に変換しなくても、ワイド・キャラクター・ストリング (L-リテラル、'wc' ストリング関数など) を処理するための ANSI C 機構を十分に利用できることです。短所としては、暗黙のうちに変換を行うことによってアプリケーション・ランタイムのパフォーマンスに影響を及ぼすことがあり、さらにメモリー要件が大きくなる恐れがあることが挙げられます。

WCHARTYPE CONVERT を選択した場合は、すべてのグラフィック・ホスト変数を、sqldbchar ではなく、wchar_t を使用して宣言してください。

WCHARTYPE CONVERT 振る舞いは希望するものの、アプリケーションはプリコンパイルする必要がない場合 (たとえば、CLI アプリケーション)、コンパイル時に C プリプロセッサー・マクロ SQL_WCHART_CONVERT を定義してください。 これによって、DB2 ヘッダー・ファイルの特定の定義で、データ・タイプ sqldbchar ではなく wchar_t が使用されます。

注: WCHARTYPE CONVERT プリコンパイル・オプションは、現在、DB2® Windows® 3.1 クライアントで実行するプログラムではサポートされていません。 それらのプログラムには、デフォルト (WCHARTYPE NOCONVERT) を使用してください。

NOCONVERT (デフォルト)

WCHARTYPE NOCONVERT オプションを選択した場合、あるいは WCHARTYPE オプションをまったく指定しない場合は、アプリケーションとデータベース・マネージャーの間で暗黙の文字コード変換は行われません。グラフィック・ホスト変数のデータは、変換されない DBCS 文字として、データベース・マネージャーとの間で送受信されます。 これにはパフォーマンスを向上させるという利点がありますが、短所としてアプリケーションが wchar_t ホスト変数内のワイド・キャラクター・データの使用をやめるか、またはデータベース・マネージャーとのインターフェースをとる際にデータのマルチバイト形式への変換のために wcstombs() および mbstowcs() 関数を明示的に呼び出さなければならないということがあります。

WCHARTYPE NOCONVERT を選択した場合は、他の DB2 クライアント/サーバー・プラットフォームへの移植性を最大限に得られるようにするため、すべてのグラフィック・ホスト変数を sqldbchar タイプを使用して宣言してください。

注意すべきその他の指針としては以下のものがあります。

- wchar_t または sqldbchar サポートは DBCS データの処理のために使用されるため、これを使用する場合は DBCS または EUC で使用可能なハードウェアとソフトウェアが必要になる。このサポートが使用可能であるのは、DBCS 環境の DB2 Universal Database か、または UCS-2 データベースに接続されている任意のアプリケーション (1 バイト・アプリケーションを含む) で GRAPHIC データを処理している場合のみです。
- DBCS 以外の文字と、DBCS 以外の文字に変換できるワイド・キャラクターは、GRAPHIC ストリング内では使用してはならない。 *DBCS 以外の文字* とは、 1 バイト文字と、2 バイト文字以外の文字のことを指します。 GRAPHIC ストリングでは、その値に 2 バイト文字のコード・ポイントのみが含まれているかどうかを確認するための妥当性検査は行われません。グラフィック・ホスト変数には、DBCS データ、または WCHARTYPE CONVERT が有効な場合には、 DBCS データに変換されるワイド・キャラクター・データしか含めることができません。2 バイト文字と 1 バイト文字が混在しているデータは、文字ホスト変数に保管してください。混合データのホスト変数は WCHARTYPE オプションの設定の影響を受けないことに注意してください。

- WCHARTYPE NOCONVERT プリコンパイル・オプションを使用しているアプリケーションでは、 L リテラルをグラフィック・ホスト変数とともに使用しない。これは、L リテラルがワイド・キャラクター形式であるためです。 L リテラルは、L という接頭部を付けた C 言語のワイド・キャラクター・ストリング・リテラルであり、データ・タイプは "array of wchar_t" です。たとえば、L"dbcs-string" は L リテラルです。
- L リテラルを使用した wchar_t ホスト変数の初期化は、WCHARTYPE CONVERT プリコンパイル・オプションを使用しているアプリケーションでは行えるものの、 SQL ステートメントでは使用できない。 SQL ステートメントでは、 L リテラルを使用する代わりに WCHARTYPE の設定から独立している GRAPHIC ストリング定数を使用してください。
- WCHARTYPE オプションの設定は、ホスト変数だけでなく SQLDA 構造体を使用してデータベース・マネージャーとの間で受け渡しするグラフィック・データに影響を与える。 WCHARTYPE CONVERT が有効な場合、 SQLDA を介してアプリケーションから受け取られるグラフィック・データはワイド・キャラクター形式と見なされ、 wcstombs() を暗黙のうちに呼び出して DBCS 形式に変換されます。同様に、アプリケーションが受け取るグラフィック出力データは、アプリケーション・ストレージに保管される前にワイド・キャラクター形式に変換されています。
- 境界域が設定されていないストアード・プロシージャーは、 WCHARTYPE NOCONVERT オプションを用いてプリコンパイルしなければならない。通常、境界域の設定されていないストアード・プロシージャーは CONVERT または NOCONVERT のいずれかのオプションを用いてプリコンパイルすることができますが、このオプションの指定はストアード・プロシージャーに含まれる SQL ステートメントに操作されるグラフィック・データの形式に影響を及ぼします。ただしどちらの場合も、SQLDA を介してストアード・プロシージャーに渡されるグラフィック・データはすべて DBCS 形式となります。同じように、 SQLDA を介してストアード・プロシージャーから渡されるデータも DBCS 形式でなければなりません。
- アプリケーションがデータベース・アプリケーション・リモート・インターフェース (DARI) のインターフェース (sqleproc() API) を介してストアード・プロシージャーを呼び出す場合、入力 SQLDA 内のグラフィック・データはすべて、呼び出しているアプリケーションの WCHARTYPE 設定に関係なく、DBCS 形式でなければならない。または UCS-2 データベースに接続されている場合は、UCS-2 でなければならない。 同じく、出力 SQLDA 内のグラフィック・データはすべて、 WCHARTYPE 設定に関係なく、 DBCS 形式、または UCS-2 データベースに接続されている場合は UCS-2 形式で戻されます。
- アプリケーションが SQL CALL ステートメントを介してストアード・プロシージャーを呼び出す場合は、呼び出しているアプリケーションの WCHARTYPE 設定に従って、 SQLDA でグラフィック・データが変換される。
- ユーザー定義関数 (UDF) に渡されるグラフィック・データは、常に DBCS 形式である。同じように、UDF から戻されるグラフィック・データもすべて、 DBCS データベースでは DBCS 形式、 EUC および UCS-2 データベースでは UCS-2 形式と見なされます。
- DBCLOB ファイル参照変数の使用により DBCLOB ファイルに保管されるデータは、 DBCS 形式か、または UCS-2 データベースの場合には、UCS-2 形式で保管

されます。同様に、DBCLOB ファイルからの入力データは、 DBCS 形式か、または UCS-2 データベースの場合には UCS-2 形式のいずれかで検索されます。

注: C 言語アプリケーションを WCHARTYPE CONVERT オプションを使用してプリコンパイルする場合、 DB2 は変換関数がデータを渡すときに、入出力両方のアプリケーションのグラフィック・データを妥当性検査します。 CONVERT オプションを使用しない場合は、グラフィック・データの変換は行われず、したがって検証も行われません。このことが CONVERT/NOCONVERT 混合環境では、無効なデータが NOCONVERT アプリケーションによって挿入され、それを CONVERT アプリケーションが取り出したりする場合に、問題の原因になります。このようなデータの変換は失敗し、CONVERT アプリケーションでの FETCH 時に、SQLCODE -1421 (SQLSTATE 22504) が返されます。

関連資料:

- 「*SQL リファレンス 第 2 巻*」の『PREPARE ステートメント』

C および C++ での日本語または中国語 (繁体字) EUC、および UCS-2 に関する考慮事項

アプリケーション・コード・ページが日本語または中国語 (繁体字) EUC の場合、またはアプリケーションが UCS-2 データベースと接続されている場合、CONVERT オプションか NOCONVERT オプションのどちらか、および `wchar_t` または `sqldbchar` グラフィック・ホスト変数、または入力/出力 SQLDA を使用することにより、データベース・サーバーで GRAPHIC 列にアクセスできます。 この節で *DBCS 形式* に言及する場合、それは EUC データ用の UCS-2 エンコード・スキーマを指します。次の 2 つのケースを考えてみてください。

- CONVERT オプションを使用する場合

  DB2® クライアントによって、グラフィック・データがワイド・キャラクター形式からご使用のアプリケーション・コード・ページに変換され、その後、入力 SQLDA をデータベース・サーバーに送信する前に UCS-2 に変換します。 グラフィック・データはすべて、UCS-2 コード・ページ ID によってタグ付けされたデータベース・サーバーに送られます。混合文字データは、アプリケーション・コード・ページ ID によってタグ付けされます。 クライアントによってデータベースからグラフィック・データが取り出されると、そのグラフィック・データは UCS-2 コード・ページ ID によってタグ付けされます。 DB2 クライアントが、データを UCS-2 からクライアント・アプリケーション・コード・ページへ変換し、さらにそれをワイド・キャラクター形式に変換します。 ホスト変数の代わりに入力 SQLDA を使用した場合は、グラフィック・データを必ずワイド・キャラクター形式でエンコードする必要があります。 このデータは UCS-2 に変換され、その後データベース・サーバーに送られます。上記の変換はパフォーマンスに影響を及ぼします。

- NOCONVERT オプションを使用する場合

  グラフィック・データは UCS-2 によってエンコードされ、 UCS-2 コード・ページでタグ付けされたものと DB2 からは見なされます。変換は行われません。 DB2 は、グラフィック・ホスト変数を単にバケットとして使用されるものと見なします。 NOCONVERT オプションを選択した場合、データベース・サーバーか

ら取り出されるグラフィック・データは、 UCS-2 によってエンコードされたアプリケーションに渡されます。アプリケーション・コード・ページから UCS-2、および UCS-2 からアプリケーション・コード・ページへの変換は、すべてユーザーの責任で行うことになります。 UCS-2 としてタグ付けされたデータは、変換や置換なしでデータベース・サーバーに送られます。

変換を最小限に抑えるには、 NOCONVERT オプションを使用してアプリケーション内で変換を処理するか、または GRAPHIC 列を使用しないかのいずれかです。 wchar_t エンコードが 2 バイト Unicode のクライアント環境 (たとえば Windows® NT または AIX® バージョン 4.3 およびそれ以降など) の場合には、 NOCONVERT オプションを使用して直接 UCS-2 で作業できます。 この場合、ご使用のアプリケーションはビッグ・エンディアンとリトル・エンディアン・アーキテクチャーとの違いを扱わなければなりません。 DB2 Universal Database は、 NOCONVERT オプションを使用する場合、常に 2 バイト・ビッグ・エンディアンである sqldbchar を使用します。

UCS-2 への変換後 (NOCONVERT 指定の場合) や、ワイド・キャラクター形式への変換 (CONVERT 指定の場合) によって、 IBM®-eucJP/IBM-eucTW CS0 (7 ビット ASCII) データおよび IBM-eucJP CS2 (カタカナ) データをグラフィック・ホスト変数に割り当てることはしないでください。これは、どちらの EUC コード・セットも UCS-2 から PC DBCS へと変換すると単一バイトになってしまうためです。

通常、eucJP および eucTW は GRAPHIC データを UCS-2 として保管しますが、これらのデータベースにある GRAPHIC データは非 ASCII eucJP または eucTW データのままです。 特に、そのような GRAPHIC データに埋め込まれるスペースは、DBCS スペースです (UCS-2、U+3000 では表意文字スペースとも呼ばれます)。しかし、UCS-2 データベースの場合には、 GRAPHIC データに UCS-2 文字を含めることができ、スペースの埋め込みは UCS-2 スペース、U+0020 を使用して実行されます。 UCS-2 データベースから UCS-2 を検索する場合と、 eucJP および eucTW データベースから UCS-2 データを検索する場合には、この違いに注意してください。

関連概念:

- 673 ページの『日本語および中国語 (繁体字) EUC および UCS-2 コード・セットに関する考慮事項』

C および C++ 用のホスト変数がある SQL 宣言セクション

以下に、サポートされている SQL データ・タイプのために宣言されたホスト変数を使用したサンプルの SQL 宣言セクションを示します。

```
EXEC SQL BEGIN DECLARE SECTION;

:
short     age = 26;              /* SQL type  500 */
short     year;                  /* SQL type  500 */
sqlint32  salary;                /* SQL type  496 */
sqlint32  deptno;                /* SQL type  496 */
float     bonus;                 /* SQL type  480 */
double    wage;                  /* SQL type  480 */
char      mi;                    /* SQL type  452 */
char      name[6];               /* SQL type  460 */
```

```
struct   {
          short len;
          char data[24];
         } address;               /* SQL type  448 */
struct   {
          short len;
          char data[32695];
         } voice;                 /* SQL type  456 */
sql type is clob(1m)
          chapter;                /* SQL type  408 */
sql type is clob_locator
          chapter_locator;        /* SQL type  964 */
sql type is clob_file
          chapter_file_ref;       /* SQL type  920 */
sql type is blob(1m)
          video;                  /* SQL type  404 */
sql type is blob_locator
          video_locator;          /* SQL type  960 */
sql type is blob_file
          video_file_ref;         /* SQL type  916 */
sql type is dbclob(1m)
          tokyo_phone_dir;        /* SQL type  412 */
sql type is dbclob_locator
          tokyo_phone_dir_lctr;   /* SQL type  968 */
sql type is dbclob_file
          tokyo_phone_dir_flref;  /* SQL type  924 */
struct   {
          short len;
          sqldbchar data[100];
         } vargraphic1;           /* SQL type  464 */
                                  /* Precompiled with
                                  WCHARTYPE NOCONVERT option */
struct   {
          short len;
          wchar_t data[100];
         } vargraphic2;           /* SQL type  464 */
                                  /* Precompiled with
                                  WCHARTYPE CONVERT option */
struct   {
          short len;
          sqldbchar data[10000];
         } long_vargraphic1;      /* SQL type  472 */
                                  /* Precompiled with
                                  WCHARTYPE NOCONVERT option */
struct   {
          short len;
          wchar_t data[10000];
         } long_vargraphic2;      /* SQL type  472 */
                                  /* Precompiled with
                                  WCHARTYPE CONVERT option */
sqldbchar graphic1[100];          /* SQL type  468 */
                                  /* Precompiled with
                                  WCHARTYPE NOCONVERT option */
wchar_t   graphic2[100];          /* SQL type  468 */
                                  /* Precompiled with
                                  WCHARTYPE CONVERT option */
char      date[11];               /* SQL type  384 */
char      time[9];                /* SQL type  388 */
char      timestamp[27];          /* SQL type  392 */
short     wage_ind;               /* Null indicator */

:
    EXEC SQL END DECLARE SECTION;
```

C および C++ におけるデータ・タイプに関する考慮事項

以下のセクションでは、SQL データ・タイプがどのように C および C++ データにマップされるか説明します。

C および C++ においてサポートされている SQL データ・タイプ

特定の事前定義済み C および C++ データ・タイプは、データベース・マネージャーの列タイプに対応しています。 これらの C/C++ データ・タイプのみが、ホスト変数として宣言できます。

それぞれの列タイプに対応する C/C++ の列タイプを次の表に示します。プリコンパイラーはホスト変数宣言を検出すると、該当する SQL タイプの値を判別します。データベース・マネージャーはこの値を使用して、アプリケーションとの間でやりとりするデータを変換します。

注: DB2 ホスト言語で、DATALINK データ・タイプをサポートするホスト変数はありません。

表 13. C/C++ 宣言にマップされた SQL データ・タイプ

SQL 列タイプ[1]	C/C++ データ・タイプ	SQL 列タイプ記述
SMALLINT (500 または 501)	short short int sqlint16	16 ビットの符号付き整数
INTEGER (496 または 497)	long long int sqlint32[2]	32 ビットの符号付き整数
BIGINT (492 または 493)	long long long __int64 sqlint64[3]	64 ビットの符号付き整数
REAL[4] (480 または 481)	float	単精度浮動小数点
DOUBLE[5] (480 または 481)	double	倍精度浮動小数点
DECIMAL(*p*,*s*) (484 または 485)	厳密な対応なし; double を使用	パック 10 進数 (パック 10 進フィールドを文字データとして操作するために CHAR および DECIMAL 関数を使用することを推奨。)
CHAR(1) (452 または 453)	char	単一文字
CHAR(*n*) (452 または 453)	厳密な対応なし; char[*n+1*] を使用 (n はデータを保持するだけの十分な大きさ) 1<=*n*<=254	固定長文字ストリング

表 13. C/C++ 宣言にマップされた SQL データ・タイプ (続き)

SQL 列タイプ[1]	C/C++ データ・タイプ	SQL 列タイプ記述

VARCHAR(*n*)
(448 または 449)
	struct tag {
	short int;
	char[*n*]
	}
	
	1<=*n*<=32 672
		2 バイトのストリング長指定子を持つ、NULL 終了可変文字以外のストリング

	代替使用; char[*n+1*] を使用 (n はデータを保持するだけの十分な大きさ) 1<=*n*<=32 672	NULL 終了可変長文字ストリング **注:** 460/461 の SQL タイプを割り当てられる。

LONG VARCHAR
(456 または 457)
	struct tag {
	short int;
	char[*n*]
	}
	
	32 673<=*n*<=32 700
		2 バイトのストリング長指定子を持つ、NULL 終了可変文字以外のストリング

CLOB(*n*) (408 または 409)	sql type is clob(*n*) 1<=*n*<=2 147 483 647	4 バイトのストリング長指定子を持つ、NULL 終了可変文字以外のストリング
CLOB ロケーター変数[6] (964 または 965)	sql type is clob_locator	サーバー上の CLOB エンティティーを識別する
CLOB ファイル参照変数[6] (920 または 921)	sql type is clob_file	CLOB データを含むファイルの記述子
BLOB(*n*) (404 または 405)	sql type is blob(*n*) 1<=*n*<=2 147 483 647	4 バイト・ストリング長標識の NULL 終了可変長バイナリー・ストリングでない
BLOB ロケーター変数[6] (960 または 961)	sql type is blob_locator	サーバー上の BLOB エンティティーを識別する
BLOB ファイル参照変数[6] (916 または 917)	sql type is blob_file	BLOB データを含むファイルの記述子
DATE (384 または 385)	NULL 終了文字形式	NULL 終止符を収容するために最低 11 文字を使用できる。
	VARCHAR 構造書式	最低 10 文字を使用できる。
TIME (388 または 389)	NULL 終了文字形式	NULL 終止符を収容するために最低 9 文字を使用できる。
	VARCHAR 構造書式	最低 8 文字を使用できる。
TIMESTAMP (392 または 393)	NULL 終了文字形式	NULL 終止符を収容するために最低 27 文字を使用できる。
	VARCHAR 構造書式	最低 26 文字が使用できる。

注: 以下のデータ・タイプは WCHARTYPE NOCONVERT オプションで再コンパイル時に DBCS または EUC 環境でのみ使用可能です。

表 13. C/C++ 宣言にマップされた SQL データ・タイプ (続き)

SQL 列タイプ[1]	C/C++ データ・タイプ	SQL 列タイプ記述
GRAPHIC(1) (468 または 469)	sqldbchar	単一の 2 バイト文字
GRAPHIC(*n*) (468 または 469)	厳密な対応なし; sqldbchar[*n+1*] を使用 (n はデータを保持するだけの十分な大きさ) 1<=*n*<=127	固定長 2 バイト文字ストリング

VARGRAPHIC(*n*)
(464 または 465)
	struct tag {
	short int;
	sqldbchar[*n*]
	}
	
	1<=*n*<=16 336
		2 バイトのストリング長指定子を持つ、NULL 終了可変 2 バイト文字以外のストリング

	代替使用; sqldbchar[*n+1*] を使用 (n はデータを保持するだけの十分な大きさ) 1<=*n*<=16 336	NULL 終了可変長 2 バイト文字ストリング **注:** 400/401 の SQL タイプを割り当てられる。

LONG VARGRAPHIC
(472 または 473)
	struct tag {
	short int;
	sqldbchar[*n*]
	}
	
	16 337<=*n*<=16 350
		2 バイトのストリング長指定子を持つ、NULL 終了可変 2 バイト文字以外のストリング

注: 以下のデータ・タイプは、WCHARTYPE CONVERT オプションを使用してプリコンパイルする場合の DBCS または EUC 環境でのみ使用できる。		
GRAPHIC(1) (468 または 469)	wchar_t	• 単一のワイド・キャラクター (C タイプの場合) • 単一の 2 バイト文字 (列タイプの場合)
GRAPHIC(*n*) (468 または 469)	厳密な対応なし; wchar[*n+1*] を使用 (n はデータを保持するだけの十分な大きさ) 1<=*n*<=127	固定長 2 バイト文字ストリング

VARGRAPHIC(*n*)
(464 または 465)
	struct tag {
	short int;
	wchar_t [*n*]
	}
	
	1<=*n*<=16 336
		2 バイトのストリング長指定子を持つ、NULL 終了可変 2 バイト文字以外のストリング

	代替使用; char[*n+1*] を使用 (n はデータを保持するだけの十分な大きさ) 1<=*n*<=16 336	NULL 終了可変長 2 バイト文字ストリング **注:** 400/401 の SQL タイプを割り当てられる。

表 13. C/C++ 宣言にマップされた SQL データ・タイプ (続き)

SQL 列タイプ[1]	C/C++ データ・タイプ	SQL 列タイプ記述

LONG VARGRAPHIC
(472 または 473)
	struct tag {
	short int;
	wchar_t [*n*]
	}
	
	16 337<=*n*<=16 350
		2 バイトのストリング長指定子を持つ、NULL 終了可変 2 バイト文字以外のストリング

注: 以下のデータ・タイプは DBCS または EUC 環境でのみ使用できる。		
DBCLOB(*n*) (412 または 413)	sql type is dbclob(*n*) 1<=*n*<=1 073 741 823	4 バイトのストリング長指定子を持つ、NULL 終了可変 2 バイト文字以外のストリング
DBCLOB ロケーター変数[6] (968 または 969)	sql type is dbclob_locator	サーバー上の DBCLOB エンティティーを識別する
DBCLOB ファイル参照変数[6] (924 または 925)	sql type is dbclob_file	DBCLOB データを含むファイルの記述子

注:

1. **SQL 列タイプ**の下の最初の数字は標識変数が提供されないことを示し、 2 番目の数字は標識変数が提供されることを示します。 標識変数は、NULL 値を示したり、切り捨てられたストリングの長さを保持するのに必要です。これらの値は、それぞれのデータ・タイプの SQLDA の SQLTYPE フィールドに現れます。
2. プラットフォームと互換性を持たせるには、sqlint32 を使用してください。 64 ビットの UNIX プラットフォームでは、 ″long″ は 64 ビット整数です。 64 ビットの Windows オペレーティング・システムおよび 32 ビットの UNIX プラットフォームでは、 ″long″ は 32 ビットの整数です。
3. プラットフォームで互換性を持たせるには、sqlint64 を使用してください。 DB2 Universal Database の sqlsystm.h ヘッダー・ファイルは、 Microsoft コンパイラーを使用する場合には、 Windows NT プラットフォームで sqlint64 を ″__int64″ とタイプ定義します。また、32 ビットの UNIX プラットフォームでは ″long long″ と、64 ビットの UNIX プラットフォームでは ″long″ とタイプ定義します。
4. FLOAT(*n*)。ここで 0 < *n* < 25 の場合、REAL と同義。 SQLDA での REAL と DOUBLE の違いは長さの値です (4 または 8)。
5. 以下の SQL タイプは、DOUBLE と同義です。
   - FLOAT
   - FLOAT(*n*)。ここで、*n* の取る範囲は 24 < *n* < 54。
   - DOUBLE PRECISION
6. これは列タイプではなく、ホスト変数タイプである。

以下に、サポートされている C/C++ データ・タイプに関するその他の規則を示します。

- データ・タイプ `char` は `char` または `unsigned char` と宣言することができる。
- データベース・マネージャーは、 NULL で終了する可変長文字ストリング・データ・タイプ char[*n*] (データ・タイプ 460) を、 VARCHAR(*m*) として処理する。
  - LANGLEVEL が SAA1 の場合、ホスト変数の長さ *m* は、 char[*n*] 内の文字ストリングの長さ *n* または最初の NULL 終止符 (¥0) の前のバイト数のいずれか小さい方と等しくなる。

  - LANGLEVEL が MIA の場合には、ホスト変数の長さ *m* は最初の NULL 終止符 (¥0) の前のバイト数と等しくなる。
- データベース・マネージャーは、 NULL で終了する可変長 GRAPHIC ストリング・データ・タイプ `wchar_t[n]` または `sqldbchar[n]` (データ・タイプ 400) を、 VARGRAPHIC(*m*) として処理する。
  - LANGLEVEL が SAA1 の場合、ホスト変数の長さ *m* は、`wchar_t[n]` または `sqldbchar[n]` 内の文字ストリングの長さ *n*、または最初のグラフィック NULL 終止符の前の文字数のいずれか小さい方と等しくなる。
  - LANGLEVEL が MIA の場合には、ホスト変数の長さ *m* は最初のグラフィック NULL 終止符の前の文字数と等しくなる。
- 無符号数値データ・タイプはサポートされていない。
- C/C++ データ・タイプの `int` は、その内部表現がマシン依存型であるため使用できない。

関連概念:

- 184 ページの『C および C++ 用のホスト変数がある SQL 宣言セクション』

C および C++ における FOR BIT DATA

標準的な C または C++ のストリング・タイプである 460 は、FOR BIT DATA に指定された列に使用しないでください。データベース・マネージャーは、NULL 文字が検出されると、このデータ・タイプを切り捨てます。 VARCHAR (SQL タイプ 448) または CLOB (SQL タイプ 408) のどちらかを使用してください。

関連概念:

- 184 ページの『C および C++ 用のホスト変数がある SQL 宣言セクション』

関連資料:

- 186 ページの『C および C++ においてサポートされている SQL データ・タイプ』

プロシージャー、関数、およびメソッド用の C/C++ データ・タイプ

次の表は、プロシージャー、UDF、およびメソッドの SQL データ・タイプおよび C データ・タイプ間のサポートされるマッピングをリストしています。

表 14. C/C++ 宣言にマップされた SQL データ・タイプ

SQL 列名	C/C++ データ・タイプ	SQL 列タイプ記述
SMALLINT (500 または 501)	short	16 ビットの符号付き整数
INTEGER (496 または 497)	sqlint32	32 ビットの符号付き整数
BIGINT (492 または 493)	sqlint64	64 ビットの符号付き整数
REAL (480 または 481)	float	単精度浮動小数点
DOUBLE (480 または 481)	double	倍精度浮動小数点

表 14. C/C++ 宣言にマップされた SQL データ・タイプ (続き)

SQL 列名	C/C++ データ・タイプ	SQL 列タイプ記述
DECIMAL(*p*,*s*) (484 または 485)	サポートされていません。	10 進数を渡すには、パラメーターを DECIMAL からキャスト可能なデータ・タイプ (たとえば CHAR または DOUBLE) に定義し、明示的に引き数をこのタイプにキャストします。
CHAR(*n*) (452 または 453)	char[*n+1*] (n はデータを保持するだけの十分な大きさ) 1<=*n*<=254	固定長、NULL 終了文字ストリング
CHAR(*n*) FOR BIT DATA (452 または 453)	char[*n+1*] (n はデータを保持するだけの十分な大きさ) 1<=*n*<=254	固定長文字ストリング
VARCHAR(*n*) (448 または 449) (460 または 461)	char[*n+1*] (n はデータを保持するだけの十分な大きさ) 1<=*n*<=32 672	NULL 終了可変長ストリング

VARCHAR(*n*) FOR BIT DATA
(448 または 449)
	struct {
	sqluint16 length;
	char[*n*]
	}
	
	1<=*n*<=32 672
		NULL 終了可変長文字ストリングでない

LONG VARCHAR
(456 または 457)
	struct {
	sqluint16 length;
	char[*n*]
	}
	
	32 673<=*n*<=32 700
		NULL 終了可変長文字ストリングでない

CLOB(*n*)
(408 または 409)
	struct {
	sqluint32 length;
	char data[n];
	}
	
	1<=*n*<=2 147 483 647
		4 バイト・ストリング長標識の NULL 終了可変長文字ストリングでない

BLOB(*n*)
(404 または 405)
	struct {
	sqluint32 length;
	char data[n];
	}
	
	1<=*n*<=2 147 483 647
		4 バイト・ストリング長標識の NULL 終了可変長バイナリー・ストリングでない

DATE (384 または 385)	char[11]	NULL 終了文字形式
TIME (388 または 389)	char[9]	NULL 終了文字形式
TIMESTAMP (392 または 393)	char[27]	NULL 終了文字形式
注: 以下のデータ・タイプは WCHARTYPE NOCONVERT オプションで再コンパイル時に DBCS または EUC 環境でのみ使用可能です。		
GRAPHIC(*n*) (468 または 469)	sqldbchar[*n+1*] (n はデータを保持するだけの十分な大きさ) 1<=*n*<=127	固定長、NULL 終了 2 バイト文字ストリング

表 14. C/C++ 宣言にマップされた SQL データ・タイプ (続き)

SQL 列名	C/C++ データ・タイプ	SQL 列タイプ記述
VARGRAPHIC(*n*) (400 または 401)	sqldbchar[*n+1*] (n はデータを保持するだけの十分な大きさ) 1<=*n*<=16 336	NULL 終了、可変長 2 バイト文字ストリングでない

LONG VARGRAPHIC
(472 または 473)
	struct {
	sqluint16 length;
	sqldbchar[*n*]
	}
	
	16 337<=*n*<=16 350
		NULL 終了、可変長 2 バイト文字ストリングでない

DBCLOB(*n*)
(412 または 413)
	struct {
	sqluint32 length;
	sqldbchar data[n];
	}
	
	1<=*n*<=1 073 741 823
		4 バイト・ストリング長標識の NULL 終了可変長文字ストリングでない

C および C++ における SQLSTATE および SQLCODE 変数

LANGLEVEL プリコンパイル・オプションを SQL92E の値とともに使用すると、次の 2 つの宣言をホスト変数として組み込めます。

```
EXEC SQL BEGIN DECLARE SECTION;
  char      SQLSTATE[6]
  sqlint32  SQLCODE;


⋮
  EXEC SQL END DECLARE SECTION;
```

これらのいずれも指定しない場合は、SQLCODE 宣言はプリコンパイル中であると見なされます。このオプションを使用するときには、INCLUDE SQLCA ステートメントを指定してはならないことに注意してください。

複数のソース・ファイルから成るアプリケーションでは、上の例のように、最初のソース・ファイルで SQLCODE および SQLSTATE 変数を定義することができます。その後のソース・ファイルは、次のようにその定義を修正する必要があります。

```
extern sqlint32 SQLCODE;
extern char     SQLSTATE[6];
```

関連概念:

- 114 ページの『戻りコード』
- 114 ページの『SQLCODE、SQLSTATE、および SQLWARN フィールドのエラー情報』

第 7 章 C および C++ アプリケーション用のマルチスレッド・データベース・アクセス

マルチスレッド・データベース・アクセスの目的 193
マルチスレッドの使用に際しての推奨事項. . . . 194
マルチスレッド UNIX アプリケーション用のコード・ページおよび国別/地域別コードに関する考慮事項 195
マルチスレッド・アプリケーションのトラブルシューティング 196
複数のスレッドの使用に関する潜在的な問題 . . 196
複数コンテキストにおけるデッドロックの回避方法 196

マルチスレッド・データベース・アクセスの目的

いくつかのオペレーティング・システムに共通する特徴は、1 つのプロセスで実行プログラムの複数のスレッドを実行できることです。複数のスレッドにより、アプリケーションが非同期のイベントを処理することができ、ポーリング機能がなくても容易にイベント・ドリブン・アプリケーションを作成できます。以下の情報では、データベース・マネージャーが複数のスレッドを処理する方法を解説し、留意すべき設計の指針を示します。

マルチスレッドのアプリケーション開発に関する用語 (クリティカル・セクションおよびセマフォーなど) について詳しくない方は、ご使用のオペレーティング・システムのプログラミングに関する資料を調べてください。

DB2 アプリケーションは、コンテキスト を使用して複数のスレッドから SQL ステートメントを実行することができます。コンテキストとは、アプリケーションがすべての SQL ステートメントおよび API 呼び出しを実行する環境のことです。すべての接続、作業単位、および他のデータベース・リソースは、特定のコンテキストに関連付けられています。各コンテキストは、アプリケーション内の 1 つ以上のスレッドに関連付けられています。

各実行可能 SQL ステートメントでは、最初のランタイム・サービス呼び出しは常にラッチを取得しようとします。成功すると処理を続行しますが、 (他のスレッドの SQL ステートメントがすでにラッチを取得しているために) 失敗すると、呼び出しは信号セマフォーでこれがポストされるまでブロックされ、それからラッチを取得し処理を続行します。ラッチは SQL ステートメントが処理を終了するまで保持され、その SQL ステートメントに対して生成された最後のランタイム・サービス呼び出しにより解放されます。

最終的な結果として、他のスレッドが SQL ステートメントを同時に実行しようとしても各 SQL ステートメントはアトミック単位で実行されます。これにより内部データ構造は、異なるスレッドによって同時に変更されることがなくなります。API もランタイム・サービスを使用したラッチを使用します。したがって、API には、各コンテキスト内のランタイム・サービス・ルーチンと同じ制限が課されます。

DB2® バージョン 8 では、すべてのバージョン 8 アプリケーションは、デフォルトでマルチスレッドになり、複数のコンテキストを使用できます。 (バージョン 8 以前のアプリケーションの動作は、変更ありません。) 必要な場合は、以下の DB2 API を使用して、複数コンテキストを使用できます。具体的に言えば、アプリケー

ションはスレッド用のコンテキストを作成でき、各スレッド用の別個のコンテキストにアタッチ、またはデタッチができ、スレッド間でコンテキストを渡すことができます。アプリケーションがこれらの API のいずれも 呼び出さない場合は、 DB2 が自動的にアプリケーション用に複数コンテキストを管理します。

- `sqleBeginCtx()`
- `sqleEndCtx()`
- `sqleAttachToCtx()`
- `sqleDetachFromCtx()`
- `sqleGetCurrentCtx()`
- `sqleInterruptCtx()`

コンテキストはプロセス内のスレッド間で交換できますが、プロセス間では交換できません。複数のコンテキストの使用方法の 1 つは、並行トランザクションのサポートです。

関連概念:

- 694 ページの『並行トランザクション』

関連資料:

- 「*管理 API リファレンス*」の『sqleAttachToCtx - コンテキストへのアタッチ』
- 「*管理 API リファレンス*」の『sqleBeginCtx - アプリケーション・コンテキストの作成およびアタッチ』
- 「*管理 API リファレンス*」の『sqleDetachFromCtx - コンテキストからの切り離し』
- 「*管理 API リファレンス*」の『sqleEndCtx - アプリケーション・コンテキストの切り離しおよび破棄』
- 「*管理 API リファレンス*」の『sqleGetCurrentCtx - 現行コンテキストの入手』
- 「*管理 API リファレンス*」の『sqleInterruptCtx - コンテキストへの割り込み』

関連サンプル:

- 『dbthrds.sqc -- How to use multiple context APIs on UNIX (C)』
- 『dbthrds.sqC -- How to use multiple context APIs on UNIX (C++)』

マルチスレッドの使用に際しての推奨事項

マルチスレッドのアプリケーションからデータベースをアクセスする際には、これらの指針に従ってください。

- 連続したデータ構造体の変更。

  アプリケーションは、SQL ステートメントまたはデータベース・マネージャー・ルーチンが、あるスレッドで処理されている間に、SQL ステートメントおよびデータベース・マネージャー・ルーチンが使用するユーザー定義のデータ構造体が、別のスレッドによって変更されていないことを確認する必要があります。 たとえば、他のスレッドの SQL ステートメントが SQLDA を使用している場合は、スレッドが SQLDA を再び割り振ることができないようにしてください。
- 個別のデータ構造体の使用を考える。

繰り返しを避けるため、各スレッドにそれぞれのユーザー定義のデータを渡す方が容易であるといえます。このガイドラインは特に、SQLCA の場合にそういえます。SQLCA は個々の実行可能 SQL ステートメントばかりでなく、すべてのデータベース・マネージャー・ルーチンによって使用されるからです。 SQLCA に関連したこの問題を避けるための代替手段が 3 つあります。

- 最初のスレッド以外のスレッドが使用するルーチンにはすべて、その先頭に `struct sqlca sqlca` を追加して、EXEC SQL INCLUDE SQLCA を使用する。
- EXEC SQL INCLUDE SQLCA を、グローバル有効範囲内に置くのではなく、SQLを含む各ルーチンの内部に置く。
- EXEC SQL INCLUDE SQLCA を `#include "sqlca.h"` に置き換え、SQL を使用するルーチンの先頭に `"struct sqlca sqlca"` を追加する。

注: デフォルト・スタック・サイズは使用せず、スタック・サイズを少なくとも 256 000 に増やすことをお勧めします。 DB2® は、DB2 関数を呼び出す場合に、最小スタック・サイズである 256 000 を必要とします。したがって、アプリケーションおよび DB2 関数呼び出しの最小要件の両方に対して十分な大きさを持つ合計スタック・サイズを割り振っていることを確認する必要があります。

マルチスレッド UNIX アプリケーション用のコード・ページおよび国別/地域別コードに関する考慮事項

AIX®、Solaris オペレーティング環境、HP-UX、および Silicon Graphics IRIX では、データベース接続で使用されるコード・ページおよび国別/地域別コードをランタイムに照会するために使用される関数に対して変更が加えられました。これらの関数は現在ではスレッド・セーフですが、多数の並行データベース接続を使用するマルチスレッド・アプリケーションでは、ロック競合 (およびその結果、パフォーマンスの低下) が起きる可能性があります。

DB2®_FORCE_NLS_CACHE 環境変数を使用して、マルチスレッド・アプリケーションでのロック競合を減らせます。 DB2_FORCE_NLS_CACHE が TRUE に設定されると、コード・ページおよび国別/地域別コード情報は、スレッドが最初にアクセスする際に保管されます。 その時点から、キャッシュされた情報は、この情報を要求する他のすべてのスレッドで使用されます。この情報を保管するとロック競合は削減され、状況によってはパフォーマンスが向上します。

接続間のロケール設定をアプリケーションが変更する場合には、DB2_FORCE_NLS_CACHE を TRUE に設定するべきではありません。この状態になると、ロケール設定が変更されても、元のロケール情報が戻されます。一般的には、マルチスレッド・アプリケーションはロケール設定を変更しないので、変更しないままにしておけばアプリケーションはスレッド・セーフになります。

関連概念:

- 「*管理ガイド: パフォーマンス*」の『DB2 レジストリー変数と環境変数』

マルチスレッド・アプリケーションのトラブルシューティング

以下のセクションでは、マルチスレッド・アプリケーションで発生する問題、およびその回避方法について説明します。

複数のスレッドの使用に関する潜在的な問題

複数のスレッドを使用するアプリケーションは、当然のことながら、単一スレッドを使用するアプリケーションよりも複雑です。この余分の複雑さにより、予期しない問題がいくつか生じる可能性が潜んでいます。マルチスレッドのアプリケーションを作成するときには、次の事柄に注意を払ってください。

- 2 つ以上のコンテキスト間でのデータベースの従属関係

  アプリケーション内の各コンテキストには、データベース・オブジェクトに対するロックなど、それぞれ固有のデータベース・リソースがあります。この特性のため、2 つのコンテキストが同じデータベース・オブジェクトにアクセスしている場合、デッドロックを引き起こす可能性があります。データベース・マネージャーがデッドロックを検出すると、一方のコンテキストが SQLCODE -911 を受け取り、その作業単位がロールバックされます。
- 2 つ以上のコンテキスト間におけるアプリケーションの従属関係

  コンテキスト間の従属関係を確立するプログラミング技法に注意してください。ラッチ、セマフォー、およびクリティカル・セクションは、そのような従属関係を確立するプログラミング技法の例です。アプリケーションに 2 つのコンテキストがあり、そのコンテキスト間にはアプリケーションの従属関係とデータベースの従属関係のどちらもが存在する場合、アプリケーションがデッドロックとなる可能性があります。従属関係のあるものがデータベース・マネージャーの管理範囲外にある場合、デッドロックが検出されないので、アプリケーションは中断またはハングします。

関連概念:

- 196 ページの『複数コンテキストにおけるデッドロックの回避方法』

複数コンテキストにおけるデッドロックの回避方法

データベース・マネージャーはスレッド間のデッドロックを検出できないため、デッドロックしないように (または少なくともデッドロックを回避できるように) アプリケーションを設計してコーディングしなければなりません。

データベース・マネージャーが検出しないデッドロックの例として、2 つのコンテキストがあり、そのどちらも共通のデータ構造体にアクセスするアプリケーションを考えてみましょう。両方のコンテキストがそのデータ構造体を同時に変更することを避けるため、データ構造体はセマフォーによって保護されます。コンテキストは次のようになります。

```
context 1
SELECT * FROM TAB1 FOR UPDATE....
UPDATE TAB1 SET....
get semaphore
access data structure
release semaphore
COMMIT
```

```
context 2
get semaphore
access data structure
SELECT * FROM TAB1...
release semaphore
COMMIT
```

最初のコンテキストが SELECT および UPDATE ステートメントを正常に実行しているときに、 2 番目のコンテキストがセマフォーを獲得してデータ構造体にアクセスするとします。最初のコンテキストがセマフォーを獲得しようとしますが、2 番目のコンテキストがセマフォーを保持しているため、獲得できません。ここで 2 番目のコンテキストは表 TAB1 から行を読み取ろうとしますが、最初のコンテキストが保持するデータベース・ロックによってその操作が停止してしまいます。アプリケーションは、コンテキスト 1 がコンテキスト 2 の前に完了できず、コンテキスト 2 がコンテキスト 1 の完了を待っている状態になります。アプリケーションはデッドロックしますが、データベース・マネージャーはセマフォーの従属関係を知らないため、コンテキストはロールバックされません。未解決の従属関係のため、アプリケーションは延期状態になってしまいます。

上記の例で起こるデッドロックを回避する方法はいくつかあります。

- セマフォーを獲得する前に保持していたロックをすべて解除する。

  コンテキスト 1 のコードを変更して、セマフォーを獲得する前にコミットを実行するようにします。
- セマフォーによって保護されたセクションの内部に、SQL ステートメントをコーディングしない。

  コンテキスト 2 のコードを変更して、SELECT を実行する前にセマフォーを解除するようにします。
- すべての SQL ステートメントをセマフォー内部にコーディングする。

  コンテキスト 1 のコードを変更して、SELECT ステートメントを実行する前にセマフォーを獲得するようにします。この技法は、動作はしますが、あまりお勧めできません。というのは、セマフォーはデータベース・マネージャーへのアクセスをシリアライズするため、複数のスレッドを使用する効果が発揮されないからです。
- *locktimeout* データベース構成パラメーターを -1 以外の値に設定する。

  -1 以外の値ではデッドロックを防ぐことはできませんが、実行を再開することはできます。コンテキスト 2 は、要求されたロックを獲得できないため、結局はロールバックされます。ロールバックのエラーを処理するときには、コンテキスト 2 はセマフォーを解除しなければなりません。セマフォーを解除すると、コンテキスト 1 が継続でき、コンテキスト 2 が解放されて作動を再試行します。

デッドロックを回避する技法を例を参考にして説明しましたが、この方法はすべてのマルチスレッド・アプリケーションに適用できます。一般に、保護リソースを扱うようにデータベース・マネージャーを扱うならば、マルチスレッド・アプリケーションで問題が生じることはありません。

関連概念:

- 196 ページの『複数のスレッドの使用に関する潜在的な問題』

第 8 章 COBOL でのプログラミング

COBOL に関するプログラミング考慮事項 199
COBOL での言語制限 199
COBOL におけるマルチスレッド・データベース・アクセス 199
COBOL の入力および出力ファイル 200
COBOL の組み込みファイル 200
COBOL における組み込み SQL 203
COBOL のホスト変数 205
COBOL のホスト変数 205
COBOL におけるホスト変数の名前 205
COBOL におけるホスト変数宣言 206
COBOL における数値ホスト変数の構文 . . . 206
COBOL における固定長ホスト変数の構文 . . . 207
COBOL における固定長グラフィック・ホスト変数の構文 209
COBOL の標識変数 209
COBOL における LOB ホスト変数の構文 . . . 210
COBOL における LOB ロケーター・ホスト変数の構文 211
COBOL におけるファイル参照ホスト変数の構文 211
COBOL でのホスト構造サポート 212
COBOL の標識表 214
COBOL グループ・データ項目での REDEFINES 214
COBOL 用のホスト変数がある SQL 宣言セクション 215
COBOL におけるデータ・タイプに関する考慮事項 215
COBOL でサポートされている SQL データ・タイプ 215
BINARY/COMP-4 COBOL データ・タイプ . . 218
COBOL での FOR BIT DATA. 218
COBOL での SQLSTATE および SQLCODE 変数 218
COBOL での日本語または中国語 (繁体字) EUC、および UCS-2 に関する考慮事項 219
オブジェクト指向 COBOL 219

COBOL に関するプログラミング考慮事項

特定のホスト言語によるプログラミングの考慮事項について、以降のセクションで説明します。言語制限、ホスト言語別の組み込みファイル、組み込み SQL ステートメント、ホスト変数、およびサポートされるホスト変数のデータ・タイプについての情報が含まれています。組み込み SQL ステートメント、言語制限、およびホスト変数にサポートされるデータ・タイプについては、 Micro Focus COBOL の資料を参照してください。

関連資料:

- 「*アプリケーション開発ガイド アプリケーションの構築および実行*」の『COBOL のサンプル』

COBOL での言語制限

API ポインターの長さは、すべて 4 バイトです。 API 呼び出しで値パラメーターとして使用する整数の変数はすべて、 USAGE COMP-5 文節で宣言しなければなりません。

COBOL におけるマルチスレッド・データベース・アクセス

COBOL はマルチスレッド・データベース・アクセスをサポートしていません。

COBOL の入力および出力ファイル

デフォルトでは、入力ファイルの拡張子は .sqb ですが、 TARGET プリコンパイル・オプション (TARGET ANSI_COBOL、TARGET IBMCOB、TARGET MFCOB、または TARGET MFCOB16) を使用した場合は、入力ファイルに任意の拡張子を付けることができます。

デフォルトでは、出力ファイルの拡張子は .cbl ですが、 OUTPUT プリコンパイル・オプションを使用すれば、出力修正後のソース・ファイルに新しい名前やパスを指定できます。

COBOL の組み込みファイル

COBOL のホスト言語固有の組み込みファイルには、ファイル拡張子 .cbl が付いています。IBM COBOL コンパイラーの「システム/390 ホスト・データ・タイプ・サポート」機能を使用する場合、ご使用のアプリケーションの DB2 組み込みファイルは以下のディレクトリーにあります。

```
$HOME/sqllib/include/cobol_i
```

DB2 サンプル・プログラムを、提供されたスクリプト・ファイルを使用して構築する場合、スクリプト・ファイルで指定されている組み込みファイルのパスを、cobol_i ディレクトリー (cobol_a ディレクトリーではない) に変更する必要があります。

IBM COBOL コンパイラーの「システム/390 ホスト・データ・タイプ・サポート」機能を使用**しない**場合、またはこのコンパイラーの以前のバージョンを使用する場合、ご使用のアプリケーションの DB2 組み込みファイルは以下のディレクトリーにあります。

```
$HOME/sqllib/include/cobol_a
```

アプリケーションで使用するためのこれらの組み込みファイルについて以下で説明します。

SQL (sql.cbl) このファイルには、バインド・プログラム、プリコンパイラー、およびエラー・メッセージ検索 API 用の言語固有プロトタイプが含まれています。また、システム定数も定義されています。

SQLAPREP (sqlaprep.cbl)
このファイルには、独自のプリコンパイラーの作成に必要な定義が入っています。

SQLCA (sqlca.cbl)
このファイルは SQL 連絡域 (SQLCA) 構造体を定義します。SQLCA には、データベース・マネージャーが SQL ステートメントおよび API 呼び出しの実行に関するエラー情報をアプリケーションに提供するために使用する変数が含まれています。

SQLCA_92 (sqlca_92.cbl)
このファイルには、SQL 連絡域 (SQLCA) 構造体の FIPS SQL92 Entry Level 準拠版が入っています。このファイルは、FIPS SQL92 Entry Level standard に準拠する DB2 アプリケーションを作成する

ときには、 sqlca.cbl の代わりに組み込む必要があります。 sqlca_92.cbl ファイルは、 LANGLEVEL プリコンパイラー・オプションを SQL92E に設定すると、 DB2 プリコンパイラーが自動的に組み込みます。

SQLCODES (sqlcodes.cbl)
このファイルは、SQLCA 構造体の SQLCODE フィールドで使用する定数を定義します。

SQLDA (sqlda.cbl)
このファイルは SQL 記述子域 (SQLDA) 構造体を定義します。 SQLDA はアプリケーションとデータベース・マネージャーとの間でデータをやりとりするために使用されます。

SQLEAU (sqleau.cbl)
このファイルには、DB2 セキュリティー監査 API に必要な定数および構造の定義が含まれています。これらの API を使用する場合は、プログラムにこのファイルを組み込む必要があります。このファイルにはまた、監査証跡レコード内のフィールドの定数およびキーワード値の定義も含まれています。これらの定義は、外部または取引先の監査証跡抽出プログラムで使用できます。

SQLENV (sqlenv.cbl)
このファイルは、データベース環境 API に対する言語固有の呼び出し、およびそれらのインターフェースの構造、定数、戻りコードを定義します。

SQLETSD (sqletsd.cbl)
このファイルは、表スペース記述子構造 SQLETSDESC を定義します。これはデータベース作成 API である sqlgcrea に渡されます。

SQLE819A (sqle819a.cbl)
データベースのコード・ページが 819 (ISO ラテン 1) の場合、このシーケンスは、ホスト CCSID 500 (EBCDIC 各国対応) バイナリー照合に従って、 FOR BIT DATA ではない文字ストリングをソートします。このファイルは CREATE DATABASE API が使用します。

SQLE819B (sqle819b.cbl)
データベースのコード・ページが 819 (ISO ラテン 1) の場合、このシーケンスは、ホスト CCSID 037 (EBCDIC 米国英語) バイナリー照合に従って、 FOR BIT DATA ではない文字ストリングをソートします。このファイルは CREATE DATABASE API が使用します。

SQLE850A (sqle850a.cbl)
データベースのコード・ページが 850 (ASCII ラテン 1) の場合、このシーケンスは、ホスト CCSID 500 (EBCDIC 各国対応) バイナリー照合に従って、 FOR BIT DATA ではない文字ストリングをソートします。このファイルは CREATE DATABASE API が使用します。

SQLE850B (sqle850b.cbl)
データベースのコード・ページが 850 (ASCII ラテン 1) の場合、

このシーケンスは、ホスト CCSID 037 (EBCDIC 米国英語) バイナリー照合に従って、 FOR BIT DATA ではない文字ストリングをソートします。このファイルは CREATE DATABASE API が使用します。

SQLE932A (sqle932a.cbl)
データベースのコード・ページが 932 (ASCII 日本語) の場合、このシーケンスは、ホスト CCSID 5035 (EBCDIC 日本語) バイナリー照合に従って、 FOR BIT DATA ではない文字ストリングをソートします。このファイルは CREATE DATABASE API が使用します。

SQLE932B (sqle932b.cbl)
データベースのコード・ページが 932 (ASCII 日本語) の場合、このシーケンスは、ホスト CCSID 5026 (EBCDIC 日本語) バイナリー照合に従って、 FOR BIT DATA ではない文字ストリングをソートします。このファイルは CREATE DATABASE API が使用します。

SQL1252A (sql1252a.cbl)
データベースのコード・ページが 1252 (Windows ラテン 1) の場合、このシーケンスは、ホスト CCSID 500 (EBCDIC 各国対応) バイナリー照合に従って、FOR BIT DATA ではない文字ストリングをソートします。このファイルは CREATE DATABASE API が使用します。

SQL1252B (sql1252b.cbl)
データベースのコード・ページが 1252 (Windows ラテン 1) の場合、このシーケンスは、ホスト CCSID 037 (EBCDIC 米国英語) バイナリー照合に従って、FOR BIT DATA ではない文字ストリングをソートします。このファイルは CREATE DATABASE API が使用します。

SQLMON (sqlmon.cbl)
このファイルは、データベース・システム・モニター API に対する言語固有の呼び出し、およびそれらのインターフェースの構造、定数、戻りコードを定義します。

SQLMONCT (sqlmonct.cbl)
このファイルには、データベース・システム・モニター API を呼び出すのに必要な定数定義とローカル・データ構造定義が含まれています。

SQLSTATE (sqlstate.cbl)
このファイルは、SQLCA 構造体の SQLSTATE フィールドで使用する定数を定義します。

SQLUTBCQ (sqlutbcq.cbl)
このファイルは、表スペース・コンテナー照会データ構造 SQLB-TBSCONTQRY-DATA を定義します。これは、表スペース・コンテナー照会 API である sqlgstsc、sqlgftcq、および sqlgtcq で使用されます。

SQLUTBSQ (sqlutbsq.cbl)
このファイルは、表スペース照会データ構造 SQLB-TBSQRY-DATA を定義します。これは、表スペース照会 API である sqlgstsq、sqlgftsq、および sqlgtsq で使用されます。

SQLUTIL (sqlutil.cbl)
このファイルは、ユーティリティー API に対する言語固有の呼び出し、およびそれらのインターフェースに必要な構造、定数、コードを定義します。

COBOL における組み込み SQL

組み込み SQL ステートメントは次の 3 つのエレメントからなります。

エレメント	正しい COBOL 構文
一組のキーワード	EXEC SQL
ステートメント・ストリング	任意の有効な SQL ステートメント
ステートメント終止符	END-EXEC.

たとえば、次のようになります。

```
EXEC SQL SELECT col INTO :hostvar FROM table END-EXEC.
```

組み込み SQL ステートメントには、以下の規則が適用されます。

- 実行可能な SQL ステートメントは、PROCEDURE DIVISION になければなりません。 SQL ステートメントの前には、COBOL ステートメントとして段落名を付けることができます。
- SQL ステートメントは、領域 A (列 8 ～ 11) または領域 B (列 12 ～ 72) のどちらからも開始することができます。
- 各 SQL ステートメントは、EXEC SQL で開始して END-EXEC で終了します。各 SQL ステートメントは、修正済みソース・ファイル内のコメントとして、SQL プリコンパイラーに含まれます。
- SQL ステートメント終止符を使用しなければならない。使用しない場合、プリコンパイラーはアプリケーション内の次の終止符まで処理を継続します。これにより、不確定のエラーが起こる恐れがあります。
- SQL コメントは、組み込み SQL ステートメントの一部となっている行であればどこでも使用することができる。このコメントは、動的に実行するステートメントでは使用できません。 SQL コメントの形式は、ダブル・ダッシュ (--) の後に 0 個以上の文字ストリングが続き、行末で終了します。 SQL コメントを SQL ステートメントの後に置かないようにしてください。これを行うと、コメントが COBOL 言語の一部のように見えるため、コンパイル・エラーの原因となるからです。
- COBOL のコメントは、組み込み SQL ステートメント内の*ほとんど* の場所で使用できます。例外は以下のとおりです。
  - コメントは EXEC と SQL との間では使用できない。
  - コメントは動的に実行されるステートメントでは使用できない。
- SQL ステートメントは、COBOL 言語と同じ行継続規則に従います。ただし、EXEC SQL キーワードを 2 行に分割しないでください。

- SQL ステートメントを含んだファイルの組み込みに COBOL COPY ステートメントは使用できません。 SQL ステートメントはモジュールがコンパイルされる前にプリコンパイルされます。プリコンパイラーは、COBOL COPY ステートメントを無視します。代わりに SQL INCLUDE ステートメントを使用して、これらのファイルを組み込んでください。

  INCLUDE ファイルを探索する際、DB2® COBOL プリコンパイラーはまず最初に現行ディレクトリーを検索し、次いで DB2INCLUDE 環境変数に指定されたディレクトリーを検索します。 次の例を考えてみてください。

  – `EXEC SQL INCLUDE payroll END-EXEC.`

    INCLUDE ステートメントに指定されたファイルが単引用符 (') で囲まれていない場合、上記のとおり、プリコンパイラーは最初 `payroll.sqb` を検索し、次に `payroll.cpy`、次に `payroll.cbl`、最後にそれがロックする各ディレクトリーを検索します。

  – `EXEC SQL INCLUDE 'pay/payroll.cbl' END-EXEC.`

    ファイル名が単引用符 (') で囲まれている場合、すでに説明したとおり、名前に拡張子は追加されません。

    単引用符 (') 内のファイル名に完全パスが含まれていない場合、DB2INCLUDE の中身によってそのファイルの検索が行われ、何らかのパスが INCLUDE ファイル名に指定されます。 たとえば、DB2 (AIX 版) では、DB2INCLUDE は ‘`/disk2:myfiles/cobol`’ に設定され、プリコンパイラーは ‘`./pay/payroll.cbl`’、 ‘`/disk2/pay/payroll.cbl`’、 ‘`./myfiles/cobol/pay/payroll.cbl`’ の順に探索します。 プリコンパイラー・メッセージには、実際にファイルが見つかったパスが表示されます。 Windows プラットフォームでは、上の例のスラッシュを円記号 (¥) に置き換えます。

  注: DB2INCLUDE の設定は、DB2 コマンド行プロセッサーによってキャッシュされます。 何らかの CLP コマンドを発行した後に DB2INCLUDE の設定を変更する場合は、 TERMINATE コマンドを入力してから、データベースに接続し直し、あとは通常どおりプリコンパイルしてください。
- ストリング定数を次の行に継続するには、継続行の列 7 に '-' を、また列 12 またはそれ以降にストリング区切り文字を入れなければなりません。
- SQL 算術演算子はブランクで区切らなければなりません。
- 全ラインの COBOL コメントは、SQL ステートメント内を含め、プログラムのどこで使用してもかまいません。
- SQL ステートメント内のホスト変数の参照時に宣言されたとおりのホスト変数を使用する。
- 行末文字およびタブ文字などの空白文字の置換は、次のように行われる。
  – 引用符の外 (ただし、SQL ステートメント内) では、行末文字または TAB 文字は単一スペースと置き換えられる。
  – 引用符内では、COBOL プログラムに適合した形でストリングが継続されていれば、行末文字は消去されます。 TAB は修正されません。

行末および TAB に使用される実際の文字は、プラットフォームごとに異なります。たとえば Windows ベースのプラットフォームでは、復帰/改行を行末に使用するのに対し、 UNIX® ベースのシステムは改行のみを使用します。

関連資料:

- 751 ページの『付録 A. サポートされる SQL ステートメント』

COBOL のホスト変数

以下のセクションでは、COBOL プログラムにおけるホスト変数の宣言と使用について説明します。

COBOL のホスト変数

ホスト変数は、SQL ステートメント内で参照される COBOL の言語変数です。これにより、アプリケーションは入力データをデータベース・マネージャーに渡し、またデータベース・マネージャーから出力データを受け取ることができます。アプリケーションがプリコンパイルされると、コンパイラーはホスト変数を他の COBOL 変数と同様に使用します。

関連概念:

- 205 ページの『COBOL におけるホスト変数の名前』
- 206 ページの『COBOL におけるホスト変数宣言』

関連資料:

- 206 ページの『COBOL における数値ホスト変数の構文』
- 207 ページの『COBOL における固定長ホスト変数の構文』
- 209 ページの『COBOL における固定長グラフィック・ホスト変数の構文』
- 210 ページの『COBOL における LOB ホスト変数の構文』
- 211 ページの『COBOL における LOB ロケーター・ホスト変数の構文』
- 211 ページの『COBOL におけるファイル参照ホスト変数の構文』

COBOL におけるホスト変数の名前

SQL プリコンパイラーは、宣言された名前によってホスト変数を識別します。以下の規則が適用されます。

- 変数名は 255 文字以内の長さで指定する。
- ホスト変数は、SQL、sql、DB2®、または db2 以外の接頭部で開始する。これらはシステムが使用する予約語です。
- これから説明する宣言構文を使用する FILLER 項目はグループ・ホスト変数宣言では許可されており、プリコンパイラーはその項目を無視します。ただし、SQL DECLARE セクション内で FILLER を複数回使用した場合、プリコンパイラーは失敗します。 VARCHAR、LONG VARCHAR、 VARGRAPHIC、または LONG VARGRAPHIC 宣言には、FILLER 項目を組み込むことができません。
- ハイフンは、ホスト変数名として使用できます。

SQL は、スペースで囲まれたハイフンを減算演算子として解釈します。ハイフンをホスト変数名として使用する場合は、スペースを入れないでください。

- REDEFINES 文節は、ホスト変数宣言では許可されています。
- レベル-88 宣言は、ホスト変数宣言セクションでは許可されていますが、無視されます。

関連概念:

- 206 ページの『COBOL におけるホスト変数宣言』

関連資料:

- 206 ページの『COBOL における数値ホスト変数の構文』
- 207 ページの『COBOL における固定長ホスト変数の構文』
- 209 ページの『COBOL における固定長グラフィック・ホスト変数の構文』
- 210 ページの『COBOL における LOB ホスト変数の構文』
- 211 ページの『COBOL における LOB ロケーター・ホスト変数の構文』
- 211 ページの『COBOL におけるファイル参照ホスト変数の構文』

COBOL におけるホスト変数宣言

ホスト変数宣言の識別には、SQL の宣言セクションを使用しなければなりません。このセクションにより、それ以降の SQL ステートメントで参照が可能なホスト変数をプリコンパイラーに知らせます。

COBOL プリコンパイラーは、有効な COBOL 宣言のサブセットのみを認識します。

関連タスク:

- 「*アプリケーション開発ガイド サーバー・アプリケーションのプログラミング*」の『構造化型ホスト変数の宣言』

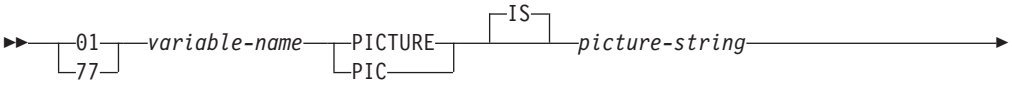
関連資料:

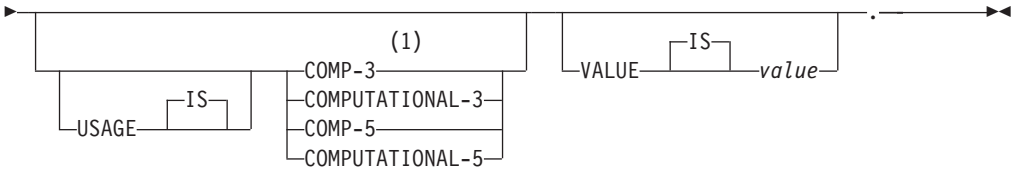
- 206 ページの『COBOL における数値ホスト変数の構文』
- 207 ページの『COBOL における固定長ホスト変数の構文』
- 209 ページの『COBOL における固定長グラフィック・ホスト変数の構文』
- 210 ページの『COBOL における LOB ホスト変数の構文』
- 211 ページの『COBOL における LOB ロケーター・ホスト変数の構文』
- 211 ページの『COBOL におけるファイル参照ホスト変数の構文』

COBOL における数値ホスト変数の構文

数値ホスト変数の構文を次に示します。

COBOL における数値ホスト変数の構文

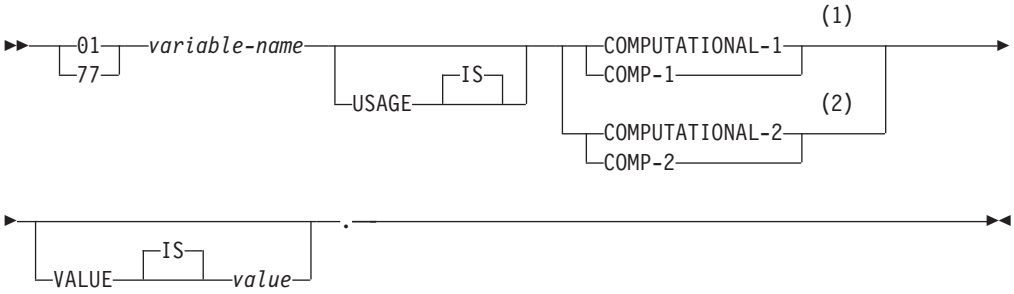




注:

1 COMP-3 の代わりに PACKED-DECIMAL を使用できます。

浮動小数点



注:

1 REAL (SQLTYPE 480)、長さ 4

2 DOUBLE (SQLTYPE 480)、長さ 8

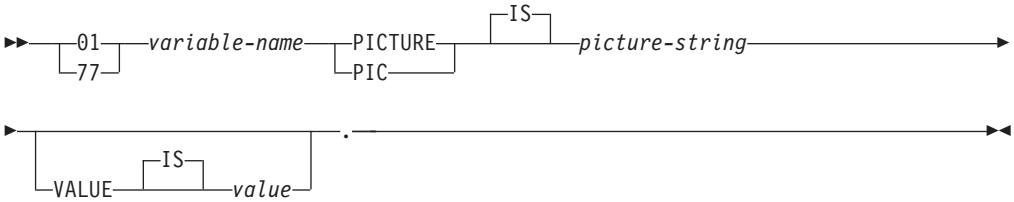
数値ホスト変数に関する考慮事項:

1. *Picture-string* は、以下のいずれかの形式でなければならない。
   - S9(m)V9(n)
   - S9(m)V
   - S9(m)
2. 数字の 9 は拡張することができる (たとえば、″S9(3)″ の代わりに ″S999″ とすることができます)。
3. *m* および *n* は、正の整数でなければならない。

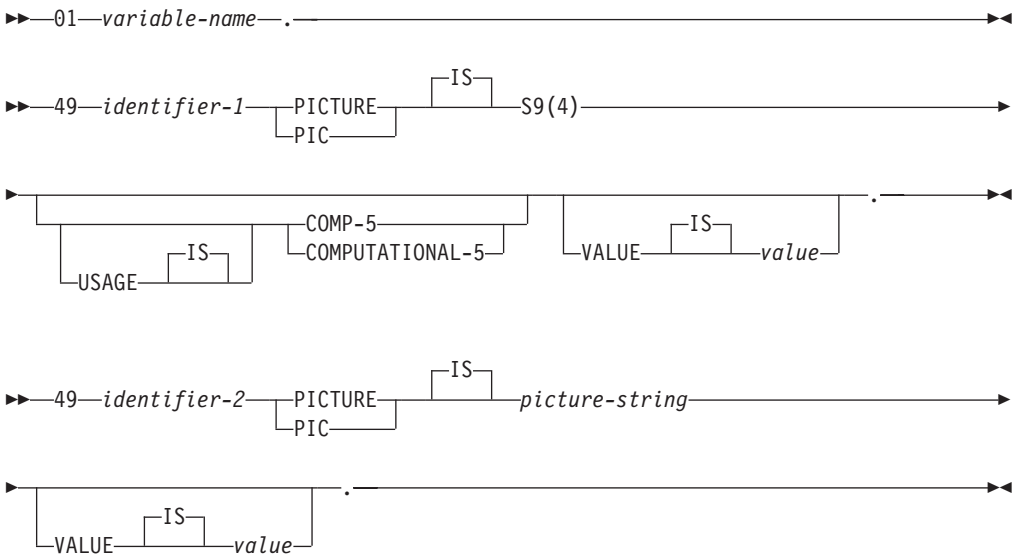
COBOL における固定長ホスト変数の構文

文字ホスト変数の構文を次に示します。

COBOL における文字ホスト変数の構文: 固定長



可変長



文字ホスト変数に関する考慮事項:

1. *Picture-string* の形式は、*X(m)* でなければならない。 X は拡張できます (たとえば、″X(3)″ の代わりに ″XXX″ にできます)。
2. *m* は、1 ～ 254 までの長さの固定長ストリングである。
3. *m* は、1 ～ 32 700 までの長さの可変長ストリングである。
4. *m* が 32 672 よりも大きい場合、そのホスト変数は LONG VARCHAR ストリングと見なされ、使用が制限される。
5. X および 9 を、PICTURE 文節内のピクチャー文字として使用する。他の文字は使用できません。
6. 可変長ストリングは、長さ項目と値項目とから構成される。適切な COBOL 名を、長さ項目およびストリング項目として使用できます。ただし、SQL ステートメント内の集合名を使用して可変長ストリングを参照してください。
7. 以下の例に示すような CONNECT ステートメントでは、 COBOL 文字ストリング・ホスト変数 `dbname` および `userid` の後続ブランクは、処理前に削除されます。

   ```
   EXEC SQL CONNECT TO :dbname USER :userid USING :p-word
   END-EXEC.
   ```

   ただし、パスワードではブランクが有効なため、 `p-word` ホスト変数を VARCHAR データ項目として宣言する必要があります。こうすることにより、アプリケーションは CONNECT ステートメントのパスワードの有効な長さを明示的に指示することができます。以下はその例です。

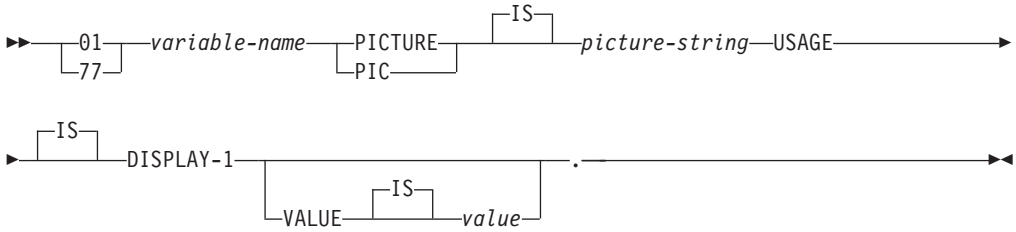
   ```
   EXEC SQL BEGIN DECLARE SECTION END-EXEC.
   01 dbname PIC X(8).
   01 userid PIC X(8).
   01 p-word.
       49 L PIC S9(4) COMP-5.
       49 D PIC X(18).
   EXEC SQL END DECLARE SECTION END-EXEC.
   PROCEDURE DIVISION.
       MOVE "sample" TO dbname.
       MOVE "userid" TO userid.
   ```

```
    MOVE "password" TO D OF p-word.
    MOVE 8          TO L of p-word.
EXEC SQL CONNECT TO :dbname USER :userid USING :p-word
END-EXEC.
```

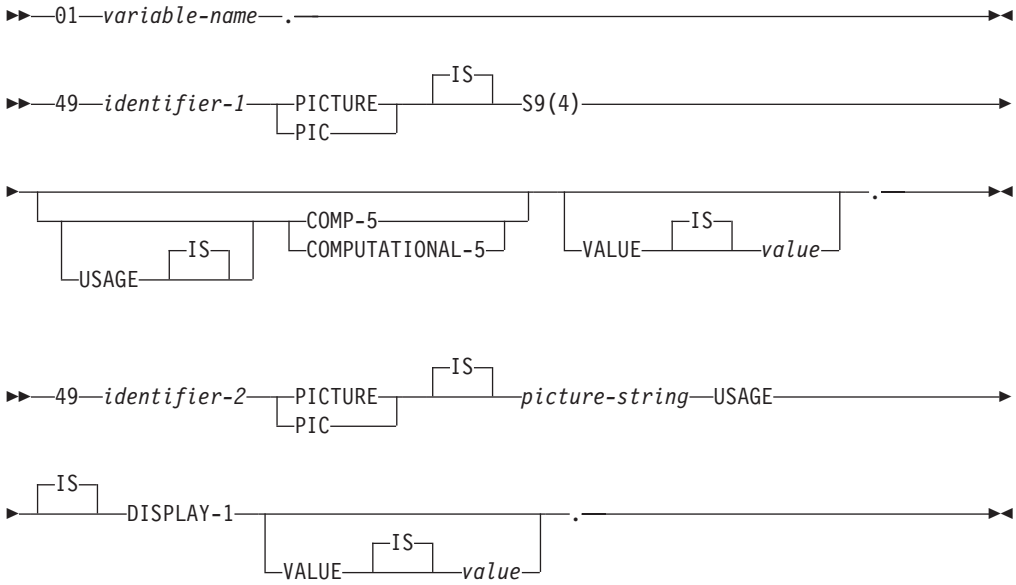
COBOL における固定長グラフィック・ホスト変数の構文

グラフィック・ホスト変数の構文を次に示します。

COBOL におけるグラフィック・ホスト変数の構文: 固定長



可変長



グラフィック・ホスト変数に関する考慮事項:

1. *Picture-string* の形式は、 *G(m)* でなければならない。 G は拡張できます (たとえば、″G(3)″ の代わりに ″GGG″ とできます)。
2. *m* は、1 ～ 127 までの長さの固定長ストリングである。
3. *m* は、1 ～ 16 350 までの長さの可変長ストリングである。
4. *m* が 16 336 よりも大きい場合、そのホスト変数は LONG VARGRAPHIC ストリングと見なされ、使用が制限される。

COBOL の標識変数

標識変数のデータ・タイプは、PIC S9(4) COMP-5 と宣言してください。

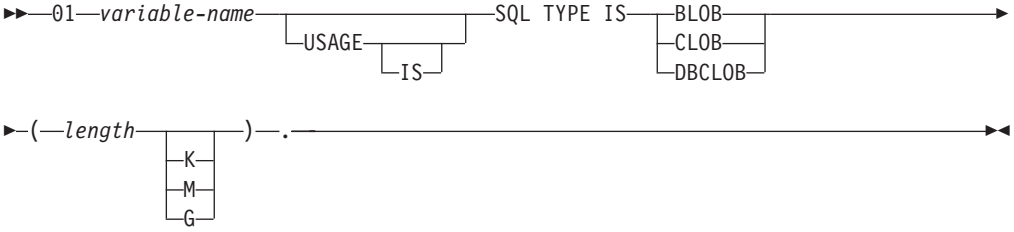
関連概念:

- 214 ページの『COBOL の標識表』

COBOL における LOB ホスト変数の構文

COBOL におけるラージ・オブジェクト (LOB) ホスト変数の宣言構文を次に示します。

COBOL における LOB ホスト変数の構文



LOB ホスト変数に関する考慮事項:

1. BLOB および CLOB の場合は、 1 <= lob-length <= 2 147 483 647 である。
2. DBCLOB の場合は、 1 <= lob-length <= 1 073 741 823 である。
3. SQL TYPE IS、BLOB、CLOB、DBCLOB、K、M、G は、大文字、小文字、またはその混合のいずれでもかまわない。
4. LOB 宣言内での初期化はできない。
5. ホスト変数名により、プリコンパイラー生成コード内の LENGTH および DATA に接頭語が付けられる。

BLOB の例:

宣言:

```
01 MY-BLOB USAGE IS SQL TYPE IS BLOB(2M).
```

この結果、以下の構造が生成されます。

```
01 MY-BLOB.
   49 MY-BLOB-LENGTH PIC S9(9) COMP-5.
   49 MY-BLOB-DATA PIC X(2097152).
```

CLOB の例:

宣言:

```
01 MY-CLOB USAGE IS SQL TYPE IS CLOB(125M).
```

この結果、以下の構造が生成されます。

```
01 MY-CLOB.
   49 MY-CLOB-LENGTH PIC S9(9) COMP-5.
   49 MY-CLOB-DATA PIC X(131072000).
```

DBCLOB の例:

宣言:

```
01 MY-DBCLOB USAGE IS SQL TYPE IS DBCLOB(30000).
```

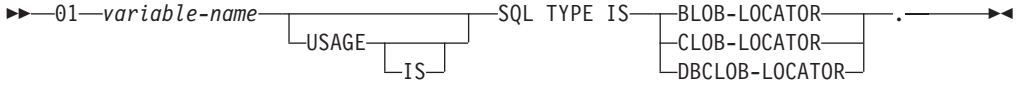
この結果、以下の構造が生成されます。

```
01 MY-DBCLOB.
   49 MY-DBCLOB-LENGTH PIC S9(9) COMP-5.
   49 MY-DBCLOB-DATA PIC G(30000) DISPLAY-1.
```

COBOL における LOB ロケーター・ホスト変数の構文

COBOL におけるラージ・オブジェクト (LOB) ロケーター・ホスト変数の宣言構文を次に示します。

COBOL における LOB ロケーター・ホスト変数の構文



LOB ロケーター・ホスト変数に関する考慮事項:

1. SQL TYPE IS、BLOB-LOCATOR、CLOB-LOCATOR、DBCLOB-LOCATOR は、大文字、小文字、またはその混合のいずれでもかまわない。
2. ロケーターの初期化はできない。

BLOB ロケーターの例 (他のタイプの LOB ロケーターの場合も同様):

宣言:

```
01 MY-LOCATOR USAGE SQL TYPE IS BLOB-LOCATOR.
```

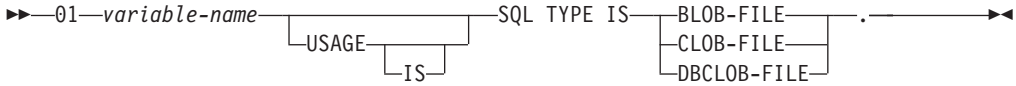
この結果、以下の宣言が生成されます。

```
01 MY-LOCATOR PIC S9(9) COMP-5.
```

COBOL におけるファイル参照ホスト変数の構文

COBOL におけるファイル参照ホスト変数の宣言構文を次に示します。

COBOL におけるファイル参照ホスト変数の構文



- SQL TYPE IS、BLOB-FILE、CLOB-FILE、DBCLOB-FILE は、大文字、小文字、またはその混合のいずれでもかまわない。

BLOB ファイル参照の例 (他のタイプの LOB の場合も同様):

宣言:

```
01 MY-FILE USAGE IS SQL TYPE IS BLOB-FILE.
```

この結果、以下の宣言が生成されます。

```
01 MY-FILE.
   49 MY-FILE-NAME-LENGTH PIC S9(9) COMP-5.
   49 MY-FILE-DATA-LENGTH PIC S9(9) COMP-5.
   49 MY-FILE-FILE-OPTIONS PIC S9(9) COMP-5.
   49 MY-FILE-NAME PIC X(255).
```

COBOL でのホスト構造サポート

COBOL プリコンパイラーは、ホスト変数宣言セクション内のグループ・データ項目をサポートします。グループ・データ項目は特に、 SQL ステートメント内の基本データ項目の集合を手早く参照する方法を提供します。たとえば、以下のグループ・データ項目は、 SAMPLE データベースの STAFF 表にある列にアクセスするために使用することができます。

```
01 staff-record.
    05 staff-id       pic s9(4) comp-5.
    05 staff-name.
        49 l          pic s9(4) comp-5.
        49 d          pic x(9).
    05 staff-info.
        10 staff-dept pic s9(4) comp-5.
        10 staff-job  pic x(5).
```

宣言セクション内のグループ・データ項目は、上記のホスト変数のタイプを従属データ項目として含むことができます。これには、すべてのタイプのラージ・オブジェクトの他に、数値および文字のすべてのタイプが含まれます。グループ・データ項目は、最高 10 レベルまでネストさせることができます。 上の例のように、レベル 49 の従属項目には VARCHAR 文字タイプを宣言しなければならないことに注意してください。レベルが 49 でない場合は、VARCHAR は 2 つの従属項目を持つグループ・データ項目と見なされ、グループ・データ項目の宣言および使用の規則に従います。上記の例では、staff-info はグループ・データ項目であり、staff-name は VARCHAR です。同じ原則は、LONG VARCHAR、VARGRAPHIC および LONG VARGRAPHIC にも当てはまります。グループ・データ項目は、02 ～ 49 の範囲のどのレベルにおいても宣言することができます。

グループ・データ項目とその従属項目には、以下の 4 とおりの使用法があります。

使用法 1

グループ全体を SQL ステートメント内の単一のホスト変数として参照します。

```
EXEC SQL SELECT id, name, dept, job
  INTO :staff-record
  FROM staff WHERE id = 10 END-EXEC.
```

プリコンパイラーは staff-record の参照を、 staff-record 内で宣言されたすべての従属項目をコンマで区切ったリストへと変換します。 名前の重複を避けるために、各基本項目はグループ名により修飾されます。これは、次の方式と同じです。

使用法 2

グループ・データ項目の 2 番目の使用法です。

```
EXEC SQL SELECT id, name, dept, job
  INTO
  :staff-record.staff-id,
  :staff-record.staff-name,
  :staff-record.staff-info.staff-dept,
  :staff-record.staff-info.staff-job
  FROM staff WHERE id = 10 END-EXEC.
```

注: staff-id への参照は、接頭部 staff-record. を使ったグループ名で修飾されており、純粋な COBOL の場合のように staff-record の staff-id によってではありません。

staff-record の従属項目と同じ名前を持つホスト変数がその他にない場合は、上記のステートメントは使用法 3 と同様に、明示的なグループ修飾を取り除いてコーディングできます。

使用法 3

ここでは、特定のグループ項目を修飾しない、通常の COBOL の方式で従属項目が参照されています。

```
EXEC SQL SELECT id, name, dept, job
  INTO
  :staff-id,
  :staff-name,
  :staff-dept,
  :staff-job
  FROM staff WHERE id = 10 END-EXEC.
```

純粋な COBOL と同様に、特定の従属項目が固有に識別できれば、プリコンパイラーに受け入れられます。たとえば、staff-job が複数のグループに現れるとすると、プリコンパイラーはあいまいな参照であることを示すエラーを出します。

```
SQL0088N Host variable "staff-job" is ambiguous.
```

使用法 4

あいまい参照を解決するために、従属項目の部分修飾を使用することができます。たとえば、以下のようにします。

```
EXEC SQL SELECT id, name, dept, job
  INTO
  :staff-id,
  :staff-name,
  :staff-info.staff-dept,
  :staff-info.staff-job
  FROM staff WHERE id = 10 END-EXEC.
```

使用法 1 のような単一のグループ項目のみの参照は、コンマで区切った従属項目のリストに対応するため、このタイプの参照はエラーとなる場合があります。たとえば、次のようにします。

```
EXEC SQL CONNECT TO :staff-record END-EXEC.
```

ここでの CONNECT ステートメントは、1 バイト文字ベースの変数を予期しています。 staff-record グループ・データ項目を与えると、このホスト変数は以下のようなプリコンパイル・エラーとなります。

```
SQL0087N Host variable "staff-record" is a structure used where
         structure references are not permitted.
```

この他に SQL0087N を引き起こすグループ項目の使用法には、 PREPARE、EXECUTE IMMEDIATE、 CALL、標識変数、および SQLDA 参照を含むものがあります。 このような状態では、前述の使用法 2、3 および 4 での個々の従属項目の参照がそうであるように、従属項目を 1 つしか持たないグループを使用することができます。

COBOL の標識表

COBOL プリコンパイラーは、グループ・データ項目での使用に便利な標識変数の表の定義をサポートします。以下のように宣言します。

```
01 <indicator-table-name>.
    05 <indicator-name> pic s9(4) comp-5
                        occurs <table-size> times.
```

たとえば、次のようになります。

```
01 staff-indicator-table.
    05 staff-indicator pic s9(4) comp-5
                       occurs 7 times.
```

この標識表は、上記のグループ項目の最初の形式で効率的に使用できます。

```
EXEC SQL SELECT id, name, dept, job
  INTO :staff-record :staff-indicator
  FROM staff WHERE id = 10 END-EXEC.
```

ここでは、プリコンパイラーが staff-indicator は標識表として宣言されていることを検出し、 SQL ステートメントの処理の際に、これを個々の標識の参照に拡張します。 staff-indicator(1) は staff-record の staff-id、 staff-indicator(2) は staff-record の staff-name、というように関連付けられます。

注: データ項目内の従属項目よりも k 個多い標識項目が標識表に存在する場合 (たとえば、staff-indicator に項目が 10 個ある場合は、 k=6 となります)、標識表の終端の k 個の余分な項目は無視されます。 同様に、標識項目が従属項目よりも k 個少ない場合は、グループ項目内の最後の k 個の項目は、対応する標識を持ちません。 *SQL ステートメント内の標識表にある個々のエレメントを参照できることに注意してください。*

関連概念:

- 209 ページの『COBOL の標識変数』

COBOL グループ・データ項目での REDEFINES

ホスト変数の宣言に、REDEFINES 文節を使用することができます。 REDEFINES 文節を使っていくつかのグループ・データ項目を宣言し、そのグループ・データ項目が SQL ステートメント内で全体として参照される場合、 REDEFINES 文節を含む従属項目は展開されません。 たとえば、次のようにします。

```
01 foo.
 10 a pic s9(4) comp-5.
 10 a1 redefines a pic x(2).
 10 b pic x(10).
```

SQL ステートメント内での foo の参照は、次のようになります。

```
... INTO :foo ...
```

上のステートメントは、次のステートメントと等価です。

```
... INTO :foo.a, :foo.b ...
```

つまり、従属項目 a1 は、REDEFINES 文節で宣言されていますが、そのような状況では自動的に展開されません。 a1 があいまいでない場合は、次のように、SQL ステートメント内の REDEFINES 文節で明示的に従属項目を参照することができます。

```
... INTO :foo.a1 ...
```

または

```
... INTO :a1 ...
```

COBOL 用のホスト変数がある SQL 宣言セクション

サポートされている SQL データ・タイプそれぞれについて宣言されたホスト変数を含んだ、 SQL 宣言のサンプルを次に示します。

```
EXEC SQL BEGIN DECLARE SECTION END-EXEC.
*
  01 age        PIC S9(4) COMP-5.
  01 divis      PIC S9(9) COMP-5.
  01 salary     PIC S9(6)V9(3) COMP-3.
  01 bonus      USAGE IS COMP-1.
  01 wage       USAGE IS COMP-2.
  01 nm         PIC X(5).
  01 varchar.
     49 leng    PIC S9(4) COMP-5.
     49 strg    PIC X(14).
  01 longvchar.
     49 len     PIC S9(4) COMP-5.
     49 str     PIC X(6027).
  01 MY-CLOB USAGE IS SQL TYPE IS CLOB(1M).
  01 MY-CLOB-LOCATOR USAGE IS SQL TYPE IS CLOB-LOCATOR.
  01 MY-CLOB-FILE USAGE IS SQL TYPE IS CLOB-FILE.
  01 MY-BLOB USAGE IS SQL TYPE IS BLOB(1M).
  01 MY-BLOB-LOCATOR USAGE IS SQL TYPE IS BLOB-LOCATOR.
  01 MY-BLOB-FILE USAGE IS SQL TYPE IS BLOB-FILE.
  01 MY-DBCLOB USAGE IS SQL TYPE IS DBCLOB(1M).
  01 MY-DBCLOB-LOCATOR USAGE IS SQL TYPE IS DBCLOB-LOCATOR.
  01 MY-DBCLOB-FILE USAGE IS SQL TYPE IS DBCLOB-FILE.
  01 MY-PICTURE PIC G(16000) USAGE IS DISPLAY-1.
  01 dt         PIC X(10).
  01 tm         PIC X(8).
  01 tmstmp     PIC X(26).
  01 wage-ind   PIC S9(4) COMP-5.
*
EXEC SQL END DECLARE SECTION END-EXEC.
```

関連資料:

- 215 ページの『COBOL でサポートされている SQL データ・タイプ』

COBOL におけるデータ・タイプに関する考慮事項

以下のセクションでは、SQL データ・タイプがどのように COBOL データにマップされるか説明します。

COBOL でサポートされている SQL データ・タイプ

一部の事前定義 COBOL データ・タイプは、列タイプと一致します。ホスト変数として宣言できるのは、その種の COBOL データ・タイプだけです。

以下の表に、各列タイプに対応する COBOL データ・タイプを示します。プリコンパイラーはホスト変数宣言を検出すると、該当する SQL タイプの値を判別します。データベース・マネージャーはこの値を使用して、アプリケーションとの間でやりとりするデータを変換します。

ホスト変数のすべての有効なデータ記述が認識されるわけではありません。COBOL データ項目は、以下の表に示す項目と一致していなければなりません。別のデータ項目を使用すると、エラーになる場合があります。

注: DB2 ホスト言語で、DATALINK データ・タイプをサポートするホスト変数はありません。

表 15. COBOL 宣言にマップされる SQL データ・タイプ

SQL 列タイプ[1]	COBOL データ・タイプ	SQL 列タイプ記述
SMALLINT (500 または 501)	01 name PIC S9(4) COMP-5	16 ビットの符号付き整数
INTEGER (496 または 497)	01 name PIC S9(9) COMP-5	32 ビットの符号付き整数
BIGINT (492 または 493)	01 name PIC S9(18) COMP-5	64 ビットの符号付き整数
DECIMAL(*p*,*s*) (484 または 485)	01 name PIC S9(*m*)V9(*n*) COMP-3	パック 10 進数
REAL[2] (480 または 481)	01 name USAGE IS COMP-1	単精度浮動小数点
DOUBLE[3] (480 または 481)	01 name USAGE IS COMP-2	倍精度浮動小数点
CHAR(*n*) (452 または 453)	01 name PIC X(*n*)	固定長文字ストリング
VARCHAR(*n*) (448 または 449)	01 name 49 length PIC S9(4) COMP-5 49 name PIC X(*n*) 1<=*n*<=32 672	可変長文字ストリング
LONG VARCHAR (456 または 457)	01 name 49 length PIC S9(4) COMP-5 49 data PIC X(*n*) 32 673<=*n*<=32 700	long 可変長文字ストリング
CLOB(*n*) (408 または 409)	01 MY-CLOB USAGE IS SQL TYPE IS CLOB(n) 1<=*n*<=2 147 483 647	ラージ・オブジェクト可変長文字ストリング
CLOB ロケーター変数[4] (964 または 965)	01 MY-CLOB-LOCATOR USAGE IS SQL TYPE IS CLOB-LOCATOR	サーバー上の CLOB エンティティーを識別する
CLOB ファイル参照変数[4] (920 または 921)	01 MY-CLOB-FILE USAGE IS SQL TYPE IS CLOB-FILE	CLOB データを含むファイルの記述子
BLOB(*n*) (404 または 405)	01 MY-BLOB USAGE IS SQL TYPE IS BLOB(n) 1<=*n*<=2 147 483 647	ラージ・オブジェクト可変長バイナリー・ストリング
BLOB ロケーター変数[4] (960 または 961)	01 MY-BLOB-LOCATOR USAGE IS SQL TYPE IS BLOB-LOCATOR	サーバーにある BLOB エンティティーを識別する
BLOB ファイル参照変数[4] (916 または 917)	01 MY-CLOB-FILE USAGE IS SQL TYPE IS CLOB-FILE	CLOB データを含むファイルの記述子

表 15. COBOL 宣言にマップされる SQL データ・タイプ (続き)

SQL 列タイプ[1]	COBOL データ・タイプ	SQL 列タイプ記述
DATE (384 または 385)	01 identifier PIC X(10)	10 バイトの文字ストリング
TIME (388 または 389)	01 identifier PIC X(8)	8 バイトの文字ストリング
TIMESTAMP (392 または 393)	01 identifier PIC X(26)	26 バイトの文字ストリング
注: 以下のデータ・タイプは、DBCS 環境のみで使用できる。		
GRAPHIC(*n*) (468 または 469)	01 name PIC G(*n*) DISPLAY-1	固定長 2 バイト文字ストリング
VARGRAPHIC(*n*) (464 または 465)	01 name 49 length PIC S9(4) COMP-5 49 name PIC G(*n*) DISPLAY-1 1<=*n*<=16 336	2 バイトのストリング長標識を持つ、可変長 2 バイト文字ストリング
LONG VARGRAPHIC (472 または 473)	01 name 49 length PIC S9(4) COMP-5 49 name PIC G(*n*) DISPLAY-1 16 337<=*n*<=16 350	2 バイトのストリング長標識を持つ、可変長 2 バイト文字ストリング
DBCLOB(*n*) (412 または 413)	01 MY-DBCLOB USAGE IS SQL TYPE IS DBCLOB(n) 1<=*n*<=1 073 741 823	4 バイトのストリング長標識を持つ、ラージ・オブジェクト可変長 2 バイト文字ストリング
DBCLOB ロケーター変数[4] (968 または 969)	01 MY-DBCLOB-LOCATOR USAGE IS SQL TYPE IS DBCLOB-LOCATOR	サーバー上の DBCLOB エンティティーを識別する
DBCLOB ファイル参照変数[4] (924 または 925)	01 MY-DBCLOB-FILE USAGE IS SQL TYPE IS DBCLOB-FILE	DBCLOB データを含むファイルの記述子

注:

1. **SQL 列タイプ**の下の最初の数字は標識変数が提供されないことを示し、 2 番目の数字は標識変数が提供されることを示します。標識変数は、NULL 値を示したり、切り捨てられたストリングの長さを保持するのに必要です。これらの値は、それぞれのデータ・タイプの SQLDA の SQLTYPE フィールドに現れます。
2. FLOAT(*n*)。ここで 0 < *n* < 25 の場合、REAL と同義。 SQLDA での REAL と DOUBLE の違いは長さの値です (4 または 8)。
3. 以下の SQL タイプは、DOUBLE と同義です。
   - FLOAT
   - FLOAT(*n*)。ここで、*n* の取る範囲は 24 < *n* < 54。
   - DOUBLE PRECISION
4. これは列タイプではなく、ホスト変数タイプです。

サポートされる COBOL データ・タイプについては、さらに次の規則があります。

- PIC S9 と COMP-3/COMP-5 が明示されている場合、これらは必須です。
- VARCHAR、LONG VARCHAR、VARGRAPHIC、LONG VARGRAPHIC、すべての LOB 変数タイプ以外の列タイプについては、レベル番号として 01 の代わりに 77 を使用できます。
- DECIMAL(p,s) 列タイプのホスト変数を宣言する際には、以下の規則を使用します。以下のサンプルを参照してください。

  ```
  01 identifier PIC S9(m)V9(n) COMP-3
  ```

  – 小数点の表記に V を使用します。

- *n* と *m* の値は 1 以上でなければなりません。
- *n* + *m* の値は 31 以下でなければなりません。
- *s* の値は *n* の値と等しくなります。
- *p* の値は *n* + *m* の値と等しくなります。
- 反復因数 *(n)* と *(m)* はオプションです。以下の例はすべて有効です。

```
01 identifier PIC S9(3)V COMP-3
01 identifier PIC SV9(3) COMP-3
01 identifier PIC S9V COMP-3
01 identifier PIC SV9 COMP-3
```

- COMP-3 の代わりに PACKED-DECIMAL を使用できます。

• 配列は、COBOL プリコンパイラーではサポートされていません。

関連概念:

• 215 ページの『COBOL 用のホスト変数がある SQL 宣言セクション』

BINARY/COMP-4 COBOL データ・タイプ

DB2® COBOL プリコンパイラーは、整数ホスト変数および標識が許可されているときには常に、BINARY、COMP、および COMP-4 データ・タイプの使用をサポートします。ただしそれは、ターゲット COBOL コンパイラーが BINARY、COMP、または COMP-4 データ・タイプを、COMP-5 データ・タイプと等しく見なす (または等しく見なすようにできる) 場合に限ります。 本書では、そのようなホスト変数および標識を、タイプ COMP-5 と示します。 COMP、COMP-4、BINARY COMP および COMP-5 を等価として扱う、 DB2 のサポートするターゲット・コンパイラーは、次のとおりです。

• IBM® COBOL Set for AIX®

• Micro Focus COBOL for AIX

COBOL での FOR BIT DATA

一定のデータベース列には FOR BIT DATA を宣言できます。通常は文字を含むこれらの列は、バイナリー情報を保持するために使用されます。バイナリー・データを含めることのできる COBOL ホスト変数のタイプは、 CHAR(*n*)、VARCHAR、LONG VARCHAR、および BLOB データ・タイプです。これらのデータ・タイプは、FOR BIT DATA 属性を用いて処理を行う場合に使用してください。

関連資料:

• 215 ページの『COBOL でサポートされている SQL データ・タイプ』

COBOL での SQLSTATE および SQLCODE 変数

LANGLEVEL プリコンパイル・オプションを SQL92E の値とともに使用すると、次の 2 つの宣言をホスト変数として組み込めます。

```
EXEC SQL BEGIN DECLARE SECTION END-EXEC.
01 SQLSTATE PICTURE X(5).
01 SQLCODE  PICTURE S9(9) USAGE COMP.
.
.
.
EXEC SQL END DECLARE SECTION END-EXEC.
```

これらのいずれも指定しない場合は、SQLCODE 宣言はプリコンパイル中であると見なされます。 01 は 77 でもかまいませんし、PICTURE は PIC でもかまいません。このオプションを使用するときには、INCLUDE SQLCA ステートメントを指定してはならないことに注意してください。

複数のソース・ファイルから成るアプリケーションでは、 SQLCODE および SQLSTATE 定義を上の例に示された最初のソース・ファイルで定義することができます。

COBOL での日本語または中国語 (繁体字) EUC、および UCS-2 に関する考慮事項

eucJp または eucTW コード・セットで実行されている、または UCS-2 データベースに接続されているアプリケーションから送られてくるグラフィック・データはすべて、 UCS-2 コード・ページ ID でタグ付けされます。アプリケーションの側では、グラフィック文字ストリングをデータベース・サーバーに送る前に UCS-2 に変換しておく必要があります。同様に、UCS-2 データベースからアプリケーションが取り出すグラフィック・データ、または EUC eucJP または eucTW コードで実行されているアプリケーションがデータベースから取り出すグラフィック・データも、UCS-2 によってエンコードされます。このため、アプリケーションの側では、UCS-2 データで表示しようとする場合を除き、内部的に UCS-2 からご使用のアプリケーションのコード・ページに変換する必要があります。

このような変換は、SQLDA へのデータのコピー前、および SQLDA からのデータのコピー後に実行する必要があるため、 UCS-2 への変換および UCS-2 からの変換は、ご使用のアプリケーションが担当することになります。DB2 Universal Database では、アプリケーションからアクセス可能な変換ルーチンは提供していません。その代わりに、ご使用のオペレーティング・システムから呼び出し可能なシステム・コールを使用してください。 UCS-2 データベースの場合は、VARCHAR および VARGRAPHIC スカラー関数の使用を考慮することができます。

関連概念:

- 673 ページの『日本語および中国語 (繁体字) EUC および UCS-2 コード・セットに関する考慮事項』

関連資料:

- 「*SQL リファレンス 第 1 巻*」の『VARCHAR スカラー関数』
- 「*SQL リファレンス 第 1 巻*」の『VARGRAPHIC スカラー関数』

オブジェクト指向 COBOL

オブジェクト指向 COBOL を使用する場合、以下の規則に従う必要があります。

- SQL ステートメントは、1 コンパイル単位内の最初のプログラムまたはクラスだけに使用できます。この制約事項は、プリコンパイラーが参照する最初の作業用ストレージ・セクションに一時作業データを挿入するために存在します。
- オブジェクト指向 COBOL プログラムでは、SQL ステートメントが含まれるそれぞれのクラスには、クラス・レベルの作業用ストレージ・セクションがなけれ

ばなりません。これは、このセクションが空である場合にも当てはまります。このセクションを使用して、プリコンパイラーが生成したデータ定義を保管します。

第 9 章 FORTRAN でのプログラミング

FORTRAN でのプログラミングに関する考慮事項 221
FORTRAN における言語制約事項 221
FORTRAN での参照による呼び出し 221
FORTRAN でのデバッグとコメント行 222
FORTRAN でのプリコンパイルに関する考慮事項 222
FORTRAN におけるマルチスレッド・データベース・アクセス. 222
FORTRAN の入力および出力ファイル 222
組み込みファイル 222
FORTRAN の組み込みファイル 222
FORTRAN アプリケーションの組み込みファイル 225
FORTRAN における組み込み SQL ステートメント 226
FORTRAN のホスト変数 227
FORTRAN のホスト変数 227
FORTRAN におけるホスト変数の名前 228
FORTRAN におけるホスト変数宣言 228
FORTRAN における数値ホスト変数の構文 . . 229
FORTRAN における文字ホスト変数の構文 . . 229
FORTRAN の標識変数 231
FORTRAN におけるラージ・オブジェクト (LOB) ホスト変数の構文 231
FORTRAN におけるラージ・オブジェクト (LOB) ロケーター・ホスト変数の構文 232
FORTRAN におけるファイル参照ホスト変数の構文 232
FORTRAN 用のホスト変数がある SQL 宣言セクション 233
FORTRAN でサポートされている SQL データ・タイプ 233
FORTRAN でのマルチバイト文字セットに関する考慮事項. 235
FORTRAN での日本語または中国語 (繁体字) EUC、および UCS-2 に関する考慮事項 235
FORTRAN の SQLSTATE および SQLCODE 変数 235

FORTRAN でのプログラミングに関する考慮事項

特定のホスト言語によるプログラミングの考慮事項について、以降のセクションで説明します。言語制限、ホスト言語別の組み込みファイル、組み込み SQL ステートメント、ホスト変数、およびサポートされるホスト変数のデータ・タイプについての情報が含まれています。

注: FORTRAN のサポートは DB2 バージョン 5 において確立され、今後 FORTRAN のサポートを拡張する予定はありません。たとえば、FORTRAN プリコンパイラーは、表名などの SQL オブジェクト ID を処理できません。これは 18 バイトより長いからです。 19 ～ 128 バイト長の表名などの、バージョン 5 より後の DB2 に追加された機能を使用したい場合には、 FORTRAN 以外の言語でアプリケーションを作成する必要があります。

FORTRAN における言語制約事項

以下のセクションでは、FORTRAN に関する言語制約事項について説明します。

FORTRAN での参照による呼び出し

API パラメーターの中には、呼び出し変数に値ではなくアドレスを必要とするものもあります。データベース・マネージャーは、それらのパラメーターの指定を単純化する GET ADDRESS、DEREFERENCE ADDRESS、および COPY MEMORY API を提供します。

関連資料:

- 「*管理 API リファレンス*」の『sqlgdref - アドレスの参照解除』
- 「*管理 API リファレンス*」の『sqlgaddr - アドレスの入手』

- 「*管理 API リファレンス*」の『sqlgmcpy - メモリーのコピー』

FORTRAN でのデバッグとコメント行

一部の FORTRAN コンパイラーは、 1 列目が 'D' または 'd' である行を条件行と見なします。これらの行は、デバッグのためにコンパイルすることも、コメントとして取り扱うことも可能です。プリコンパイラーは、1 列目が 'D' または 'd' である行を常にコメントと見なします。

FORTRAN でのプリコンパイルに関する考慮事項

以下の項目は、プリコンパイル処理に影響を及ぼします。

- プリコンパイラーは、連続記入行の 1 〜 5 列目には数字、ブランク、およびタブ文字しか認めない。
- `.sqf` ソース・ファイルでは、ホレリス定数はサポートされない。

FORTRAN におけるマルチスレッド・データベース・アクセス

FORTRAN はマルチスレッド・データベース・アクセスをサポートしていません。

FORTRAN の入力および出力ファイル

デフォルトでは、入力ファイルの拡張子は `.sqf` ですが、 TARGET プリコンパイル・オプションを使用した場合は、入力ファイルに任意の拡張子を付けることができます。

出力ファイルの拡張子は、デフォルトでは UNIX® ベースのプラットフォームでは `.f` であり、 Windows® ベースのプラットフォームでは `.for` ですが、 OUTPUT プリコンパイル・オプションを使用すれば出力修正後のソース・ファイルに新しい名前やパスを指定できます。

関連資料:

- 「コマンド・リファレンス」の『PRECOMPILE コマンド』

組み込みファイル

以下のセクションでは、FORTRAN の組み込みファイルについて説明します。

FORTRAN の組み込みファイル

FORTRAN ホスト言語に固有の組み込みファイルのファイル拡張子は、 UNIX ベースのプラットフォームの場合は `.f` です。 Windows ベースの場合は `.for` です。以下の FORTRAN 組み込みファイルをアプリケーションで使用することができます。

SQL (sql.f) このファイルには、バインド・プログラム、プリコンパイラー、およびエラー・メッセージ検索 API 用の言語固有プロトタイプが含まれています。また、システム定数も定義されています。

SQLAPREP (sqlaprep.f)

このファイルには、独自のプリコンパイラーの作成に必要な定義が入っています。

SQLCA (sqlca_cn.f, sqlca_cs.f)

このファイルは SQL 連絡域 (SQLCA) 構造体を定義します。 SQLCA には、データベース・マネージャーが SQL ステートメントおよび API 呼び出しの実行に関するエラー情報をアプリケーションに提供するために使用する変数が含まれています。

FORTRAN アプリケーションには 2 つの SQLCA ファイルが提供されます。 `sqlca_cs.f` はデフォルトで、IBM SQL 互換フォーマットで SQLCA 構造体を定義します。 `SQLCA NONE` オプションを指定して `sqlca_cn.f` ファイルをプリコンパイルすると、定義される SQLCA 構造体のパフォーマンスが向上します。

SQLCA_92 (sqlca_92.f)

このファイルには、SQL 連絡域 (SQLCA) 構造体の FIPS SQL92 Entry Level 準拠版が入っています。このファイルは、FIPS SQL92 Entry Level standard に適合する DB2 アプリケーションを作成する際に、 `sqlca_cn.f` または `sqlca_cs.f` のどちらかのファイルの代わりに組み込まれる必要があります。 LANGLEVEL プリコンパイラー・オプションに SQL92E が設定されていれば、 `sqlca_92.f` ファイルは DB2 プリコンパイラーによって自動的に組み込まれます。

SQLCODES (sqlcodes.f)

このファイルは、SQLCA 構造体の SQLCODE フィールドで使用する定数を定義します。

SQLDA (sqldact.f)

このファイルは SQL 記述子域 (SQLDA) 構造体を定義します。 SQLDA はアプリケーションとデータベース・マネージャーとの間でデータをやりとりするために使用されます。

SQLEAU (sqleau.f)

このファイルには、DB2 セキュリティー監査 API に必要な定数および構造の定義が含まれています。これらの API を使用する場合は、プログラムにこのファイルを組み込む必要があります。このファイルにはまた、監査証跡レコード内のフィールドの定数およびキーワード値の定義も含まれています。これらの定義は、外部または取引先の監査証跡抽出プログラムで使用できます。

SQLENV (sqlenv.f)

このファイルは、データベース環境 API に対する言語固有の呼び出し、およびそれらのインターフェースの構造、定数、戻りコードを定義します。

SQLE819A (sqle819a.f)

データベースのコード・ページが 819 (ISO ラテン 1) の場合、このシーケンスは、ホスト CCSID 500 (EBCDIC 各国対応) バイナリ

ー照合に従って、 FOR BIT DATA ではない文字ストリングをソートします。このファイルは CREATE DATABASE API が使用します。

SQLE819B (sqle819b.f)
データベースのコード・ページが 819 (ISO ラテン 1) の場合、このシーケンスは、ホスト CCSID 037 (EBCDIC 米国英語) バイナリー照合に従って、 FOR BIT DATA ではない文字ストリングをソートします。このファイルは CREATE DATABASE API が使用します。

SQLE850A (sqle850a.f)
データベースのコード・ページが 850 (ASCII ラテン 1) の場合、このシーケンスは、ホスト CCSID 500 (EBCDIC 各国対応) バイナリー照合に従って、 FOR BIT DATA ではない文字ストリングをソートします。このファイルは CREATE DATABASE API が使用します。

SQLE850B (sqle850b.f)
データベースのコード・ページが 850 (ASCII ラテン 1) の場合、このシーケンスは、ホスト CCSID 037 (EBCDIC 米国英語) バイナリー照合に従って、 FOR BIT DATA ではない文字ストリングをソートします。このファイルは CREATE DATABASE API が使用します。

SQLE932A (sqle932a.f)
データベースのコード・ページが 932 (ASCII 日本語) の場合、このシーケンスは、ホスト CCSID 5035 (EBCDIC 日本語) バイナリー照合に従って、 FOR BIT DATA ではない文字ストリングをソートします。このファイルは CREATE DATABASE API が使用します。

SQLE932B (sqle932b.f)
データベースのコード・ページが 932 (ASCII 日本語) の場合、このシーケンスは、ホスト CCSID 5026 (EBCDIC 日本語) バイナリー照合に従って、 FOR BIT DATA ではない文字ストリングをソートします。このファイルは CREATE DATABASE API が使用します。

SQL1252A (sql1252a.f)
データベースのコード・ページが 1252 (Windows ラテン 1) の場合、このシーケンスは、ホスト CCSID 500 (EBCDIC 各国対応) バイナリー照合に従って、FOR BIT DATA ではない文字ストリングをソートします。このファイルは CREATE DATABASE API が使用します。

SQL1252B (sql1252b.f)
データベースのコード・ページが 1252 (Windows ラテン 1) の場合、このシーケンスは、ホスト CCSID 037 (EBCDIC 米国英語) バイナリー照合に従って、FOR BIT DATA ではない文字ストリングをソートします。このファイルは CREATE DATABASE API が使用します。

SQLMON (sqlmon.f)
このファイルは、データベース・システム・モニター API に対する言語固有の呼び出し、およびそれらのインターフェースの構造、定数、戻りコードを定義します。

SQLSTATE (sqlstate.f)
このファイルは、SQLCA 構造体の SQLSTATE フィールドで使用する定数を定義します。

SQLUTIL (sqlutil.f)
このファイルは、ユーティリティー API に対する言語固有の呼び出し、およびそれらのインターフェースに必要な構造、定数、コードを定義します。

関連概念:

- 225 ページの『FORTRAN アプリケーションの組み込みファイル』

FORTRAN アプリケーションの組み込みファイル

ファイルの組み込みには、EXEC SQL INCLUDE ステートメントを使用する方法と、 FORTRAN INCLUDE ステートメントを使用する方法の 2 つがあります。プリコンパイラーは FORTRAN INCLUDE ステートメントを無視し、 EXEC SQL ステートメントを使用して組み込まれたファイルのみを処理します。

INCLUDE ファイルを探索する際、 DB2® FORTRAN プリコンパイラーはまず最初に現行ディレクトリーを検索し、次いで DB2INCLUDE 環境変数に指定されたディレクトリーを検索します。 次の例を考えてみてください。

- `EXEC SQL INCLUDE payroll`

  INCLUDE ステートメントに指定されたファイルが単引用符 (') で囲まれていない場合、上記のとおり、プリコンパイラーは最初 `payroll.sqf` を検索し、次に `payroll.f` (Windows® ベースのプラットフォームでは `payroll.for`)、最後にそれがロックする各ディレクトリーを検索します。

- `EXEC SQL INCLUDE 'pay/payroll.f'`

  ファイル名が単引用符 (') で囲まれている場合、すでに説明したとおり、名前に拡張子は追加されません。 (Windows ベースのプラットフォームの場合、ファイルは `'pay¥payroll.for'` と指定されます。)

  単引用符 (') 内のファイル名に完全パスが含まれていない場合、 DB2INCLUDE の中身によってそのファイルの検索が行われ、何らかのパスが INCLUDE ファイル名に指定されます。たとえば、DB2 (UNIX® ベースのプラットフォーム版) では、 DB2INCLUDE は `'/disk2:myfiles/fortran'` に設定され、プリコンパイラーは `'./pay/payroll.f'`、 `'/disk2/pay/payroll.f'`、 `'./myfiles/cobol/pay/payroll.f'` の順に探索します。プリコンパイラー・メッセージには、実際にファイルが見つかったパスが表示されます。 Windows ベースのプラットフォームでは、上記の例でスラッシュ (/) の代わりに円記号 (¥) を使用し、拡張子 `'f'` の代わりに `'for'` を使用します。

注: DB2INCLUDE の設定は、DB2 コマンド行プロセッサーによってキャッシュされます。 何らかの CLP コマンドを発行した後に DB2INCLUDE の設定を変更する場合は、 TERMINATE コマンドを入力してから、データベースに接続し直し、あとは通常どおりプリコンパイルしてください。

関連概念:

- 「*管理ガイド: パフォーマンス*」の『DB2 レジストリー変数と環境変数』

関連資料:

- 222 ページの『FORTRAN の組み込みファイル』

FORTRAN における組み込み SQL ステートメント

組み込み SQL ステートメントは次の 3 つのエレメントからなります。

エレメント	正しい FORTRAN 構文
キーワード	EXEC SQL
ステートメント・ストリング	ブランクを区切り文字として使用した有効な SQL ステートメント
ステートメント終止符	ソース行の終わり

ソース行の終端は、ステートメント終止符として機能します。行が継続する場合、ステートメント終止符は連続記入行の最終行の終端に位置します。

たとえば、次のようになります。

```
EXEC SQL SELECT COL INTO :hostvar FROM TABLE
```

組み込み SQL ステートメントには、以下の規則が適用されます。

- SQL ステートメントは 7 ～ 72 列までの間でコーディングする。
- 全行の FORTRAN コメント、または SQL コメントを使用する。ただし、SQL ステートメント内で FORTRAN 行末コメント '!' 文字は使用できません。それ以外の場所であれば、このコメント文字はホスト変数宣言を含めてどこでも使用できます。
- 組み込み SQL ステートメントをコーディングする際は、 FORTRAN ステートメントにその必要がない場合でも、ブランクを区切り文字として使用する。
- 各 FORTRAN ソース行に対しては、SQL ステートメントのみを使用する。複数の行が必要なステートメントには、通常の FORTRAN の連結規則が適用されます。ただし、対になった EXEC SQL キーワードを複数行に分割してはなりません。
- SQL コメントは、組み込み SQL ステートメントの一部となっている行であればどこでも使用することができる。このコメントは、動的に実行するステートメントでは使用できません。 SQL コメントの形式は、ダブル・ダッシュ (--) の後に 0 個以上の文字ストリングが続き、行末で終了します。
- FORTRAN のコメントは、組み込み SQL ステートメント内のほとんど の場所で使用することができる。例外は以下のとおりです。
  - コメントは EXEC と SQL との間では使用できない。
  - コメントは動的に実行されるステートメントでは使用できない。

  - 行末で ! を使用して FORTRAN のコメントをコーディングすることは、組み込み SQL ではサポートされていない。
- SQL ステートメント内で実定数を指定する際には、指数表記法を使用する。データベース・マネージャーは、SQL ステートメント内にある 小数点を持った数字ストリングを、実定数ではなく 10 進定数と解釈します。
- 最初の実行可能 FORTRAN ステートメントよりも前にある SQL ステートメントでは、ステートメント番号は無効になる。 SQL ステートメントにこれと対応するステートメント番号が付けられている場合、プリコンパイラーは、SQL ステートメントの直前に置かれるラベル付き CONTINUE ステートメントを生成します。
- SQL ステートメント内のホスト変数の参照時に宣言されたとおりのホスト変数を使用する。
- 行末文字およびタブ文字などの空白文字の置換は、次のように行われる。
  - 引用符の外 (ただし、SQL ステートメント内) では、行末文字または TAB 文字は単一スペースと置き換えられる。
  - 引用符内では、FORTRAN プログラムに適合した形でストリングが継続されていれば、行末文字は消去される。 TAB は修正されません。

  行末および TAB に使用される実際の文字は、プラットフォームごとに異なります。たとえば Windows® ベースのプラットフォームの場合は、復帰/改行を行末に使用するのに対し、 UNIX® ベースのプラットフォームは改行のみを使用します。

関連資料:

- 751 ページの『付録 A. サポートされる SQL ステートメント』

FORTRAN のホスト変数

以下のセクションでは、FORTRAN プログラムにおけるホスト変数の宣言と使用について説明します。

FORTRAN のホスト変数

ホスト変数は、SQL ステートメント内で参照される FORTRAN 言語変数となります。これにより、アプリケーションは入力データをデータベース・マネージャーに渡し、またデータベース・マネージャーから出力データを受け取ることができます。アプリケーションがコンパイルされると、コンパイラーはホスト変数を他の FORTRAN 変数として使用します。

関連概念:

- 228 ページの『FORTRAN におけるホスト変数の名前』
- 228 ページの『FORTRAN におけるホスト変数宣言』
- 231 ページの『FORTRAN の標識変数』

関連資料:

- 229 ページの『FORTRAN における数値ホスト変数の構文』
- 229 ページの『FORTRAN における文字ホスト変数の構文』

- 231 ページの『FORTRAN におけるラージ・オブジェクト (LOB) ホスト変数の構文』
- 232 ページの『FORTRAN におけるラージ・オブジェクト (LOB) ロケーター・ホスト変数の構文』
- 232 ページの『FORTRAN におけるファイル参照ホスト変数の構文』

FORTRAN におけるホスト変数の名前

SQL プリコンパイラーは、宣言された名前によってホスト変数を識別します。この場合、以下の規則が適用されます。

- 変数名は 255 文字以内の長さで指定する。
- ホスト変数は、SQL、sql、DB2®、または db2 以外の接頭部で開始する。これらはシステムが使用する予約語です。

関連概念:

- 228 ページの『FORTRAN におけるホスト変数宣言』

関連資料:

- 229 ページの『FORTRAN における数値ホスト変数の構文』
- 229 ページの『FORTRAN における文字ホスト変数の構文』
- 231 ページの『FORTRAN におけるラージ・オブジェクト (LOB) ホスト変数の構文』
- 232 ページの『FORTRAN におけるラージ・オブジェクト (LOB) ロケーター・ホスト変数の構文』
- 232 ページの『FORTRAN におけるファイル参照ホスト変数の構文』

FORTRAN におけるホスト変数宣言

ホスト変数宣言の識別には、SQL の宣言セクションを使用しなければなりません。このようにして、それ以降の SQL ステートメントで参照が可能なホスト変数をプリコンパイラーに知らせます。

FORTRAN プリコンパイラーは、有効な FORTRAN 宣言のサブセットのみを有効な変数宣言として認識します。これらの宣言は、数値変数または文字変数のいずれかを宣言します。数値ホスト変数は、数値の SQL 入出力値に対する入出力値として使用することができます。文字ホスト変数は任意の文字、日付、時間またはタイム・スタンプの SQL 入出力値に対する入出力値として使用できます。プログラマーは、出力変数が受け取る値を含めることのできる長さを持つようにコーディングしなければなりません。

関連タスク:

- 「*アプリケーション開発ガイド サーバー・アプリケーションのプログラミング*」の『構造化型ホスト変数の宣言』

関連資料:

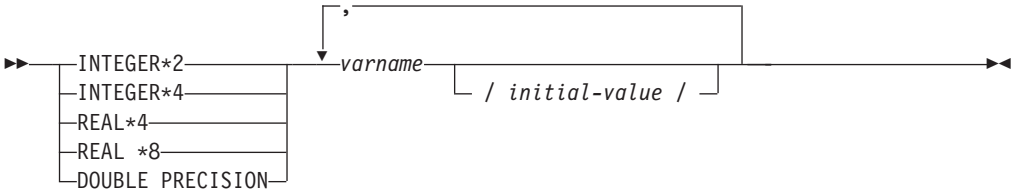
- 229 ページの『FORTRAN における数値ホスト変数の構文』
- 229 ページの『FORTRAN における文字ホスト変数の構文』

- 231 ページの『FORTRAN におけるラージ・オブジェクト (LOB) ホスト変数の構文』
- 232 ページの『FORTRAN におけるラージ・オブジェクト (LOB) ロケーター・ホスト変数の構文』
- 232 ページの『FORTRAN におけるファイル参照ホスト変数の構文』

FORTRAN における数値ホスト変数の構文

FORTRAN における数値ホスト変数の構文を次に示します。

FORTRAN における数値ホスト変数の構文



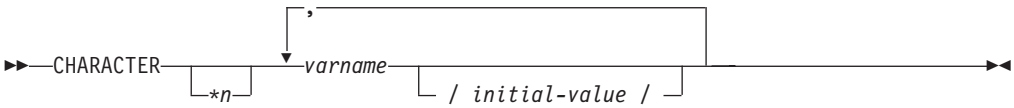
数値ホスト変数に関する考慮事項:

1. REAL*8 と DOUBLE PRECISION は同値である。
2. REAL*8 定数のインディケーターには、D ではなく E を使用する。

FORTRAN における文字ホスト変数の構文

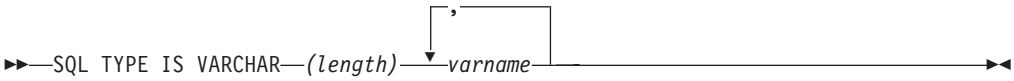
固定長文字ホスト変数の構文を次に示します。

FORTRAN における文字変数の構文: 固定長



可変長文字ホスト変数の構文を次に示します。

可変長



文字ホスト変数に関する考慮事項:

1. *n の最大値は 254。
2. 長さ (length) が 1 と 32 672 の間である場合、ホスト変数のタイプは VARCHAR(448)。
3. 長さ (length) が 32 673 と 32 700 の間である場合、ホスト変数のタイプは LONG VARCHAR(SQLTYPE 456)。
4. 宣言内で VARCHAR ホスト変数および LONG VARCHAR ホスト変数を初期設定することは許可されていない。

VARCHAR の例:

宣言:

```
sql type is varchar(1000) my_varchar
```

この結果、以下の構造が生成されます。

```
character    my_varchar(1000+2)
integer*2    my_varchar_length
character    my_varchar_data(1000)
equivalence( my_varchar(1),
+            my_varchar_length )
equivalence( my_varchar(3),
+            my_varchar_data )
```

アプリケーションは、たとえば、ホスト変数の内容の設定や検査のために、my_varchar_length および my_varchar_data の両方を処理できます。 SQL ステートメントでベース名 (ここでは、my_varchar) を使用して、 VARCHAR 全体を参照します。

LONG VARCHAR の例:

宣言:

```
sql type is varchar(10000) my_lvarchar
```

この結果、以下の構造が生成されます。

```
character    my_lvarchar(10000+2)
integer*2    my_lvarchar_length
character    my_lvarchar_data(10000)
equivalence( my_lvarchar(1),
+            my_lvarchar_length )
equivalence( my_lvarchar(3),
+            my_lvarchar_data )
```

アプリケーションは、たとえば、ホスト変数の内容の設定や検査のために、my_lvarchar_length および my_lvarchar_data の両方を処理できます。 SQL ステートメントでベース名 (ここでは、my_lvarchar) を使用して、 LONG VARCHAR 全体を参照します。

注: 以下の例に示すような CONNECT ステートメントでは、 FORTRAN 文字ストリング・ホスト変数 dbname および userid の後続ブランクは、処理前に削除されます。

```
EXEC SQL CONNECT TO :dbname USER :userid USING :passwd
```

しかし、パスワードにはブランクも有効であるため、パスワードのホスト変数を VARCHAR として宣言し、実際のパスワードの長さを反映するように長さ (length) フィールドを設定しなければなりません。

```
EXEC SQL BEGIN DECLARE SECTION
  character*8 dbname, userid
  sql type is varchar(18) passwd
EXEC SQL END DECLARE SECTION
character*18 passwd_string
equivalence(passwd_data,passwd_string)
dbname = 'sample'
userid = 'userid'
passwd_length= 8
passwd_string = 'password'
EXEC SQL CONNECT TO :dbname USER :userid USING :passwd
```

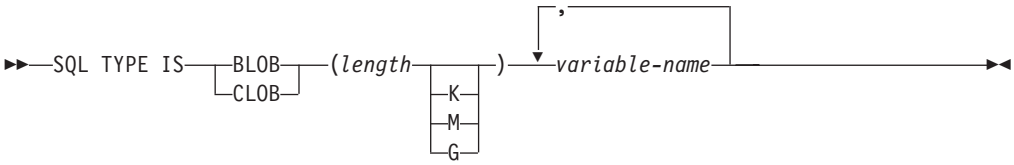
FORTRAN の標識変数

標識変数は、INTEGER*2 データ・タイプとして宣言します。

FORTRAN におけるラージ・オブジェクト (LOB) ホスト変数の構文

FORTRAN におけるラージ・オブジェクト (LOB) ホスト変数の宣言構文を次に示します。

FORTRAN におけるラージ・オブジェクト (LOB) ホスト変数の構文



LOB ホスト変数に関する考慮事項:

1. GRAPHIC タイプは、FORTRAN ではサポートされていない。
2. SQL TYPE IS、BLOB、CLOB、K、M、G は、大文字、小文字、またはその混合のいずれでもかまわない。
3. BLOB および CLOB の場合は、 1 <= lob-length <= 2 147 483 647 である。
4. LOB 宣言内での LOB の初期化はできない。
5. ホスト変数名には、プリコンパイラー生成コードにおいて 'length' と 'data' という接頭部がある。

BLOB の例:

宣言:

```
sql type is blob(2m) my_blob
```

この結果、以下の構造が生成されます。

```
 character    my_blob(2097152+4)
 integer*4    my_blob_length
 character    my_blob_data(2097152)
 equivalence( my_blob(1),
+             my_blob_length )
 equivalence( my_blob(5),
+             my_blob_data )
```

CLOB の例:

宣言:

```
sql type is clob(125m) my_clob
```

この結果、以下の構造が生成されます。

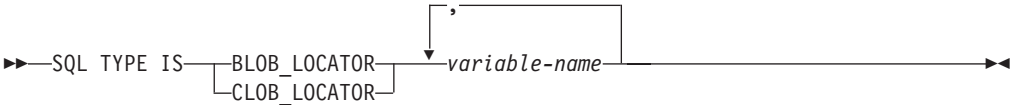
```
 character    my_clob(131072000+4)
 integer*4    my_clob_length
 character    my_clob_data(131072000)
```

```
 equivalence( my_clob(1),
+             my_clob_length )
 equivalence( my_clob(5),
+             my_clob_data )
```

FORTRAN におけるラージ・オブジェクト (LOB) ロケーター・ホスト変数の構文

FORTRAN におけるラージ・オブジェクト (LOB) ロケーター・ホスト変数の宣言構文を次に示します。

FORTRAN におけるラージ・オブジェクト (LOB) ロケーター・ホスト変数の構文



LOB ロケーター・ホスト変数に関する考慮事項:

1. GRAPHIC タイプは、FORTRAN ではサポートされていない。
2. SQL TYPE IS、BLOB_LOCATOR、CLOB_LOCATOR は、大文字、小文字、またはその混合のいずれでもかまわない。
3. ロケーターの初期化はできない。

CLOB ロケーターの例 (BLOB ロケーターと同様):

宣言:

```
SQL TYPE IS CLOB_LOCATOR my_locator
```

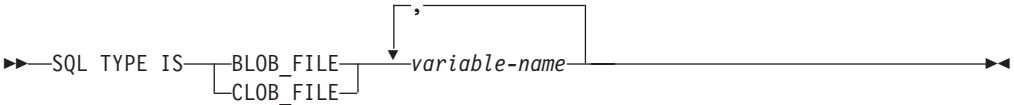
この結果、以下の宣言が生成されます。

```
integer*4 my_locator
```

FORTRAN におけるファイル参照ホスト変数の構文

FORTRAN におけるファイル参照ホスト変数の宣言構文を次に示します。

FORTRAN におけるファイル参照ホスト変数の構文



ファイル参照ホスト変数に関する考慮事項:

1. GRAPHIC タイプは、FORTRAN ではサポートされていない。
2. SQL TYPE IS、BLOB_FILE、CLOB_FILE は、大文字、小文字、またはその混合のいずれでもかまわない。

BLOB ファイル参照変数の例 (CLOB ファイル参照変数と同様):

```
SQL TYPE IS BLOB_FILE my_file
```

この結果、以下の宣言が生成されます。

```
 character     my_file(267)
 integer*4     my_file_name_length
 integer*4     my_file_data_length
 integer*4     my_file_file_options
 character*255 my_file_name
 equivalence(  my_file(1),
+              my_file_name_length )
 equivalence(  my_file(5),
+              my_file_data_length )
 equivalence(  my_file(9),
+              my_file_file_options )
 equivalence(  my_file(13),
+              my_file_name )
```

FORTRAN 用のホスト変数がある SQL 宣言セクション

サポートされているデータ・タイプそれぞれについて宣言されたホスト変数を含む、 SQL 宣言のサンプルを以下に示します。

```
EXEC SQL BEGIN DECLARE SECTION
  INTEGER*2     AGE  /26/
  INTEGER*4     DEPT
  REAL*4        BONUS
  REAL*8        SALARY
  CHARACTER     MI
  CHARACTER*112 ADDRESS
  SQL TYPE IS VARCHAR (512) DESCRIPTION
  SQL TYPE IS VARCHAR (32000) COMMENTS
  SQL TYPE IS CLOB (1M) CHAPTER
  SQL TYPE IS CLOB_LOCATOR CHAPLOC
  SQL TYPE IS CLOB_FILE  CHAPFL
  SQL TYPE IS BLOB (1M) VIDEO
  SQL TYPE IS BLOB_LOCATOR VIDLOC
  SQL TYPE IS BLOB_FILE VIDFL
  CHARACTER*10  DATE
  CHARACTER*8   TIME
  CHARACTER*26  TIMESTAMP
  INTEGER*2     WAGE_IND
EXEC SQL END DECLARE SECTION
```

関連資料:

- 233 ページの『FORTRAN でサポートされている SQL データ・タイプ』

FORTRAN でサポートされている SQL データ・タイプ

特定の定義済み FORTRAN のデータ・タイプは、データベース・マネージャーの列のタイプに対応しています。ホスト変数として宣言できるのは、これらの FORTRAN データ・タイプのみです。

それぞれの列タイプに対応する FORTRAN データ・タイプを次に示します。プリコンパイラーはホスト変数宣言を検出すると、該当する SQL タイプの値を判別します。データベース・マネージャーはこの値を使用して、アプリケーションとの間でやりとりするデータを変換します。

注: DB2 ホスト言語で、DATALINK データ・タイプをサポートするホスト変数はありません。

表 16. FORTRAN 宣言にマップされる SQL データ・タイプ

SQL 列タイプ[1]	FORTRAN データ・タイプ	SQL 列タイプ記述
SMALLINT (500 または 501)	INTEGER*2	16 ビットの符号付き整数
INTEGER (496 または 497)	INTEGER*4	32 ビットの符号付き整数
REAL[2] (480 または 481)	REAL*4	単精度浮動小数点
DOUBLE[3] (480 または 481)	REAL*8	倍精度浮動小数点
DECIMAL(*p*,*s*) (484 または 485)	厳密に対応するものがない ; REAL*8 を使用	パック 10 進数
CHAR(*n*) (452 または 453)	CHARACTER**n*	長さが *n* の固定長文字ストリング (*n* の範囲は 1 ～ 254 まで)
VARCHAR(*n*) (448 または 449)	SQL TYPE IS VARCHAR(*n*) *n* の範囲は 1 ～ 32 672	可変長文字ストリング
LONG VARCHAR (456 または 457)	SQL TYPE IS VARCHAR(*n*) *n* の範囲は 32 673 ～ 32 700	long 可変長文字ストリング
CLOB(*n*) (408 または 409)	SQL TYPE IS CLOB (n) *n* の範囲は 1 ～ 2 147 483 647	ラージ・オブジェクト可変長文字ストリング
CLOB ロケーター変数[4] (964 または 965)	SQL TYPE IS CLOB_LOCATOR	サーバー上の CLOB エンティティーを識別する
CLOB ファイル参照変数[4] (920 または 921)	SQL TYPE IS CLOB_FILE	CLOB データを含むファイルの記述子
BLOB(*n*) (404 または 405)	SQL TYPE IS BLOB(n) *n* の範囲は 1 ～ 2 147 483 647	ラージ・オブジェクト可変長バイナリー・ストリング
BLOB ロケーター変数[4] (960 または 961)	SQL TYPE IS BLOB_LOCATOR	サーバー上の BLOB エンティティーを識別する
BLOB ファイル参照変数[4] (916 または 917)	SQL TYPE IS BLOB_FILE	BLOB データを含むファイルの記述子
DATE (384 または 385)	CHARACTER*10	10 バイトの文字ストリング
TIME (388 または 389)	CHARACTER*8	8 バイトの文字ストリング
TIMESTAMP (392 または 393)	CHARACTER*26	26 バイトの文字ストリング

注:

1. **SQL 列タイプ**の下の最初の数字は標識変数が提供されないことを示し、 2 番目の数字は標識変数が提供されることを示します。 標識変数は、NULL 値を示したり、切り捨てられたストリングの長さを保持するのに必要です。これらの値は、それぞれのデータ・タイプの SQLDA の SQLTYPE フィールドに現れます。
2. FLOAT(*n*)。ここで 0 < *n* < 25 の場合、REAL と同義。 SQLDA での REAL と DOUBLE の違いは長さの値です (4 または 8)。
3. 以下の SQL タイプは、DOUBLE と同義です。
   - FLOAT
   - FLOAT(*n*)。ここで、*n* の取る範囲は 24 < *n* < 54。
   - DOUBLE PRECISION
4. これは列タイプではなく、ホスト変数タイプである。

以下に、サポートされる FORTRAN データ・タイプのその他の規則を示します。

- VARCHAR、LONG VARCHAR、または CLOB ホスト変数を使用して、 254 文字よりも長い動的 SQL ステートメントを定義できる。

関連概念:

- 233 ページの『FORTRAN 用のホスト変数がある SQL 宣言セクション』

FORTRAN でのマルチバイト文字セットに関する考慮事項

FORTRAN では、グラフィック (マルチバイト) ホスト変数データ・タイプはサポートされていません。 `character` データ・タイプによって、混合文字ホスト変数だけがサポートされています。 グラフィック・データを含むユーザー SQLDA を作成することは可能です。

FORTRAN での日本語または中国語 (繁体字) EUC、および UCS-2 に関する考慮事項

eucJp または eucTW コード・セットで実行されている、または UCS-2 データベースに接続されているアプリケーションから送られてくるグラフィック・データはすべて、 UCS-2 コード・ページ ID でタグ付けされます。アプリケーションの側では、グラフィック文字ストリングをデータベース・サーバーに送る前に UCS-2 に変換しておく必要があります。同様に、UCS-2 データベースからアプリケーションが取り出すグラフィック・データ、または EUC eucJP または eucTW コードで実行されているアプリケーションがデータベースから取り出すグラフィック・データも、UCS-2 によってエンコードされます。このため、アプリケーションの側では、UCS-2 データで表示しようとする場合を除き、内部的に UCS-2 からご使用のアプリケーションのコード・ページに変換する必要があります。

このような変換は、SQLDA へのデータのコピー前、および SQLDA からのデータのコピー後に実行する必要があるため、 UCS-2 への変換および UCS-2 からの変換は、ご使用のアプリケーションが担当することになります。DB2 Universal Database では、アプリケーションからアクセス可能な変換ルーチンは提供していません。その代わりに、ご使用のオペレーティング・システムから呼び出し可能なシステム・コールを使用してください。 UCS-2 データベースの場合は、VARCHAR および VARGRAPHIC スカラー関数の使用を考慮することができます。

関連概念:

- 673 ページの『日本語および中国語 (繁体字) EUC および UCS-2 コード・セットに関する考慮事項』

関連資料:

- 「*SQL リファレンス 第 1 巻*」の『VARCHAR スカラー関数』
- 「*SQL リファレンス 第 1 巻*」の『VARGRAPHIC スカラー関数』

FORTRAN の SQLSTATE および SQLCODE 変数

LANGLEVEL プリコンパイル・オプションを SQL92E の値とともに使用すると、次の 2 つの宣言をホスト変数として組み込めます。

```
EXEC SQL BEGIN DECLARE SECTION;
  CHARACTER*5 SQLSTATE
  INTEGER     SQLCOD
  .
  .
  .
EXEC SQL END DECLARE SECTION
```

これらのどちらも指定されない場合、プリコンパイル・ステップの間、SQLCOD 宣言が仮定されます。 変数には SQLSTATE または SQLSTA と名前が付けられます。 このオプションを使用するときには、INCLUDE SQLCA ステートメントを指定してはならないことに注意してください。

複数のソース・ファイルがあるアプリケーションの場合、各ソース・ファイルに SQLCOD と SQLSTATE の宣言が上記のように組み込まれることがあります。

関連資料:

- 「コマンド・リファレンス」の『PRECOMPILE コマンド』

第 3 部 ADO.NET、OLE DB、および ODBC

第 10 章 DB2 .NET Data Provider

DB2 .NET Data Provider の概要 239
DB2 .NET Data Provider のシステム要件 239
DB2 .NET Data Provider を使用するアプリケーションのプログラミング 240
DB2 .NET Data Provider を使用したアプリケーションからのデータベースへの接続 240
DB2 .NET Data Provider を使用したアプリケーションからの SQL ステートメントの実行 . . . 240
DB2 .NET Data Provider を使用したアプリケーションからの結果セットの読み取り 242
DB2 .NET Data Provider を使用したアプリケーションからのストアード・プロシージャーの呼び出し 242
DB2 .NET Data Provider でサポートされている SQL データ・タイプ 244

DB2 .NET Data Provider の概要

DB2® .NET Data Provider は ADO.NET インターフェースの拡張であり、 .NET アプリケーションが、セキュア接続を通して DB2 データベースに接続し、コマンドを実行し、結果セットを検索することを可能します。

DB2 .NET Data Provider には、すべての DB2 .NET Data Provider オブジェクトとそれらのメンバーについての詳細情報が示された参照資料が組み込まれています。この資料は、DB2 インストール・プロセス中に、Microsoft® Visual Studio .NET に登録されます。 Microsoft Visual Studio .NET から DB2 .NET Data Provider の資料を表示するには、 「ヘルプ」メニュー・オプション、および「目次 (Contents)」を選択します。ヘルプ・ビューアーが開いたら、「*IBM® DB2 .NET Data Provider ヘルプ (IBM® DB2 .NET Data Provider Help)*」でフィルターに掛けてください。

DB2 .NET Data Provider のシステム要件

DB2® .NET Data Provider により、 .NET アプリケーションは、次のデータベース管理システムにアクセスできます。

- Windows®、UNIX®、および Linux ベースのコンピューター用の DB2 Universal Database™ バージョン 8
- DB2 Connect™ 経由で、DB2 Universal Database for OS/390® and z/OS™ バージョン 6 (以降)
- DB2 Connect 経由で、DB2 Universal Database for AS/400® and iSeries™ バージョン 5 リリース 1 (以降)
- DB2 Connect 経由で、DB2 Universal Database for VSE & VM バージョン 7.3 (以降)

DB2 インストール・プログラムを使用して DB2 .NET Data Provider をインストールする前に、 .NET Framework (バージョン 1.0 または バージョン 1.1) をコンピューターにインストールしておく必要があります。 .NET Framework がインストールされていないと、 DB2 インストール・プログラムは DB2 .NET Data Provider をインストールしません。

DB2 Universal Database for AS/400 and iSeries の場合は、サーバーに APAR ii13348 の修正を適用する必要があります。

DB2 for VSE & VM サーバーおよび DB2 for iSeries サーバーと共に使用できるのは、 .NET Framework バージョン 1.1 と Visual Studio .NET 2003 のみです。 .NET Framework バージョン 1.0 および Visual Studio .NET 2002 は、これらのサーバーと共に使用することはできません。

DB2 .NET Data Provider を使用するアプリケーションのプログラミング

以下のセクションでは、 DB2 データベースのデータを使用または操作する .NET アプリケーションをプログラミングする際の主なステップについて説明します。 C# および Visual Basic .NET の例を用いて、各ステップを説明します。

DB2 .NET Data Provider を使用したアプリケーションからのデータベースへの接続

DB2 .NET Data Provider を使用する場合、データベース接続は、DB2Connection クラスを通して確立されます。まず、接続パラメーターを保管するストリングを作成する必要があります。

以下は、考えられる接続ストリングの例です。

- ```
  String connectString = "Database=SAMPLE";
  // When used, attempts to connect to the SAMPLE database.
  ```
- ```
  String connectString = "Server=srv:50000;Database=SAMPLE;UID=db2adm;PWD=ab1cd";
  // When used, attempts to connect to the SAMPLE database on the server
  // 'srv' through port 50000 using 'db2adm' and 'ab1cd' as the user id and
  // password respectively.
  ```

データベース接続を作成するには、 DB2Connection コンストラクターに connectString を渡します。その後、DB2Connection オブジェクトの Open メソッドを使用して、 connectString で識別されたデータベースに正式に接続します。

C# でデータベースに接続する

```
String connectString = "Database=SAMPLE";
DB2Connection conn = new DB2Connection(connectString);
conn.Open();
return conn;
```

Visual Basic .NET でデータベースに接続する

```
Dim connectString As String = "Database=SAMPLE"
Dim conn As DB2Connection = new DB2Connection(connectString)
conn.Open()
Return conn
```

DB2 .NET Data Provider を使用したアプリケーションからの SQL ステートメントの実行

DB2 .NET Data Provider を使用する場合、 SQL ステートメントの実行は、DB2Command クラスを通し、このクラスのメソッド ExecuteReader() および ExecuteNonQuery() と、このクラスのプロパティー CommandText、CommandType、および Transaction を使用して行います。出力を生成する SQL ステートメントの場合は ExecuteReader() メソッドを使用する必要があり、その結果は DB2DataReader オブジェクトから検索できます。他のすべての SQL ステートメントの場合は、ExecuteNonQuery() メソッドを使用する必要があります。 DB2Command オブジェク

トの Transaction プロパティーは、 DB2Transaction に初期設定してください。DB2Transaction オブジェクトは、データベース・トランザクションのロールバックとコミットを担当します。

C# で UPDATE ステートメントを実行する

```
// assume a DB2Connection conn
DB2Command cmd = conn.CreateCommand();
DB2Transaction trans = conn.BeginTransaction();
cmd.Transaction = trans;
cmd.CommandText = "UPDATE staff " +
                  "  SET salary = (SELECT MIN(salary) " +
                  "                  FROM staff " +
    "                  WHERE id >= 310) " +
                  "  WHERE id = 310";
cmd.ExecuteNonQuery();
```

Visual Basic .NET で UPDATE ステートメントを実行する

```
' assume a DB2Connection conn
DB2Command cmd = conn.CreateCommand();
DB2Transaction trans = conn.BeginTransaction();
cmd.Transaction = trans;
cmd.CommandText = "UPDATE staff " +
                  "  SET salary = (SELECT MIN(salary) " +
                  "                  FROM staff " +
                  "                  WHERE id >= 310) " +
                  "  WHERE id = 310";
cmd.ExecuteNonQuery();
```

C# で SELECT ステートメントを実行する

```
// assume a DB2Connection conn
DB2Command cmd = conn.CreateCommand();
DB2Transaction trans = conn.BeginTransaction();
cmd.Transaction = trans;
cmd.CommandText = "SELECT deptnumb, location " +
                  "  FROM org " +
                  "  WHERE deptnumb < 25";
DB2DataReader reader = cmd.ExecuteReader();
```

Visual Basic .NET で SELECT ステートメントを実行する

```
' assume a DB2Connection conn
Dim cmd As DB2Command = conn.CreateCommand()
Dim trans As DB2Transaction = conn.BeginTransaction()
cmd.Transaction = trans
cmd.CommandText = "UPDATE staff " +
                  "  SET salary = (SELECT MIN(salary) " +
                  "                  FROM staff " +
                  "                  WHERE id >= 310) " +
                  "  WHERE id = 310"
cmd.ExecuteNonQuery()
```

アプリケーションがデータベース・トランザクションを実行したら、それをロールバックするか、コミットする必要があります。これは、 DB2Transaction オブジェクトのメソッド Commit() および Rollback() を通して行います。

C# でトランザクションをロールバックまたはコミットする

```
// assume a DB2Transaction object conn
trans.Rollback();
...
trans.Commit();
```

C# でトランザクションをロールバックまたはコミットする

```
' assume a DB2Transaction object conn
trans.Rollback()
...
trans.Commit()
```

DB2 .NET Data Provider を使用したアプリケーションからの結果セットの読み取り

DB2 .NET Data Provider を使用する場合、結果セットの読み取りは、DB2DataReader オブジェクトを通して行います。結果セットの次の行に進むためには、 DB2DataReader メソッドの Read() を使用します。出力の個々の列からのデータの抽出には、メソッド GetString()、GetInt32()、 GetDecimal()、および他のすべての使用可能データ・タイプ用のメソッドを使用します。 DB2DataReader をクローズするためには、 DB2DataReader の Close() メソッドを使用します。出力の読み取りを終了する際には、必ずこれを行います。

C# で結果セットを読み取る

```
// assume a DB2DataReader reader
Int16 deptnum = 0;
String location="";

// Output the results of the query
while(reader.Read())
{
  deptnum = reader.GetInt16(0);
  location = reader.GetString(1);
  Console.WriteLine("    " + deptnum + " " + location);
}
reader.Close();
```

Visual Basic .NET で結果セットを読み取る

```
' assume a DB2DataReader reader
Dim deptnum As Int16 = 0
Dim location As String ""

' Output the results of the query
Do While (reader.Read())
  deptnum = reader.GetInt16(0)
  location = reader.GetString(1)
  Console.WriteLine("    " & deptnum & " " & location)
Loop
reader.Close();
```

DB2 .NET Data Provider を使用したアプリケーションからのストアード・プロシージャーの呼び出し

DB2 .NET Data Provider を使用する場合は、 DB2Command オブジェクトを使用することによって、ストアード・プロシージャーを呼び出せます。 CommandType プロパティーのデフォルト値は、CommandType.Text です。これは SQL ステートメントに適した値であり、ストアード・プロシージャーを呼び出すためにも使用できます。ただし、CommandType に CommandType.StoredProcedure を設定したほうが、ストアード・プロシージャーの呼び出しはより容易になります。この場合は、ストアード・プロシージャー名とパラメーターのみを指定する必要があります。

以下の例は、CommandType プロパティーに CommandType.StoredProcedure または CommandType.Text が指定された、INOUT_PARAM というストアード・プロシージャーを呼び出す方法を示しています。

C# で DB2Command の CommandType プロパティーに CommandType.StoredProcedure を設定することによって、ストアード・プロシージャーを呼び出す

```
// assume a DB2Connection comm
DB2Transaction trans = conn.BeginTransaction();
DB2Command cmd = conn.CreateCommand();
String procName = "INOUT_PARAM";
cmd.Transaction = trans;
cmd.CommandType = CommandType.StoredProcedure;
cmd.CommandText = procName;

// Register input-output and output parameters for the DB2Command
...

// Call the stored procedure
Console.WriteLine("  Call stored procedure named " + procName);
cmd.ExecuteNonQuery();
```

C# で DB2Command の CommandType プロパティーに CommandType.Text を設定することによって、ストアード・プロシージャーを呼び出す

```
// assume a DB2Connection comm
DB2Transaction trans = conn.BeginTransaction();
DB2Command cmd = conn.CreateCommand();
String procName = "INOUT_PARAM";
String procCall = "CALL INOUT_PARAM (?, ?, ?)";
cmd.Transaction = trans;
cmd.CommandType = CommandType.Text;
cmd.CommandText = procCall;

// Register input-output and output parameters for the DB2Command
...

// Call the stored procedure
Console.WriteLine("  Call stored procedure named " + procName);
cmd.ExecuteNonQuery();
```

Visual Basic .NET で DB2Command の CommandType プロパティーに CommandType.StoredProcedure を設定することによって、ストアード・プロシージャーを呼び出す

```
' assume a DB2DataReader reader
Dim trans As DB2Transaction = conn.BeginTransaction()
Dim cmd As DB2Command = conn.CreateCommand()
Dim procName As String = "INOUT_PARAM"
cmd.Transaction = trans
cmd.CommandType = CommandType.StoredProcedure
cmd.CommandText = procName

' Register input-output and output parameters for the DB2Command
...

' Call the stored procedure
Console.WriteLine("  Call stored procedure named " & procName)
cmd.ExecuteNonQuery()
```

Visual Basic .NET で DB2Command の CommandType プロパティーに CommandType.Text を設定することによって、ストアード・プロシージャーを呼び出す

```
' assume a DB2DataReader reader
Dim trans As DB2Transaction = conn.BeginTransaction()
Dim cmd As DB2Command = conn.CreateCommand()
Dim procName As String = "INOUT_PARAM"
Dim procCall As String = "CALL INOUT_PARAM (?, ?, ?)"
cmd.Transaction = trans
cmd.CommandType = CommandType.Text
cmd.CommandText = procCall

' Register input-output and output parameters for the DB2Command
...

' Call the stored procedure
Console.WriteLine("  Call stored procedure named " & procName)
cmd.ExecuteNonQuery()
```

DB2 .NET Data Provider でサポートされている SQL データ・タイプ

以下の表には、DB2 .NET Data Provider の DB2Type データ・タイプと DB2 データ・タイプ、および対応する .NET Framework のデータ・タイプのマッピングをリストしています。

表 17. DB2 データ・タイプと .NET データ・タイプのマッピング

DB2Type Enum	DB2 データ・タイプ	.NET データ・タイプ
SmallInt	SMALLINT	Int16
Integer	INTEGER	Int32
BigInt	BIGINT	Int64
Real	REAL	Single
Double	DOUBLE PRECISION	Double
Float	FLOAT	Double
Decimal	DECIMAL	Decimal
Numeric	DECIMAL	Decimal
Date	DATE	DateTime
Time	TIME	TimeSpan
Timestamp	TIMESTAMP	DateTime
Char	CHAR	String
VarChar	VARCHAR	String
LongVarChar(1)	LONG VARCHAR	String
Binary	CHAR FOR BIT DATA	Byte[]
VarBinary	VARCHAR FOR BIT DATA	Byte[]
LongVarBinary(1)	LONG VARCHAR FOR BIT DATA	Byte[]
Graphic	GRAPHIC	String
VarGraphic	VARGRAPHIC	String
LongVarGraphic(1)	LONG GRAPHIC	String
Clob	CLOB	String
Blob	BLOB	Byte[]
DbClob	DBCLOB(N)	String

注:

1. これらのデータ・タイプは、DB2 .NET 共通言語ランタイム・ルーチンではサポートされていません。これらは、クライアント・アプリケーションでのみサポートされます。

注: dbinfo 構造は、CLR 関数およびプロシージャーにパラメーターとして渡されます。 CLR UDF のスクラッチパッドと呼び出しタイプも、CLR ルーチンにパラメーターとして渡されます。これらのパラメーターに対する適切な CLR データ・タイプについては、以下の関連トピックを参照してください。

- CLR ルーチンのパラメーター (Parameters in CLR routines)

関連概念:

- 「アプリケーション開発ガイド サーバー・アプリケーションのプログラミング」の『外部ルーチン用のパラメーター・スタイル 』
- 「アプリケーション開発ガイド サーバー・アプリケーションのプログラミング」の『共通言語ランタイム (CLR) ルーチン』
- 「アプリケーション開発ガイド サーバー・アプリケーションのプログラミング」の『CLR ルーチンのパラメーター』

関連タスク:

- 「アプリケーション開発ガイド サーバー・アプリケーションのプログラミング」の『構造化型パラメーターを外部ルーチンに渡す』
- 「アプリケーション開発ガイド サーバー・アプリケーションのプログラミング」の『CLR ルーチンの作成』
- 「アプリケーション開発ガイド サーバー・アプリケーションのプログラミング」の『C# の CLR ユーザー定義関数の例』
- 「アプリケーション開発ガイド サーバー・アプリケーションのプログラミング」の『C# の CLR プロシージャーの例』

関連サンプル:

- 『SpCreate.db2 -- Creates the external procedures implemented in spserver.cs』
- 『SpServer.cs -- C# external code implementation of procedures created in spcat.db2』
- 『SpCreate.db2 -- Creates the external procedures implemented in spserver.vb』
- 『SpServer.vb -- VB.NET implementation of procedures created in SpCat.db2』

第 11 章 IBM OLE DB Provider for DB2

IBM OLE DB Provider for DB2 の目的. 247
IBM OLE DB Provider for DB2 でサポートされているアプリケーション・タイプ 248
OLE DB サービス. 249
IBM OLE DB Provider でサポートされているスレッド・モデル. 249
IBM OLE DB Provider によるラージ・オブジェクトの操作 249
IBM OLE DB Provider でサポートされているスキーマ行セット. 249
IBM OLE DB Provider で自動的に使用可能になる OLE DB サービス 251
データ・サービス 251
IBM OLE DB Provider でサポートされているカーソル・モード. 252
DB2 と OLE DB の間のデータ・タイプ・マッピング. 252
OLE DB タイプから DB2 タイプにデータを設定するためのデータ変換. 253
DB2 タイプから OLE DB タイプにデータを設定するためのデータ変換. 255
IBM OLE DB Provider の制限. 256
IBM OLE DB Provider での OLE DB コンポーネントおよびインターフェースのサポート 257
IBM OLE DB Provider での OLE DB プロパティーのサポート 259
IBM OLE DB Provider によるデータ・ソースへの接続 262
ADO アプリケーション 263
ADO 接続ストリング・キーワード 263
Visual Basic ADO アプリケーションによるデータ・ソースへの接続 264
ADO アプリケーションにおける更新可能な両方向スクロール・カーソル. 264
ADO アプリケーションに関する制限 264
IBM OLE DB Provider での ADO メソッドおよびプロパティーのサポート 264
C および C++ アプリケーション. 268
C/C++ アプリケーションのコンパイルおよびリンクと IBM OLE DB Provider. 269
IBM OLE DB Provider による、C/C++ アプリケーションでのデータ・ソースへの接続 269
MTS および COM+ 分散トランザクション . . . 269
MTS および COM+ 分散トランザクションのサポートと IBM OLE DB Provider 269
DB2 Universal Database における、C/C++ アプリケーション用の MTS サポートの使用可能化 . 270

IBM OLE DB Provider for DB2 の目的

Microsoft® OLE DB は、さまざまな情報ソースに保管されているデータへの一様なアクセスをアプリケーションに提供する、 OLE/COM インターフェースのセットです。 OLE DB のアーキテクチャーでは、 OLE DB Consumer と OLE DB Provider を定義しています。 OLE DB Consumer は、 OLE DB インターフェースを使用するシステムまたはアプリケーションで、 OLE DB Provider は、OLE DB インターフェースを公開するコンポーネントです。

IBM® OLE DB Provider for DB2® を使用すれば、 DB2 は OLE DB Provider のリソース管理プログラムとして機能できます。このサポートにより、OLE DB2 ベースのアプリケーションは、OLE インターフェースを使用して DB2 データを抽出したり照会したりできます。 IBM OLE DB Provider for DB2 (Provider 名は IBMDADB2) を使用すれば、 OLE DB Consumer は DB2 Universal Database™ サーバー上のデータにアクセスできます。 DB2 Connect™ がインストールされていれば、これらの OLE DB Consumer は、DB2 for MVS™、 DB2 for VM/VSE、または SQL/400 などのホスト DBMS 上のデータにもアクセスできます。

IBM OLE DB Provider for DB2 には以下の機能が備わっています。

- OLE DB Provider 仕様のサポート・レベル 0。いくつかの付加的なレベル 1 インターフェースが含まれます。

- フリー・スレッド Provider のインプリメンテーション。アプリケーションは、1 つのスレッドで作成したコンポーネントを他の任意のスレッドで使用できます。
- DB2 エラー・メッセージを戻すエラー検索サービス。

IBM OLE DB Provider はクライアントに存在し、OLE DB 表関数 (これも DB2 UDB でサポートされる) とは異なるものであることに注意してください。

本書の以下の節では、IBM OLE DB Provider for DB2 固有の仕様について説明します。 Microsoft OLE DB 2.0 仕様の詳細については、 Microsoft OLE DB 2.0 Programmer's Reference and Data Access SDK, Microsoft Press を参照してください。

準拠バージョン:

IBM OLE DB Provider for DB2 は、Microsoft OLE DB 仕様のバージョン 2.7 に準拠しています。

システム要件:

サポートされている Windows® オペレーティング・システムについては、 IBM OLE DB Provider for DB2 Server に関するアナウンス・レターを参照してください。

IBM OLE DB Provider for DB2 をインストールするには、上記のサポートされているオペレーティング・システムのいずれかを実行している必要があります。また、DB2 Application Development Client と、Microsoft Data Access Components (MDAC) Version 2.7 以降 (本書の執筆時点では、 `http://www.microsoft.com/data` (日本語版につきましては、 http://www.microsoft.com/japan/products/ntserver/option_pack/download.htm をご覧ください。) から入手可能) をインストールする必要もあります。

関連資料:

- 257 ページの『IBM OLE DB Provider での OLE DB コンポーネントおよびインターフェースのサポート』

IBM OLE DB Provider for DB2 でサポートされているアプリケーション・タイプ

IBM® OLE DB Provider for DB2® を使用すれば、以下のタイプのアプリケーションを作成できます。

- ADO アプリケーション。以下のものが含まれます。
  - Microsoft® Visual Studio C++ アプリケーション
  - Microsoft Visual Basic アプリケーション
- OLE DB .NET Data Provider を使用する ADO.NET アプリケーション
- OLE DB インターフェースを使用して IBMDADB2 に直接アクセスする C/C++ アプリケーション。 Data Access Consumer Object が ATL COM AppWizard によって生成される ATL アプリケーションが含まれます。

OLE DB サービス

以下の節では、OLE DB サービスについて説明します。

IBM OLE DB Provider でサポートされているスレッド・モデル

IBM® OLE DB Provider for DB2® では、フリー・スレッド・モデルがサポートされています。このモデルでは、アプリケーションが、1 つのスレッドで作成したコンポーネントを他の任意のスレッドで使用できます。

IBM OLE DB Provider によるラージ・オブジェクトの操作

IBMDADB2 でデータをストレージ・オブジェクト (DBTYPE_IUNKNOWN) として取得したり設定したりするには、以下のように ISequentialStream インターフェースを使用します。

- ストレージ・オブジェクトをパラメーターにバインドするには、 DBBINDING 構造の DBOBJECT で、dwFlag フィールドに値 STGM_READ だけを含めることができます。 IBMDADB2 では、バインド済みオブジェクトの ISequentialStream インターフェースの Read メソッドが実行されます。
- ストレージ・オブジェクトからデータを取得するには、アプリケーションがそのストレージ・オブジェクトの ISequentialStream インターフェースで Read メソッドを実行する必要があります。
- データを取得したとき、長さ部分の値は実データの長さであり、IUnknown ポインターの長さではありません。

IBM OLE DB Provider でサポートされているスキーマ行セット

以下の表は、`IDBSchemaRowset` によってサポートされているスキーマ行セットを示しています。行セットのうちサポートされていない列は NULL に設定されることに注意してください。

表 18. IBM OLE DB Provider for DB2 でサポートされているスキーマ行セット

サポートされている GUID	サポートされている制限	サポートされている列	注
DBSCHEMA _COLUMN_PRIVILEGES	COLUMN_NAME TABLE_NAME TABLE_SCHEMA	COLUMN_NAME GRANTEE GRANTOR IS_GRANTABLE PRIVILEGE_TYPE TABLE_NAME TABLE_SCHEMA	
DB_SCHEMA_COLUMNS	COLUMN_NAME TABLE_NAME TABLE_SCHEMA	CHARACTER_MAXIMUM_LENGTH CHARACTER_OCTET_LENGTH COLUMN_DEFAULT COLUMN_FLAGS COLUMN_HASDEFAULT COLUMN_NAME DATA_TYPE DESCRIPTION IS_NULLABLE NUMERIC_PRECISION NUMERIC_SCALE ORDINAL_POSITION TABLE_NAME TABLE_SCHEMA	

表 18. IBM OLE DB Provider for DB2 でサポートされているスキーマ行セット (続き)

サポートされている GUID	サポートされている制限	サポートされている列	注
DBSCHEMA_FOREIGN_KEYS	FK_TABLE_NAME FK_TABLE_SCHEMA PK_TABLE_NAME PK_TABLE_SCHEMA	DEFERRABILITY DELETE_RULE FK_COLUMN_NAME FK_NAME FK_TABLE_NAME FK_TABLE_SCHEMA ORDINAL PK_COLUMN_NAME PK_NAME PK_TABLE_NAME PK_TABLE_SCHEMA UPDATE_RULE	PK_TABLE_NAME または FK_TABLE_NAME の制限のうち少なくとも 1 つを指定する必要があります。 “%” ワイルドカードは使用できません。
DBSCHEMA_INDEXES	TABLE_NAME TABLE_SCHEMA	CARDINALITY CLUSTERED COLLATION COLUMN_NAME INDEX_NAME INDEX_SCHEMA ORDINAL_POSITION PAGES TABLE_NAME TABLE_SCHEMA TYPE UNIQUE	ソート順序はサポートされていません。指定されているソート順序は無視されます。
DBSCHEMA_PRIMARY_KEYS	TABLE_NAME TABLE_SCHEMA	COLUMN_NAME ORDINAL PK_NAME TABLE_NAME TABLE_SCHEMA	少なくとも TABLE_NAME の制限を指定する必要があります。 “%” ワイルドカードは使用できません。
DBSCHEMA _PROCEDURE_PARAMETERS	PARAMETER_NAME PROCEDURE_NAME PROCEDURE_SCHEMA	CHARACTER_MAXIMUM_LENGTH CHARACTER_OCTET_LENGTH DATA_TYPE DESCRIPTION IS_NULLABLE NUMERIC_PRECISION NUMERIC_SCALE ORDINAL_POSITION PARAMETER_DEFAULT PARAMETER_HASDEFAULT PARAMETER_NAME PARAMETER_TYPE PROCEDURE_NAME PROCEDURE_SCHEMA TYPE_NAME	
DBSCHEMA_PROCEDURES	PROCEDURE_NAME PROCEDURE_SCHEMA	DESCRIPTION PROCEDURE_NAME PROCEDURE_SCHEMA PROCEDURE_TYPE	

表 18. IBM OLE DB Provider for DB2 でサポートされているスキーマ行セット (続き)

サポートされている GUID	サポートされている制限	サポートされている列	注
DBSCHEMA_PROVIDER_TYPES	DATA_TYPE BEST_MATCH	AUTO_UNIQUE_VALUE BEST_MATCH CASE_SENSITIVE CREATE_PARAMS COLUMN_SIZE DATA_TYPE FIXED_PREC_SCALE IS_FIXEDLENGTH IS_LONG IS_NULLABLE LITERAL_PREFIX LITERAL_SUFFIX LOCAL_TYPE_NAME MINIMUM_SCALE MAXIMUM_SCALE SEARCHABLE TYPE_NAME UNSIGNED_ATTRIBUTE	
DBSCHEMA_STATISTICS	TABLE_NAME TABLE_SCHEMA	CARDINALITY TABLE_NAME TABLE_SCHEMA	ソート順序はサポートされていません。指定されているソート順序は無視されます。
DBSCHEMA _TABLE_PRIVILEGES	TABLE_NAME TABLE_SCHEMA	GRANTEE GRANTOR IS_GRANTABLE PRIVILEGE_TYPE TABLE_NAME TABLE_SCHEMA	
DBSCHEMA_TABLES	TABLE_NAME TABLE_SCHEMA TABLE_TYPE	DESCRIPTION TABLE_NAME TABLE_SCHEMA TABLE_TYPE	

IBM OLE DB Provider で自動的に使用可能になる OLE DB サービス

デフォルトでは、IBM® OLE DB Provider for DB2® は、 Provider のクラス ID (CLSID) の下に DWORD 値が 0xFFFFFFFF であるレジストリー項目 OLEDB_SERVICES を追加することにより、すべての OLE DB サービスを自動的に使用可能にします。この値の意味は以下のとおりです。

表 19. OLE DB サービス

使用可能になっているサービス	DWORD 値
すべてのサービス (デフォルト)	0xFFFFFFFF
プールおよび AutoEnlistment を除くすべて	0xFFFFFFFC
クライアント・カーソルを除くすべて	0xFFFFFFFB
プール、エンリスト、およびカーソルを除くすべて	0xFFFFFFF8
サービスなし	0x000000000

データ・サービス

以下の節では、データ・サービスに関する考慮事項について説明します。

IBM OLE DB Provider でサポートされているカーソル・モード

IBM® OLE DB Provider for DB2® では、読み取り専用、順方向専用、読み取り専用の両方向スクロール・カーソル、および更新可能な両方向スクロール・カーソルがネイティブでサポートされています。

DB2 と OLE DB の間のデータ・タイプ・マッピング

IBM OLE DB Provider では、DB2 データ・タイプと OLE DB データ・タイプの間のデータ・タイプ・マッピングがサポートされています。以下の表は、サポートされているマッピングと使用可能な名前の完全なリストを示すことにより、列およびパラメーターのデータ・タイプを示しています。

表 20. DB2 データ・タイプと OLE DB データ・タイプの間のデータ・タイプ・マッピング

DB2 データ・タイプ	OLE DB データ・タイプ標識	OLE DB 標準タイプ名	DB2 固有の名前
SMALLINT	DBTYPE_I2	“DBTYPE_I2”	“SMALLINT”
INTEGER	DBTYPE_I4	“DBTYPE_I4”	“INTEGER” または “INT”
BIGINT	DBTYPE_I8	“DBTYPE_I8”	“BIGINT”
REAL	DBTYPE_R4	“DBTYPE_R4”	“REAL”
FLOAT	DBTYPE_R8	“DBTYPE_R8”	“FLOAT”
DOUBLE	DBTYPE_R8	“DBTYPE_R8″	“DOUBLE” または “DOUBLE PRECISION”
DECIMAL	DBTYPE_NUMERIC	“DBTYPE_NUMERIC”	“DEC” または “DECIMAL”
NUMERIC	DBTYPE_NUMERIC	“DBTYPE_NUMERIC”	“NUM” または “NUMERIC”
DATE	DBTYPE_DBDATE	“DBTYPE_DBDATE”	“DATE”
TIME	DBTYPE_DBTIME	“DBTYPE_DBTIME”	“TIME”
TIMESTAMP	DBTYPE_DBTIMESTAMP	“DBTYPE_DBTIMESTAMP”	“TIMESTAMP”
CHAR	DBTYPE_STR	“DBTYPE_CHAR”	“CHAR” または “CHARACTER”
VARCHAR	DBTYPE_STR	“DBTYPE_VARCHAR”	“VARCHAR”
LONG VARCHAR	DBTYPE_STR	“DBTYPE_LONGVARCHAR”	“LONG VARCHAR”
CLOB	DBTYPE_STR および DBCOLUMNFLAGS_ISLONG または DBPARAMFLAGS_ISLONG	“DBTYPE_CHAR” “DBTYPE_VARCHAR” “DBTYPE_LONGVARCHAR” および DBCOLUMNFLAGS_ISLONG または DBPARAMFLAGS_ISLONG	“CLOB”
GRAPHIC	DBTYPE_WSTR	“DBTYPE_WCHAR”	“GRAPHIC”
VARGRAPHIC	DBTYPE_WSTR	“DBTYPE_WVARCHAR”	“VARGRAPHIC”
LONG VARGRAPHIC	DBTYPE_WSTR	“DBTYPE_WLONGVARCHAR”	“LONG VARGRAPHIC”
DBCLOB	DBTYPE_WSTR および DBCOLUMNFLAGS_ISLONG または DBPARAMFLAGS_ISLONG	“DBTYPE_WCHAR” “DBTYPE_WVARCHAR” “DBTYPE_WLONGVARCHAR” および DBCOLUMNFLAGS_ISLONG または DBPARAMFLAGS_ISLONG	“DBCLOB”

表 20. DB2 データ・タイプと OLE DB データ・タイプの間のデータ・タイプ・マッピング (続き)

DB2 データ・タイプ	OLE DB データ・タイプ標識	OLE DB 標準タイプ名	DB2 固有の名前
CHAR(n) FOR BIT DATA	DBTYPE_BYTES	“DBTYPE_BINARY”	
VARCHAR(n) FOR BIT DATA	DBTYPE_BYTES	“DBTYPE_VARBINARY”	
LONG VARCHAR FOR BIT DATA	DBTYPE_BYTES	“DBTYPE_LONGVARBINARY”	
BLOB	DBTYPE_BYTES および DBCOLUMNFLAGS_ISLONG または DBPARAMFLAGS_ISLONG	“DBTYPE_BINARY” “DBTYPE_VARBINARY” “DBTYPE_LONGVARBINARY” および DBCOLUMNFLAGS_ISLONG または DBPARAMFLAGS_ISLONG	“BLOB”
DATA LINK	DBTYPE_STR	“DBTYPE_CHAR”	“DATA LINK”

OLE DB タイプから DB2 タイプにデータを設定するためのデータ変換

IBM OLE DB Provider では、OLE DB タイプから DB2 タイプにデータを設定するためのデータ変換がサポートされています。データのタイプや値に応じて、一部のケースではデータの切り捨てが起こる場合があることに注意してください。

表 21. OLE DB タイプから DB2 タイプへのデータ変換

	DB2 データ・タイプ																					
																		ビット・データの場合				
OLE DB タイプ標識	SMALLINT	INTEGER	BIGINT	REAL	FLOAT DOUBLE	DECIMAL NUMERIC	DATE	TIME	TIMESTAMP	CHAR	VARCHAR	LONG VARCHAR	CLOB	GRAPHIC	VARGRAPHIC	LONG VARGRAPHIC	DBCLOB	CHAR	VARCHAR	LONG VARCHAR	BLOB	DATALINK
DBTYPE_EMPTY																						
DBTYPE_NULL																						
DBTYPE_RESERVED																						
DBTYPE_I1	X	X	X	X	X	X				X	X											
DBTYPE_I2	X	X	X	X	X	X				X	X											
DBTYPE_I4	X	X	X	X	X	X				X	X											
DBTYPE_I8	X	X	X	X	X	X				X	X											
DBTYPE_UI1	X	X	X	X	X	X				X	X											
DBTYPE_UI2	X	X	X	X	X	X				X	X											

表 21. OLE DB タイプから DB2 タイプへのデータ変換 (続き)

OLE DB タイプ標識	DB2 データ・タイプ																					
																		ビット・データの場合				
	SMALLINT	INTEGER	BIGINT	REAL	FLOAT DOUBLE	DECIMAL NUMERIC	DATE	TIME	TIMESTAMP	CHAR	VARCHAR	LONG VARCHAR	CLOB	GRAPHIC	VARGRAPHIC	LONG VARGRAPHIC	DBCLOB	CHAR	VARCHAR	LONG VARCHAR	BLOB	DATALINK
DBTYPE_UI4	X	X	X	X	X	X				X	X											
DBTYPE_UI8	X	X	X	X	X	X				X	X											
DBTYPE_R4	X	X	X	X	X	X				X	X											
DBTYPE_R8	X	X	X	X	X	X				X	X											
DBTYPE_CY																						
DBTYPE_DECIMAL	X	X	X	X	X	X				X	X											
DBTYPE_NUMERIC	X	X	X	X	X	X				X	X											
DBTYPE_DATE																						
DBTYPE_BOOL	X	X	X	X	X	X				X	X											
DBTYPE_BYTES			X			X				X	X	X				X		X	X	X		
DBTYPE_BSTR - 今後決定されます																						
DBTYPE_STR	X	X	X	X	X	X	X	X	X	X	X	X		X	X	X		X	X	X		X
DBTYPE_WSTR														X	X	X						
DBTYPE_VARIANT - 今後決定されます																						
DBTYPE_IDISPATCH																						
DBTYPE_IUNKNOWN										X	X	X	X	X	X	X	X	X	X	X		
DBTYPE_GUID																						
DBTYPE_ERROR																						
DBTYPE_BYREF																						
DBTYPE_ARRAY																						
DBTYPE_VECTOR																						
DBTYPE_UDT																						
DBTYPE_DBDATE							X		X	X	X											
DBTYPE_DBTIME								X	X	X	X											
DBTYPE_DBTIMESTAMP							X	X	X	X	X											
DBTYPE_FILETIME																						
DBTYPE_PROP_VARIANT																						
DBTYPE_HCHAPTER																						
DBTYPE_VARNUMERIC																						

関連資料:

- 255 ページの『DB2 タイプから OLE DB タイプにデータを設定するためのデータ変換』

DB2 タイプから OLE DB タイプにデータを設定するためのデータ変換

データを取得するため、IBM OLE DB Provider では DB2 タイプから OLE DB タイプへのデータ変換を行えます。データのタイプや値に応じて、一部のケースではデータの切り捨てが起こる場合があることに注意してください。

表 22. DB2 タイプから OLE DB タイプへのデータ変換

	DB2 データ・タイプ																					
																		ビット・データの場合				
OLE DB タイプ標識	SMALLINT	INTEGER	BIGINT	REAL	FLOATDOUBLE	DECIMALNUMERIC	DATE	TIME	TIMESTAMP	CHAR	VARCHAR	LONGVARCHAR	CLOB	GRAPHIC	VARGRAPHIC	LONGVARGRAPHIC	DBCLOB	CHAR	VARCHAR	LONGVARCHAR	BLOB	DATALINK
DBTYPE_EMPTY																						
DBTYPE_NULL																						
DBTYPE_RESERVED																						
DBTYPE_I1	X	X		X	X	X				X	X	X		X	X	X		X	X	X		X
DBTYPE_I2	X	X		X	X	X				X	X	X		X	X	X		X	X	X		X
DBTYPE_I4	X	X		X	X	X				X	X	X		X	X	X		X	X	X		X
DBTYPE_I8	X	X	X	X	X	X				X	X	X		X	X	X		X	X	X		X
DBTYPE_UI1	X	X		X	X	X				X	X	X		X	X	X		X	X	X		X
DBTYPE_UI2	X	X		X	X	X				X	X	X		X	X	X		X	X	X		X
DBTYPE_UI4	X	X		X	X	X				X	X	X		X	X	X		X	X	X		X
DBTYPE_UI8	X	X	X	X	X	X				X	X	X		X	X	X		X	X	X		X
DBTYPE_R4	X	X		X	X	X				X	X	X		X	X	X		X	X	X		X
DBTYPE_R8	X	X		X	X	X				X	X	X		X	X	X		X	X	X		X
DBTYPE_CY	X	X		X	X	X				X	X	X		X	X	X		X	X	X		X
DBTYPE_DECIMAL	X	X		X	X	X				X	X	X		X	X	X		X	X	X		X
DBTYPE_NUMERIC	X	X		X	X	X				X	X	X		X	X	X		X	X	X		X
DBTYPE_DATE	X	X		X	X		X	X	X	X	X	X		X	X	X						X
DBTYPE_BOOL	X	X		X	X	X				X	X	X		X	X	X		X	X	X		X
DBTYPE_BYTES	X	X		X	X	X	X	X	X	X	X	X		X	X	X		X	X	X		X
DBTYPE_BSTR	X	X	X	X	X	X	X	X	X	X	X	X		X	X	X		X	X	X		X
DBTYPE_STR	X	X	X	X	X	X	X	X	X	X	X	X		X	X	X		X	X	X		X

表 22. DB2 タイプから OLE DB タイプへのデータ変換 (続き)

OLE DB タイプ標識	DB2 データ・タイプ																					
	SMALLINT	INTEGER	BIGINT	REAL	FLOATDOUBLE	DECIMALNUMERIC	DATE	TIME	TIMESTAMP	CHAR	VARCHAR	LONGVARCHAR	CLOB	GRAPHIC	VARGRAPHIC	LONGVARGRAPHIC	DBCLOB	ビット・データの場合			BLOB	DATALINK
																		CHAR	VARCHAR	LONGVARCHAR		
DBTYPE_WSTR	X	X	X	X	X	X	X	X	X	X	X	X		X	X	X		X	X	X		X
DBTYPE_VARIANT	X	X	X	X	X	X	X	X	X	X	X	X		X	X	X		X	X	X		X
DBTYPE_IDISPATCH																						
DBTYPE_IUNKNOWN	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
DBTYPE_GUID										X	X	X		X	X	X		X	X	X		X
DBTYPE_ERROR																						
DBTYPE_BYREF																						
DBTYPE_ARRAY																						
DBTYPE_VECTOR																						
DBTYPE_UDT																						
DBTYPE_DBDATE							X	X	X	X	X	X		X	X	X		X	X	X		X
DBTYPE_DBTIME							X	X	X	X	X	X		X	X	X						X
DBTYPE_DBTIMESTAMP							X	X	X	X	X	X		X	X	X		X	X	X		X
DBTYPE_FILETIME			X				X	X	X	X	X	X		X	X	X		X	X	X		X
DBTYPE_PROP_VARIANT	X	X	X	X	X					X	X	X		X	X	X		X	X	X		X
DBTYPE_HCHAPTER																						
DBTYPE_VARNUMERIC																						

注: アプリケーションが ISequentialStream::Read を実行してストレージ・オブジェクトからデータを取得するとき、戻されるデータのフォーマットは列のデータ・タイプによって決まります。

- 文字でないデータおよびバイナリー・データの場合、列のデータはそのオペレーティング・システムでその値を表すバイトのシーケンスとして表されます。
- 文字データの場合、データは最初に DBTYPE_STR に変換されます。
- DBCLOB の場合、データは最初に DBTYPE_WCHAR に変換されます。

関連資料:

- 253 ページの『OLE DB タイプから DB2 タイプにデータを設定するためのデータ変換』

IBM OLE DB Provider の制限

IBM® OLE DB Provider の制限は以下のとおりです。

- IBMDADB2 では、ITransactionLocal インターフェースにより、自動コミットおよびユーザー制御のトランザクション有効範囲がサポートされています。デフォルトの有効範囲は自動コミット・トランザクション有効範囲です。ネストされたトランザクションはサポートされていません。
- RestartPosition は、コマンド・テキストにパラメーターが含まれている場合にサポートされません。
- IBMDADB2 では、DBID パラメーター (IOpenRowset インターフェースで使用されるパラメーター) を介して渡される表名が引用符で囲まれません。その代わりに、引用符が必要な場合に OLE DB Consumer で表名に引用符を追加する必要があります。

IBM OLE DB Provider での OLE DB コンポーネントおよびインターフェースのサポート

以下の表は、IBM OLE DB Provider および Microsoft OLE DB Provider for ODBC でサポートされている OLE DB コンポーネントおよびインターフェースを示しています。

表 23. IBM OLE DB Provider for DB2 および Microsoft OLE DB Provider for ODBC でサポートされている OLE DB コンポーネントおよびインターフェースの比較

	インターフェース	DB2	ODBC Provider
BLOB			
	ISequentialStream	はい	はい
コマンド			
	IAccessor	はい	はい
	ICommand	はい	はい
	ICommandPersist	いいえ	いいえ
	ICommandPrepare	はい	はい
	ICommandProperties	はい	はい
	ICommandText	はい	はい
	ICommandWithParameters	はい	はい
	IColumnsInfo	はい	はい
	IColumnsRowset	はい	はい
	IConvertType	はい	はい
	ISupportErrorInfo	はい	はい
データ・ソース			
	IConnectionPoint	いいえ	はい
	IDBAsynchNotify (Consumer)	いいえ	いいえ
	IDBAsynchStatus	いいえ	いいえ
	IDBConnectionPointContainer	いいえ	はい
	IDBCreateSession	はい	はい
	IDBDataSourceAdmin	いいえ	いいえ
	IDBInfo	はい	はい
	IDBInitialize	はい	はい

表 23. IBM OLE DB Provider for DB2 および Microsoft OLE DB Provider for ODBC でサポートされている OLE DB コンポーネントおよびインターフェースの比較 (続き)

	インターフェース	DB2	ODBC Provider
	IDBProperties	はい	はい
	IPersist	はい	いいえ
	IPersistFile	はい	はい
	ISupportErrorInfo	はい	はい
列挙型定数			
	IDBInitialize	はい	はい
	IDBProperties	はい	はい
	IParseDisplayName	はい	いいえ
	ISourcesRowset	はい	はい
	ISupportErrorInfo	はい	はい
エラー検索サービス			
	IErrorLookUp	はい	はい
エラー・オブジェクト			
	IErrorInfo	はい	いいえ
	IErrorRecords	はい	いいえ
	ISQLErrorInfo (カスタム)	はい	いいえ
複数結果			
	IMultipleResults	はい	はい
	ISupportErrorInfo	はい	はい
行セット			
	IAccessor	はい	はい
	IColumnsRowset	はい	はい
	IColumnsInfo	はい	はい
	IConvertType	はい	はい
	IChapteredRowset	いいえ	いいえ
	IConnectionPointContainer	はい	はい
	IDBAsynchStatus	いいえ	いいえ
	IParentRowset	いいえ	いいえ
	IRowset	はい	はい
	IRowsetChange	はい	はい
	IRowsetChapterMember	いいえ	いいえ
	IRowsetFind	いいえ	いいえ
	IRowsetIdentity	はい	はい
	IRowsetIndex	いいえ	いいえ
	IRowsetInfo	はい	はい
	IRowsetLocate	はい	はい
	IRowsetNotify (Consumer)	はい	いいえ
	IRowsetRefresh	Cursor Service Component	はい
	IRowsetResynch	Cursor Service Component	はい

表 23. *IBM OLE DB Provider for DB2* および *Microsoft OLE DB Provider for ODBC* でサポートされている *OLE DB* コンポーネントおよびインターフェースの比較 *(続き)*

	インターフェース	DB2	ODBC Provider
	IRowsetScroll	はい[1]	はい
	IRowsetUpdate	Cursor Service Component	はい
	IRowsetView	いいえ	いいえ
	ISupportErrorInfo	はい	はい
注:			
1. 戻される値は近似値です。削除された行はスキップされません。			
セッション			
	IAlterIndex	いいえ	いいえ
	IAlterTable	いいえ	いいえ
	IDBCreateCommand	はい	はい
	IDBSchemaRowset	はい	はい
	IGetDataSource	はい	はい
	IIndexDefinition	いいえ	いいえ
	IOpenRowset	はい	はい
	ISessionProperties	はい	はい
	ISupportErrorInfo	はい	はい
	ITableDefinition	いいえ	いいえ
	ITableDefinitionWithConstraints	いいえ	いいえ
	ITransaction	はい	はい
	ITransactionJoin	はい	はい
	ITransactionLocal	はい	はい
	ITransactionObject	いいえ	いいえ
	ITransactionOptions	いいえ	はい
ビュー・オブジェクト			
	IViewChapter	いいえ	いいえ
	IViewFilter	いいえ	いいえ
	IViewRowset	いいえ	いいえ
	IViewSort	いいえ	いいえ

IBM OLE DB Provider での OLE DB プロパティーのサポート

以下の表は、IBM OLE DB Provider でサポートされている OLE DB プロパティーを示しています。

表 24. *IBM OLE DB Provider for DB2* でサポートされているプロパティー

プロパティー・グループ	プロパティー・セット	プロパティー	デフォルト値	R/W (読み取り/書き込み)
データ・ソース	DBPROPSET_DATASOURCE	DBPROP_MULTIPLECONNECTIONS	VARIANT_FALSE	R

表 24. *IBM OLE DB Provider for DB2* でサポートされているプロパティー *(続き)*

プロパティー・グループ	プロパティー・セット	プロパティー	デフォルト値	R/W (読み取り/書き込み)
		DBPROP_RESETDATASOURCE	DBPROPVAL_RD_RESETALL	R/W
データ・ソース情報	DBPROPSET_DATASOURCEINFO	DBPROP_ACTIVESESSIONS	0	R
		DBPROP_ASYNCTXNABORT	VARIANT_FALSE	R
		DBPROP_ASYNCTXNCOMMIT	VARIANT_FALSE	R
		DBPROP_BYREFACCESSORS	VARIANT_FALSE	R
		DBPROP_COLUMNDEFINITION	DBPROPVAL_CD_NOTNULL	R
		DBPROP_CONCATNULLBEHAVIOR	DBPROPVAL_CB_NULL	R
		DBPROP_CONNECTIONSTATUS	DBPROPVAL_CS_INITIALIZED	R
		DBPROP_DATASOURCENAME	なし	R
		DBPROP_DATASOURCEREADONLY	VARIANT_FALSE	R
		DBPROP_DBMSNAME	なし	R
		DBPROP_DBMSVER	なし	R
		DBPROP_DSOTHREADMODEL	DBPROPVAL_RT_FREETHREAD	R
		DBPROP_GROUPBY	DBPROPVAL_GB_CONTAINS_SELECT	R
		DBPROP_IDENTIFIERCASE	DBPROPVAL_IC_UPPER	R
		DBPROP_MAXINDEXSIZE	0	R
		DBPROP_MAXROWSIZE	0	R
		DBPROP_MAXROWSIZEINCLUDESBLOB	VARIANT_TRUE	R
		DBPROP_MAXTABLEINSELECT	0	R
		DBPROP_MULTIPLEPARAMSETS	VARIANT_FALSE	R
		DBPROP_MULTIPLERESULTS	DBPROPVAL_MR_SUPPORTED	R
		DBPROP_MULTIPLESTORAGEOBJECTS	VARIANT_TRUE	R
		DBPROP_MULTITABLEUPDATE	VARIANT_FALSE	R
		DBPROP_NULLCOLLATION	DBPROPVAL_NC_LOW	R
		DBPROP_OLEOBJECTS	DBPROPVAL_OO_BLOB	R
		DBPROP_ORDERBYCOLUMNSINSELECT	VARIANT_FALSE	R
		DBPROP_OUTPUTPARAMETERAVAILABILITY	DBPROPVAL_OA_ATEXECUTE	R
		DBPROP_PERSISTENTIDTYPE	DBPROPVAL_PT_NAME	R
		DBPROP_PREPAREABORTBEHAVIOR	DBPROPVAL_CB_DELETE	R
		DBPROP_PROCEDURETERM	“STORED PROCEDURE”	R
		DBPROP_PROVIDERFRIENDLYNAME	“IBM OLE DB Provider for DB2”	R
		DBPROP_PROVIDERNAME	“IBMDADB2.DLL”	R
		DBPROP_PROVIDEROLEDBVER	“02.7”	R
		DBPROP_PROVIDERVER	なし	R
		DBPROP_QUOTEIDENTIFIERCASE	DBPROPVAL_IC_SENSITIVE	R
		DBPROP_ROWSETCONVERSIONSONCOMMAND	VARIANT_TRUE	R
		DBPROP_SCHEMATERM	“SCHEMA”	R
		DBPROP_SCHEMAUSAGE	DBPROPVAL_SU_DML_STATEMENTS | DBPROPVAL_SU_TABLE_DEFINITION | DBPROPVAL_SU_INDEX_DEFINITION | DBPROPVAL_SU_PRIVILEGE_DEFINITION	R
		DBPROP_SQLSUPPORT	DBPROPVAL_SQL_ODBC_EXTENDED | DBPROPVAL_SQL_ESCAPECLAUSES | DBPROPVAL_SQL_ANSI92_ENTRY	R
		DBPROP_SERVERNAME	なし	R
		DBPROP_STRUCTUREDSTORAGE	DBPROPVAL_SS_ISEQUENTIALSTREAM	R
		DBPROP_SUBQUERIES	DBPROPVAL_SQ_CORRELATEDSUBQUERIES | DBPROPVAL_SQ_COMPARISON | DBPROPVAL_SQ_EXISTS | DBPROPVAL_SQ_IN | DBPROPVAL_SQ_QUANTIFIED |	R
		DBPROP_SUPPORTEDTXNDDL	DBPROPVAL_TC_ALL	R

表 24. *IBM OLE DB Provider for DB2* でサポートされているプロパティー *(続き)*

プロパティー・グループ	プロパティー・セット	プロパティー	デフォルト値	R/W (読み取り/書き込み)
		DBPROP_SUPPORTEDTXNISOLEVELS	DBPROPVAL_TI_CURSORSTABILITY | DBPROPVAL_TI_READCOMMITTED | DBPROPVAL_TI_READUNCOMMITTED | DBPROPVAL_TI_SERIALIZABLE |	R
		DBPROP_SUPPORTEDTXNISORETAIN	DBPROPVAL_TR_COMMIT_DC | DBPROPVAL_TR_ABORT_NO |	R
		DBPROP_TABLETERM	“TABLE”	R
		DBPROP_USERNAME	なし	R
初期化	DBPROPSET_DBINIT	DBPROP_AUTH_PASSWORD	なし	R/W
		DBPROP_AUTH_PERSIST_SENSITIVE_AUTHINFO	VARIANT_FALSE	R/W
		DBPROP_AUTH_USERID	なし	R/W
		DBPROP_INIT_DATASOURCE	なし	R/W
		DBPROP_INIT_HWND	なし	R/W
		DBPROP_INIT_MODE	DB_MODE_READWRITE	R/W
		DBPROP_INIT_OLEDBSERVICES	0xFFFFFFFF	R/W
		DBPROP_INIT_PROMPT	DBPROMPT_NOPROMPT	R/W
		DBPROP_INIT_PROVIDERSTRING	なし	R/W
行セット	DBPROPSET_ROWSET	DBPROP_ABORTPRESERVE	VARIANT_FALSE	R
		DBPROP_ACCESSORDER	DBPROPVAL_AO_RANDOM	R
		DBPROP_BLOCKINGSTORAGEOBJECTS	VARIANT_FALSE	R
		DBPROP_BOOKMARKS	VARIANT_FALSE	R/W
		DBPROP_BOOKMARKSKIPPED	VARIANT_FALSE	R
		DBPROP_BOOKMARKTYPE	DBPROPVAL_BMK_NUMERIC	R
		DBPROP_CACHEDEFERRED	VARIANT_FALSE	R/W
		DBPROP_CANFETCHBACKWARDS	VARIANT_FALSE	R/W
		DBPROP_CANHOLDROWS	VARIANT_FALSE	R
		DBPROP_CANSCROLLBACKWARDS	VARIANT_FALSE	R/W
		DBPROP_CHANGEINSERTEDROWS	VARIANT_FALSE	R
		DBPROP_COMMITPRESERVE	VARIANT_TRUE	R/W
		DBPROP_COMMANDTIMEOUT	0	R/W
		DBPROP_DEFERRED	VARIANT_FALSE	R
		DBPROP_IAccessor	VARIANT_TRUE	R
		DBPROP_IColumnsInfo	VARIANT_TRUE	R
		DBPROP_IColumnsRowset	VARIANT_TRUE	R/W
		DBPROP_IConvertType	VARIANT_TRUE	R
		DBPROP_IMultipleResults	VARIANT_FALSE	R/W
		DBPROP_IRowset	VARIANT_TRUE	R
		DBPROP_IRowChange	VARIANT_FALSE	R/W
		DBPROP_IRowsetFind	VARIANT_FALSE	R
		DBPROP_IRowsetIdentity	VARIANT_TRUE	R
		DBPROP_IRowsetInfo	VARIANT_TRUE	R
		DBPROP_IRowsetLocate	VARIANT_FALSE	R/W
		DBPROP_IRowsetScroll	VARIANT_FALSE	R/W
		DBPROP_IRowsetUpdate	VARIANT_FALSE	R
		DBPROP_ISequentialStream	VARIANT_TRUE	R
		DBPROP_ISupportErrorInfo	VARIANT_TRUE	R
		DBPROP_LITERALBOOKMARKS	VARIANT_FALSE	R
		DBPROP_LITERALIDENTITY	VARIANT_TRUE	R
		DBPROP_LOCKMODE	DBPROPVAL_LM_SINGLEROW	R/W
		DBPROP_MAXOPENROWS	32767	R
		DBPROP_MAXROWS	0	R/W
		DBPROP_NOTIFICATIONGRANULARITY	DBPROPVAL_NT_SINGLEROW	R/W

表 24. IBM OLE DB Provider for DB2 でサポートされているプロパティー (続き)

プロパティー・グループ	プロパティー・セット	プロパティー	デフォルト値	R/W (読み取り/書き込み)
		DBPROP_NOTIFICATION PHASES	DBPROPVAL_NP_OKTODO DBPROPBAL_NP_ABOUTTODO DBPROPVAL_NP_SYNCHAFTER DBPROPVAL_NP_FAILEDTODO DBPROPVAL_NP_DIDEVENT	R
		DBPROP_NOTIFYROWSETRELEASE	DBPROPVAL_NP_OKTODO DBPROPVAL_NP_ABOUTTODO	R
		DBPROP _NOTIFYROWSETFETCHPOSITIONCHANGE	DBPROPVAL_NP_OKTODO DBPROPVAL_NP_ABOUTTODO	R
		DBPROP_NOTIFYCOLUMNSET	DBPROPVAL_NP_OKTODO DBPROPVAL_NP_ABOUTTODO	R
		DBPROP_NOTIFYROWDELETE	DBPROPVAL_NP_OKTODO DBPROPVAL_NP_ABOUTTODO	R
		DBPROP_NOTIFYROWINSERT	DBPROPVAL_NP_OKTODO DBPROPVAL_NP_ABOUTTODO	R
		DBPROP_ORDEREDBOOKMARKS	VARIANT_FALSE	R
		DBPROP_OTHERINSERT	VARIANT_FALSE	R
		DBPROP_OTHERUPDATEDELETE	VARIANT_FALSE	R/W
		DBPROP_OWNINSERT	VARIANT_FALSE	R
		DBPROP_OWNUPDATEDELETE	VARIANT_FALSE	R
		DBPROP_QUICKRESTART	VARIANT_FALSE	R/W
		DBPROP_REMOVEDELETED	VARIANT_FALSE	R/W
		DBPROP_ROWTHREADMODEL	DBPROPVAL_RT_FREETHREAD	R
		DBPROP_SERVERCURSOR	VARIANT_TRUE	R
		DBPROP_SERVERDATAONINSERT	VARIANT_FALSE	R
		DBPROP_UNIQUEROWS	VARIANT_FALSE	R/W
		DBPROP_UPDATABILITY	0	R/W
行セット	DBPROPSET_DB2ROWSET	DBPROP_ISLONGMINLENGTH	32000	R/W
セッション	DBPROPSET_SESSION	DBPROP_SESS_AUTOCOMMITISOLEVELS	DBPROPVAL_TI_CURSORSTABILITY	R/W

IBM OLE DB Provider によるデータ・ソースへの接続

以下の例は、IBM® OLE DB Provider for DB2 を使用して DB2® データ・ソースに接続する方法を示しています。

例 1: ADO を使用する Visual Basic アプリケーション:

```
Dim db As ADODB.Connection
Set db = New ADODB.Connection
db.Provider = "IBMDADB2"
db.CursorLocation = adUseClient
...
```

例 2: IDBPromptInitialize とデータ・リンクを使用する C/C++ アプリケーション:

```
// Create DataLinks
hr = CoCreateInstance (
     CLSID_DataLinks,
     NULL,
     CLSCTX_INPROC_SERVER,
     IID_IDBPromptInitialize,
     (void**)&pIDBPromptInitialize);

// Invoke the DataLinks UI to select the provider and data source
hr = pIDBPromptInitialize->PromptDataSource (
     NULL,
```

```
     GetDesktopWindow(),
     DBPROMPTOPTIONS_PROPERTYSHEET,
     0,
     NULL,
     NULL,
     IID_IDBInitialize,
     (IUnknown**)&pIDBInitialize);
```

例 3: IDataInitialize と Service Component を使用する C/C++ アプリケーション:

```
hr = CoCreateInstance (
     CLSID_MSDAINITIALIZE,
     NULL,
     CLSCTX_INPROC_SERVER,
     IID_IDataInitialize,
     (void**)&pIDataInitialize);

hr = pIDataInitialize->CreateDBInstance(
     CLSID_IBMDADB2, // ClassID of IBMDADB2
     NULL,
     CLSCTX_INPROC_SERVER,
     NULL,
     IID_IDBInitialize,
     (IUnknown**)&pIDBInitialize);
```

ADO アプリケーション

以下の節では、ADO アプリケーションに関する考慮事項について説明します。

ADO 接続ストリング・キーワード

ADO (ActiveX Data Objects) 接続ストリング・キーワードを指定するには、Provider (接続) ストリングで keyword=*value* のフォーマットを使用してキーワードを指定します。複数のキーワードを指定する場合はセミコロン (;) で区切ります。

以下の表は、IBM® OLE DB Provider for DB2® でサポートされているキーワードを示しています。

表 25. IBM OLE DB Provider for DB2 でサポートされているキーワード

キーワード	値	意味
DSN	データベース別名の名前	データベース・ディレクトリーで使用される DB2 データベース別名。
UID	ユーザー ID	DB2 サーバーへの接続に使用するユーザー ID。
PWD	UID のパスワード	DB2 サーバーへの接続に使用するユーザー ID のパスワード。

他の DB2 CLI 構成キーワードも、IBM OLE DB Provider の動作に影響します。

関連資料:

- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 1 巻*」の『CLI/ODBC 構成キーワード (カテゴリー別)』

Visual Basic ADO アプリケーションによるデータ・ソースへの接続

IBM® OLE DB Provider for DB2 を使用して DB2® データ・ソースに接続するには、 IBMDADB2 Provider 名を使用します。

関連概念:

- 262 ページの『IBM OLE DB Provider によるデータ・ソースへの接続』

関連タスク:

- 「*アプリケーション開発ガイド アプリケーションの構築および実行*」の『Visual Basic による ADO アプリケーションの構築』

ADO アプリケーションにおける更新可能な両方向スクロール・カーソル

IBM® OLE DB Provider for DB2® では、読み取り専用、順方向専用、読み取り専用の両方向スクロール・カーソル、および更新可能な両方向スクロール・カーソルがネイティブでサポートされています。更新可能な両方向スクロール・カーソルにアクセスする ADO アプリケーションはカーソル位置を `adUseClient` または `adUseServer` に設定する必要があります。カーソル位置を `adUseServer` に設定すると、カーソルがサーバー上でマテリアライズします。

ADO アプリケーションに関する制限

以下に ADO アプリケーションの制限を示します。

- ストアード・プロシージャーを呼び出す ADO アプリケーションでは、事前にパラメーターが作成されて明示的にバインドされていなければなりません。パラメーターを自動生成するための `Parameters.Refresh` 方式は、DB2 Server for VSE & VM ではサポートされていません。
- デフォルトのパラメーター値はサポートされていません。
- サーバー・サイドの両方向スクロール・カーソルを使用して新しい行を挿入する場合、AddNew() 方式を Fieldlist および Values 引き数値を指定して使用します。これは、各列に対して Update() 呼び出しに続いて引き数なしの AddNew() を呼び出すより効率的です。 AddNew() および Update() の各呼び出しはサーバーに対する別個の要求です。したがって、これは AddNew() を単独で呼び出すより効率が悪くなります。
- 新規に挿入された行は、サーバー・サイドの両方向スクロール・カーソルでは更新できません。
- サーバー・サイドの両方向スクロール・カーソルを使用している場合には、長いデータ、LOB、または Datalink 列を持つ表は更新できません。

IBM OLE DB Provider での ADO メソッドおよびプロパティーのサポート

IBM OLE DB Provider は、以下の ADO メソッドおよびプロパティーをサポートしています。

表 26. *IBM OLE DB Provider for DB2* でサポートされている *ADO* メソッドおよびプロパティー

ADO	メソッド/プロパティー	OLE DB インターフェース/プロパティー	IBM OLE DB でのサポート
コマンド・メソッド	Cancel	ICommand	あり
	CreateParameter		あり
	Execute		あり
コマンド・プロパティー	ActiveConnection	(ADO 固有)	
	Command Text	ICommandText	あり
	Command Timeout	ICommandProperties::SetProperties DBPROP_COMMANDTIMEOUT	あり
	CommandType	(ADO 固有)	
	Prepared	ICommandPrepare	あり
	State	(ADO 固有)	
コマンド・コレクション	Parameters	ICommandWithParameter DBSCHEMA _PROCEDURE_PARAMETERS	あり
	Properties	ICommandProperties IDBProperties	あり
接続メソッド	BeginTrans CommitTrans RollbackTrans	ITransactionLocal	はい (ただし、非ネスト) はい (ただし、非ネスト) はい (ただし、非ネスト)
	Execute	ICommand IOpenRowset	あり
	Open	IDBCreateSession IDBInitialize	あり
	OpenSchema adSchemaColumnPrivileges adSchemaColumns adSchemaForeignKeys adSchemaIndexes adSchemaPrimaryKeys adSchemaProcedureParam adSchemaProcedures adSchemaProviderType adSchemaStatistics adSchemaTablePrivileges adSchemaTables	IDBSchemaRowset	 あり あり あり あり あり あり あり あり あり あり あり
	Cancel		あり
接続プロパティー	Attributes adXactCommitRetaining adXactRollbackRetaining	ITransactionLocal	 あり あり
	CommandTimeout	ICommandProperties DBPROP_COMMAND_TIMEOUT	あり
	ConnectionString	(ADO 固有)	
	ConnectionTimeout	IDBProperties DBPROP_INIT_TIMEOUT	なし

表 26. IBM OLE DB Provider for DB2 でサポートされている ADO メソッドおよびプロパティー (続き)

ADO	メソッド/プロパティー	OLE DB インターフェース/プロパティー	IBM OLE DB での サポート
	CursorLocation:		
	adUseClient	(OLE DB Cursor Service を使用)	あり
	adUseNone	(使用されず)	なし
	adUseServer		あり
	DefaultDataBase	IDBProperties DBPROP_CURRENTCATALOG	なし
	IsolationLevel	ITransactionLocal DBPROP_SESS _AUTOCOMMITISOLEVELS	あり
	Mode	IDBProperties	
	adModeRead	DBPROP_INIT_MODE	なし
	adModeReadWrite		あり
	adModeShareDenyNone		なし
	adModeShareDenyRead		なし
	adModeShareDenyWrite		なし
	adModeShareExclusive		なし
	adModeUnknown		なし
	adModeWrite		なし
	Provider	ISourceRowset::GetSourceRowset	あり
	State	(ADO 固有)	
	Version	(ADO 固有)	
接続コレクション	Errors	IErrorRecords	あり
	Properties	IDBProperties	あり
エラー・プロパティー	Description	IErrorRecords	あり
	NativeError		あり
	Number		あり
	Source		あり
	SQLState		あり
	HelpContext		なし
	HelpFile		なし
フィールド・メソッド	AppendChunk	ISequentialStream	あり
	GetChunk		あり
フィールド・プロパティー	Actual Size	IAccessor IRowset	あり
	Attributes	IColumnInfo	
	DataFormat		あり
	DefinedSize		あり
	Name		あり
	NumericScale		あり
	Precision		あり
	Type		あり
	OriginalValue	IRowsetUpdate	あり (Cursor Service)
	UnderlyingValue	IRowsetRefresh	あり (Cursor Service)
		IRowsetResynch	あり (Cursor Service)
	Value	IAccessor IRowset	あり

表 26. IBM OLE DB Provider for DB2 でサポートされている ADO メソッドおよびプロパティー (続き)

ADO	メソッド/プロパティー	OLE DB インターフェース/プロパティー	IBM OLE DB でのサポート
フィールド・コレクション	Properties	IDBProperties IRowsetInfo	あり
パラメーター・メソッド	AppendChunk	ISequentialStream	あり
	Attributes Direction Name NumericScale Precision Scale Size Type	ICommandWithParameter DBSCHEMA _PROCEDURE_PARAMETERS	 あり なし あり あり あり あり あり
	Value	IAccessor ICommand	あり
パラメーター・コレクション	Properties		あり
レコード・セット・メソッド	AddNew	IRowsetChange	あり
	Cancel		あり
	CancelBatch	IRowsetUpdate::Undo	あり (Cursor Service)
	CancelUpdate		あり (Cursor Service)
	Clone	IRowsetLocate	あり
	Close	IAccessor IRowset	あり
	CompareBookmarks		なし
	Delete	IRowsetChange	あり
	GetRows	IAccessor IRowset	あり
	Move	IRowset IRowsetLocate	あり
	MoveFirst	IRowset IRowsetLocate	あり
	MoveNext	IRowset IRowsetLocate	あり
	MoveLast	IRowsetLocate	あり
	MovePrevious	IRowsetLocate	あり
	NextRecordSet	IMultipleResults	あり
	Open	ICommand IOpenRowset	あり
	Requery	ICommand IOpenRowset	あり
	Resync	IRowsetRefresh	あり (Cursor Service)
	Supports	IRowsetInfo	あり
	Update UpdateBatch	IRowsetChange IRowsetUpdate	あり あり (Cursor Service)

表 26. IBM OLE DB Provider for DB2 でサポートされている ADO メソッドおよびプロパティー (続き)

ADO	メソッド/プロパティー	OLE DB インターフェース/プロパティー	IBM OLE DB での サポート
レコード・セット・プロパティー	AbsolutePage	IRowsetLocate IRowsetScroll	あり はい[1]
	AbsolutePosition	IRowsetLocate IRowsetScroll	あり はい[1]
	ActiveConnection	IDBCreateSession IDBInitialize	あり
	BOF	(ADO 固有)	
	Bookmark	IAccessor IRowsetLocate	あり
	CacheSize	IRowsetLocate 内の cRows IRowset	あり
	CursorType adOpenDynamic adOpenForwardOnly adOpenKeySet adOpenStatic	ICommandProperties	 なし あり あり あり
	EditMode	IRowsetUpdate	あり (Cursor Service)
	EOF	(ADO 固有)	
	Filter	IRowsetLocate IRowsetView IRowsetUpdate IViewChapter IViewFilter	なし
	LockType	ICommandProperties	あり
	MarshallOption		なし
	MaxRecords	ICommandProperties IOpenRowset	あり
	PageCount	IRowsetScroll	はい[1]
	PageSize	(ADO 固有)	
	Sort	(ADO 固有)	
	Source	(ADO 固有)	
	State	(ADO 固有)	
	Status	IRowsetUpdate	あり (Cursor Service)
注: 1. 戻される値は近似値です。削除された行はスキップされません。			
レコード・セット・コレクション	Fields	IColumnInfo	あり
	Properties	IDBProperties IRowsetInfo::GetProperties	あり

C および C++ アプリケーション

以下の節では、C および C++ アプリケーションに関する考慮事項について説明します。

C/C++ アプリケーションのコンパイルおよびリンクと IBM OLE DB Provider

定数 CLSID_IBMDADB2 を使用する C/C++ アプリケーションには、SQLLIB¥include ディレクトリーにある ibmdadb2.h ファイルを組み込む必要があります。これらのアプリケーションでは、include ステートメントの前に DBINITCONSTANTS を定義する必要があります。以下の例は、ステートメントの正しい順序を示しています。

```
#define DBINITCONSTANTS
#include "ibmdadb2.h"
```

IBM OLE DB Provider による、C/C++ アプリケーションでのデータ・ソースへの接続

C/C++ アプリケーションで IBM® OLE DB Provider for DB2 を使用して DB2® データ・ソースに接続するには、 2 つの OLE DB コア・インターフェース (IDBPromptInitialize および IDataInitialize) のいずれかを使用するか、 COM API CoCreateInstance を呼び出すことができます。 IDataInitialize インターフェースは OLE DB Service Component によって提供され、IDBPromptInitialize は Data Links Component によって提供されます。

関連概念:

- 262 ページの『IBM OLE DB Provider によるデータ・ソースへの接続』

関連タスク:

- 「*アプリケーション開発ガイド アプリケーションの構築および実行*」の『Visual C++ による ADO アプリケーションの構築』

MTS および COM+ 分散トランザクション

以下の節では、MTS および COM+ 分散トランザクションに関する考慮事項について説明します。

MTS および COM+ 分散トランザクションのサポートと IBM OLE DB Provider

Windows® NT 上の Microsoft® Transaction Server (MTS) 環境か Windows 2000 上の Component Services (COM+) 環境で実行されている OLE DB アプリケーションは、ITransactionJoin インターフェースを使用することにより、 複数の DB2® Universal Database、ホスト、および iSeries データベース・サーバーや MTS/COM+ 仕様に準拠する他のリソース・マネージャーとの間で行われる分散トランザクションに参加することができます。

前提条件:

IBM® OLE DB Provider for DB2 に備わっている MTS または COM+ 分散トランザクション・サポートを使用するには、ご使用のサーバーが以下の前提条件を満たしていることを確かめてください。

注: これらの要件は、DB2 クライアントがインストールされている Windows マシンにのみ適用されます。

- Windows NT® および MTS Version 2.0 (Microsoft Hotfix 0772 以降)

  MTS Version 2.0 for Windows NT は Windows NT 4.0 Option Pack の一部として入手できます。 Option Pack は以下のサイトからダウンロードできます。

  ```
  http://www.microsoft.com/ntserver/nts/downloads/recommended/NT4OptPk/
  ```

- Windows 2000 (Service Pack 3 以降)

関連概念:

- 709 ページの『トランザクション・マネージャーとしての Microsoft Transaction Server (MTS) および Microsoft Component Services (COM+)』
- 711 ページの『Microsoft Component Services (COM+) での疎結合サポート』

DB2 Universal Database における、C/C++ アプリケーション用の MTS サポートの使用可能化

C または C++ アプリケーションを MTS または COM+ トランザクション・モードで実行するには、 `DataLink` インターフェースを使用して IBMDADB2 データ・ソース・インスタンスを作成できます。また、`CoCreateInstance` を使用し、セッション・オブジェクトを入手し、 `JoinTransaction` を使用することもできます。詳しくは、C または C++ アプリケーションをデータ・ソースに接続する方法に関する説明を参照してください。

ADO アプリケーションを MTS または COM+ トランザクション・モードで実行するには、 C または C++ アプリケーションをデータ・ソースに接続する方法に関する説明を参照してください。

MTS または COM+ パッケージ内のコンポーネントをトランザクション・モードで使用するには、そのコンポーネントの Transactions プロパティーを以下の値のいずれかに設定します。

- 『Required』
- 『Required New』
- 『Supported』

これらの値については、MTS の資料を参照してください。

関連概念:

- 709 ページの『トランザクション・マネージャーとしての Microsoft Transaction Server (MTS) および Microsoft Component Services (COM+)』
- 711 ページの『Microsoft Component Services (COM+) での疎結合サポート』

第 12 章 OLE DB .NET Data Provider

OLE DB .NET Data Provider 271
OLE DB .NET Data Provider の制約事項 272
OLE DB .NET Data Provider アプリケーションでの接続プーリング 276
OLE DB .NET Data Provider アプリケーションの時刻列. 277
OLE DB .NET Data Provider アプリケーションの ADORecordset オブジェクト 278

OLE DB .NET Data Provider

OLE DB .NET Data Provider は、ConnectionString オブジェクト内では IBMDADB2 として示される IBM® DB2® OLE DB Driver を使用します。 OLE DB .NET Data Provider でサポートされている接続ストリング・キーワードは、 IBM OLE DB Provider for DB2 でサポートされている接続ストリング・キーワードと同じです。また、OLE DB .NET Data Provider には、IBM DB2 OLE DB Provider と同じ制約事項があります。 OLE DB .NET Data Provider に対しては追加の制約事項があり、それについては OLE DB .NET Data Provider の制約事項のトピックで説明しています。

OLE DB .NET Data Provider を使用するには、 .NET Framework バージョン 1.1 がインストールされている必要があります。

DB2 Universal Database™ for AS/400® and iSeries™ の場合は、サーバーに APAR ii13348 の修正を適用する必要があります。

以下は、OLE DB .NET Data Provider でサポートされているすべての接続キーワードです。

表 27. OLE DB .NET Data Provider の **ConnectionString** キーワード

キーワード	値	意味
PROVIDER	IBMDADB2	IBM OLE DB Provider for DB2 を指定します (必須)
DSN または データ・ソース	データベース別名	データベース・ディレクトリーにカタログされた DB2 データベース別名。
UID	user ID	DB2 サーバーへの接続に使用するユーザー ID
PWD	password	DB2 サーバーへの接続に使用するユーザー ID のパスワード

以下に、OleDbConnection を作成して SAMPLE データベースに接続する例を示します。

```
[Visual Basic .NET]
Dim con As New OleDbConnection("Provider=IBMDADB2;" +
    "Data Source=sample;UID=userid;PWD=password;")
con.Open()
```

```
[C#]
OleDbConnection con = new OleDbConnection("Provider=IBMDADB2;" +
                "Data Source=sample;UID=userid;PWD=password;" );
con.Open()
```

OLE DB .NET Data Provider の制約事項

以下の表は、IBM OLE DB .NET Data Provider の使用に関係した制約事項を示しています。

表 28. IBM OLE DB .NET Data Provider の制約事項

クラスまたはフィーチャー	制約事項の説明	影響を受ける DB2 サーバー
ASCII 文字ストリーム	DbType.AnsiString または DbType.AnsiStringFixedLength を使用している場合、OleDbParameters で ASCII 文字ストリームを使用することはできません。 OLE DB .NET Data Provider が次の例外を出します。 "Specified cast is not valid" **予備手段:** DbType.AnsiString または DbType.AnsiStringFixedLength の代わりに DbType.Binary を使用します。	すべて
ADORecord	ADORecord はサポートされていません。	すべて
ADORecordSet および Timestamp	MSDN で説明されているように、ADORecordSet の変化時間単位は、1 秒として解決されます。そのため、DB2 Timestamp 列が ADORecordSet に保管されるとき、小数点未満の秒数はすべて失われます。同様に、DataSet に ADORecordSet からデータを取り込んだ後は、 DataSet の Timestamp 列に、小数点未満の秒数は含まれません。 **予備手段:** この予備手段は、DB2 Universal Database for Linux, UNIX, and Windows バージョン 8.1 フィックスパック 4 以降にのみ有効です。小数点未満の秒数が失われないようにするためには、次の CLI キーワードを設定することができます。 MAPTIMESTAMPDESCRIBE = 2 このキーワードにより、Timestamp は WCHAR(26) で記述されます。このキーワードを設定するには、DB2 コマンド・ウィンドウから次のコマンドを実行します。 db2 update cli cfg for section common using MAPTIMESTAMPDESCRIBE 2	すべて
Chapters	Chapters はサポートされていません。	すべて
キー情報	OLE DB .NET Data Provider は、IDataReader を開くのと同時にキー情報を検索することはできません。	DB2 for VM/VSE

表 28. IBM OLE DB .NET Data Provider の制約事項 (続き)

クラスまたはフィーチャー	制約事項の説明	影響を受ける DB2 サーバー
ストアード・プロシージャーのキー情報	OLE DB .NET Data Provider は、 DB2 Universal Database for Linux, UNIX, and Windows からのみ、ストアード・プロシージャーによって戻された結果セットに関するキー情報を検索できます。なぜなら、Linux、UNIX、および Windows 以外のプラットフォーム用の DB2 サーバーは、ストアード・プロシージャーで開かれた結果セットの拡張記述情報を戻さないからです。 DB2 Universal Database for Linux, UNIX, and Windows 上でストアード・プロシージャーによって戻された結果セットのキー情報を検索するには、 DB2 サーバーに対して、次のレジストリー変数を設定する必要があります。 `db2set DB2_APM_PERFORMANCE=8` このサーバー・サイドの DB2 レジストリー変数を設定すると、サーバー上で結果セットのメタデータがより長い期間使用可能になり、 OLE DB が正常にキー情報を検索できるようになります。しかし、サーバーのワークロードによっては、 OLE DB Provider が情報を照会するのに十分な期間メタデータが使用可能にならないことがあります。したがって、ストアード・プロシージャーから戻された結果セットに関しては、キー情報を常に入手できるという保障はありません。 CALL ステートメントに関するキー情報を検索するためには、アプリケーションが CALL ステートメントを実行しなければなりません。`OleDbDataAdapter.FillSchema()` または `OleDbCommand.ExecuteReader(CommandBehavior.SchemaOnly | CommandBehavior.KeyInfo)` を呼び出しても、ストアード・プロシージャー呼び出しは実際には実行されません。したがって、ストアード・プロシージャーによって戻される結果セットのキー情報は検索されません。	すべて

バッチ SQL ステートメントからのキー情報	複数の結果を戻すバッチ SQL ステートメントを使用している場合、`FillSchema()` メソッドは、バッチ SQL ステートメント・リストの中の最初の SQL ステートメントについてのみ、スキーマ情報の検索を試行します。このステートメントが結果セットを戻さなければ、表は作成されません。たとえば、次のようになります。
	`[C#]`
	`cmd.CommandText = "INSERT INTO ORG(C1) VALUES(1000); SELECT C1 FROM ORG;";`
	`da = new OleDbDataAdapter(cmd);`
	`da.FillSchema(ds, SchemaType.Source);`
	
	バッチ SQL ステートメントの最初のステートメントが、結果セットを戻さない INSERT ステートメントなので、データ・セット内に表は作成されません。
		すべて

表 28. IBM OLE DB .NET Data Provider の制約事項 (続き)

クラスまたはフィーチャー	制約事項の説明	影響を受ける DB2 サーバー
OleDbCommandBuilder	SELECT ステートメントに以下のデータ・タイプの列が含まれている場合、OleDbCommandBuilder によって自動的に生成された UPDATE、DELETE、および INSERT ステートメントは正しくありません。 • CLOB • BLOB • DBCLOB • LONG VARCHAR • LONG VARCHAR FOR BIT DATA • LONG VARGRAPHIC DB2 Universal Database for Linux, Unix, and Windows 以外の DB2 サーバーに接続している場合は、以下のデータ・タイプの列も問題を引き起こします。 • VARCHAR[1] • VARCHAR FOR BIT DATA[1] • VARGRAPHIC[1] • REAL • FLOAT または DOUBLE • TIMESTAMP **注:** 1. これらのデータ・タイプの列は、254 バイトより大きい VARCHAR 値、 254 バイトより大きい VARCHAR 値 FOR BIT DATA、または 127 バイトより大きい VARGRAPHIC として定義されている場合は適切です。この条件は、DB2 Universal Database for Linux, Unix, and Windows 以外の DB2 サーバーに接続している場合にのみ有効です。 OleDbCommandBuilder は、 WHERE 文節の等価比較で選択されたすべての列を使用する SQL ステートメントを生成しますが、以前にリストされたデータ・タイプを等価比較に使用することはできません。 **注:** この制限は、OleDbCommandBuilder を利用して UPDATE、 DELETE、および INSERT ステートメントを自動生成する IDbDataAdapter.Update() メソッドに影響することにご注意ください。生成されたステートメントに、前にリストされたデータ・タイプがいずれか 1 つ含まれていると、 UPDATE 操作は失敗します。 **予備手段:** 前にリストされたデータ・タイプのすべての列を、生成された SQL ステートメントの WHERE 文節から明示的に除去する必要があります。自分で UPDATE、DELETE、および INSERT ステートメントをコーディングすることをお勧めします。	すべて
OleDbCommandBuilder.DeriveParameters	DeriveParameters() を使用する際は、大文字小文字の区別が重要です。OleDbCommand.CommandText で指定されたストアード・プロシージャー名は、 DB2 カタログ表に保管されている名前と同じケースでなければなりません。ストアード・プロシージャー名がどのように保管されているかを確認するには、プロシージャー名制限なしで OpenSchema(OleDbSchemaGuid.Procedures) を呼び出します。これは、すべてのストアード・プロシージャー名を戻します。デフォルトで、DB2 はストアード・プロシージャー名を大文字で保管します。したがって、ほとんどの場合、ストアード・プロシージャー名は大文字で指定する必要があります。	すべて

表 28. IBM OLE DB .NET Data Provider の制約事項 (続き)

クラスまたはフィーチャー	制約事項の説明	影響を受ける DB2 サーバー
OleDbCommandBuilder.DeriveParameters	OleDbCommandBuilder.DeriveParameters() メソッドは、生成される OleDbParameterCollection 内に ReturnValue パラメーターを含めません。SqlClient および DB2 .NET Data Provider はデフォルトで、生成される ParameterCollection に、 ParameterDirection.ReturnValue と共にパラメーターを追加します。	すべて
OleDbCommandBuilder.DeriveParameters	多重定義されたストアード・プロシージャーの場合、OleDbCommandBuilder.DeriveParameters() メソッドは失敗します。 MYPROC という名前の複数のストアード・プロシージャーがあり、それぞれが異なる数または異なるタイプのパラメーターを取る場合、 OleDbCommandBuilder.DeriveParameters() では、多重定義されているすべてのストアード・プロシージャーのすべてのパラメーターが検索されます。	すべて
OleDbCommandBuilder.DeriveParameters	アプリケーションがストアード・プロシージャーをスキーマで修飾していない場合、 DeriveParameters() では、そのプロシージャー名のすべてのパラメーターが戻されます。したがって、同一のプロシージャー名に対して複数のスキーマが存在する場合、 DeriveParameters() では、同じ名前を持つすべてのプロシージャーのすべてのパラメーターが戻されます。	すべて
OleDbConnection.ChangeDatabase	OleDbConnection.ChangeDatabase() メソッドはサポートされてません。	すべて
OleDbConnection.ConnectionString	接続ストリングの中で ¥b、¥a、または ¥O などの印刷不可文字を使用すると、例外が出されます。 以下のキーワードには、制約があります。 **Data Source** データ・ソースは、サーバーではなく、データベースの名前です。SERVER キーワードは指定できますが、IBMDADB2 プロバイダーからは無視されます。 **Initial Catalog および Connect Timeout** これらのキーワードはサポートされていません。通常、OLE DB .NET Data Provider は、認識されない非サポート・キーワードはすべて無視します。しかし、これらのキーワードを指定すると、次の例外が発生します。 `Multiple-step OLE DB operation generated errors. Check each OLE DB status value, if available. No work was done.` **ConnectionTimeout** ConnectionTimeout は読み取り専用です。	すべて

表 28. IBM OLE DB .NET Data Provider の制約事項 (続き)

クラスまたはフィーチャー	制約事項の説明	影響を受ける DB2 サーバー

OleDbConnection.GetOleDbSchemaTable	制限値は、大文字小文字の区別があり、システム・カタログ表に保管されているデータベース・オブジェクトとケースが一致している必要があります。デフォルトではこれは大文字です。
	
	たとえば、次のようにして表を作成したとします。
	`CREATE TABLE abc(c1 SMALLINT)`
	
	DB2 は表の名前を大文字 ("ABC") でシステム・カタログに保管します。そのため、"ABC" を制限値として使用する必要があります。たとえば、次のようにします。
	`schemaTable = con.GetOleDbSchemaTable(OleDbSchemaGuid.Tables, new object[] { null, null, "ABC", "TABLE" });`
	
	予備手段:
	
	データ定義の中に大文字小文字の区別やスペースが必要な場合は、それらの前後に引用符を付けます。たとえば、次のようにします。
	`cmd.CommandText = "create table ¥"Case Sensitive¥"(c1 int)";`
	`cmd.ExecuteNonQuery();`
	`tablename = "¥"Case Sensitive¥"";`
	`schemaTable = con.GetOleDbSchemaTable(OleDbSchemaGuid.Tables, new object[] { null, null, tablename, "TABLE" });`
		すべて

OleDbDataAdapter および DataColumnMapping	ソース列名には、大文字小文字の区別があります。これは、DB2 カタログに保管されるケースと一致している必要があり、それはデフォルトで大文字です。 たとえば、次のようにします。 `colMap = new DataColumnMapping("EMPNO", "Employee ID");`	すべて
OleDbDataReader.GetSchemaTable	OLE DB .NET Data Provider は、拡張記述情報を戻さないサーバーからは、拡張記述情報を検索できません。拡張記述をサポートしていないサーバー (影響を受けるサーバー) に接続している場合、 IDataReader.GetSchemaTable() から戻されたメタデータ表の中の以下の列は無効となります。 • IsReadOnly • IsUnique • IsAutoIncrement • BaseSchemaName • BaseCatalogName	DB2 for OS/390 バージョン 7 以降 DB2 for OS/400 DB2 for VM/VSE
ストアード・プロシージャー: 結果セットの列名はなし	DB2 for OS/390 バージョン 6.1 サーバーは、ストアード・プロシージャーから戻された結果セットの列名を戻しません。 OLE DB .NET Data Provider は、これらの無名列を、その順序位置 (たとえば、"1"、"2"、"3") にマップします。これは、MSDN で説明されているマッピング ("Column1", "Column2", "Column3") とは異なります。	DB2 for OS/390 バージョン 6.1

OLE DB .NET Data Provider アプリケーションでの接続プーリング

OLE DB .NET Data Provider は、OLE DB セッション・プーリングを使用して、自動的に接続をプールします。プーリングを含む OLE DB サービスを使用可能または使用不可にするには、接続ストリング引き数を使用します。たとえば、次の接続ストリングは、 OLE DB セッション・プーリングおよび自動トランザクション・エンリストメントを使用不可にします。

```
Provider=IBMDADB2;OLE DB Services=-4;Data Source=SAMPLE;
```

次の表では、OLE DB サービスを設定するために使用できる、 ADO 接続ストリング属性について説明しています。

表 29. ADO 接続ストリング属性を使用した OLE DB サービスの設定

使用可能サービス	接続ストリングの値
すべてのサービス (デフォルト)	″OLE DB Services = -1;″
プーリングを除くすべてのサービス	″OLE DB Services = -2;″
プーリングおよび自動エンリストメントを除くすべてのサービス	″OLE DB Services = -4;″
クライアント・カーソルを除くすべてのサービス	″OLE DB Services = -5;″
クライアント・カーソルおよびプーリングを除くすべてのサービス	″OLE DB Services = -6;″
サービスなし	″OLE DB Services = 0;″

OLE DB セッション・プーリングまたはリソース・プーリングについて、および OLE DB プロバイダー・サービスのデフォルトをオーバーライドしてプーリングを使用不可にする方法について詳しくは、 MSDN ライブラリー (http://msdn.microsoft.com/library) の「OLE DB Programmer's Reference」を参照してください。

OLE DB .NET Data Provider アプリケーションの時刻列

以下のセクションでは、OLE DB .NET Data Provider アプリケーションに、時刻列をインプリメントする方法について説明します。

パラメーター・マーカーを使用した挿入:

次のように、時刻列に時刻値を挿入しようとしています。

```
command.CommandText = "insert into mytable(c1) values( ? )";
```

列 c1 は時刻列です。ここでは、時刻値をパラメーター・マーカーにバインドする 2 つの方法があります。

`OleDbParameter.OleDbType = OleDbType.DBTime` を使用する

OleDbType.DBTime は TimeSpan オブジェクトにマップするので、 TimeSpan オブジェクトをパラメーター値として指定する必要があります。パラメーター値は、String または DateTime オブジェクトにはできません。これは TimeSpan オブジェクトでなければなりません。たとえば、次のようにします。

```
p1.OleDbType = OleDbType.DBTime;
p1.Value = TimeSpan.Parse("0.11:20:30");
rowsAffected = cmd.ExecuteNonQuery();
```

MSDN の資料で説明されているとおり、 TimeSpan の形式は、[-]d.hh:mm:ss.ff の形式のストリングで表記されます。

`OleDbParameter.OleDbType = OleDbType.DateTime` を使用する

この場合、OLE DB .NET Data Provider は、パラメーター値を TimeSpan オブジェクトではなく、DateTime オブジェクトに変換します。したがって、パラメーター値は、 DateTime オブジェクトに変換できる任意の有効なストリング/オブジェクトにできます。つまり、11:20:30 などの値を用いることができます。値を DateTime オブジェクトにすることもできます。値を TimeSpan オブジェクトにすることはできません。なぜなら、TimeSpan オブジェクトは DateTime オブジェクトに変換できないからです。 TimeSpan は IConvertible をインプリメントしていません。

たとえば、次のようにします。

```
p1.OleDbType = OleDbType.DBTimeStamp;
p1.Value = "11:20:30";
rowsAffected = cmd.ExecuteNonQuery();
```

検索:

時刻列を検索するには、 IDataRecord.GetValue() メソッドまたは OleDbDataReader.GetTimeSpan() メソッドを使用する必要があります。

たとえば、次のようにします。

```
TimeSpan ts1 = ((OleDbDataReader)reader).GetTimeSpan( 0 );
TimeSpan ts2 = (TimeSpan) reader.GetValue( 0 );
```

OLE DB .NET Data Provider アプリケーションの ADORecordset オブジェクト

以下は、ADORecordset オブジェクトの使用に関係した考慮事項です。

- ADO タイプ adDBTime クラスは、 .NET Framework の DateTime クラスにマップされます。 OleDbType.DBTime は、TimeSpan オブジェクトに対応します。
- TimeSpan オブジェクトを ADORecordset オブジェクトの Time フィールドに割り当てることはできません。なぜなら、ADORecordset オブジェクトの Time フィールドでは、 DateTime オブジェクトが期待されているからです。 TimeSpan オブジェクトを ADORecordset オブジェクトに割り当てると、次のメッセージが表示されます。

  ```
  Method's type signature is not Interop compatible.
  ```

  Time フィールドに入れることができるのは、DateTime オブジェクトか、DateTime オブジェクトに解析できる String のみです。
- OleDbDataAdapter を使用して DataSet に ADORecordset を取り込むとき、ADORecordset の Time フィールドは、 DataSet の TimeSpan 列に変換されます。
- Recordsets には主キーや制約は保管されません。したがって、MissingSchemaAction.AddWithKey を使用して DataSet に Recordset からデータを取り込む際は、キー情報は追加されません。

第 13 章 ODBC .NET Data Provider

ODBC .NET Data Provider . . . . . 279 | ODBC .NET Data Provider の制約事項 279

ODBC .NET Data Provider

ODBC .NET Data Provider は、DB2 CLI ドライバーを使用して、 DB2® データ・ソースに対して ODBC 呼び出しを行います。したがって、ODBC .NET Data Provider がサポートする接続ストリング・キーワードは、 DB2 CLI ドライバーがサポートする接続ストリング・キーワードと同じです。また、ODBC .NET Data Provider には、DB2 CLI ドライバーと同じ制約事項があります。 ODBC .NET Data Provider に対しては追加の制約事項があり、それについてはODBC .NET Data Provider の制約事項のトピックで説明しています。

ODBC .NET Data Provider を使用するには、 .NET Framework バージョン 1.1 がインストールされている必要があります。 DB2 Universal Database for AS/400® and iSeries™ の場合は、サーバーに APAR II13348 の修正を適用する必要があります。

以下は、ODBC .NET Data Provider でサポートされている接続キーワードです。

表 30. ODBC .NET Data Provider の **ConnectionString** *キーワード*

キーワード	値	意味
DSN	データベース別名	データベース・ディレクトリーにカタログされた DB2 データベース別名。
UID	ユーザー ID	DB2 サーバーへの接続に使用するユーザー ID
PWD	パスワード	DB2 サーバーへの接続に使用するユーザー ID のパスワード

以下に、OdbcConnection を作成して SAMPLE データベースに接続する例を示します。

```
[Visual Basic .NET]
Dim con As New OdbcConnection("DSN=sample;UID=userid;PWD=password;")
con.Open()

[C#]
OdbcConnection con = new OdbcConnection("DSN=sample;UID=userid;PWD=password;");
con.Open()
```

ODBC .NET Data Provider の制約事項

以下の表は、IBM ODBC .NET Data Provider の使用に関係した制約事項を示しています。

表 31. IBM ODBC .NET Data Provider の制約事項

クラスまたはフィーチャー	制約事項の説明	影響を受ける DB2 サーバー
ASCII 文字ストリーム	DbType.AnsiString または DbType.AnsiStringFixedLength を使用している場合、OdbcParameters で ASCII 文字ストリームを使用することはできません。 ODBC .NET Data Provider が次の例外を出します。 "Specified cast is not valid" **予備手段:** DbType.AnsiString または DbType.AnsiStringFixedLength の代わりに DbType.Binary を使用します。	すべて

Command.Prepare	最後の準備の後に CommandText が変更されている場合は、コマンド (Command.ExecuteNonQuery または Command.ExecuteReader) を実行する前に、OdbcCommand.Prepare() を明示的に実行する必要があります。OdbcCommand.Prepare() を再び呼び出さないと、 ODBC .NET Data Provider は以前に準備された CommandText を実行します。
	
	たとえば、次のとおりです。
	[C#]
	command.CommandText="select CLOB('ABC') from table1";
	command.Prepare();
	command.ExecuteReader();
	command.CommandText="select CLOB('XYZ') from table2";
	command.ExecuteReader(); // This ends up re-executing the first statement
		すべて

表 31. IBM ODBC .NET Data Provider の制約事項 (続き)

クラスまたはフィーチャー	制約事項の説明	影響を受ける DB2 サーバー

CommandBehavior.
SequentialAccess
	CommandBehavior.SequentialAccess で作成されたリーダーから、IDataReader.GetChars() を使用して読み取る場合は、列全体を保持するのに十分な大きさのバッファーを割り振る必要があります。そうしないと、次の例外が発生します。
	`Requested range extends past the end of the array.`
	`  at System.Runtime.InteropServices.Marshal.Copy(Int32 source,`
	`  Char[] destination, Int32 startIndex, Int32 length)`
	`  at System.Data.Odbc.OdbcDataReader.GetChars(Int32 i,`
	`  Int64 dataIndex, Char[] buffer, Int32 bufferIndex, Int32 length)`
	`  at OleRestrict.TestGetCharsAndBufferSize(IDbConnection con)`
	次の例で、十分なバッファーを割り振る方法を示します。
	`CREATE TABLE myTable(c0 int, c1 CLOB(10K))`
	`SELECT c1 FROM myTable;`
	`[C#]`
	`cmd.CommandText = "SELECT c1 from myTable";`
	`IDataReader reader = cmd.ExecuteReader(CommandBehavior.SequentialAccess);`
	`Int32 iChunkSize = 10;`
	`Int32 iBufferSize = 10;`
	`Int32 iFieldOffset = 0;`
	`Char[] buffer = new Char[ iBufferSize ];`
	`reader.Read();`
	`reader.GetChars(0, iFieldOffset, buffer, 0, iChunkSize);`
	GetChars() を呼び出すと、次の例外が出されます。
	`"Requested range extends past the end of the array"`
	GetChars() によって上記の例外が出されないようにするためには、次のようにして、BufferSize に列のサイズを設定します。
	`Int32 iBufferSize = 10000;`
	iBufferSize の値 10,000 は、 CLOB 列 c1 に割り振られている値 10K と対応します。
		すべて

CommandBehavior.
SequentialAccess
	ODBC .NET Data Provider は、 OdbcDataReader.GetChars() を使用しているときに読み取るデータがなくなると、次の例外を出します。
	`NO_DATA - no error information available`
	`  at System.Data.Odbc.OdbcConnection.HandleError(HandleRef hrHandle,`
	`  SQL_HANDLE hType, RETCODE retcode)`
	`  at System.Data.Odbc.OdbcDataReader.GetData(Int32 i, SQL_C sqlctype,`
	`  Int32 cb)`
	`  at System.Data.Odbc.OdbcDataReader.GetChars(Int32 i, Int64 dataIndex,`
	`  Char[] buffer, Int32 bufferIndex, Int32 length)`
		すべて

CommandBehavior.
SequentialAccess
	OdbcDataReader.GetChars() を使用するときは、値 5000 などの大きなチャンク・サイズは使用できません。大きなチャンク・サイズを使用しようとすると、ODBC .NET Data Provider が次の例外を出します。
	`Object reference not set to an instance of an object.`
	`  at System.Runtime.InteropServices.Marshal.Copy(Int32 source,`
	`  Char[] destination, Int32 startIndex, Int32 length)`
	`  at System.Data.Odbc.OdbcDataReader.GetChars(Int32 i, Int64 dataIndex,`
	`  Char[] buffer, Int32 bufferIndex, Int32 length)`
	`  at OleRestrict.TestGetCharsAndBufferSize(IDbConnection con)`
		すべて

接続プール	ODBC .NET Data Provider は接続プーリングを制御しません。接続プーリングは、ODBC Driver Manager によって取り扱われます。接続プーリングについて詳しくは、 MSDN ライブラリー (http://msdn.microsoft.com/library) の「ODBC Programmer's Reference」を参照してください。	すべて

表 31. IBM ODBC .NET Data Provider の制約事項 (続き)

クラスまたはフィーチャー	制約事項の説明	影響を受ける DB2 サーバー
DataColumnMapping	ソース列名の大文字小文字は、システム・カタログ表で使用されている大文字小文字と一致している必要があります。これは、デフォルトで大文字です。	すべて

10 進列	10 進列では、パラメーター・マーカーはサポートされません。
	
	通常、ターゲット SQLType が Decimal 列の場合は、 OdbcParameter に OdbcType.Decimal を使用します。しかし、ODBC .NET Data Provider は OdbcType.Decimal を見付けると、SQL_C_WCHAR の C タイプと SQL_VARCHAR の SQLType を使用してパラメーターをバインドし、それは無効となります。
	
	以下に例を示します。
	`[C#]`
	`cmd.CommandText = "SELECT dec_col FROM MYTABLE WHERE dec_col > ? ";`
	`OdbcParameter p1 = cmd.CreateParameter();`
	`p1.DbType = DbType.Decimal;`
	`p1.Value = 10.0;`
	`cmd.Parameters.Add(p1);`
	`IDataReader rdr = cmd.ExecuteReader();`
	
	次の例外が戻されます。
	`ERROR [07006] [IBM][CLI Driver][SQLDS/VM] SQL0301N The value of input host variable or parameter number "" cannot be used because of its data type. SQLSTATE=07006`
	
	予備手段:
	
	OdbcParameter 値を使用する代わりに、リテラルのみを使用します。
		DB2 for VM/VSE

キー情報	表名を修飾するために使用されるスキーマ名 (たとえば、MYSCHEMA.MYTABLE) は、接続ユーザー ID と一致していなければなりません。 ODBC .NET Data Provider は、指定されたスキーマが接続ユーザー ID と異なるキー情報は検索できません。
	
	以下に例を示します。
	`CREATE TABLE USERID2.TABLE1(c1 INT NOT NULL PRIMARY KEY);`
	
	`[C#]`
	`// Connect as user bob`
	`odbcCon = new OdbcConnection("DSN=sample;UID=bob;PWD=mypassword");`
	
	`OdbcCommand cmd = odbcCon.CreateCommand();`
	
	`// Select from table with schema USERID2`
	`cmd.CommandText="SELECT * FROM USERID2.TABLE1";`
	
	`// Fails - No key info retrieved`
	`da.FillSchema(ds, SchemaType.Source);`
	
	`// Fails - SchemaTable has no primary key`
	`cmd.ExecuteReader(CommandBehavior.KeyInfo)`
	
	`// Throws exception because no primary key`
	`cbuilder.GetUpdateCommand();`
		すべて

表 31. IBM ODBC .NET Data Provider の制約事項 (続き)

クラスまたはフィーチャー	制約事項の説明	影響を受ける DB2 サーバー

キー情報	ODBC .NET Data Provider は、 IDataReader を開くのと同時にキー情報を検索することはできません。 ODBC .NET Data Provider が IDataReader を開くと、サーバー上でカーソルが開きます。キー情報が要求された場合、これは SQLPrimaryKeys() または SQLStatistic() を呼び出してキー情報を取得しますが、これらのスキーマ関数は他のカーソルを開きます。 DB2 for VM/VSE はカーソル保留をサポートしていないため、最初のカーソルはクローズされます。その結果、IDataReader.Read() が IDataReader を呼び出すと、次の例外が発生します。
	`System.Data.Odbc.OdbcException: ERROR [HY010] [IBM][CLI Driver] CLI0125E Function sequence error. SQLSTATE=HY010`
	予備手段:
	先にキー情報を検索してから、データを検索する必要があります。
	たとえば、次のようにします。
	`[C#]`
	`OdbcCommand cmd = odbcCon.CreateCommand();`
	`OdbcDataAdapter da = new OdbcDataAdapter(cmd);`
	`cmd.CommandText = "SELECT * FROM MYTABLE";`
	`// Use FillSchema to retrieve just the schema information`
	`da.FillSchema(ds, SchemaType.Source);`
	`// Use FillSchema to retrieve just the schema information`
	`da.Fill(ds);`
		DB2 for VM/VSE

キー情報	SQL ステートメントの中では、データベース・オブジェクトは、データベース・オブジェクトをシステム・カタログ表に保管するのに用いられているケース (大文字小文字) と同じケースを使用して参照する必要があります。デフォルトでは、データベース・オブジェクトは大文字でシステム・カタログ表に保管されるので、ほとんどの場合は、大文字を使用する必要があります。 ODBC .NET Data Provider は、SQL ステートメントをスキャンしてデータベース・オブジェクト名を検索し、それらを、システム・カタログ表内のこれらのオブジェクトについての照会を発行する SQLPrimaryKeys および SQLStatistics などのスキーマ関数に渡します。データベース・オブジェクトの参照は、システム・カタログ表にそれらが保管されている状態と完全に一致していなければなりません。そうでないと、空の結果セットが戻されます。	DB2 for OS/390 DB2 for OS/400 DB2 for VM/VSE
バッチの非選択 SQL ステートメントのキー情報	ODBC .NET Data Provider は、 SELECT で始まっていないバッチ・ステートメントのキー情報は検索できません。	DB2 for OS/390 DB2 for OS/400 DB2 for VM/VSE

表 31. IBM ODBC .NET Data Provider の制約事項 (続き)

クラスまたはフィーチャー	制約事項の説明	影響を受ける DB2 サーバー

LOB 列	ODBC .NET Data Provider は LOB データ・タイプをサポートしていません。そのため、DB2 サーバーが SQL_CLOB (-99)、SQL_BLOB (-98)、または SQL_DBCLOB (-350) を戻すと、 ODBC .NET Data Provider は次の例外を出します。
	`"Unknown SQL type - -98"` (Blob 列の場合)
	`"Unknown SQL type - -99"` (Clob 列の場合)
	`"Unknown SQL type - -350"` (DbClob 列の場合)
	
	直接または間接的に LOB 列にアクセスするメソッドはどれも失敗します。
	
	予備手段:
	
	CLI/ODBC `LongDataCompat` キーワードに 1 を設定します。こうすると、DB2 CLI ドライバーが、次のような ODBC .NET Data Provider が認識するデータ・タイプへのデータ・タイプ・マッピングを行います。
	• SQL_CLOB から SQL_LONGVARCHAR へ
	• SQL_BLOB から SQL_LONGVARBINARY へ
	• SQL_DBCLOB から SQL_WLONGVARCHAR へ
	
	`LongDataCompat` キーワードを設定するには、クライアント・マシンの DB2 コマンド・ウィンドウから次の DB2 コマンドを実行します。
	`db2 update cli cfg for section common using longdatacompat 1`
	
	このキーワードは、次のように、アプリケーションで接続ストリングを使用して設定することもできます。
	`[C#]`
	`OdbcConnection con =`
	`  new OdbcConnection("DSN=SAMPLE;UID=uid;PWD=mypwd;LONGDATACOMPAT=1;");`
	
	すべての CLI/ODBC キーワードのリストについては、「DB2 Universal Database コール・レベル・インターフェースの手引きおよび解説書」の『UID CLI/ODBC 構成キーワード』を参照してください。
		すべて

`OdbcCommand.Cancel`	`OdbcCommand.Cancel` の実行後にステートメントを実行すると、次の例外が発生することがあります。 `"ERROR [24000] [Microsoft][ODBC Driver Manager] Invalid cursor state"`	すべて

表 31. IBM ODBC .NET Data Provider の制約事項 (続き)

クラスまたはフィーチャー	制約事項の説明	影響を受ける DB2 サーバー

OdbcCommandBuilder	OdbcCommandBuilder を使用して UPDATE、DELETE、および INSERT ステートメントを自動生成する場合、大文字小文字の区別は重要です。システム・カタログ表が、(作成時にデータベース・オブジェクトの前後に引用符を追加することによって) 明示的に大文字小文字の区別付きで作成されているのでない限り、デフォルトで DB2 はスキーマ情報 (表名や列名など) を大文字でシステム・カタログ表に保管します。したがって、SQL ステートメントは、カタログに保管されているケース (デフォルトでは大文字) と一致している必要があります。
	
	たとえば、次のステートメントを使用して表を作成したとします。
	`"db2 create table mytable (c1 int) "`
	
	この場合、DB2 は表名 mytable をシステム・カタログ表に MYTABLE として保管します。
	
	次のコード例は、OdbcCommandBuilder クラスの正しい使用法を例示しています。
	`[C#]`
	`OdbcCommand cmd = odbcCon.CreateCommand();`
	`cmd.CommandText = "SELECT * FROM MYTABLE";`
	`OdbcDataAdapter da = new OdbcDataAdapter(cmd);`
	`OdbcCommandBuilder cb = new OdbcCommandBuilder(da);`
	`OdbcCommand updateCmd = cb.GetUpdateCommand();`
	
	この例では、表名を大文字で参照しないと、次の例外を受け取ります。
	`"Dynamic SQL generation for the UpdateCommand is not supported against a SelectCommand that does not return any key column information."`
		すべて

OdbcCommandBuilder	SELECT ステートメントに以下の列データ・タイプが含まれている場合、OdbcCommandBuilder によって生成されたコマンドは正しくありません。
	`REAL`
	`FLOAT または DOUBLE`
	`TIMESTAMP`
	
	これらのデータ・タイプは、SELECT ステートメントの WHERE 文節では使用できません。
		DB2 for OS/390
		DB2 for OS/400
		DB2 for VM/VSE

OdbcCommandBuilder.DeriveParameters	DeriveParameters() メソッドは、SQLProcedureColumns にマップされ、ストアード・プロシージャーの名前に CommandText プロパティーを使用します。CommandText には、 (完全な ODBC 呼び出し構文を使用する) ストアード・プロシージャーの名前は含まれないので、 SQLProcedureColumns は、 ODBC 呼び出し構文に基づいて識別されたプロシージャー名で呼び出されます。以下に例を示します。 `"{ CALL myProc(?) }"` この結果は、空の結果セットとなり、ここからはプロシージャーの列は見付かりません。	すべて
OdbcCommandBuilder.DeriveParameters	DeriveParameters() を使用するには、 CommandText にストアード・プロシージャーの名前を指定します (たとえば、cmd.CommandText = "MYPROC")。プロシージャー名は、システム・カタログ表に保管されているケースと一致していなければなりません。 DeriveParameters() では、システム・カタログ表内で見付かる、そのプロシージャー名のすべてのパラメーターが戻されます。ステートメントを実行する前に、 CommandText を元の完全な ODBC 呼び出し構文に戻すのを忘れないでください。	すべて
OdbcCommandBuilder.DeriveParameters	ODBC .NET Data Provider に関しては、ReturnValue パラメーターは戻されません。	すべて

表 31. IBM ODBC .NET Data Provider の制約事項 (続き)

クラスまたはフィーチャー	制約事項の説明	影響を受ける DB2 サーバー
OdbcCommandBuilder.DeriveParameters	DeriveParameters() は、完全修飾ストアード・プロシージャー名をサポートしていません。たとえば、CommandText = "MYSCHEMA.MYPROC" に対して DeriveParameters() を呼び出すと失敗します。このとき、パラメーターは戻されません。	すべて
OdbcCommandBuilder.DeriveParameters	DeriveParameters() は、多重定義のストアード・プロシージャーに対しては機能しません。 SQLProcedureColumns は、ストアード・プロシージャーのすべてのバージョンのすべてのパラメーターを戻します。	すべて
OdbcConnection.ChangeDatabase	OdbcConnection.ChangeDatabase() メソッドはサポートされていません。	すべて
OdbcConnection.ConnectionString	• Server キーワードは無視されます。 • Connect Timeout キーワードは無視されます。 DB2 CLI は接続タイムアウトをサポートしていないので、このプロパティーを設定しても、ドライバーに影響はありません。 • 接続プーリング・キーワードは無視されます。具体的には、これは Pooling、Min Pool Size、Max Pool Size、 Connection Lifetime、および Connection Reset というキーワードに影響します。	すべて
OdbcDataReader.GetSchemaTable	ODBC .NET Data Provider は、拡張記述情報を戻さないサーバーからは、拡張記述情報を検索できません。そのため、拡張記述をサポートしていないサーバー (影響を受けるサーバー) に接続している場合、 IDataReader.GetSchemaTable() から戻されたメタデータ表の中の以下の列は無効となります。 • IsReadOnly • IsUnique • IsAutoIncrement • BaseSchemaName • BaseCatalogName	DB2 for OS/390 バージョン 7 以降 DB2 for OS/400 DB2 for VM/VSE

ストアード・プロシージャー	ストアード・プロシージャーを呼び出すには、完全な ODBC 呼び出し構文を指定する必要があります。
	
	たとえば、VARCHAR(10) をパラメーターとして取るストアード・プロシージャー MYPROC を呼び出すには、次のようにします。
	[C#]
	OdbcCommand cmd = odbcCon.CreateCommand();
	cmd.CommandType = CommandType.Text;
	cmd.CommandText = "{ CALL MYPROC(?) }"
	OdbcParameter p1 = cmd.CreateParameter();
	p1.Value = "Joe";
	p1.OdbcType = OdbcType.NVarChar;
	cmd.Parameters.Add(p1);
	cmd.ExecuteNonQuery();
	
	注: CommandType.StoredProcedure を使用している場合であっても、完全な ODBC 呼び出し構文を使用する必要があることに注意してください。これは、MSDN の中の OdbcCommand.CommandText プロパティーの項で説明されています。
		すべて

ストアード・プロシージャー: 結果セットの列名はなし	DB2 for OS/390 バージョン 6.1 サーバーは、ストアード・プロシージャーから戻された結果セットの列名を戻しません。 ODBC .NET Data Provider は、これらの無名列を、その順序位置 (たとえば、″1″、″2″、″3″) にマップします。これは、MSDN で説明されているマッピング ("Column1", "Column2", "Column3") とは異なります。	DB2 for OS/390 バージョン 6.1
ユニーク索引の主キーへのプロモーション	ODBC .NET Data Provider は、NULL 可能なユニーク索引を主キーにプロモートします。これは MSDN の説明と異なります。 MSDN では、NULL 可能なユニーク索引は主キーにプロモートすべきでないと説明されています。	すべて

第 4 部 Java

第 14 章 Java アプリケーション・サポートの概要

DB2® Universal Database は、 JDBC を使用して Java™ で作成されたクライアント・アプリケーションおよびアプレット用、および組み込み SQL for Java (SQLJ) 用のドライバー・サポートを提供します。

JDBC とは、Java アプリケーションがリレーショナル・データベースにアクセスする際に使用する、アプリケーション・プログラミング・インターフェース (API) です。 JDBC 用の DB2 Universal Database™ サポートを使用すると、 DRDA® をサポートするサーバー上で、ローカル DB2 データまたはリモート・リレーショナル・データにアクセスする Java アプリケーションを作成することができます。

SQLJ は、Java アプリケーションでの組み込み静的 SQL のサポートを提供します。もともと SQLJ は、静的 SQL モデルで動的 SQL JDBC モデルを補うために、 IBM®、Oracle®、および Tandem によって開発されたものです。

Java アプリケーションは一般的に、動的 SQL の場合は JDBC を、静的 SQL の場合は SQLJ を使用します。ただし、SQLJ は JDBC との相互運用が可能であるため、アプリケーションは同じ作業単位内で JDBC と SQLJ を使用できます。

このトピックでは、DB2 Universal Database によって提供される、 Java アプリケーション開発環境について説明します。

JDBC の仕様によると、JDBC ドライバー・アーキテクチャーには、以下の 4 つのタイプがあります。

タイプ 1
Open Database Connectivity (ODBC) など、他のデータ・アクセス API へのマッピングとして、 JDBC API をインプリメントするドライバー。一般的にこのタイプのドライバーは、ネイティブ・ライブラリーに依存しているため、移植性が限られています。 JDBC-ODBC ブリッジ・ドライバーは、タイプ 1 ドライバーの一例です。

タイプ 2
一部が Java プログラム言語で、一部がネイティブ・コードで作成されたドライバー。このドライバーは、接続先のデータ・ソースに固有のネイティブ・クライアント・ライブラリーを使用します。ネイティブ・コードであるため、移植性は限られています。

タイプ 3
pure Java クライアントを使用し、データベース非依存のプロトコルを使用してサーバーと通信するドライバー。そして、サーバーはクライアントの要求をデータ・ソースへ送ります。

タイプ 4
pure Java であり、特定のデータ・ソース用のネットワーク・プロトコルをインプリメントするドライバー。クライアントはデータ・ソースに直接接続します。

DB2 バージョン 8 では、タイプ 2 ドライバー、およびタイプ 2 とタイプ 4 の JDBC インプリメンテーションを結合したドライバーがサポートされています。

DB2 バージョン 8 では、タイプ 3 ドライバーもサポートされています。ただし、このドライバーは推奨されていません。これまでのリリースの DB2 UDB for Linux, UNIX®, and Windows® での JDBC ドライバーは、 DB2 CLI (Call Level Interface) に基づいて構築されていました。 DB2 バージョン 8 のタイプ 2 およびタイプ 3 ドライバーは、 DB2 UDB サーバーと通信するために、引き続き DB2 CLI インターフェースを使用します。 DB2 バージョン 8 では、完全に Java で作成された新しい DB2 Universal JDBC ドライバーが追加されています。 DB2 バージョン 8 でサポートされているドライバーは、以下のとおりです。

Linux、UNIX、および Windows 用の DB2 JDBC タイプ 2 ドライバー (DB2 JDBC タイプ 2 ドライバー) (DB2 V8.2 では推奨されていません):

DB2 JDBC タイプ 2 ドライバーでは、Java アプリケーションが JDBC を介して DB2 への呼び出しを行います。 DB2 JDBC タイプ 2 ドライバーへの呼び出しは、Java ネイティブ・メソッドに変換されます。このドライバーを使用する Java アプリケーションは、DB2 クライアントで実行されます。 JDBC 要求は、この DB2 クライアントを介して DB2 サーバーへ送られます。 DB2 JDBC アプリケーション・ドライバーを使用して、DB2 UDB for iSeries™ データ・ソース、または DB2 for OS/390 or z/OS 環境内のデータ・ソースへアクセスするには、その前に、DB2 Connect™ バージョン 8 をインストールしなければなりません。

DB2 JDBC タイプ 2 ドライバーは、以下の JDBC および SQLJ 関数をサポートしています。

- JDBC 1.2 仕様で記述されているメソッドのほとんどと、 JDBC 2.0 仕様で記述されているメソッドの一部。 『JDBC API 用ドライバー・サポートの比較』を参照してください。
- すべての JDBC メソッドに同等の機能を実行する SQLJ ステートメント
- 接続プール
- 分散トランザクション
- Java ユーザー定義関数およびストアード・プロシージャー

Linux、UNIX、および Windows 用の DB2 JDBC タイプ 2 ドライバーは、 DB2 の将来のリリースではサポートされなくなります。そのため、DB2 Universal JDBC ドライバーへの移行を考慮する必要があります。

Linux、UNIX、 および Windows 用の DB2 JDBC タイプ 3 ドライバー (DB2 V8.1 では推奨されていません):

DB2 JDBC タイプ 3 ドライバー (アプレットやネット・ドライバーとも呼ばれる) は、 JDBC クライアントと JDBC サーバーから成り立っています。 DB2 JDBC アプレット・ドライバーは、Web ブラウザーによってアプレットとともにロードできます。または、アプレット・ドライバーをスタンドアロン Java アプリケーションとして使用することもできます。アプレットが DB2 データベース・サーバーへの接続を要求すると、アプレット・ドライバーは、 Web サーバーが実行されているマシン上の DB2 JDBC アプレット・サーバーに対して TCP/IP ソケットをオープンします。接続がセットアップされると、アプレット・ドライバーは以後のそれぞれのデータベース・アクセス要求を、 TCP/IP 接続を介してアプレットから JDBC サーバーへ送信します。それから JDBC サーバーは、タスクを実行するために対応す

る DB2 呼び出しを行います。完了すると、JDBC サーバーはその接続を介して JDBC クライアントに結果を送り返します。 JDBC サーバー・プロセスは db2jd です。

Linux、UNIX、および Windows 用の DB2 JDBC タイプ 3 ドライバーは、 DB2 の将来のリリースではサポートされなくなります。そのため、DB2 Universal JDBC ドライバーへの移行を考慮する必要があります。

DB2 Universal JDBC ドライバー (タイプ 2 およびタイプ 4):

DB2 Universal JDBC ドライバーは、SQLJ サポートに加えて、 JDBC タイプ 2 および JDBC タイプ 4 の動作を組み込んだ単一のドライバーです。アプリケーションが DB2 Universal JDBC ドライバーをロードすると、タイプ 2 およびタイプ 4 インプリメンテーションに関して単一のドライバー・インスタンスがロードされます。アプリケーションはこの単一のドライバー・インスタンスを使用して、タイプ 2 および 4 の接続を作成します。タイプ 2 およびタイプ 4 の接続は同時に作成できます。 DB2 Universal JDBC ドライバーのタイプ 2 ドライバーの動作は、*DB2 Universal JDBC Driver type 2 connectivity* と呼ばれます。 DB2 Universal JDBC ドライバーのタイプ 4 ドライバーの動作は、*DB2 Universal JDBC Driver type 4 connectivity* と呼ばれます。

DB2 Universal JDBC ドライバーは、他の何らかの DB2 JDBC ドライバーの後継ではなく、全く新しいドライバーです。したがって、このドライバーと他のドライバーとの間には、動作の点でいくらかの違いが予想されます。

DB2 Universal JDBC ドライバーは、以下の JDBC および SQLJ 機能をサポートしています。

- JDBC 1.2 および JDBC 2.0 仕様で記述されているメソッドのほとんどと、JDBC 3.0 仕様で記述されているメソッドの一部。 『JDBC API 用ドライバー・サポートの比較』を参照してください。
- すべての JDBC メソッドに同等の機能を実行する SQLJ ステートメント。
- 接続プールに使用可能な接続。 WebSphere Application Server または別のアプリケーション・サーバーによって、接続プールが行われます。
- Java ユーザー定義関数およびストアード・プロシージャーのインプリメンテーション (Universal Type 2 Connectivity のみ)。
- WebSphere® Application Server バージョン 5.0 以降で実行されるグローバル・トランザクション。
- 分散トランザクション管理のサポート。このサポートは、 Java 2 Platform, Enterprise Edition (J2EE) Java Transaction Service (JTS) および Java Transaction API (JTA) の仕様をインプリメントします。これは、グローバル・トランザクションに関する X/Open 規格に準拠しています (*Distributed Transaction Processing: The XA Specification* (www.opengroup.org から入手可能)。

関連資料:

- 470 ページの『DB2 Universal JDBC ドライバーと他の DB2 JDBC ドライバーの JDBC との相違点』
- 477 ページの『DB2 Universal JDBC ドライバーと他の DB2 JDBC ドライバーとの間の SQLJ の相違』
- 416 ページの『JDBC API 用ドライバー・サポートの比較』

第 15 章 JDBC アプリケーション・プログラミング

以降のセクションでは、 JDBC アプリケーションの作成に関する情報が含まれています。

JDBC アプリケーション・プログラミングの基本的な概念

以降のトピックでは、 JDBC アプリケーションの作成に関する基本的な情報が含まれています。

JDBC アプリケーション作成時の基本的なステップ

JDBC アプリケーションの作成は、他の言語での SQL アプリケーションの作成と多くの点で共通しています。一般に、以下のことを行う必要があります。

- JDBC メソッドが入っている Java™ パッケージへアクセスする。
- DB2® 表からのデータ送信またはデータ検索用の変数を宣言する。
- データ・ソースに接続する。
- SQL ステートメントを実行する。
- SQL エラーおよび警告を処理する。
- データ・ソースから切断する。

実行する必要のあるタスクは、他の言語でも同様ですが、それらのタスクを実行する方法はいくらか異なります。

294 ページの図 4 は、各タスクを例示する簡単なプログラムです。このプログラムは、DB2 Universal JDBC ドライバーで実行されます。

```
import java.sql.*;                                                          1

public class EzJava
{
  public static void main(String[] args)
  {
    String urlPrefix = "jdbc:db2:";
    String url;
    String empNo;                                                           2
    Connection con;
    Statement stmt;
    ResultSet rs;

    System.out.println ("**** Enter class EzJava");

    // Check the that first argument has the correct form for the portion
    // of the URL that follows jdbc:db2:, as described
    // in the Connecting to a data source using the DriverManager
    // interface with the DB2 Universal JDBC Driver topic.
    // For example, for Universal Driver type 2 connectivity,
    // args[0] might be MVS1DB2M. For Universal
    // Driver type 4 connectivity, args[0] might
    // be //stlmvs1:10110/MVS1DB2M.
    if (args.length==0)
    {
      System.err.println ("Invalid value. First argument appended to "+
       "jdbc:db2: must specify a valid URL.");
      System.exit(1);
    }
    url = urlPrefix + args[0];

    try
    {
      // Load the DB2 Universal JDBC ドライバー
      Class.forName("com.ibm.db2.jcc.DB2Driver");                           3a
      System.out.println("**** Loaded the JDBC driver");

      // Create the connection using the DB2 Universal JDBC Driver
      con = DriverManager.getConnection (url);                              3b
      // Commit changes manually
      con.setAutoCommit(false);
      System.out.println("**** Created a JDBC connection to the data source");

      // Create the Statement
      stmt = con.createStatement();                                         4a
      System.out.println("**** Created JDBC Statement object");

      // Execute a query and generate a ResultSet instance
      rs = stmt.executeQuery("SELECT EMPNO FROM EMPLOYEE");                 4b
      System.out.println("**** Creaed JDBC ResultSet object");

      // Print all of the employee numbers to standard output device
      while (rs.next()) {
        empNo = rs.getString(1);
        System.out.println("Employee number = " + empNo);
      }
      System.out.println("**** Fetched all rows from JDBC ResultSet");
```

図 4. 簡単な JDBC アプリケーション (1/2)

```
      // Close the ResultSet
      rs.close();
      System.out.println("**** Closed JDBC ResultSet");

      // Close the Statement
      stmt.close();
      System.out.println("**** Closed JDBC Statement");

      // Connection must be on a unit-of-work boundary to allow close
      con.commit();
      System.out.println ( "**** Transaction committed" );

      // Close the connection
      con.close();                                                          6
      System.out.println("**** Disconnected from data source");

      System.out.println("**** JDBC Exit from class EzJava - no errors");

    }

    catch (ClassNotFoundException e)
    {
      System.err.println("Could not load JDBC driver");
      System.out.println("Exception: " + e);
      e.printStackTrace();
    }

    catch(SQLException ex)                                                  5
    {
      System.err.println("SQLException information");
      while(ex!=null) {
        System.err.println ("Error msg: " + ex.getMessage());
        System.err.println ("SQLSTATE: " + ex.getSQLState());
        System.err.println ("Error code: " + ex.getErrorCode());
        ex.printStackTrace();
        ex = ex.getNextException(); // For drivers that support chained exceptions
      }
    }
  }  // End main
}    // End EzJava
```

図 4. 簡単な JDBC アプリケーション (2/2)

294 ページの図 4 の注:

1 このステートメントは、java.sql パッケージをインポートします。これには、JDBC コア API が入っています。アクセスする必要のある他の Java パッケージの詳細については、 『JDBC サポート用の Java パッケージへのアクセス』を参照してください。

2 String 変数 empNo は、ホスト変数の機能を実行します。つまり、SQL 照会から検索されたデータを保持するために使用されます。詳細については、 『JDBC アプリケーションでの変数の宣言』を参照してください。

3a および **3b** これら 2 つのステートメントのセットは、使用可能な 2 つのインターフェースのいずれかを使用して、データ・ソースに接続する方法を示します。詳細については、 『JDBC を使用したデータ・ソースへの接続』を参照してください。

4a および **4b** ステートメントのこれらの 2 つのセットは、 JDBC で SELECT を実行する方法を示します。他の SQL 操作を実行する方法の詳細については、 『JDBC アプリケーションでの SQL の実行』を参照してください。

5 この try/catch ブロックは、 SQL エラー処理での SQLException クラスの使用を示しています。 SQL エラーの処理の詳細については、 『DB2 Universal JDBC ドライバー使用時の SQLException の処理』を参照してください。 SQL 警告の処理の詳細については、 『JDBC アプリケーションでの SQL 警告の処理』を参照してください。

6 このステートメントは、アプリケーションをデータ・ソースから切断します。『データ・ソースへの接続のクローズ』を参照してください。

関連概念:

- 296 ページの『JDBC サポート用の Java パッケージ』
- 296 ページの『JDBC アプリケーションでの変数』
- 307 ページの『SQL を実行するための JDBC インターフェース』
- 297 ページの『JDBC アプリケーションによるデータ・ソースへの接続の方法』

関連タスク:

- 315 ページの『DB2 Universal JDBC ドライバー使用時の SQLException の処理』
- 319 ページの『DB2 Universal JDBC ドライバー使用時の SQLWarning の処理』

JDBC サポート用の Java パッケージ

JDBC メソッドを呼び出す前に、それらのメソッドを含むさまざまな Java™ パッケージのすべて、または一部にアクセスできなければなりません。そのために、パッケージか特定クラスをインポートするか、完全修飾クラス名を使用することができます。 JDBC プログラムでは、以下のパッケージまたはクラスが必要になる可能性があります。

java.sql
コア JDBC API を含みます。

javax.naming
Java Naming and Directory Interface (JNDI) のクラスおよびインターフェースを含みます。これは、`DataSource` をインプリメントするためにしばしば使用されます。

javax.sql
JDBC 2.0 規格の拡張機能を含みます。

javax.transaction
DB2® JDBC Type 2 Driver for Linux, UNIX® and Windows® (DB2 JDBC Type 2 Driver) 用の、分散トランザクションのための JDBC サポートを含みます。

com.ibm.db2.jcc
DB2 Universal JDBC ドライバー用の JDBC の、 DB2 固有のインプリメンテーションを含みます。

COM.ibm.db2.jdbc
DB2 JDBC Type 2 Driver 用の JDBC の、DB2 固有のインプリメンテーションを含みます。

JDBC アプリケーションでの変数

他のすべての Java™ アプリケーションの場合のように、 JDBC アプリケーションを作成する際に変数を宣言します。 Java アプリケーションでは、それらの変数のことを Java ID と呼びます。それらの ID のいくつかは、他の言語でのホスト変数と同じ機能を持っています。それらは、DB2® 表に渡すデータまたは DB2® 表から検

索するデータを保持します。 『JDBC アプリケーション作成時の基本的なステップ』 でのサンプル・プログラムの ID empNo は、 DB2 表の CHAR 列から検索するデータを保持する Java String ID の例です。

Java 変数のデータ・タイプが DB2 データ・タイプに緊密にマップされている場合、 DB2 はより適切なアクセス・パスを選択するので、 Java データ・タイプの選択はパフォーマンスに影響を与える可能性があります。 『Java、JDBC、および SQL データ・タイプ』 では、 Java データ・タイプおよび JDBC データ・タイプと、SQL データ・タイプの推奨マッピングを示しています。

関連概念:

- 293 ページの『JDBC アプリケーション作成時の基本的なステップ』

関連資料:

- 403 ページの『Java、JDBC、および SQL のデータ・タイプ』

JDBC アプリケーションによるデータ・ソースへの接続の方法

SQL プログラムで SQL ステートメントを実行する前に、データベース・サーバーに接続する必要があります。 JDBC では、データベース・サーバーのことをデータ・ソース と呼びます。

図 5 には、 Java™ アプリケーションで、 Type 2 ドライバーまたは DB2 Universal JDBC Driver type 2 connectivity のデータ・ソースに接続する方法を示しています。

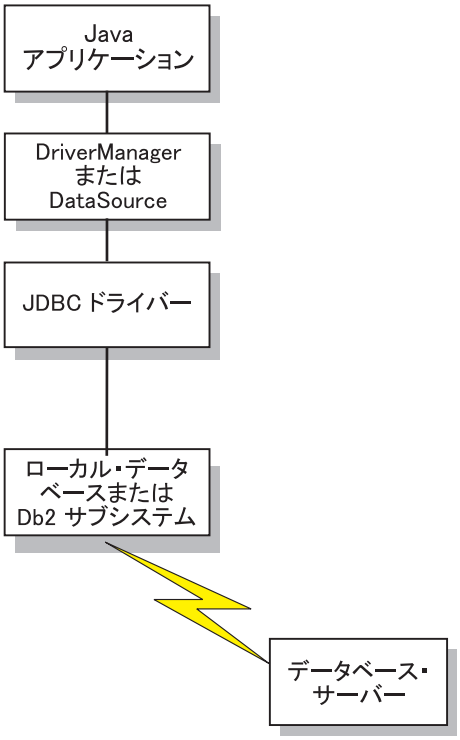
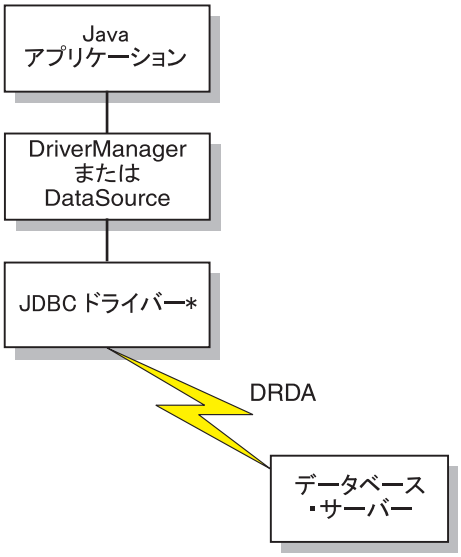


図 5. Type 2 ドライバーまたは DB2 Universal JDBC Driver type 2 connectivity での Java アプリケーションの流れ

図 6 には、Java アプリケーションで、 DB2 Universal JDBC Driver type 4 connectivity のデータ・ソースに接続する方法が示されています。



* JVM の下で実行される Java バイト・コード

図 6. DB2 Universal JDBC Driver type 4 connectivity での Java アプリケーションの流れ

データ・ソースに接続する方法は、使用する JDBC のバージョンによって異なります。 `DriverManager` インターフェースを使用して接続する方法は、すべてのレベルの JDBC で使用可能です。 `DataSource` インターフェースを使用して接続する方法は、JDBC 2.0 以降で使用可能です。

関連概念:

- 298 ページの『DB2 アプリケーションによる DriverManager インターフェースと DB2 JDBC Type 2 ドライバーを使用したデータ・ソースへの接続』

関連タスク:

- 303 ページの『DataSource インターフェースを使用したデータ・ソースへの接続』
- 300 ページの『DriverManager インターフェースと DB2 Universal JDBC ドライバーを使用したデータ・ソースへの接続』

DB2 アプリケーションによる DriverManager インターフェースと DB2 JDBC Type 2 ドライバーを使用したデータ・ソースへの接続

JDBC アプリケーションは、 `java.sql` パッケージの一部である JDBC `DriverManager` インターフェースを使用して、データ・ソースへの接続を確立できます。

最初に Java™ アプリケーションは、 Class.forName メソッドを呼び出して、JDBC ドライバーをロードします。アプリケーションがドライバーをロードした後で、 DriverManager.getConnection メソッドを呼び出すことにより、データベース・サーバーに接続します。

Linux、UNIX®、および Windows® 用 DB2® JDBC Type 2 ドライバー (DB2 JDBC Type 2 ドライバー) の場合、以下の引き数を指定した Class.forName メソッドを呼び出して、ドライバーをロードします。

```
COM.ibm.db2.jdbc.app.DB2Driver
```

以下のコードは、DB2 JDBC Type 2 ドライバーのロードを示しています。

```
try {
  // Load the DB2 JDBC Type 2 Driver with DriverManager
  Class.forName("COM.ibm.db2.jdbc.app.DB2Driver");
} catch (ClassNotFoundException e) {
   e.printStackTrace();
}
```

catch ブロックは、ドライバーが見つからなかった場合にエラーをプリントするのに使用されます。

ドライバーのロード後、 DriverManager.getConnection メソッドを呼び出して、データ・ソースへ接続します。以下のいずれかの形式の getConnection を使用できます。

```
getConnection(String url);
getConnection(String url, user, password);
getConnection(String url, java.util.Properties info);
```

url 引き数は、データ・ソースを表します。

DB2 JDBC Type 2 ドライバーの場合、次の形式の URL を指定します。

DB2 JDBC Type 2 ドライバーの URL の構文:

URL の各部には以下のような意味があります。

jdbc:db2:
jdbc:db2: は、接続が DB2 UDB サーバーに対するものであることを示します。

database
データベース別名。別名は、DB2 クライアント上の DB2 データベース・カタログ項目を参照します。

info 引き数は、接続に関するドライバー・プロパティーのセットを含む、タイプ java.util.Properties のオブジェクトです。 *info* 引き数の指定は、 URL での *property=value* ストリングの指定の代わりとなります。

接続のユーザー ID およびパスワードの指定: 接続のユーザー ID およびパスワードを指定するには、いくつかの方法があります。

- *user* および *password* を指定する、 getConnection メソッドの形式を使用する。

- java.util.Properties オブジェクトでユーザーおよびパスワード・プロパティーを設定してから、 *info* を指定する getConnection メソッドの形式を使用します。

例: ユーザーおよびパスワード・パラメーターでのユーザー ID およびパスワードの設定

```
String url = "jdbc:db2:toronto";
                                        // Set URL for data source
String user = "db2adm";
String password = "db2adm";
Connection con = DriverManager.getConnection(url, user, password);
                                        // Create connection
```

例: java.util.Properties オブジェクトでのユーザー ID およびパスワードの設定

```
Properties properties = new Properties(); // Create Properties object
properties.put("user", "db2adm");         // Set user ID for connection
properties.put("password", "db2adm");     // Set password for connection
String url = "jdbc:db2:toronto";
                                        // Set URL for data source
Connection con = DriverManager.getConnection(url, properties);
                                        // Create connection
```

関連概念:

- 487 ページの『DB2 JDBC Type 2 ドライバー使用時のセキュリティー』

DriverManager インターフェースと DB2 Universal JDBC ドライバーを使用したデータ・ソースへの接続

JDBC アプリケーションは、 java.sql パッケージの一部である JDBC DriverManager インターフェースを使用して、データ・ソースへの接続を確立できます。

最初に Java™ アプリケーションは、Class.forName メソッドを呼び出して、 JDBC ドライバーをロードします。アプリケーションがドライバーをロードした後で、DriverManager.getConnection メソッドを呼び出すことにより、データベース・サーバーに接続します。

DB2 Universal JDBC ドライバーの場合、以下の引き数を設定して Class.forName メソッドを呼び出し、ドライバーをロードできます。

```
com.ibm.db2.jcc.DB2Driver
```

以前の JDBC ドライバーとの互換性を保つためには、以下の引き数を代わりに使用できます。

```
COM.ibm.db2os390.sqlj.jdbc.DB2SQLJDriver
```

以下のコードは、DB2 Universal JDBC ドライバーのロードを示しています。

```
try {
  // Load the DB2® Universal JDBC Driver with DriverManager
  Class.forName("com.ibm.db2.jcc.DB2Driver");
} catch (ClassNotFoundException e) {
   e.printStackTrace();
}
```

catch ブロックは、ドライバーが見つからなかった場合にエラーをプリントするのに使用されます。

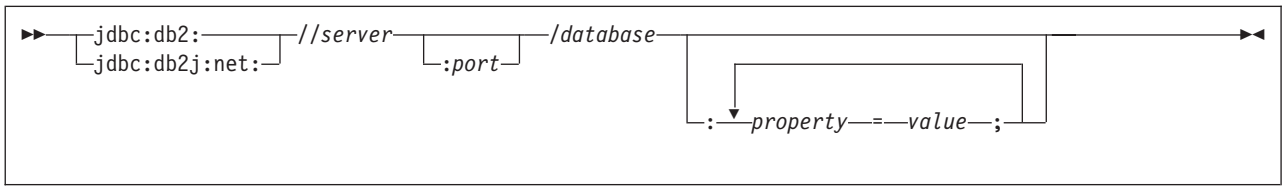
ドライバーのロード後、DriverManager.getConnection メソッドを呼び出して、データ・ソースへ接続します。以下のいずれかの形式の getConnection を使用できます。

```
getConnection(String url);
getConnection(String url, user, password);
getConnection(String url, java.util.Properties info);
```

url 引き数はデータ・ソースを表し、使用する JDBC 接続のタイプを示します。

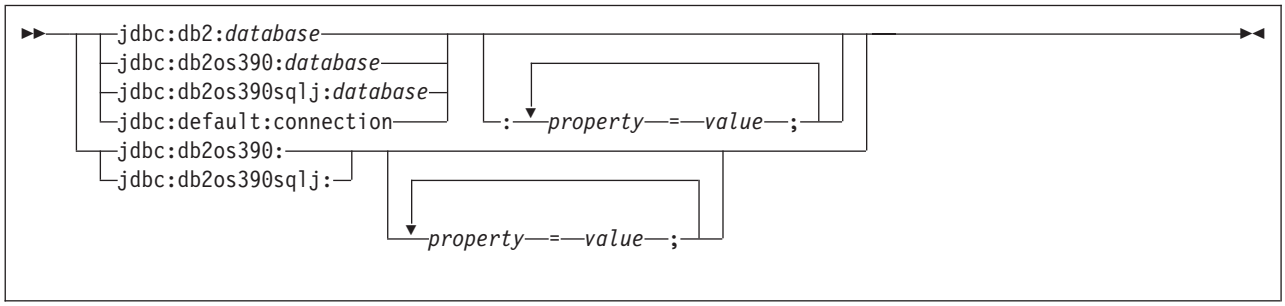
DB2 Universal JDBC Driver type 4 connectivity の場合、以下の形式の URL を指定します。

Universal Type 4 Connectivity での URL の構文:



DB2 Universal JDBC Driver type 2 connectivity の場合、以下のいずれかの形式の URL を指定します。

Universal Type 2 Connectivity での URL の構文:



URL の各部には以下のような意味があります。

jdbc:db2:、jdbc:db2j:net:
URL の最初の部分の意味は、以下のとおりです。

jdbc:db2:
接続が DB2 UDB ファミリー内のサーバーへ行われることを示しています。

jdbc:db2j:net:
接続がリモート IBM® Cloudscape™ サーバーへ行われることを示しています。

server
データベース・サーバーのドメイン・ネームまたは IP アドレス。

port
データベース・サーバーに割り当てられる TCP/IP サーバーのポート番号。これは、0 から 65535 の間の整数です。デフォルトは 446 です。

database
データベース・サーバーの名前。この名前は、Universal Type 4 Connectivity か Universal Type 2 Connectivity のどちらが使用されるかによって異なります。

Universal Type 4 Connectivity の場合:

- 接続を DB2 for z/OS サーバーへ行う場合、*database* 値は、インストール中に定義される DB2 ロケーション名です。この値の中の文字はすべて大文字でなければなりません。サーバーで以下の SQL ステートメントを実行することにより、ロケーション名を判別できます。

  ```
  SELECT CURRENT SERVER FROM SYSIBM.SYSDUMMY1;
  ```

- 接続を DB2 UDB for Linux、for UNIX、および for Windows サーバーへ行う場合、 *database* は、インストール中に定義されるデータベース名です。
- 接続を IBM Cloudscape サーバーへ行う場合、 *database* は、データベースを含むファイルの完全修飾名です。この名前は、二重引用符 (") で囲まなければなりません。たとえば、次のようにします。

  ```
  "c:/databases/testdb"
  ```

Universal Type 2 Connectivity の場合:

- serverName 接続プロパティーの値がヌルの場合、 *database* は、インストール中に定義されるデータベース名になります。 serverName プロパティーの値がヌルではない場合、 *database* はデータベース別名になります。

property=*value*;
JDBC 接続のプロパティー。これらのプロパティーの定義については、 『DB2 Universal JDBC ドライバーのプロパティー』を参照してください。

info 引き数は、接続に関するドライバー・プロパティーのセットを含む、タイプ `java.util.Properties` のオブジェクトです。 *info* 引き数の指定は、 URL での *property*=*value* ストリングの指定の代わりとなります。指定できるプロパティーについては、 『DB2 Universal JDBC ドライバーのプロパティー』を参照してください。

接続のユーザー ID およびパスワードの指定: 接続のユーザー ID およびパスワードを指定するには、いくつかの方法があります。

- `property=value;` 文節を設定して *url* を指定する、 `getConnection` メソッドの形式を使用し、 URL 内にユーザーおよびパスワード・プロパティーを含めます。
- *user* および *password* を指定する、 `getConnection` メソッドの形式を使用します。
- `java.util.Properties` オブジェクトでユーザーおよびパスワード・プロパティーを設定してから、 *info* を指定する `getConnection` メソッドの形式を使用します。

例: URL でのユーザー ID およびパスワードの設定

```
String url = "jdbc:db2://sysmvs1.stl.ibm.com:5021/san_jose:" +
  "user=db2adm;password=db2adm;";
                                        // Set URL for data source
Connection con = DriverManager.getConnection(url);
                                        // Create connection
```

例: ユーザーおよびパスワード・パラメーターでのユーザー ID およびパスワードの設定

```
String url = "jdbc:db2://sysmvs1.stl.ibm.com:5021/san_jose";
                                        // Set URL for data source
String user = "db2adm";
String password = "db2adm";
Connection con = DriverManager.getConnection(url, user, password);
                                        // Create connection
```

例: java.util.Properties オブジェクトでのユーザー ID およびパスワードの設定

```
Properties properties = new Properties(); // Create Properties object
properties.put("user", "db2adm");         // Set user ID for connection
properties.put("password", "db2adm");     // Set password for connection
String url = "jdbc:db2://sysmvs1.stl.ibm.com:5021/san_jose";
                                        // Set URL for data source
Connection con = DriverManager.getConnection(url, properties);
                                        // Create connection
```

関連概念:

- 488 ページの『DB2 Universal JDBC ドライバー使用時のセキュリティー』

関連資料:

- 408 ページの『DB2 Universal JDBC ドライバーのプロパティー』

DataSource インターフェースを使用したデータ・ソースへの接続

DriverManager を使用してデータ・ソースに接続する場合、アプリケーションが特定の JDBC ドライバー・クラス名とドライバー URL を識別しなければならないため、移植性が低下します。ドライバー・クラス名およびドライバー URL は、JDBC ベンダー、ドライバー・インプリメンテーション、およびデータ・ソースに固有のものです。アプリケーションがデータ・ソース間で移植可能でなければならない場合、 DataSource インターフェースを使用しなければなりません。

DataSource インターフェースを使用してデータ・ソースに接続するときには、DataSource オブジェクトを使用します。 DriverManager インターフェースを使用して行う場合のように、同じアプリケーションで DataSource オブジェクトを作成および使用できます。 図 7 では、DB2 Universal JDBC ドライバーでの例が示されています。

図 7. 同じアプリケーション内での DataSource オブジェクトの作成および使用

```
import java.sql.*;          // JDBC base
import javax.sql.*;        // JDBC 2.0 standard extension APIs
import com.ibm.db2.jcc.*; // DB2® Universal JDBC Driver   1
                          // interfaces
DB2SimpleDataSource db2ds=new DB2SimpleDataSource();      2
db2ds.setDatabaseName("db2loc1");                          3
                          // Assign the location name
db2ds.setDescription("Our Sample Database");
                          // Description for documentation
db2ds.setUser("john");
```

```
                        // Assign the user ID
db2ds.setPassword("db2");
                        // Assign the password
Connection con=db2ds.getConnection();                    4
                        // Create a Connection object
```

1 DataSource インターフェースのインプリメンテーションを含むパッケージをインポートします。

2 DB2DataSource オブジェクトを作成します。 DB2DataSource は、 DataSource インターフェースの DB2 インプリメンテーションの 1 つです。 DB2 の DataSource インプリメンテーションの詳細については、 『DataSource オブジェクトの作成および展開』を参照してください。

3 setDatabaseName、setDescription、 setUser、および setPassword メソッドは、属性を DB2DataSource オブジェクトに割り当てます。 DB2 Universal JDBC ドライバーで DB2DataSource オブジェクトに設定できる属性については、 『DB2 Universal JDBC ドライバーのプロパティー』を参照してください。

4 DB2DataSource オブジェクト db2ds が表すデータ・ソースへの接続を確立します。

しかし、DataSource オブジェクトの使用に関してより柔軟性のある方法は、システム管理者が WebSphere® や他の何らかのツールを使用して、それを個別に作成および管理することです。また、DataSource オブジェクトを作成および管理するプログラムは、 Java™ Naming and Directory Interface (JNDI) を使用して、 DataSource オブジェクトに論理名を割り当てます。そのような場合、DataSource オブジェクトを使用する JDBC アプリケーションは、論理名でオブジェクトを参照できるため、基礎データ・ソースに関する情報は必要ありません。さらに、システム管理者がデータ・ソース属性を変更できるため、ユーザーがアプリケーション・プログラムを変更する必要もありません。

WebSphere を使用した DataSource オブジェクトのデプロイについて調べるには、以下の Web の URL を参照してください。

http://www.ibm.com/software/webservers/appserv/

DataSource オブジェクトを自分でデプロイする方法を調べるには、『DataSource オブジェクトの作成およびデプロイ』を参照してください。

DataSource インターフェースと DriverManager インターフェースを同じアプリケーションで使用できますが、移植性を最大にするために、 DataSource インターフェースだけを使用して、接続を取得するようにお勧めします。

このトピックの残りの部分では、システム管理者がすでにオブジェクトを作成し、それに論理名を割り当てているというという状況で、 DataSource オブジェクトを使用して接続を作成する方法について説明します。

DataSource オブジェクトを使用して接続を取得するには、以下のステップに従う必要があります。

1. システム管理者から、接続するデータ・ソースの論理名を取得します。
2. 次のステップで使用する Context オブジェクトを作成します。 Context インターフェースは、JDBC ではなく、 Java Naming and Directory Interface (JNDI) の一部です。
3. アプリケーション・プログラムで JNDI を使用し、論理データ・ソース名に関連する DataSource オブジェクトを入手します。
4. DataSource.getConnection メソッドを使用して、接続を獲得します。

以下のいずれかの形式の getConnection メソッドを使用できます。

```
getConnection();
getConnection(String user, String password);
```

DataSource のデプロイ時に指定したのとは異なる、接続のユーザー ID およびパスワードを指定する必要がある場合は、2 番目の形式を使用してください。

図 8 は、接続先のデータ・ソースの論理名が jdbc/sampledb であるという状況で、DataSource オブジェクトを使用して接続を獲得するために、アプリケーション・プログラム内で必要なコードの例を示しています。特定のステートメントの右側にある番号は、前述のステップに対応しています。

図 8. DataSource オブジェクトを使用した接続の獲得

```
import java.sql.*;
import javax.naming.*;
import javax.sql.*;
...
Context ctx=new InitialContext();                                  2
DataSource ds=(DataSource)ctx.lookup("jdbc/sampledb");             3
Connection con=ds.getConnection();                                 4
```

関連タスク:

- 345 ページの『DataSource オブジェクトの作成およびデプロイ』

関連資料:

- 408 ページの『DB2 Universal JDBC ドライバーのプロパティー』

JDBC トランザクションの分離レベルの設定

JDBC プログラム内で作業単位の分離レベルを設定するには、Connection.setTransactionIsolation(int *level*) メソッドを使用します。 表 32 は、Connection.setTransactionIsolation メソッドと DB2® でそれらの相当するものに指定する *level* の値を示しています。

表 32. 相当する JDBC および DB2 分離レベル

JDBC 値	DB2 分離レベル
TRANSACTION_SERIALIZABLE	反復可能読み取り
TRANSACTION_REPEATABLE_READ	読み取り固定
TRANSACTION_READ_COMMITTED	カーソル固定
TRANSACTION_READ_UNCOMMITTED	非コミット読み取り (Uncommitted read)

分離レベルを変更できるのは、トランザクションの開始時のみです。

関連概念:

- 306 ページの『JDBC 接続オブジェクト』

JDBC 接続オブジェクト

いずれかの接続方法でデータ・ソースに接続する際に、データ・ソースへの接続を表す、 Connection オブジェクトを作成します。この Connection オブジェクトを使用して、以下のことを実行します。

- SQL ステートメントを実行するために、Statement、PreparedStatement、および CallableStatement オブジェクトを作成します。このことについては、 『JDBC アプリケーションでの SQL の実行』を参照してください。
- 接続先のデータ・ソースについての情報を集めます。このプロセスについては、『データ・ソースの情報入手のための DatabaseMetaData の使用』 で説明されています。
- トランザクションをコミットまたはロールバックします。トランザクションをコミットするときは、手動でも自動でも構いません。これらの操作については、『JDBC トランザクションのコミットまたはロールバック』を参照してください。
- データ・ソースへの接続をクローズします。この操作については、 『JDBC データ・ソースへの接続のクローズ』を参照してください。

関連概念:

- 307 ページの『SQL を実行するための JDBC インターフェース』

関連タスク:

- 307 ページの『JDBC データ・ソースへの接続のクローズ』
- 306 ページの『JDBC トランザクションのコミットまたはロールバック』
- 334 ページの『DatabaseMetaData を使用したデータ・ソースの情報入手』

JDBC トランザクションのコミットまたはロールバック

JDBC では、トランザクションを明示的にコミットまたはロールバックするには、commit または rollback メソッドを使用します。たとえば、次のようにします。

```
Connection con;
...
con.commit();
```

自動コミット・モードがオンになっている場合、DB2® は、各 SQL ステートメントの完了のたびにコミット操作を実行します。自動コミット・モードがオンかどうかを判別するには、 Connection.getAutoCommit メソッドを呼び出します。自動コミット・モードをオンに設定するには、 Connection.setAutoCommit(true) メソッドを呼び出します。自動コミット・モードをオフに設定するには、 Connection.setAutoCommit(false) メソッドを呼び出します。

関連概念:

- 327 ページの『JDBC アプリケーションでのセーブポイント』

関連タスク:

- 337 ページの『JDBC アプリケーションでのバッチ更新の作成』
- 307 ページの『JDBC データ・ソースへの接続のクローズ』

JDBC データ・ソースへの接続のクローズ

データ・ソースへの接続が完了したら、データ・ソースへの接続をクローズすることは *重要* です。そうすることにより、Connection オブジェクトの DB2® および JDBC リソースがすぐに解放されます。データ・ソースへの接続をクローズするには、close メソッドを使用します。たとえば、次のようにします。

```
Connection con;
...
con.close();
```

自動コミット・モードがオンになっていない場合、接続をクローズする前に、接続を作業単位境界に移す必要があります。

関連概念:

- 297 ページの『JDBC アプリケーションによるデータ・ソースへの接続の方法』

SQL を実行するための JDBC インターフェース

従来の SQL プログラムでは、SQL ステートメントを実行して、表に対してデータを挿入、更新、および削除したり、表からデータを検索したり、またはストアード・プロシージャーを呼び出したりします。 JDBC プログラムで同じ機能を実行するには、以下のインターフェースで定義されているメソッドを呼び出します。

- Statement インターフェースは、すべての SQL ステートメントの実行をサポートしています。以下のインターフェースは、Statement インターフェースからメソッドを継承します。
  - PreparedStatement インターフェースは、入力パラメーター・マーカーを含む SQL ステートメントをサポートしています。パラメーター・マーカーは入力変数を表します。 PreparedStatement インターフェースは、パラメーター・マーカーを使わない SQL ステートメントで使用することもできます。

    DB2 Universal JDBC ドライバーを使用する場合、PreparedStatement インターフェースを使用すると、入力パラメーターがあって出力パラメーターがなく、結果セットを戻さないストアード・プロシージャーを呼び出すことができます。
  - CallableStatement インターフェースは、ストアード・プロシージャーの呼び出しをサポートします。

    CallableStatement インターフェースを使用して、入力パラメーター、出力パラメーター、または入出力パラメーターを持つストアード・プロシージャー、あるいはパラメーターのないストアード・プロシージャーを呼び出せます。 DB2 Universal JDBC ドライバーを使用する場合、Statement インターフェースを使用して、ストアード・プロシージャーを呼び出すこともできますが、それらのストアード・プロシージャーはパラメーターを持つことができません。
- ResultSet インターフェースは、照会で生成された結果にアクセスできます。 ResultSet インターフェースには、他の言語の SQL アプリケーションで使用されるカーソルと同じ目的があります。

JDBC インターフェースに関する DB2® サポートの完全なリストは、 『JDBC API 用ドライバー・サポートの比較』を参照してください。

関連タスク:

- 312 ページの『DB2 からデータを検索するための PreparedStatement.executeQuery メソッドの使用』
- 310 ページの『DB2 表のデータを更新するための PreparedStatement.executeUpdate メソッドの使用』
- 309 ページの『DB2 表からデータを検索するための Statement.executeQuery メソッドの使用』
- 308 ページの『DB2 オブジェクトを作成および変更するための Statement.executeUpdate メソッドの使用』

関連資料:

- 416 ページの『JDBC API 用ドライバー・サポートの比較』

DB2 オブジェクトを作成および変更するための Statement.executeUpdate メソッドの使用

Statement.executeUpdate メソッドを使用して、以下を行うことができます。

- CREATE、ALTER、DROP、GRANT、REVOKE などのデータ定義ステートメントを実行する。
- パラメーター・マーカーを含まない INSERT、UPDATE および DELETE ステートメントを実行する。
- DB2 Universal JDBC ドライバーの場合、CALL ステートメントを実行して、パラメーターを含まず、結果表を戻さないストアード・プロシージャーを呼び出す。

これらの SQL ステートメントを実行するには、以下のステップを実行する必要があります。

1. Connection.createStatement メソッドを呼び出して、Statement オブジェクトを作成する。
2. Statement.executeUpdate メソッドを呼び出して、SQL 操作を実行する。
3. Statement.close メソッドを呼び出して、Statement オブジェクトをクローズする。

たとえば、次の SQL ステートメントを実行したいとします。

```
UPDATE EMPLOYEE SET PHONENO='4657' WHERE EMPNO='000010'
```

以下のコードでは、Statement オブジェクト stmt を作成して UPDATE ステートメントを実行し、numUpd に更新された行の数を戻します。特定のステートメントの右側にある番号は、前述のステップに対応しています。

```
Connection con;
Statement stmt;
int numUpd;
...
stmt = con.createStatement();                // Create a Statement object  1
numUpd = stmt.executeUpdate(
  "UPDATE EMPLOYEE SET PHONENO='4657' WHERE EMPNO='000010'");              2
                                             // Perform the update
stmt.close();                                // Close Statement object     3
```

図 9. Statement.executeUpdate の使用

関連資料:

- 470 ページの『DB2 Universal JDBC ドライバーと他の DB2 JDBC ドライバーの JDBC との相違点』
- 416 ページの『JDBC API 用ドライバー・サポートの比較』

DB2 表からデータを検索するための Statement.executeQuery メソッドの使用

パラメーター・マーカーを含まない SELECT ステートメントを使用してデータを表から検索するには、 Statement.executeQuery メソッドを使用できます。このメソッドは、 ResultSet オブジェクトに結果表を戻します。結果表が得られたら、 ResultSet メソッドを使用して結果表の中を移動し、各行の列値を個々に取得する必要があります。

DB2 Universal JDBC ドライバーでは、Statement.executeQuery メソッドを使用して、ストアード・プロシージャー呼び出しからも結果セットを検索することができます。ただし、ストアード・プロシージャーが戻す結果セットが 1 つだけの場合に限ります。ストアード・プロシージャーが複数の結果セットを戻す場合は、Statement.execute メソッドを使用する必要があります。詳しくは、『 JDBC アプリケーションでのストアード・プロシージャーからの複数の結果セットの検索』を参照してください。

このトピックでは、最も単純な種類の ResultSet を扱います。これは一度に 1 行だけ順方向に移動できる読み取り専用 ResultSet です。 DB2 Universal JDBC ドライバーは、更新可能およびスクロール可能な ResultSet もサポートしています。これらについては、 『JDBC アプリケーションでの ResultSet の更新可能性、スクロール可能性、および保持可能性の指定』で説明しています。

パラメーター・マーカーを含まない SELECT ステートメントを使用して表から行を検索するには、以下のステップを実行する必要があります。

1. Connection.createStatement メソッドを呼び出して、Statement オブジェクトを作成する。
2. Statement.executeQuery メソッドを呼び出して、 ResultSet オブジェクトで SELECT ステートメントによる結果表を取得する。
3. ループでの実行として、next メソッドを使用してカーソルの位置を指定し、ResultSet オブジェクトの現在行の各列のデータを、getXXX メソッドを使用し

て検索する。 XXX はデータ・タイプを表します。サポートされている getXXX および setXXX メソッドのリストについては、 『JDBC API 用ドライバー・サポートの比較』を参照してください。

4. ResultSet.close メソッドを呼び出し、ResultSet オブジェクトをクローズする。
5. Statement オブジェクトの使用を完了した後、Statement.close メソッドを呼び出し、クローズする。

たとえば、以下のコードは、従業員表からすべての行を検索する方法を示しています。特定のステートメントの右側にある番号は、前述のステップに対応しています。

```
String empNo;
Connection con;
Statement stmt;
ResultSet rs;
...
stmt = con.createStatement();     // Create a Statement object          1
rs = stmt.executeQuery("SELECT EMPNO FROM EMPLOYEE");                   2
                                  // Get the result table from the query
while (rs.next()) {               // Position the cursor                3
 empNo = rs.getString(1);         // Retrieve only the first column value
 System.out.println("Employee number = " + empNo);
                                  // Print the column value
}
rs.close();                       // Close the ResultSet                4
stmt.close();                     // Close the Statement                5
```

図 10. Statement.executeQuery の使用

関連タスク:

- 330 ページの『JDBC アプリケーションでのストアード・プロシージャーからの複数の結果セットの検索』
- 342 ページの『JDBC アプリケーションでの ResultSet の更新可能性、スクロール可能性、および保持可能性の指定』

関連資料:

- 416 ページの『JDBC API 用ドライバー・サポートの比較』

DB2 表のデータを更新するための PreparedStatement.executeUpdate メソッドの使用

Statement.executeUpdate メソッドは、DB2® 表を定数値で更新する場合に機能します。ただし、更新では多くの場合、値を変数で DB2 表に渡す必要があります。そのためには、PreparedStatement.executeUpdate メソッドを使用します。

DB2 Universal JDBC ドライバーでは、PreparedStatement.executeUpdate を使用して、ストアード・プロシージャーを呼び出すこともできます。ただし、ストアード・プロシージャーは、入力パラメーターがあり出力パラメーターのないもの、そして結果セットを戻さないものに限ります。

SQL ステートメントを何回も実行する場合は、SQL ステートメントを PreparedStatement として作成すると、パフォーマンスを向上できます。

たとえば、以下の UPDATE ステートメントで従業員表を更新できるのは、1 つの電話番号と 1 つの従業員番号についてのみです。

```
UPDATE EMPLOYEE SET PHONENO='4657' WHERE EMPNO='000010'
```

任意のセットの電話番号および従業員番号について従業員表を更新できるように、操作を汎用化したいとします。定数の電話番号および従業員番号を、以下のように変数で置き換える必要があります。

```
UPDATE EMPLOYEE SET PHONENO=? WHERE EMPNO=?
```

この形式の変数は、パラメーター・マーカーと呼ばれます。パラメーター・マーカーを含む SQL ステートメントを実行するには、 以下のステップを実行する必要があります。

1. Connection.prepareStatement メソッドを呼び出し、 PreparedStatement オブジェクトを作成する。
2. PreparedStatement.setXXX メソッドを呼び出し、変数に値を渡す。
3. PreparedStatement.executeUpdate メソッドを呼び出し、表を変数値で更新する。
4. PreparedStatement オブジェクトの使用を完了した後、PreparedStatement.close メソッドを呼び出し、クローズする。

以下のコードは、前述のステップを実行して、従業員番号が「000010」の従業員の電話番号を「4657」に更新します。特定のステートメントの右側にある番号は、前述のステップに対応しています。

```
Connection con;
PreparedStatement pstmt;
int numUpd;
...
pstmt = con.prepareStatement(
  "UPDATE EMPLOYEE SET PHONENO=? WHERE EMPNO=?");
                                // Create a PreparedStatement object    1
pstmt.setString(1,"4657");      // Assign value to first parameter      2
pstmt.setString(2,"000010");    // Assign value to second parameter
numUpd = pstmt.executeUpdate(); // Perform the update                   3
pstmt.close();                  // Close the PreparedStatement object   4
```

図 11. パラメーター・マーカーを含む SQL ステートメントにおける PreparedStatement.executeUpdate の使用

パラメーター・マーカーを含まないステートメントについても、PreparedStatement.executeUpdate メソッドを使用することができます。パラメーター・マーカーを含まない PreparedStatement オブジェクトを実行するステップは、ステップ 2 をスキップすることを除けば、パラメーター・マーカーを含む PreparedStatement オブジェクトを実行する場合に似ています。以下の例は、そのステップを示しています。

```
Connection con;
PreparedStatement pstmt;
int numUpd;
...
pstmt = con.prepareStatement(
  "UPDATE EMPLOYEE SET PHONENO='4657' WHERE EMPNO='000010'");
                                  // Create a PreparedStatement object   1
numUpd = pstmt.executeUpdate();   // Perform the update                  3
pstmt.close();                    // Close the PreparedStatement object  4
```

図 12. パラメーター・マーカーを含まない SQL ステートメントにおける PreparedStatement.executeUpdate の使用

関連資料:

- 470 ページの『DB2 Universal JDBC ドライバーと他の DB2 JDBC ドライバーの JDBC との相違点』
- 416 ページの『JDBC API 用ドライバー・サポートの比較』

DB2 からデータを検索するための PreparedStatement.executeQuery メソッドの使用

パラメーター・マーカーを含む SELECT ステートメントを使用して表からデータを検索するには、 PreparedStatement.executeQuery メソッドを使用します。このメソッドは、 ResultSet オブジェクトに結果表を戻します。結果表が得られたら、 ResultSet メソッドを使用して結果表の中を移動し、各行の列値を個々に取得する必要があります。

DB2 Universal JDBC ドライバーでは、PreparedStatement.executeQuery メソッドを使用してストアード・プロシージャー呼び出しからも結果セットを検索することができます。ただし、ストアード・プロシージャーが戻す結果セットが 1 つだけで、入力パラメーターしか持たない場合に限ります。ストアード・プロシージャーが複数の結果セットを戻す場合は、 Statement.execute メソッドを使用する必要があります。詳しくは、『 JDBC アプリケーションでのストアード・プロシージャーからの複数の結果セットの検索』を参照してください。

パラメーター・マーカーを含む SELECT ステートメントを使用して表から行を検索するには、以下のステップを実行する必要があります。

1. Connection.prepareStatement メソッドを呼び出し、 PreparedStatement オブジェクトを作成する。
2. PreparedStatement.setXXX メソッドを呼び出し、値を入力パラメーターに渡す。
3. PreparedStatement.executeQuery メソッドを呼び出し、 ResultSet オブジェクトで SELECT ステートメントによる結果表を取得する。
4. ループでの実行として、ResultSet.next メソッドを使用してカーソルの位置を指定し、 ResultSet オブジェクトの現在行の各列のデータを、getXXX メソッドを使用して検索する。
5. ResultSet.close メソッドを呼び出し、ResultSet オブジェクトをクローズする。

6. PreparedStatement オブジェクトの使用を完了した後、
   PreparedStatement.close メソッドを呼び出し、クローズする。

たとえば、以下のコードは、特定の従業員の従業員表から行を検索する方法を示しています。特定のステートメントの右側にある番号は、前述のステップに対応しています。

```
String empnum, phonenum;
Connection con;
PreparedStatement pstmt;
ResultSet rs;
...
pstmt = con.prepareStatement(
  "SELECT EMPNO, PHONENO FROM EMPLOYEE WHERE EMPNO=?");
                                  // Create a PreparedStatement object   1
pstmt.setString(1,"000010");      // Assign value to input parameter     2

rs = pstmt.executeQuery();        // Get the result table from the query 3
while (rs.next()) {               // Position the cursor                 4
 empnum = rs.getString(1);        // Retrieve the first column value
 phonenum = rs.getString(2);      // Retrieve the first column value
 System.out.println("Employee number = " + empnum +
   "Phone number = " + phonenum);
                                  // Print the column values
}
rs.close();                            // Close the ResultSet          5
pstmt.close();                    // Close the PreparedStatement       6
```

図 13. PreparedStatement.executeQuery の使用

パラメーター・マーカーを含まないステートメントについても、
PreparedStatement.executeQuery メソッドを使用することができます。照会を何回も行う場合は、SQL ステートメントを PreparedStatement として作成すると、パフォーマンスを向上できます。

関連タスク:

- 330 ページの『JDBC アプリケーションでのストアード・プロシージャーからの複数の結果セットの検索』

関連資料:

- 416 ページの『JDBC API 用ドライバー・サポートの比較』

CallableStatement メソッドを使用したストアード・プロシージャーの呼び出し

ストアード・プロシージャーを呼び出すには、CallableStatement クラスでメソッドを呼び出します。基本的なステップは以下のとおりです。

1. Connection.prepareCall メソッドを呼び出して、CallableStatement オブジェクトを作成します。
2. CallableStatement.setXXX メソッドを呼び出して、値を入力 (IN) パラメーターに渡します。
3. CallableStatement.registerOutParameter メソッドを呼び出して、出力専用 (OUT) パラメーターとするパラメーター、または入出力 (INOUT) パラメーターとするパラメーターを指定します。

4. 以下のいずれかのメソッドを呼び出して、ストアード・プロシージャーを呼び出します。

   CallableStatement.executeUpdate
   このメソッドは、ストアード・プロシージャーが結果セットを戻さない場合に呼び出します。

   CallableStatement.executeQuery
   このメソッドは、ストアード・プロシージャーが 1 つの結果セットを戻す場合に呼び出します。

   CallableStatement.execute
   このメソッドは、ストアード・プロシージャーが複数の結果セットを戻す場合に呼び出します。
5. ストアード・プロシージャーが結果セットを戻す場合、その結果セットを検索します。 JDBC アプリケーションでのストアード・プロシージャーからの複数の結果セットの検索を参照してください。
6. CallableStatement.getXXX メソッドを呼び出して、 OUT パラメーターまたは INOUT パラメーターから値を検索します。
7. CallableStatement オブジェクトの使用が終了したら、CallableStatement.close メソッドを呼び出してオブジェクトをクローズします。

次のコードは、1 つの入力パラメーターと 4 つの出力パラメーターが備えられていて、 ResultSet が戻されないストアード・プロシージャーを呼び出すものです。特定のステートメントの右側にある番号は、前述のステップに対応しています。

```
int ifcaret;
int ifcareas;
int xsbytes;
String errbuff;
Connection con;
CallableStatement cstmt;
ResultSet rs;
...
cstmt = con.prepareCall("CALL DSN8.DSN8ED2(?,?,?,?,?)");                1
                                // Create a CallableStatement object
cstmt.setString (1, "DISPLAY THREAD(*)");                                2
                                // Set input parameter (DB2 command)
cstmt.registerOutParameter (2, Types.INTEGER);                           3
                                // Register output parameters
cstmt.registerOutParameter (3, Types.INTEGER);
cstmt.registerOutParameter (4, Types.INTEGER);
cstmt.registerOutParameter (5, Types.VARCHAR);
cstmt.executeUpdate();          // Call the stored procedure             4
ifcaret = cstmt.getInt(2);      // Get the output parameter values       6
ifcareas = cstmt.getInt(3);
xsbytes = cstmt.getInt(4);
errbuff = cstmt.getString(5);
cstmt.close();                                                           7
```

図 14. ストアード・プロシージャー呼び出しでの CallableStatement メソッドの使用 (パラメーター・マーカー付き)

関連タスク:

- 330 ページの『JDBC アプリケーションでのストアード・プロシージャーからの複数の結果セットの検索』

関連資料:

- 470 ページの『DB2 Universal JDBC ドライバーと他の DB2 JDBC ドライバーの JDBC との相違点』
- 416 ページの『JDBC API 用ドライバー・サポートの比較』

DB2 Universal JDBC ドライバー使用時の SQLException の処理

すべての Java™ プログラムと同じく、エラー処理は try/catch ブロックを使用して行われます。メソッドは、エラーが発生すると例外を出し、 catch ブロックのコードがそれらの例外を処理します。

JDBC には、エラーを処理するための SQLException クラスが提供されています。すべての JDBC メソッドは、実行時にエラーが発生すると、 SQLException のインスタンスを出します。 JDBC の仕様によると、 SQLException オブジェクトには以下の情報が含まれています。
- エラーの記述を含む String オブジェクト、または NULL。
- SQLSTATE を含む String オブジェクト、または NULL。
- エラー・コードを含む int 値。
- 次の SQLException へのポインター、または NULL。

DB2 Universal JDBC ドライバーは、 SQLException クラスを拡張した、com.ibm.db2.jcc.DB2Diagnosable インターフェースを提供しています。DB2Diagnosable インターフェースでは、 DB2® へのアクセス時に発生したエラーに関する詳細な情報を入手できます。 JDBC ドライバーがエラーを検出すると、DB2Diagnosable は標準の SQLException クラスと同じ情報を提供します。ただし、DB2 がエラーを検出すると、 DB2Diagnosable は、エラーに関する追加情報を提供する以下のメソッドを追加します。

getSqlca
以下の情報を含む DB2Sqlca オブジェクトを戻します。
- SQL エラー・コード
- SQLERRMC 値
- SQLERRP 値
- SQLERRD 値
- SQLWARN 値
- SQLSTATE

getThrowable
SQLException の原因となった java.lang.Throwable オブジェクトを戻します。そのようなオブジェクトがない場合、ヌルを戻します。

printTrace
診断情報をプリントします。

DB2 Universal JDBC ドライバーの使用時に実行する JDBC プログラムで、SQLException を処理する際の基本的なステップは、以下のとおりです。

1. プログラムで com.ibm.db2.jcc.DB2Diagnosable インターフェースと com.ibm.db2.jcc.DB2Sqlca クラスにアクセスできるようにします。それらへの参照をすべて完全に修飾するか、または以下のようにしてそれらをインポートすることができます。

```
import com.ibm.db2.jcc.DB2Diagnosable;
import com.ibm.db2.jcc.DB2Sqlca;
```

2. SQLException を生成できるコードを try ブロックに組み込みます。
3. catch ブロックで、以下のステップをループで実行します。
   a. 最後の SQLException を検索したかどうかテストします。まだであれば、次のステップへ進みます。
   b. DB2Diagnosable オブジェクトがあるかどうかテストして、 DB2 専用情報が存在するかどうかを検査します。オブジェクトがある場合、以下のようにします。
      1) オプション: DB2Diagnosable.printTrace メソッドを呼び出して、すべての SQLException 情報を java.io.PrintWriter オブジェクトに書き込みます。
      2) DB2Diagnosable.getThrowable メソッドを呼び出して、基礎となる java.lang.Throwable が SQLException の原因となったのかどうかを判別します。
      3) DB2Diagnosable.getSqlca メソッドを呼び出して、 DB2Sqlca オブジェクトを検索します。
      4) DB2Sqlca.getSqlCode メソッドを呼び出して、 SQL エラー・コード値を検索します。
      5) DB2Sqlca.getSqlErrmc メソッドを呼び出して、すべての SQLERRMC 値を含むストリングを検索するか、 DB2Sqlca.getSqlErrmcTokens メソッドを呼び出して、配列内の SQLERRMC 値を検索します。
      6) DB2Sqlca.getSqlErrp メソッドを呼び出して、SQLERRP 値を検索します。
      7) DB2Sqlca.getSqlErrd メソッドを呼び出して、配列内の SQLERRD 値を検索します。
      8) DB2Sqlca.getSqlWarn メソッドを呼び出して、配列内の SQLWARN 値を検索します。
      9) DB2Sqlca.getSqlState メソッドを呼び出して、SQLSTATE 値を検索します。
      10) DB2Sqlca.getMessage メソッドを呼び出して、データベース・サーバーからエラー・メッセージのテキストを検索します。
   c. SQLException.getNextException メソッドを呼び出して、次の SQLException を検索します。

次のコードは、 DB2 Universal JDBC ドライバーに付属する DB2 バージョンの SQLException から情報を入手する方法を示しています。特定のステートメントの右側にある番号は、前述のステップに対応しています。

```
import java.sql.*;                      // Import JDBC API package
import com.ibm.db2.jcc.DB2Diagnosable;  // Import packages for DB2       1
import com.ibm.db2.jcc.DB2Sqlca;        // SQLException support
java.io.PrintWriter printWriter;        // For dumping all SQLException
                                        // information
...
try {                                                                   2
  // Code that could generate SQLExceptions
 ...
} catch(SQLException sqle) {
    while(sqle != null) {             // Check whether there are more    3a
                                      // SQLExceptions to process
      //=====> Optional DB2-only error processing
      if (sqle instanceof DB2Diagnosable) {                             3b
                                      // Check if DB2-only information exists
          com.ibm.db2.jcc.DB2Diagnosable diagnosable =
            (com.ibm.db2.jcc.DB2Diagnosable)sqle;
          diagnosable.printTrace (printWriter, "");                     3b1
          java.lang.Throwable throwable =
            diagnosable.getThrowable();                                 3b2
          if (throwable != null) {
            // Extract java.lang.Throwable information
            // such as message or stack trace.
            ...
          }
          DB2Sqlca sqlca = diagnosable.getSqlca();                      3b3
                                        // Get DB2Sqlca object
          if (sqlca != null) {          // Check that DB2Sqlca is not null
            int sqlCode = sqlca.getSqlCode(); // Get the SQL error code  3b4
            String sqlErrmc = sqlca.getSqlErrmc();                      3b5
                                        // Get the entire SQLERRMC
            String[] sqlErrmcTokens = sqlca.getSqlErrmcTokens();
                                        // You can also retrieve the
                                        // individual SQLERRMC tokens
            String sqlErrp = sqlca.getSqlErrp();                        3b6
                                        // Get the SQLERRP
            int[] sqlErrd = sqlca.getSqlErrd();                         3b7
                                        // Get SQLERRD fields
            char[] sqlWarn = sqlca.getSqlWarn();                        3b8
                                        // Get SQLWARN fields
            String sqlState = sqlca.getSqlState();                      3b9
                                        // Get SQLSTATE
            String errMessage = sqlca.getMessage();                     3b10
                                        // Get error message

            System.err.println ("Server error message: " + errMessage);

            System.err.println ("--------------- SQLCA ---------------");
            System.err.println ("Error code: " + sqlCode);
            System.err.println ("SQLERRMC: " + sqlErrmc);
            for (int i=0; i< sqlErrmcTokens.length; i++) {
              System.err.println ("  token " + i + ": " + sqlErrmcTokens[i]);
            }
```

図 15. DB2 Universal JDBC ドライバーの使用時の SQLException の処理 (1/2)

```
            System.err.println ( "SQLERRP: " + sqlErrp );
            System.err.println (
              "SQLERRD(1): " + sqlErrd[0] + "¥n" +
              "SQLERRD(2): " + sqlErrd[1] + "¥n" +
              "SQLERRD(3): " + sqlErrd[2] + "¥n" +
              "SQLERRD(4): " + sqlErrd[3] + "¥n" +
              "SQLERRD(5): " + sqlErrd[4] + "¥n" +
              "SQLERRD(6): " + sqlErrd[5] );
            System.err.println (
              "SQLWARN1: " + sqlWarn[0] + "¥n" +
              "SQLWARN2: " + sqlWarn[1] + "¥n" +
              "SQLWARN3: " + sqlWarn[2] + "¥n" +
              "SQLWARN4: " + sqlWarn[3] + "¥n" +
              "SQLWARN5: " + sqlWarn[4] + "¥n" +
              "SQLWARN6: " + sqlWarn[5] + "¥n" +
              "SQLWARN7: " + sqlWarn[6] + "¥n" +
              "SQLWARN8: " + sqlWarn[7] + "¥n" +
              "SQLWARN9: " + sqlWarn[8] + "¥n" +
              "SQLWARNA: " + sqlWarn[9] );
            System.err.println ("SQLSTATE: " + sqlState);
                                              // portion of SQLException
          }
        sqle=sqle.getNextException();     // Retrieve next SQLException    3c
      }
}
```

図 15. DB2 Universal JDBC ドライバーの使用時の SQLException の処理 (2/2)

関連資料:

• 479 ページの『DB2 Universal JDBC ドライバーが発行するエラー・コード』

DB2 JDBC Type 2 ドライバー使用時の SQLException の処理

すべての Java™ プログラムと同じく、エラー処理は try/catch ブロックを使用して行われます。メソッドは、エラーが発生すると例外を出し、 catch ブロックのコードがそれらの例外を処理します。

JDBC には、エラーを処理するための SQLException クラスが提供されています。すべての JDBC メソッドは、実行時にエラーが発生すると、 SQLException のインスタンスを出します。 JDBC の仕様によると、 SQLException オブジェクトには以下の情報が含まれています。

- エラーの記述を含む String オブジェクト、または NULL。
- SQLSTATE を含む String オブジェクト、または NULL。
- エラー・コードを含む int 値。
- 次の SQLException へのポインター、または NULL。

Linux、UNIX®、および Windows® 用 DB2® JDBC Type 2 ドライバー (DB2 JDBC Type 2 ドライバー) の下で実行する JDBC プログラムで SQLException を処理する際の基本的なステップは、以下のとおりです。

1. SQLException を生成できるコードを try ブロックに組み込みます。
2. catch ブロックで、以下のステップをループで実行します。
   a. 最後の SQLException を検索したかどうかテストします。まだであれば、次のステップへ進みます。
   b. SQLException からエラー情報を検索します。

c. SQLException.getNextException メソッドを呼び出して、次の SQLException を検索します。

次のコードは、 DB2 JDBC Type 2 ドライバーに付属する DB2 バージョンの SQLException を使用する catch ブロックを示しています。特定のステートメントの右側にある番号は、前述のステップに対応しています。

```
import java.sql.*;                      // Import JDBC API package
...
try {
  // Code that could generate SQLExceptions
 ...
} catch(SQLException sqle) {
    while(sqle != null) {             // Check whether there are more   1
      System.out.println("Message:  " + sqle.getMessage());              2
      System.out.println("SQLSTATE: " + sqle.getSQLState());
      System.out.println("SQL error code: " + sqle.getErrorCode());
      sqle=sqle.getNextException();     // Retrieve next SQLException    3
    }
}
```

図 16. DB2 Universal JDBC ドライバーの使用時の SQLException の処理

関連タスク:

• 319 ページの『DB2 Universal JDBC ドライバー使用時の SQLWarning の処理』

DB2 Universal JDBC ドライバー使用時の SQLWarning の処理

SQL エラーの場合と異なり、SQL 警告では JDBC メソッドから例外が出されません。代わりに、Connection、Statement、 PreparedStatement、CallableStatement、および ResultSet クラスには、 getWarnings メソッドが含まれており、 SQL ステートメントを実行した後はこのメソッドを呼び出して、 SQL 警告が生成されていないかどうか判別する必要があります。 getWarnings を呼び出すと SQLWarning オブジェクトが検索されます。汎用 SQLWarning オブジェクトには、以下の情報が含まれます。

- 警告の記述を含む String オブジェクト、または NULL。
- SQLSTATE を含む String オブジェクト、または NULL。
- エラー・コードを含む int 値。
- 次の SQLWarning を指すポインター、または NULL。

DB2 Universal JDBC ドライバーの使用時は、SQLException オブジェクトの場合と同様に、 SQLWarning オブジェクトにも DB2® 固有の情報が含まれます。SQLWarning オブジェクトについての DB2 固有の情報は、 SQLException オブジェクトについての DB2 固有の情報と同じです。

SQL 警告情報を取り出す基本的なステップは、以下のとおりです。

1. SQL ステートメントを実行するメソッドを呼び出した直後に、 getWarnings メソッドを呼び出して SQLWarning オブジェクトを検索する。
2. 以下のステップをループで実行する。
   a. SQLWarning オブジェクトが NULL かどうかをテストする。NULL でない場合は、次のステップを続けます。
   b. SQLWarning.getMessage メソッドを呼び出して、警告の記述を検索する。

c. SQLWarning.getSQLState メソッドを呼び出して、SQLSTATE 値を検索する。

d. SQLWarning.getErrorCode メソッドを呼び出して、エラー・コード値を検索する。

e. DB2 固有の警告情報が必要な場合は、 SQLException の場合に DB2 固有の情報を取得するときに実行するステップと同じステップを実行する。

f. SQLWarning.getNextWarning メソッドを呼び出して、次の SQLWarning を検索する。

以下のコードは、汎用 SQLWarning 情報を取得する方法を示しています。特定のステートメントの右側にある番号は、前述のステップに対応しています。

```
Connection con;
Statement stmt;
ResultSet rs;
SQLWarning sqlwarn;
...
stmt = con.createStatement();               // Create a Statement object
rs = stmt.executeQuery("SELECT * FROM EMPLOYEE");
                                  // Get the result table from the query
sqlwarn = stmt.getWarnings();     // Get any warnings generated          1
while (sqlwarn != null) {         // While there are warnings, get and   2a
                                  // print warning information
  System.out.println ("Warning description: " + sqlwarn.getMessage());   2b
  System.out.println ("SQLSTATE: " + sqlwarn.getSQLState());             2c
  System.out.println ("Error code: " + sqlwarn.getErrorCode());         2d
  sqlwarn=sqlwarn.getNextWarning();     // Get next SQLWarning           2f
}
```

図 17. SQLWarning の処理

関連タスク:

- 315 ページの『DB2 Universal JDBC ドライバー使用時の SQLException の処理』

DB2 JDBC Type 2 ドライバーの使用時の SQLWarning の処理

SQL エラーの場合と異なり、SQL 警告では JDBC メソッドから例外が出されません。代わりに、Connection、Statement、 PreparedStatement、CallableStatement、および ResultSet クラスには、 getWarnings メソッドが含まれており、 SQL ステートメントを実行した後はこのメソッドを呼び出して、 SQL 警告が生成されていないかどうか判別する必要があります。 getWarnings を呼び出すと SQLWarning オブジェクトが検索されます。

Linux、UNIX®、および Windows® 用 DB2® JDBC Type 2 ドライバー (DB2 JDBC Type 2 ドライバー) では、汎用 SQLWarning オブジェクトが生成されます。汎用 SQLWarning オブジェクトには、以下の情報が含まれます。

- 警告の記述を含む String オブジェクト、または NULL。
- SQLSTATE を含む String オブジェクト、または NULL。
- エラー・コードを含む int 値。
- 次の SQLWarning を指すポインター、または NULL。

SQL 警告情報を取り出す基本的なステップは、以下のとおりです。

1. SQL ステートメントを実行するメソッドを呼び出した直後に、 getWarnings メソッドを呼び出して SQLWarning オブジェクトを検索します。
2. 以下のステップをループで実行します。
   a. SQLWarning オブジェクトが NULL かどうかをテストします。まだであれば、次のステップへ進みます。
   b. SQLWarning.getMessage メソッドを呼び出して、警告の記述を取り出します。
   c. SQLWarning.getSQLState メソッドを呼び出して、SQLSTATE 値を取り出します。
   d. SQLWarning.getErrorCode メソッドを呼び出して、エラー・コード値を取り出します。
   e. SQLWarning.getNextWarning メソッドを呼び出して、次の SQLWarning を取り出します。

以下のコードは、汎用 SQLWarning 情報を取得する方法を示しています。特定のステートメントの右側にある番号は、前述のステップに対応しています。

```
Connection con;
Statement stmt;
ResultSet rs;
SQLWarning sqlwarn;
...
stmt = con.createStatement();     // Create a Statement object
rs = stmt.executeQuery("SELECT * FROM EMPLOYEE");
                                  // Get the result table from the query
sqlwarn = stmt.getWarnings();     // Get any warnings generated          1
while (sqlwarn != null) {         // While there are warnings, get and   2a
                                  // print warning information
  System.out.println ("Warning description: " + sqlwarn.getMessage());   2b
  System.out.println ("SQLSTATE: " + sqlwarn.getSQLState());             2c
  System.out.println ("Error code: " + sqlwarn.getErrorCode());         2d
  sqlwarn=sqlwarn.getNextWarning();     // Get next SQLWarning           2f
}
```

図 18. SQLWarning の処理

関連タスク:

• 318 ページの『DB2 JDBC Type 2 ドライバー使用時の SQLException の処理』

JDBC アプリケーション・プログラミングの高度な概念

以降のトピックでは、 JDBC アプリケーションの作成に関する高度な情報が含まれています。

DB2 Universal JDBC ドライバーがある JDBC アプリケーションでの LOB

DB2 Universal JDBC ドライバーには、JDBC 2.0 仕様のすべての LOB サポートが含まれています。このドライバーでは、追加メソッドでの LOB と追加データ・タイプもサポートされています。

CLOB データは、常に Unicode ストリームとしてデータベース・サーバーに送信されます。データベース・サーバーは、そのデータをターゲット・コード・ページに変換します。

LOB ロケーターのサポート: DB2 Universal JDBC ドライバー では、LOB ロケーターを使用して LOB 列のデータを検索できます。 JDBC で LOB ロケーターを使用して、 LOB 列からデータを検索するには、 `fullyMaterializeLobData` プロパティーを `false` に設定する必要があります。プロパティーについては、 『DB2® Universal JDBC ドライバーのプロパティー』で説明されています。

`fullyMaterializeLobData` は、両方向スクロール・カーソルを使用して取り出されたストアード・プロシージャー・パラメーターまたは LOB には影響を与えません。両方向スクロール・カーソルを使用して、 OS/390® または z/OS™ 環境の DB2 UDB サーバーからデータを取り出すときには、 JDBC は必ず LOB ロケーターを使用して LOB 列からデータを検索します。

他の言語の場合と同様、Java アプリケーションにおける LOB ロケーターは、単一のデータベースと関連付けられています。単一の LOB ロケーターを使用して、 2 つの異なるデータベース の間でデータを移動させることはできません。 LOB データを 2 つのデータベース間で移動するには、最初のデータベースにある表から LOB データを検索するときにそれをマテリアライズし、次いでそのデータを 2 番目のデータベースにある表に挿入する必要があります。

DB2 Universal JDBC ドライバーでサポートされている他のメソッド: JDBC 仕様のメソッドに加えて、DB2 Universal JDBC ドライバーには以下のメソッドでの LOB サポートがあります。

- BLOB 列を以下の `ResultSet` メソッドの引き数として指定し、 BLOB 列からデータを検索することができます。
  - `getBinaryStream`
  - `getBytes`
- CLOB 列を以下の `ResultSet` メソッドの引き数として指定し、 CLOB 列からデータを検索することができます。
  - `getAsciiStream`
  - `getCharacterStream`
  - `getString`
  - `getUnicodeStream`
- 以下の `PreparedStatement` メソッドを使用して、 BLOB 列に対応するパラメーターの値を設定できます。
  - `setBytes`
  - `setBinaryStream`
- 以下の `PreparedStatement` メソッドを使用して、 CLOB 列に対応するパラメーターの値を設定できます。
  - `setString`
  - `setAsciiStream`
  - `setUnicodeStream`
  - `setCharacterStream`
- 次の `CallableStatement` メソッドを使用して、 JDBC `CLOB` パラメーターの値を検索することができます。

– getString

DB2 Universal JDBC ドライバーで LOB を使用するときの制約事項: Universal Type 2 Connectivity を使用する場合、 DBCLOB OUT または INOUT パラメーターを持つストアード・プロシージャーは呼び出すことができません。

関連資料:

- 408 ページの『DB2 Universal JDBC ドライバーのプロパティー』
- 470 ページの『DB2 Universal JDBC ドライバーと他の DB2 JDBC ドライバーの JDBC との相違点』
- 416 ページの『JDBC API 用ドライバー・サポートの比較』

JDBC アプリケーションでの LOB 列データの検索または更新のための Java データ・タイプ

deferPrepares プロパティーが true に設定されており、なおかつ DB2 Universal JDBC ドライバーが PreparedStatement.setXXX 呼び出しを処理する場合、ドライバーがデータ・タイプを判別するための追加的な処理が必要になることがあります。この追加的な処理はパフォーマンスに影響を及ぼす可能性があります。

LOB 列とともに使用されるパラメーターのデータ・タイプを、 JDBC ドライバーがすぐに判別できない場合、 LOB データ・タイプと互換性のあるパラメーターのデータ・タイプを選択する必要があります。

BLOB 列の入力パラメーター:

BLOB 列の入力パラメーター、または BLOB 列への入力に使用される入出力パラメーターで、以下の技法のいずれかを使用できます。

- 以下のように、BLOB 列と完全に一致する java.sql.Blob 入力変数を使用します。

```
cstmt.setBlob(parmIndex, blobData);
```

- 以下のように、ターゲット・データ・タイプとして BLOB を指定する、CallableStatement.setObject 呼び出しを使用します。

```
byte[] byteData = {(byte)0x1a, (byte)0x2b, (byte)0x3c};
cstmt.setObject(parmInd, byteData, java.sql.Types.BLOB);
```

- CallableStatement.setBinaryStream 呼び出しで、java.io.ByteArrayInputStream のタイプの入力パラメーターを使用します。java.io.ByteArrayInputStream オブジェクトは、BLOB データ・タイプと互換性があります。この呼び出しでは、入力データの正確な長さを指定する必要があります。

```
java.io.ByteArrayInputStream byteStream =
  new java.io.ByteArrayInputStream(byteData);
int numBytes = byteData.length;
cstmt.setBinaryStream(parmIndex, byteStream, numBytes);
```

BLOB 列の出力パラメーター:

BLOB 列の出力パラメーター、または BLOB 列からの出力に使用される入出力パラメーターで、以下の技法を使用できます。

- CallableStatement.registerOutParameter 呼び出しを使用して、出力パラメーターのタイプが BLOB であるように指定します。そのようにすると、BLOB デー

タ・タイプと互換性のあるデータ・タイプの変数へ、パラメーター値を取り出すことができます。たとえば、以下のコードは byte[] 変数へ BLOB 値を取り出します。

```
cstmt.registerOutParameter(parmIndex, java.sql.Types.BLOB);
cstmt.execute();
byte[] byteData = cstmt.getBytes(parmIndex);
```

CLOB 列の入力パラメーター:

CLOB 列の入力パラメーター、または CLOB 列への入力に使用される入出力パラメーターで、以下の技法のいずれかを使用できます。

- 以下のように、CLOB 列と完全に一致する java.sql.CLOB 入力変数を使用します。

  ```
  cstmt.setClob(parmIndex, clobData);
  ```

- 以下のように、ターゲット・データ・タイプとして CLOB を指定する、CallableStatement.setObject 呼び出しを使用します。

  ```
  String charData = "CharacterString";
  cstmt.setObject(parmInd, charData, java.sql.Types.CLOB);
  ```

- 以下のいずれかのタイプのストリームの入力パラメーターを使用します。
  - cstmt.setCharacterStream 呼び出しでの java.io.StringReader 入力パラメーター:

    ```
    java.io.StringReader reader = new java.io.StringReader(charData);
    cstmt.setCharacterStream(parmIndex, reader, charData.length);
    ```

  - cstmt.setAsciiStream 呼び出しでの java.io.ByteArrayInputStream パラメーター (ASCII データ用):

    ```
    byte[] charDataBytes = charData.getBytes("US-ASCII");
    java.io.ByteArrayInputStream byteStream =
      new java.io.ByteArrayInputStream (charDataBytes);
    cstmt.setAsciiStream(parmIndex, byteStream, charDataBytes.length);
    ```

  これらの呼び出しでは、入力データの正確な長さを指定する必要があります。

- 以下のように、cstmt.setString 呼び出しで String の入力パラメーターを使用します。

  ```
  cstmt.setString(charData);
  ```

  データの長さが 32KB より大きい場合、 JDBC ドライバーは CLOB データ・タイプを入力データに割り当てます。

- 以下のように、 cstmt.setObject 呼び出しで String の入力パラメーターを使用し、ターゲット・データ・タイプとして VARCHAR または LONGVARCHAR を指定します。

  ```
  cstmt.setObject(parmIndex, charData, java.sql.Types.VARCHAR);
  ```

  データの長さが 32KB より大きい場合、 JDBC ドライバーは CLOB データ・タイプを入力データに割り当てます。

CLOB 列の出力パラメーター:

CLOB 列の出力パラメーター、または CLOB 列からの出力に使用される入出力パラメーターで、以下の技法のいずれかを使用できます。

- CallableStatement.registerOutParameter 呼び出しを使用して、出力パラメーターのタイプが CLOB であるように指定します。そのようにすると、CLOB デー

タ・タイプと互換性のあるデータ・タイプの変数へ、パラメーター値を取り出すことができます。たとえば、以下のコードは String 変数へ CLOB 値を取り出します。

```
cstmt.registerOutParameter(parmIndex, java.sql.Types.CLOB);
cstmt.execute();
String charData = cstmt.getString(parmIndex);
```

- CallableStatement.registerOutParameter 呼び出しを使用して、出力パラメーターのタイプが VARCHAR または LONGVARCHAR であるように指定します。

```
cstmt.registerOutParameter(parmIndex, java.sql.Types.VARCHAR);
cstmt.execute();
String charData = cstmt.getString(parmIndex);
```

この技法は、検索されるデータの長さが 32KB 以下であることが分かっている場合にのみ使用してください。そうでない場合、データが切り捨てられます。

関連概念:

- 321 ページの『DB2 Universal JDBC ドライバーがある JDBC アプリケーションでの LOB』

関連資料:

- 403 ページの『Java、JDBC、および SQL のデータ・タイプ』

DB2 Universal JDBC ドライバーを使用した JDBC での ROWID

DB2® UDB for z/OS® および DB2 UDB for iSeries™ では、 DB2 表内の列に対して ROWID データ・タイプをサポートしています。 ROWID は、表内の行を固有に識別する値です。

以下の ResultSet メソッドを使用して、 ROWID 列からデータを検索します。

- getBytes
- getObject

getObject では、 DB2 Universal JDBC ドライバーは DB2 のみのクラスである com.ibm.db2.jcc.DB2RowID のインスタンスを戻します。

以下の PreparedStatement メソッドを使用して、 ROWID 列に関連するパラメーターの値を設定します。

- setBytes
- setObject

setObject では、パラメーターのターゲット・タイプとして、 DB2 のみのタイプである com.ibm.db2.jcc.Types.ROWID、または com.ibm.db2.jcc.DB2RowID クラスのインスタンスを使用します。

例: com.ibm.db2.jcc.DB2Types.ROWID ターゲット・タイプでの PreparedStatement.setObject の使用: パラメーターを 1 に設定するには、以下の形式の SetObject メソッドを使用します。

```
ps.setObject(1, bytes[], com.ibm.db2.jcc.DB2Types.ROWID);
```

例: com.ibm.db2.jcc.DB2RowID ターゲット・タイプでの PreparedStatement.setObject の使用: rwid が com.ibm.db2.jcc.DB2RowID のインスタンスであると仮定します。パラメーターを 1 に設定するには、以下の形式の SetObject メソッドを使用します。

```
ps.setObject (1, rwid);
```

ROWID 出力パラメーターを設定して定義するストアード・プロシージャーを呼び出すには、そのパラメーターを、 com.ibm.db2.jcc.DB2Types.ROWID タイプとして登録します。

*例: com.ibm.db2.jcc.DB2Types.ROWID パラメーター・タイプでの CallableStatement.registerOutParameter の使用:*com.ibm.db2.jcc.DB2Types.ROWID データ・タイプとして、 CALL ステートメントのパラメーター 1 を登録するには、以下の形式の registerOutParameter メソッドを使用します。

```
cs.registerOutParameter(1, com.ibm.db2.jcc.DB2Types.ROWID)
```

関連資料:

- 403 ページの『Java、JDBC、および SQL のデータ・タイプ』

JDBC アプリケーションでの特殊タイプ

特殊タイプとは、組み込み SQL データ・タイプとして内部的に表される、ユーザー定義のデータ・タイプです。特殊タイプは、SQL ステートメント CREATE DISTINCT TYPE を実行して作成します。

JDBC プログラムでは、 executeUpdate メソッドを使用して、 CREATE DISTINCT TYPE ステートメントを実行することで、特殊タイプを作成することができます。また、executeUpdate を使用して、該当するタイプの列を含む表を作成することも可能です。該当するタイプの列からデータを検索するか、該当するタイプの列を更新するときには、特殊タイプのベースとなる組み込みタイプに対応するデータ・タイプと共に、 Java™ ID を使用します。

以下の例では、INTEGER タイプをベースにした特殊タイプを作成し、該当タイプの列を含む表を作成し、行をその表に挿入し、そして表から行を検索します。

```
Connection con;
Statement stmt;
ResultSet rs;
String empNumVar;
int shoeSizeVar;
...
stmt = con.createStatement();                // Create a Statement object
stmt.executeUpdate(
  "CREATE DISTINCT TYPE SHOESIZE AS INTEGER");
                                            // Create distinct type
stmt.executeUpdate(
  "CREATE TABLE EMP_SHOE (EMPNO CHAR(6), EMP_SHOE_SIZE SHOESIZE)");
                                            // Create table with distinct type
stmt.executeUpdate("INSERT INTO EMP_SHOE " +
  "VALUES ('000010', 6)");                  // Insert a row
rs=stmt.executeQuery("SELECT EMPNO, EMP_SHOE_SIZE FROM EMP_SHOE);
                                            // Create ResultSet for query
while (rs.next()) {
  empNumVar = rs.getString(1);              // Get employee number
  shoeSizeVar = rs.getInt(2);               // Get shoe size (use int
                                            // because underlying type
                                            // of SHOESIZE is INTEGER)
  System.out.println("Employee number = " + empNumVar +
    " Shoe size = " + shoeSizeVar);
}
rs.close();                                 // Close ResultSet
stmt.close();                               // Close Statement
```

図 19. 特殊タイプの作成および使用

関連資料:

- 「*SQL リファレンス 第 2 巻*」の『CREATE DISTINCT TYPE ステートメント』

JDBC アプリケーションでのセーブポイント

SQL セーブポイントは、作業単位内でのある特定のポイント・イン・タイムにおけるデータおよびスキーマの状態を表します。セーブポイントの設定、セーブポイントの解除、セーブポイントが表す状態へのデータとスキーマのリストアを行う SQL ステートメントがあります。

DB2 Universal JDBC ドライバーでは、セーブポイントを使用するために以下のメソッドをサポートしています。

Connection.setSavepoint() または Connection.setSavepoint(String *name*)
セーブポイントを設定します。これらのメソッドは、後から releaseSavepoint または rollback 操作で使用される Savepoint オブジェクトを戻します。

これらのいずれかのメソッドを実行すると、 DB2® は、 ON ROLLBACK RETAIN CURSORS を含む、以下のフォームの SAVEPOINT ステートメントを実行します。

Connection.releaseSavepoint(Savepoint *savepoint*)
指定されたセーブポイント、およびそれ以後に確立されたすべてのセーブポイントを解放します。

Connection.rollback(Savepoint *savepoint*)
指定されたセーブポイントまで作業をロールバックします。

DatabaseMetaData.supportsSavepoints()
データ・ソースがセーブポイントをサポートしているかどうかを示します。

以下の例は、セーブポイントの設定、セーブポイントへのロールバック、およびセーブポイントの解除を行う方法を示しています。

```
Connection con;
Statement stmt;
ResultSet rs;
String empNumVar;
int shoeSizeVar;
...
con.setAutoCommit(false);                    // set autocommit OFF
stmt = con.createStatement();                // Create a Statement object
stmt.executeUpdate(
  "CREATE DISTINCT TYPE SHOESIZE AS INTEGER");
                                            // Create distinct type
con.commit();                                // Commit the create
stmt.executeUpdate(
  "CREATE TABLE EMP_SHOE (EMPNO CHAR(6), EMP_SHOE_SIZE SHOESIZE)");
                             // Create table with distinct type
con.commit();                                // Commit the create
stmt.executeUpdate("INSERT INTO EMP_SHOE " +
  "VALUES ('000010', 6)");                   // Insert a row
Savepoint savept = con.setSavepoint();       // Create a savepoint
...
stmt.executeUpdate("INSERT INTO EMP_SHOE " +
  "VALUES ('000020', 10)");                  // Insert another row
conn.rollback(savept);                       // Roll back work to the point
                              //  after the first insert
...
con.releaseSavepoint(savept);                // Release the savepoint
stmt.close();                                // Close the Statement
```

図 20. JDBC アプリケーションでのセーブポイントの設定、ロールバック、および解除

関連タスク:

- 306 ページの『JDBC トランザクションのコミットまたはロールバック』

関連資料:

- 416 ページの『JDBC API 用ドライバー・サポートの比較』

JDBC アプリケーションでの識別列の値の検索

識別列とは、DB2® がそれぞれの行に自動的に数値を生成するための方法を提供する、DB2 表の列です。 0 の位取りの完全な数値タイプ (SMALLINT、 INTEGER、BIGINT、ゼロの位取りの DECIMAL、またはこれらのいずれかのタイプに基づく特殊タイプ) を持つ列を定義するときに、 AS IDENTITY 文節を指定することにより、 CREATE TABLE または ALTER TABLE ステートメントに識別列を定義します。

DB2 Universal JDBC ドライバーを使用している場合、 JDBC 3.0 メソッドを使用して DB2 表から識別列を検索できます。 JDBC プログラムでは、識別列は自動生成キーと呼ばれます。表から自動生成キーを検索できるようにするには、自動生成キーの値を検索する行をいつ挿入するか示す必要があります。このことは、Connection.prepareStatement、Statement.executeUpdate、または Statement.execute メソッド呼び出しにフラグを設定することによって実行しま

す。実行されるステートメントは、 INSERT ステートメントまたは SELECT ステートメント内の INSERT でなければなりません。それ以外の場合、JDBC ドライバーはフラグを設定したパラメーターを無視します。

DB2 表から自動生成キーを検索するには、以下のステップを実行する必要があります。

1. 以下のメソッドのいずれかを使用して、自動生成キーを戻すことを示します。
   - PreparedStatement.executeUpdate メソッドを使用して行を挿入する予定の場合、いずれかのフォームの Connection.prepareStatement メソッドを呼び出して、 PreparedStatement オブジェクトを作成します。

     このフォームは、識別列をサポートする任意のデータベース・サーバー上の表に使用します。

     ```
     Connection.prepareStatement(sql-statement,
       Statement.RETURN_GENERATED_KEYS);
     ```

     このフォームは、識別列および SELECT 内の INSERT をサポートする任意のデータベース・サーバー上の表にのみ使用します。

     ```
     Connection.prepareStatement(sql-statement, String [] columnNames);
     ```
   - Statement.executeUpdate メソッドを使用して行を挿入する場合、いずれかのフォームの Statement.executeUpdate メソッドを呼び出します。

     このフォームは、識別列をサポートする任意のデータベース・サーバー上の表に使用します。

     ```
     Statement.executeUpdate(sql-statement, Statement.RETURN_GENERATED_KEYS);
     ```

     このフォームは、識別列および SELECT 内の INSERT をサポートする任意のデータベース・サーバー上の表にのみ使用します。

     ```
     Statement.executeUpdate(sql-statement, String [] columnNames);
     ```
   - Statement.execute メソッドを使用して行を挿入する場合、いずれかのフォームの Statement.execute メソッドを呼び出します。

     このフォームは、識別列をサポートする任意のデータベース・サーバー上の表に使用します。

     ```
     Statement.execute(sql-statement, Statement.RETURN_GENERATED_KEYS);
     ```

     このフォームは、識別列および SELECT 内の INSERT をサポートする任意のデータベース・サーバー上の表にのみ使用します。

     ```
     Statement.execute(sql-statement, String [] columnNames);
     ```
2. PreparedStatement.getGeneratedKeys メソッドまたは Statement.getGeneratedKeys メソッドを呼び出して、自動生成キーの値を含む ResultSet オブジェクトを検索します。

   ResultSet での自動生成キーのデータ・タイプは、対応する列のデータ・タイプに関係なく DECIMAL です。

次のコードは、識別列を含む表を作成し、その表に行を挿入し、識別列の自動生成キーの値を検索します。特定のステートメントの右側にある番号は、前述のステップに対応しています。

```
Connection con;
Statement stmt;
ResultSet rs;
java.math.BigDecimal iDColVar;
...
stmt = con.createStatement();         // Create a Statement object

stmt.executeUpdate(
  "CREATE TABLE EMP_PHONE (EMPNO CHAR(6), PHONENO CHAR(4), " +
    "IDENTCOL INTEGER GENERATED ALWAYS AS IDENTITY)");
                                      // Create table with identity column
stmt.executeUpdate("INSERT INTO EMP_PHONE " +                          1
  "VALUES ('000010', "5555")",        // Insert a row
  Statement.RETURN_GENERATED_KEYS);   // Indicate you want automatically
                                      // generated keys
rs = stmt.getGeneratedKeys();         // Retrieve the automatically       2
                                      // generated key value in a ResultSet.
                                      // Only one row is returned.
                                             // Create ResultSet for query
while (rs.next()) {
  idColVar = rs.getBigDecimal(1);     // Get automatically generated key
                                      // value
  System.out.println("automatically generated key value = " + idColVar);
}
rs.close();                           // Close ResultSet
stmt.close();                         // Close Statement
```

図 21. 自動生成キーの検索

関連概念:

- 734 ページの『ID 列』

関連タスク:

- 310 ページの『DB2 表のデータを更新するための PreparedStatement.executeUpdate メソッドの使用』
- 308 ページの『DB2 オブジェクトを作成および変更するための Statement.executeUpdate メソッドの使用』

関連資料:

- 416 ページの『JDBC API 用ドライバー・サポートの比較』

JDBC アプリケーションでのストアード・プロシージャーからの複数の結果セットの検索

結果セットを戻すストアード・プロシージャーを呼び出す場合、結果セットを検索するコードを組み込む必要があります。実行するステップは、戻される結果セットの数を知っているかどうか、およびそれらの結果セットの内容を知っているかどうかによって異なります。

結果セットの既知の数の検索:

結果セットの数とその内容を知っている場合に結果セットを検索するには、以下のステップを実行します。

1. Statement.execute メソッドまたは PreparedStatement.execute メソッドを呼び出して、ストアード・プロシージャーを呼び出します。ストアード・プロシージャーに入力パラメーターが含まれている場合、 PreparedStatement.execute を使用します。
2. getResultSet メソッドを呼び出して、最初の結果セットを入手します。これは、 ResultSet オブジェクトにあります。
3. ループ内で next メソッドを使用してカーソルを配置し、 getXXX メソッドを使用して、 ResultSet オブジェクトの現在の行の各列からデータを検索します。
4. *n* 個の結果セットが存在する場合、以下のステップを *n*-1 回繰り返します。
   a. getMoreResults メソッドを呼び出し、現在の結果セットをクローズして次の結果セットを指します。
   b. getResultSet メソッドを呼び出して、次の結果セットを入手します。これは、ResultSet オブジェクトにあります。
   c. ループ内で next メソッドを使用してカーソルを配置し、 getXXX メソッドを使用して、 ResultSet オブジェクトの現在の行の各列からデータを検索します。

次のコードは、2 つの結果セットの検索を示しています。最初の結果セットには INTEGER 列が、2 番目の結果セットには CHAR 列があります。特定のステートメントの右側にある番号は、前述のステップに対応しています。

```
CallableStatement cstmt;
ResultSet rs;
int i;
String s;
...
cstmt.execute();                          // Call the stored procedure      1
rs = cstmt.getResultSet();                // Get the first result set       2
while (rs.next()) {                       // Position the cursor            3
 i = rs.getInt(1);                        // Retrieve current result set value
 System.out.println("Value from first result set = " + i);
                                          // Print the value
}
cstmt.getMoreResults();                   // Point to the second result set 4a
                                          // and close the first result set
rs = cstmt.getResultSet();                // Get the second result set      4b
while (rs.next()) {                       // Position the cursor            4c
 s = rs.getString(1);                     // Retrieve current result set value
 System.out.println("Value from second result set = " + s);
                                          // Print the value
}
rs.close();                               // Close the result set
cstmt.close();                            // Close the statement
```

図 22. ストアード・プロシージャーからの既知の結果セットの検索

数が分からない結果セットの検索:

結果セットの数とその内容を知らない場合に結果セットを検索するには、 ResultSet が戻されなくなるまで ResultSet を検索する必要があります。 ResultSet ごとに、 ResultSetMetaData メソッドを使用して内容を判別します。 ResultSet の内容を判別することの詳細については、 『ResultSetMetaData を使用した ResultSet の情報入手』を参照してください。

ストアード・プロシージャーを呼び出したら、以下の基本的なステップに従い、数が分からない結果セットの内容を検索します。

1. ストアード・プロシージャーを呼び出した execute ステートメントで戻された値を検査します。戻された値が true であれば、少なくとも 1 つの結果セットがあるので、次のステップに進む必要があります。
2. 以下のステップをループで繰り返します。
   a. getResultSet メソッドを呼び出して、結果セットを入手します。これは、ResultSet オブジェクトにあります。このメソッドを呼び出すと、前の結果セットはクローズされます。
   b. ResultSet を、 『ResultSetMetaData を使用した ResultSet の情報入手』に示しているとおりに処理します。
   c. getMoreResults メソッドを呼び出して、別の結果セットがあるかどうかを判別します。 getMoreResults が true を戻す場合、ステップ 2a に進んで次の結果セットを入手します。

次のコードは、結果セットの数や内容を知らない場合に結果セットを検索する方法を示しています。特定のステートメントの右側にある番号は、前述のステップに対応しています。

```
CallableStatement cstmt;
ResultSet rs;
...
boolean resultsAvailable = cstmt.execute(); // Call the stored procedure
while (resultsAvailable) {                  // Test for result sets       1
 ResultSet rs = cstmt.getResultSet();       // Get a result set           2a
 ...                                       // process ResultSet
 resultsAvailable = cstmt.getMoreResults(); // Check for next result set  2c
                                            // (Also closes the
                                            // previous result set)
}
```

図 23. ストアード・プロシージャーからの数が分からない結果セットの検索

結果セットをオープンしたままにする:

図 23 では、 getMoreResults() を呼び出すと、前の getResultSet の呼び出しで戻される ResultSet オブジェクトがクローズされます。ただし、DB2 Universal JDBC ドライバーを使用している場合、現在の ResultSet または前にオープンした ResultSet をクローズするかどうかを決定するパラメーターを含む、 JDBC 3 形式の getMoreResults を呼び出すことが可能です。この形式の getMoreResults では、 JDK 1.4 以降が必要です。

以下の定数を指定できます。

Statement.KEEP_CURRENT_RESULT
次の ResultSet を検査しますが、現在の ResultSet のクローズは行いません。

Statement.CLOSE_CURRENT_RESULT
次の ResultSet を検査し、現在の ResultSet をクローズします。

Statement.CLOSE_ALL_RESULTS
すでにオープンされたままになっていた、すべての ResultSet をクローズします。

例えば、図 24 のコードでは、最後の ResultSet が検索されるまで、すべての ResultSet をオープンしておき、その後にすべての ResultSet をクローズします。

```
CallableStatement cstmt;
ResultSet rs;
...
boolean resultsAvailable = cstmt.execute(); // Call the stored procedure
while (resultsAvailable) {                  // Test for result sets
 ResultSet rs = cstmt.getResultSet();       // Get a result set
 ...                                       // process ResultSet
 resultsAvailable = cstmt.getMoreResults(Statement.KEEP_CURRENT_RESULT);
                                            // Check for next result set
                                            // but do not close
                                            // previous result set
}
resultsAvailable = cstmt.getMoreResults(Statement.CLOSE_ALL_RESULTS);
                                            // Close the result sets
```

図 24. 検索されたストアード・プロシージャー結果セットをオープンしたままにする

関連タスク:

- 333 ページの『ResultSetMetaData を使用した ResultSet の情報入手』

ResultSetMetaData を使用した ResultSet の情報入手

これまでに述べた表またはストアード・プロシージャーの結果セットのデータ検索は、表または結果セットの列数およびデータ・タイプが分かっていることを前提としていました。しかし実際にはいつも分かっているとは限りません。リモート・データ・ソースからデータを検索している場合などは特にそうです。不明の ResultSet を検索するプログラムを作成する場合は、ResultSetMetaData メソッドを使用して ResultSet の特性を判別してから、そのデータを検索する必要があります。

ResultSetMetaData メソッドは、以下のタイプの情報を提供します。

- ResultSet の列数。
- ResultSet の基礎表の修飾子。
- 列に関する情報。データ・タイプ、長さ、精度、位取り、および NULL 可能かどうかといった情報です。
- 列が読み取り専用かどうか。

executeQuery メソッドを呼び出して表の照会の ResultSet を生成した後に、以下の基本的なステップを実行して ResultSet の内容を判別してください。

1. ResultSet オブジェクトに対して getMetaData メソッドを呼び出し、ResultSetMetaData オブジェクトを作成する。
2. getColumnCount メソッドを呼び出し、ResultSet に含まれる列の数を判別する。
3. ResultSet の列ごとに ResultSetMetaData メソッドを実行し、列の特性を判別する。

同じ表定義に対して ResultSetMetaData.getColumnName を実行しても、データ・ソースによってはその結果が異なる可能性があります。その場合でも、戻される情報は、そのデータ・ソースの DB2® カタログに保管されている列名情報を正しく反映しています。

たとえば、以下のコードは、従業員表のすべての列のデータ・タイプを判別する方法を示しています。特定のステートメントの右側にある番号は、前述のステップに対応しています。

```
String s;
Connection con;
Statement stmt;
ResultSet rs;
ResultSetMetaData rsmtadta;
int colCount
int mtadtaint;
int i;
String colName;
String colType;
...
stmt = con.createStatement();     // Create a Statement object
rs = stmt.executeQuery("SELECT * FROM EMPLOYEE");
                                  // Get the ResultSet from the query
rsmtadta = rs.getMetaData();      // Create a ResultSetMetaData object  1
colCount = rsmtadta.getColumnCount();                                  2
                                  // Find number of columns in EMP
for (i=1; i<= colCount; i++) {                                         3
 colName = rsmtadta.getColumnName();    // Get column name
 colType = rsmtadta.getColumnTypeName();
                                        // Get column data type
 System.out.println("Column = " + colName +
  " is data type " + colType);
                                        // Print the column value
}
```

図 25. ResultSet に関する情報を取得するための ResultSetMetaData メソッドの使用

関連タスク:

- 313 ページの『CallableStatement メソッドを使用したストアード・プロシージャーの呼び出し』
- 309 ページの『DB2 表からデータを検索するための Statement.executeQuery メソッドの使用』

DatabaseMetaData を使用したデータ・ソースの情報入手

DatabaseMetaData インターフェースには、データ・ソースに関する情報を検索するメソッドが含まれています。これらのメソッドは、さまざまなデータ・ソースにアクセスできる汎用アプリケーションを作成する際に役立ちます。これらのタイプのアプリケーションでは、実行する前に、データ・ソースがさまざまなデータベース操作を処理できるかどうかをテストする必要があります。たとえば、ドライバーに対して JDBC 2.0 メソッドを呼び出す前に、データ・ソースのドライバーが JDBC 2.0 レベルかどうかを判別する必要があります。

DatabaseMetaData メソッドは、以下のタイプの情報を提供します。

- ANSI SQL レベルなど、データ・ソースがサポートするフィーチャー

- データ・ソースに関する特定の情報 (ドライバー・レベルなど)
- 制限 (索引に含めることのできる列の最大数など)
- データ・ソースがデータ定義ステートメント (CREATE、 ALTER、 DROP、 GRANT、 REVOKE) をサポートしているかどうか
- データ・ソースでのオブジェクトのリスト (表、索引、またはプロシージャーなど)
- バッチ更新やスクロール可能 `ResultSet` など、データ・ソースがさまざまな JDBC 2.0 機能をサポートするかどうか

アプリケーションが DB2® UDB for z/OS™ または OS/390® サーバーに接続する場合、 DB2 カタログ情報を必要とするいくつかの `DatabaseMetaData` メソッドを呼び出す前に、多数のストアード・プロシージャーをそのサーバーにインストールする必要があります。それらのストアード・プロシージャーは以下のとおりです。

- SQLCOLPRIVILEGES
- SQLCOLUMNS
- SQLFOREIGNKEYS
- SQLGETTYPEINFO
- SQLPRIMARYKEYS
- SQLPROCEDURECOLS
- SQLPROCEDURES
- SQLSPECIALCOLUMNS
- SQLSTATISTICS
- SQLTABLEPRIVILEGES
- SQLTABLES
- SQLUDTS

DB2 UDB for OS/390 and z/OS バージョン 7 または DB2 UDB for OS/390 バージョン 6 では、ストアード・プロシージャーは PTF に付属しています。これらの PTF は、以下の PTF 番号を使って、通常のサービス・チャネルを通して注文することができます。

表 33. DB2 Universal Database for z/OS and OS/390 のPTF

DB2 Universal Database for z/OS and OS/390 のバージョン	**PTF 番号**
バージョン 6	UQ72081 および UQ72082
バージョン 7	UQ72083

DB2 UDB for z/OS システム管理者に、これらのストアード・プロシージャーがインストールされているかどうか尋ねてください。

`DatabaseMetaData` メソッドを呼び出すには、以下の基本的なステップを実行する必要があります。

1. 接続で `getMetaData` メソッドを呼び出し、 `DatabaseMetaData` オブジェクトを作成します。
2. `DatabaseMetaData` メソッドを呼び出し、データ・ソースに関する情報を入手します。
3. メソッドが `ResultSet` を戻す場合、以下を行います。

a. ループ内で next メソッドを使用してカーソルを配置し、 getXXX メソッドを使用して、 ResultSet オブジェクトの現在の行の各列からデータを検索します。

b. close メソッドを呼び出し、 ResultSet オブジェクトをクローズします。

たとえば、以下のコードは DatabaseMetaData メソッドを使用して、ドライバー・バージョンを判別し、データ・ソースで使用可能なストアード・プロシージャーのリストを入手する方法を示すものです。特定のステートメントの右側にある番号は、前述のステップに対応しています。

```
Connection con;
DatabaseMetaData dbmtadta;
ResultSet rs;
int mtadtaint;
String procSchema;
String procName;
...
dbmtadta = con.getMetaData();     // Create the DatabaseMetaData object  1
mtadtaint = dmtadta.getDriverVersion();                                  2
                                  // Check the driver version
System.out.println("Driver version: " + mtadtaint);
rs = dbmtadta.getProcedures(null, null, "%");
                                  // Get information for all procedures
while (rs.next()) {               // Position the cursor                  3a
 procSchema = rs.getString("PROCEDURE_SCHEM");
                                  // Get procedure schema
 procName = rs.getString("PROCEDURE_NAME");
                                  // Get procedure name
 System.out.println(procSchema + "." + procName);
                                  // Print the qualified procedure name
}
rs.close();                       // Close the ResultSet                  3b
```

図 26. DatabaseMetaData メソッドを使用した、データ・ソースについての情報の入手

関連資料:

- 470 ページの『DB2 Universal JDBC ドライバーと他の DB2 JDBC ドライバーの JDBC との相違点』
- 416 ページの『JDBC API 用ドライバー・サポートの比較』

PreparedStatement のパラメーターの情報入手のための ParameterMetaData の使用

DB2 Universal JDBC ドライバーには、ParameterMetaData インターフェースのサポートが組み込まれています。 ParameterMetaData インターフェースには、PreparedStatement オブジェクト内のパラメーター・マーカーについての情報を検索するメソッドが含まれています。

ParameterMetaData メソッドは、以下のタイプの情報を提供します。

- 10 進パラメーターの精度および位取りを含む、パラメーターのデータ・タイプ。
- パラメーターの、データベース特有のタイプ名。パラメーターが特殊タイプで定義された表列に対応する場合、その名前は特殊タイプ名になります。
- パラメーターが NULL 可能かどうか。
- パラメーターが入力パラメーターか、出力パラメーターか。

- 数値パラメーターの値に符号を付けられるかどうか。
- PreparedStatement.setObject がパラメーター値を設定するときに使用する完全修飾 Java™ クラス名。

ParameterMetaData メソッドを呼び出すには、以下の基本的なステップを実行する必要があります。

1. Connection.prepareStatement メソッドを呼び出し、 PreparedStatement オブジェクトを作成する。
2. PreparedStatement.getParameterMetaData メソッドを呼び出し、ParameterMetaData オブジェクトを取り出す。
3. ParameterMetaData.getParameterCount を呼び出し、 PreparedStatement のパラメーターの数を判別する。
4. 個々のパラメーターに対して ParameterMetaData メソッドを呼び出す。

たとえば、以下のコードは、 ParameterMetaData メソッドを使用して SQL UPDATE ステートメント内のパラメーターの数およびデータ・タイプを判別する方法を示しています。特定のステートメントの右側にある番号は、前述のステップに対応しています。

```
Connection con;
ParameterMetaData pmtadta;
int mtadtacnt;
int sqlType;
...
pstmt = con.prepareStatement(
  "UPDATE EMPLOYEE SET PHONENO=? WHERE EMPNO=?");
                                  // Create a PreparedStatement object    1
pmtadta = pstmt.getParameterMetaData();                                   2
                                  // Create a ParameterMetaData object
mtadtacnt = pmtadta.getParameterCount();                                  3
                                  // Determine the number of parameters
System.out.println("Number of statement parameters: " + mtadtacnt);
for (int i = 1; i <= mtadtacnt; i++) {
  sqlType = pmtadta.getParameterType(i);                                  4
                                  // Get SQL type for each parameter
 System.out.println("SQL type of parameter " + i " is " + sqlType);
}
...
pstmt.close();                        // Close the PreparedStatement
```

図 27. ParameterMetaData メソッドを使用して PreparedStatement についての情報を入手する

関連資料:

- 416 ページの『JDBC API 用ドライバー・サポートの比較』

JDBC アプリケーションでのバッチ更新の作成

JDBC 2.0 以降をサポートする JDBC ドライバーは、バッチ更新をサポートします。バッチ更新を使用すると、DB2® 表の行を一度に 1 行ずつ更新するのではなく、 JDBC で一群の更新を同時に実行することができます。更新の同じバッチに含められるステートメントのことを、*バッチ可能* ステートメントといいます。

ステートメントに入力パラメーターかホスト式が含まれる場合、そのステートメントを含められるのは、同じステートメントの別のインスタンスを持つバッチだけで

す。このタイプのバッチを、*同種バッチ* といいます。ステートメントに入力パラメーターが含まれない場合、そのステートメントをバッチに含められるのは、そのバッチの他のステートメントに入力パラメーターまたはホスト式が含まれない場合だけです。このタイプのバッチを、*異種バッチ* といいます。同じバッチに含められる 2 つのステートメントを、*バッチ互換* といいます。

SQL 更新のバッチを作成、実行、および除去するには、以下の Statement メソッドを使用します。

- addBatch
- executeBatch
- clearBatch

パラメーターのバッチを作成して、1 つのステートメントを 1 つのバッチの中で複数回実行し、各実行で別のパラメーターを指定するには、次の PreparedStatement および CallableStatement メソッドを使用します。

- addBatch

入力パラメーターのない複数のステートメントを使用してバッチ更新を作成するには、次の基本ステップに従ってください。

1. Connection オブジェクトで AutoCommit を使用不可にします。
2. createStatement メソッドを呼び出して、Statement オブジェクトを作成します。
3. バッチで実行する SQL ステートメントごとに、addBatch メソッドを呼び出します。
4. executeBatch メソッドを呼び出して、ステートメントのバッチを実行します。
5. エラーを検査します。エラーがない場合、次のようにします。
   a. executeBatch 呼び出しが戻す配列から、各 SQL ステートメントに影響された行数を入手します。この数には、トリガーや参照保全制約によって影響された行は含まれません。
   b. commit メソッドを呼び出して、変更をコミットします。

複数の入力パラメーターがある 1 つのステートメントを使用してバッチ更新を作成するには、次の基本ステップに従ってください。

1. Connection オブジェクトで AutoCommit を使用不可にします。
2. prepareStatement メソッドを呼び出して、入力パラメーターのある SQL ステートメントのために、 PreparedStatement オブジェクトを作成します。
3. 入力パラメーター値ごとに、次のようにします。
   a. setXXX メソッドを実行して、入力パラメーターに値を割り当てます。
   b. addBatch メソッドを呼び出して、入力パラメーターをバッチに追加します。
4. executeBatch メソッドを呼び出して、すべてのパラメーターが含まれるステートメントを実行します。
5. エラーを検査します。エラーがない場合、次のようにします。
   a. executeBatch 呼び出しが戻す配列から、SQL ステートメントのそれぞれの実行で更新された行数を入手します。
   b. commit メソッドを呼び出して、変更をコミットします。

バッチ更新の例: 次のコード断片では、 2 つのパラメーターがバッチ化されます。2 つの入力パラメーターをとる UPDATE ステートメントが、各パラメーターごと

に 1 回、合計で 2 回実行されます。特定のステートメントの右側にある番号は、前述のステップに対応しています。

```
try {
...
  connection con.setAutoCommit(false);                    1
  PreparedStatement prepStmt = con.prepareStatement(
    "UPDATE DEPT SET MGRNO=? WHERE DEPTNO=?");            2
  prepStmt.setString(1,mgrnum1);                          3a
  prepStmt.setString(2,deptnum1);
  prepStmt.addBatch();                                    3b

  prepStmt.setString(1,mgrnum2);
  prepStmt.setString(2,deptnum2);
  prepStmt.addBatch();
  int [] numUpdates=prepStmt.executeBatch();              4
  for (int i=0; i < numUpdates.length; i++) {             5a
    if (numUpdates[i] == -2)
      System.out.println("Execution " + i +
        ": unknown number of rows updated");
    else
      System.out.println("Execution " + i +
        "successful: " numUpdates[i] + " rows updated");
  }
  con.commit();                                           5b
} catch(BatchUpdateException b) {
  // process BatchUpdateException
}
```

図 28. バッチ更新の実行

関連タスク:

- 306 ページの『JDBC トランザクションのコミットまたはロールバック』

関連資料:

- 470 ページの『DB2 Universal JDBC ドライバーと他の DB2 JDBC ドライバーの JDBC との相違点』

BatchUpdateException からの情報の取り出し

バッチでステートメントの実行中にエラーが発生した場合、処理は継続されます。ただし、executeBatch が BatchUpdateException を出します。
BatchUpdateException オブジェクトには以下の項目が含まれます。

- エラーの記述を含む String オブジェクト、または null
- 失敗した SQL ステートメントに関する SQLSTATE を含む String オブジェクト、または null
- エラー・コードを含む整数値、またはゼロ
- バッチ内の SQL ステートメントに関する更新カウントの整数配列、または null
- SQLException オブジェクトを指すポインター、または null

バッチ全体で 1 つの BatchUpdateException が出されます。 1 つ以上の SQLException オブジェクトが BatchUpdateException オブジェクトにチェーニングされます。 SQLException オブジェクトのチェーニングは、対応するステートメントがバッチに追加されたのと同じ順序で行われます。 SQLException オブジェクトをバッチ内のステートメントと付き合わせるのに役立つように、各 SQLException オブジェクトのエラー記述フィールドは、以下のストリングで始まっています。

```
Error for batch element #n:
```

n は、バッチ内のステートメントの番号です。

BatchUpdateException から情報を検索するには、以下のステップに従ってください。

1. BatchUpdateException.getUpdateCounts メソッドを使用して、各 SQL ステートメントが更新した行の番号を判別します。更新された行の番号を判別できない場合、getUpdateCounts は -2 を戻します。または、更新中にエラーが発生した場合、-3 を戻します。
2. SQLException メソッド getMessage、getSQLState、および getErrorCode を使用して、エラー、SQLSTATE、および最初のエラーのエラー・コードを検索します。
3. BatchUpdateException.getNextException メソッドを使用して、チェーニングされた SQLException を入手します。
4. ループ内で getMessage、getSQLState、getErrorCode、および getNextException メソッドの呼び出しを実行し、 SQLException に関する情報を取得して、次の SQLException を入手します。

BatchUpdateException からの情報の取得に関する例: 以下のコードの断片は、BatchUpdateException およびチェーニングされた SQLException オブジェクトのフィールドを取得する方法を示しています。特定のステートメントの右側にある番号は、前述のステップに対応しています。

```
try {
  // Batch updates
} catch(BatchUpdateException buex) {
    System.err.println("Contents of BatchUpdateException:");
    System.err.println(" Update counts: ");
    int [] updateCounts = buex.getUpdateCounts();             1
    for (int i = 0; i < updateCounts.length; i++) {
      System.err.println("  Statement " + i + ":" + updateCounts[i]);
    }
    System.err.println(" Message: " + buex.getMessage());     2
    System.err.println(" SQLSTATE: " + buex.getSQLState());
    System.err.println(" Error code: " + buex.getErrorCode());
    SQLException ex = buex.getNextException();                3
    while (ex != null) {                                      4
      System.err.println("SQL exception:");
      System.err.println(" Message: " + ex.getMessage());
      System.err.println(" SQLSTATE: " + ex.getSQLState());
      System.err.println(" Error code: " + ex.getErrorCode());
      ex = ex.getNextException();
    }
}
```

図 29. BatchUpdateException フィールドの検索

警告に関する情報を取得するには、executeBatch メソッドを実行したオブジェクトで、 Statement.getWarnings メソッドを使用します。それから、各 SQLWarning オブジェクトのエラー記述、SQLSTATE、エラー・コードを検索できます。

バッチ内のステートメントの実行に関する制約事項

- バッチ内の SELECT ステートメントの実行を試行する場合、BatchUpdateException が出されます。
- バッチ内で実行する CallableStatement オブジェクトには、出力パラメーターを含めることができます。ただし、出力パラメーターの値を検索することはできません。それを試行すると、BatchUpdateException が出されます。
- バッチ内で実行する CallableStatement オブジェクトからの ResultSet オブジェクトの検索は行えません。 BatchUpdateException は出されませんが、getResultSet メソッドの呼び出しで NULL 値が戻されます。

関連タスク:

- 337 ページの『JDBC アプリケーションでのバッチ更新の作成』

DB2 Universal JDBC ドライバー使用時の JDBC ResultSet の特性

ResultSet を順方向に 1 行ずつ移動することに加えて、以下のことも実行したい場合があります。

- 逆方向に移動、または特定の行に直接移動する。
- ResultSet の行を更新または削除する。
- COMMIT 後に ResultSet をオープンしたままにしておく。

以下の用語は、ResultSet の特性を表すために使用します。

スクロール可能性
カーソルを順方向、逆方向、あるいは特定の行に移動できるかどうか。

更新可能性
カーソルを使用して行を更新または削除できるかどうか。ストアード・プロシージャー ResultSet は更新できないため、この特性は、ストアード・プロシージャーから戻された ResultSet には当てはまりません。

保持可能性
COMMIT 後にカーソルがオープンしたままかどうか。

JDBC におけるスクロール可能 ResultSet は、 SCROLL と宣言された DB2® カーソルの結果表に相当します。両方向スクロール・カーソルは、*インセンシティブ* または*センシティブ* のいずれかです。インセンシティブの場合、カーソルをオープンした後の基礎表の変更はカーソルからは見えません。インセンシティブ・カーソルは読み取り専用です。センシティブの場合は、以下のようになります。

- カーソルにより基礎表に対して加えられた変更は、常にカーソルから見ることができる。
- 他の手段で基礎表に対して加えられた変更は、カーソルから見ることができる。DB2 では、行が FETCH INSENSITIVE で取り出された場合、他の手段で加えられた変更はカーソルからは見えません。行が FETCH SENSITIVE で取り出された場合、他の手段で加えられた変更はカーソルから見ることができます。 JDBC では、refreshRow メソッドを呼び出した後に getXXX メソッドを呼び出すことで、 FETCH SENSITIVE と同じになります。

JDBCの ResultSet は、データベース・サーバーが両方の属性をサポートしている場合、*静的* にも *動的* にもすることができます。プログラムの両方向スクロール・

カーソルを静的にするか動的にするかは、 cursorSensitivity プロパティーの設定により決定します。 cursorSensitivity についての詳細は、 『DB2 Universal JDBC ドライバーのプロパティー』を参照してください。

JDBC の ResultSet が静的である場合、 カーソルをオープンした後も、結果表のサイズや結果表の行の順序は変わりません。つまり、結果表に挿入を行うことはできません。そのため、結果表の行を削除すると削除ホールが発生します。現在行が削除ホールかどうかをテストするには、rowDeleted メソッドを使用します。 ResultSet をサポートしているメソッドの完全なリストは、 『JDBC API 用ドライバー・サポートの比較』を参照してください。

関連タスク:

- 342 ページの『JDBC アプリケーションでの ResultSet の更新可能性、スクロール可能性、および保持可能性の指定』

JDBC アプリケーションでの ResultSet の更新可能性、スクロール可能性、および保持可能性の指定

ResultSet のスクロール可能性、更新可能性、および保持可能性を指定するには、以下のステップを実行する必要があります。

1. ResultSet を定義する SELECT ステートメントに入力パラメーターがない場合は、 createStatement メソッドを呼び出して Statement オブジェクトを作成する。そうでない場合は、prepareStatement メソッドを呼び出し、 PreparedStatement オブジェクトを作成する。

   resultSetType、 *resultSetConcurrency*、または*resultSetHoldability* パラメーターを含む createStatement または prepareStatement メソッドの形式を指定する必要があります。

   スクロール可能性および更新可能性をサポートする createStatement メソッドの形式は、以下のようにします。

   ```
   createStatement(int resultSetType, int resultSetConcurrency);
   ```

   スクロール可能性、更新可能性、および保持可能性をサポートする createStatement メソッドの形式は、以下のようにします。

   ```
   createStatement(int resultSetType, int resultSetConcurrency,
    int resultSetHoldability);
   ```

   スクロール可能性および更新可能性をサポートする prepareStatement メソッドの形式は、以下のようにします。

   ```
   prepareStatement(String sql, int resultSetType,
    int resultSetConcurrency);
   ```

   スクロール可能性、更新可能性、保持可能性をサポートする prepareStatement メソッドの形式は、以下のようにします。

   ```
   prepareStatement(String sql, int resultSetType,
    int resultSetConcurrency, int resultSetHoldability);
   ```

   resultSetType および *resultSetConcurrency* の有効な値のリストについては、 343 ページの表 34 を参照してください。

表 34. スクロール可能 ResultSet の resultSetType および resultSetConcurrency の有効な組み合わせ

resultSetType の値	*resultSetConcurrency* の値
TYPE_FORWARD_ONLY	CONCUR_READ_ONLY
TYPE_FORWARD_ONLY	CONCUR_UPDATABLE
TYPE_SCROLL_INSENSITIVE	CONCUR_READ_ONLY
TYPE_SCROLL_SENSITIVE	CONCUR_READ_ONLY
TYPE_SCROLL_SENSITIVE	CONCUR_UPDATABLE

resultSetHoldability がとることのできる値は、 HOLD_CURSORS_OVER_COMMIT および CLOSE_CURSORS_AT_COMMIT の 2 つです。これらの値のいずれかを、*resultSetConcurrency* と *resultSetHoldability* の有効な組み合わせで指定します。ここで設定した値は、接続のデフォルトの保持可能性をオーバーライドします。

制約事項: ResultSet がスクロール可能で、 DB2 UDB for Linux, UNIX, Windows サーバーの表の列を指定するために ResultSet を使用する場合、ResultSet を定義する SELECT ステートメントでは以下のデータ・タイプの列を選択することができません。

- LONG VARCHAR
- LONG VARGRAPHIC
- DATALINK
- BLOB
- CLOB
- このリストにあるデータ・タイプのいずれかを基にした特殊タイプ。
- 構造化タイプ。

2. SELECT ステートメントに入力パラメーターがある場合は、setXXX メソッドを呼び出して入力パラメーターに値を渡す。
3. executeQuery メソッドを呼び出し、ResultSet オブジェクトで SELECT ステートメントによる結果表を取得する。
4. アクセスしたい行ごとに、以下を行う。
   a. 表 35 にリストされているメソッドのいずれかを使用して、カーソルの位置を指定する。

表 35. 両方向スクロール・カーソルの位置指定のための ResultSet メソッド

メソッド	カーソルの位置
first()	ResultSet の最初の行。
last()	ResultSet の最後の行。
next()[1]	ResultSet の次の行。
previous()[2]	ResultSet の直前の行。
absolute(int *n*)[3]	*n*>0 の場合、ResultSet の行 *n*。 *n*<0 の場合、 *m* を ResultSet の行数として、 ResultSet の行 *m+n+1*。
relative(int *n*)[4,5]	*n*>0 の場合、現在行から *n* 行後の行。*n*<0 の場合、現在行から *n* 行前の行。*n*=0 の場合、現在行。
afterLast()	ResultSet の最後の行の後。
beforeFirst()	ResultSet の最初の行の前。

表 35. 両方向スクロール・カーソルの位置指定のための ResultSet メソッド (続き)

メソッド	カーソルの位置

注:

1. カーソルが ResultSet の最初の行の前にある場合、このメソッドによるカーソルの位置は最初の行になります。
2. カーソルが ResultSet の最後の行の後にある場合、このメソッドによるカーソルの位置は最後の行になります。
3. *n* の絶対値が結果セットの行数よりも大きい場合、このメソッドによるカーソルの位置は、*n* が正のときは最後の行の後に、*n* が負のときは最初の行の前になります。
4. このメソッドを使用するには、カーソルが ResultSet の有効な行になければなりません。カーソルが最初の行の前または最後の行の後にある場合は、このメソッドによって SQLException が出されます。
5. *m* を ResultSet の行数、 *x* を ResultSet での現在行番号とします。 *n*>0 で、*x*+*n*>*m* の場合、ドライバーによってカーソルは最後の行の後に置かれます。 *n*<0 で *x*+*n*<1 の場合、ドライバーによってカーソルは最初の行の前に置かれます。

   b. 現行カーソル位置を知る必要があるときは、 getRow、isFirst、isLast、isBeforeFirst、または isAfterLast メソッドを使用してその情報を取得する。

   c. ステップ 1 (342 ページ) で *resultSetType* の値に TYPE_SCROLL_SENSITIVE を指定していて、現在行の最新の値を知る必要がある場合は、refreshRow メソッドを呼び出す。

      推奨: ResultSet の行をリフレッシュするとアプリケーションのパフォーマンスに悪影響を及ぼすことがあるため、 refreshRow を呼び出すのは最新データを知る必要があるときのみ にしてください。

   d. 以下の操作の 1 つ以上を実行する。

      - ResultSet オブジェクトの現在行の各列のデータを検索するには、getXXX メソッドを使用する。
      - 基礎表の現在行を更新するには、updateXXX メソッドを使用して ResultSet の現在行に列値を割り当てる。次に updateRow を使用して、基礎表の対応する行を更新します。基礎表を更新しないことにする場合は、updateRow メソッドではなく cancelRowUpdates メソッドを呼び出します。

        これらのメソッドを使用するには、 ResultSet の *resultSetConcurrency* の値を CONCUR_UPDATABLE にする必要があります。
      - 基礎表の現在行を削除するには、deleteRow メソッドを使用する。deleteRow を呼び出すと、ドライバーによって ResultSet の現在行がホールに置き換えられます。

        このメソッドを使用するには、 ResultSet の *resultSetConcurrency* の値を CONCUR_UPDATABLE にする必要があります。
5. close メソッドを呼び出し、ResultSet オブジェクトをクローズする。
6. close メソッドを呼び出し、Statement または PreparedStatement オブジェクトをクローズする。

たとえば、以下のコードは、従業員表のすべての行を逆順に検索して、従業員番号「000010」の電話番号を更新する方法を示しています。特定のステートメントの右側にある番号は、前述のステップに対応しています。

```
String s;
Connection con;
Statement stmt;
ResultSet rs;
...
stmt = con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
                           ResultSet.CONCUR_UPDATABLE);                1
                                      // Create a Statement object
                                      // for a scrollable, updatable
                                      // ResultSet
rs = stmt.executeQuery("SELECT EMPNO FROM EMPLOYEE FOR UPDATE OF PHONENO");
                                     // Create the ResultSet          3
rs.afterLast();                      // Position the cursor at the end of
                                     // the ResultSet                 4a
while (rs.previous()) {              // Position the cursor backward
 s = rs.getString("EMPNO");          // Retrieve the employee number  4d
                                     // (column 1 in the result
                                     // table)
 System.out.println("Employee number = " + s);
                                     // Print the column value
 if (s.compareTo("000010") == 0) {   // Look for employee 000010
  updateString("PHONENO","4657");    // Update their phone number
  updateRow();                       // Update the row
 }
}
rs.close();                          // Close the ResultSet           5
stmt.close();                        // Close the Statement           6
```

図 30. 両方向スクロール・カーソルの使用

DataSource オブジェクトの作成およびデプロイ

バージョン 2.0 以降の JDBC バージョンは、データ・ソースへの接続に使用する DataSource インターフェースを提供しています。 DataSource インターフェースの使用は、データ・ソースへ接続するための好ましい方法です。 DataSource インターフェースの使用には、以下の 2 つの部分が関係しています。

- DataSource オブジェクトの作成およびデプロイ。これは通常、WebSphere® Application Server などのツールを使用して、システム管理者によって行われます。
- DataSource オブジェクトを使用した接続の作成。これは、アプリケーション・プログラムで行われます。

このトピックには、 DataSource オブジェクトを自分で作成してデプロイするときに必要な情報を記載しています。

DB2 Universal JDBC ドライバーは、以下の DataSource インプリメンテーションを提供します。

- com.ibm.db2.jcc.DB2SimpleDataSource。これは、接続プールをサポートしていません。このインプリメンテーションは、Universal Type 2 Connectivity または Universal Type 4 Connectivity で使用できます。

DB2® JDBC Type 2 ドライバーは、以下の DataSource インプリメンテーションを提供します。

- COM.ibm.db2.jdbc.DB2DataSource。これには、接続プールの組み込みサポートが含まれています。このインプリメンテーションでは、接続プールが内部的に処理され、アプリケーションには透過的です。
- COM.ibm.db2.jdbc.DB2XADataSource。これには、分散トランザクションおよび接続プールの組み込みサポートが含まれていません。このインプリメンテーションでは、独自のコードを作成するか、 WebSphere Application Server などのツールを使用して、分散トランザクションおよび接続プールを自分で管理しなければなりません。

DataSource オブジェクトを作成してデプロイするときには、以下のタスクを実行する必要があります。

1. 適切な DataSource インプリメンテーションのインスタンスを作成します。
2. DataSource オブジェクトのプロパティーを設定します。
3. そのオブジェクトを Java™ Naming and Directory Interface (JNDI) ネーム・サービスに登録します。

図 31 の例は、これらのタスクの実行方法を示しています。

```
import java.sql.*;          // JDBC base
import javax.naming.*;     // JNDI Naming Services
import javax.sql.*;        // JDBC 2.0 standard extension APIs
import com.ibm.db2.jcc.*; // DB2 implementation of JDBC 2.0
                           // standard extension APIs

DB2SimpleDataSource db2ds = new com.ibm.db2.jcc.DB2SimpleDataSource(); 1

db2ds.setDatabaseName("db2loc1");                                      2
db2ds.setDescription("Our Sample Database");
db2ds.setUser("john");
db2ds.setPassword("db2");
:
Context ctx=new InitialContext();                                       3
Ctx.bind("jdbc/sampledb",db2ds);                                        4
```

図 31. DataSource オブジェクトの作成およびデプロイの例

1 DB2SimpleDataSource クラスのインスタンスを作成します。
2 このステートメントと次の 3 つのステートメントは、この DB2SimpleDataSource オブジェクトのプロパティーに値を設定します。
3 JNDI によって使用されるコンテキストを作成します。
4 DBSimple2DataSource オブジェクト db2ds と論理名 jdbc/sampledb とを関連付けます。このオブジェクトを使用するアプリケーションは、jdbc/sampledb という名前でそれを参照できます。

関連資料:

- 408 ページの『DB2 Universal JDBC ドライバーのプロパティー』

DB2 Universal JDBC ドライバーのクライアント転送サポート

フェイルオーバーは、別のサーバーが失敗するときに、操作を引き継ぐサーバーの機能です。 DB2 Universal JDBC ドライバーのクライアント転送サポートでは、DB2® UDB for Linux, UNIX® and Windows® 環境でのフェイルオーバーがサポートされています。これにより、DB2 UDB for Linux, UNIX and Windows クライアントは、クライアントが DB2 UDB for Linux, UNIX and Windows データベースに接続しているときの通信障害からリカバリーできます。通信障害が生じると、DB2 Universal JDBC ドライバーのクライアント転送サポートは、基礎となる接続を、データベースのフェイルオーバー・レプリカが存在する代替ロケーションに転送します。クライアント転送によって接続が維持されると、例外がスローされて、転送が行われたユーザーに通知され、トランザクションはロールバックされます。

DB2 Universal JDBC ドライバーのクライアント転送サポートを使用できるのは、`javax.sql.DataSource` インターフェースを使用する接続の場合だけです。

代替ロケーションに関する接続情報は、1 次 JDBC `DataSource` インスタンスの activeServerListJNDIName プロパティーにより、 Java™ クライアントに通知されます。 activeServerListJNDIName は、代替サーバー情報の JNDI リポジトリー内で、`DB2ActiveServerList` インスタンスへの JNDI 参照を示します。

`DB2ActiveServerList` は、 `alternateServerName` および `alternatePortNumber` の 2 つのプロパティーを備えた、シリアライズ可能な Java Bean です。 `getXXX` および `setXXX` メソッドは、プロパティーごとに定義されます。 Java Bean は、次のようになります。

```
package com.ibm.db2.jcc;
public class DB2ActiveServerList implements java.io.Serializable,
  javax.naming.Referenceable
{
  public String[] alternateServerName;
  public synchronized void
    setAlternateServerName(String[] alternateServer);
  public String[] getAlternateServerName();
  public int[] alternatePortNumber;
  public synchronized void
    setAlternatePortNumber(int[] alternatePortNumberList);
  public int[] getAlternatePortNumber();
}
```

クライアントが CONNECT または CONNECT RESET を発行すると、代替情報はサーバーからクライアントへ動的に伝搬します。この動的に伝搬された代替サーバー情報は、グローバルなドライバー・メモリーに保管され、 DB2 アクティブ・サーバーの JNDI 保管場所でも更新されます。 DB2 Universal JDBC ドライバーは、フェイルオーバー後に、更新された情報を代替 JNDI に伝搬しようとします。

新しく確立されたフェイルオーバー接続は、サーバー名とポート番号は除き、元の `DataSource` プロパティーで構成されます。さらに、元の接続で変更された DB2 特殊レジスターがあれば、フェイルオーバー接続で再確立されます。

通信障害が発生すると、DB2 Universal JDBC ドライバーは、まず元のサーバーに対するリカバリーを試行します。元のサーバーへの再接続のことを、フェイルバックといいます。フェイルバックが失敗すると、ドライバーは代替ロケーションに接続しようとします (フェイルオーバー)。フェイルオーバーまたはフェイルバック接続

が再確立された後、ドライバーは、SQLCODE -4498 の java.sql.SQLException をアプリケーションにスローし、フェイルオーバーまたはフェイルバックが発生しトランザクションが失敗したことをアプリケーションに通知します。その後、アプリケーションはトランザクションを再試行できます。

代替サーバーのセットアップ方式: JNDI を使用して、代替サーバーをセットアップします。以下のステップが関係します。

1. 初期コンテキストの環境を設定します。 jndi.properties ファイルを作成して設定を行い、その名前を CLASSPATH に追加できます。

   例: jndi.properties ファイル:

   ```
   java.naming.factory.initial=com.sun.jndi.fscontext.RefFSContextFactory
   java.naming.provider.url=file:/tmp
   ```

2. DB2ActiveServerList のインスタンスを作成して、そのインスタンスを JNDI レジストリーにバインドします。

   例: DB2ActiveServerList のインスタンスを作成し、そのインスタンスを JNDI レジストリーにバインドするコード:

   ```
   // Create a starting context for naming operations
   InitialContext registry = new InitialContext();
   // Create a DB2ActiveServerList object
   DB2ActiveServerList address = new DB2ActiveServerList();
   // Set the port number and server name for the alternate server
   int[] a = {50000};
   String[] s = {"mvs3.sj.ibm.com"};
   address.setActivePortNumber(a);
   address.setActiveServerName(s);
   // Bind the DB2ActiveServerList instance to the JNDI registry
   registry.rebind("jdbc/alternate", address);
   ```

3. 代替サーバーのロケーション情報を含む、DB2ActiveServerList オブジェクトの論理名を、元の DataSource の activeServerListJNDIName プロパティーに割り当てます。

   例: DB2ActiveServerList オブジェクトの論理名を DataSource インスタンス指定データ・ソースの activeServerListJNDIName プロパティーに割り当てるコード:

   ```
   datasource.setActiveServerListJNDIName("jdbc/alternate");
   ```

関連資料:

- 408 ページの『DB2 Universal JDBC ドライバーのプロパティー』
- 457 ページの『JDBC に対する DB2 Universal JDBC ドライバーの拡張のサマリー』

DB2 Universal JDBC ドライバーを使用した DB2 サーバーへの拡張クライアント情報の提供

DB2 Universal JDBC ドライバーには、クライアントに関する追加の情報をサーバーに提供するために使用できる DB2® 固有のメソッドがあります。この情報は、アカウンティングやワークロード管理に使用することができます。この情報は、アプリケーションが、SQL の実行などのサーバーにアクセスするアクションを実行するときに、 DB2 サーバーへ送信されます。

メソッドについては、表 36 にリストします。

表 36. DB2 サーバーにクライアント情報を提供するメソッド

メソッド	提供される情報
setDB2ClientUser	接続のためのユーザー名
setDB2ClientWorkstation	接続のためのワークステーション名
setDB2ClientApplicationInformation	接続に関して作動しているアプリケーション名
setDB2ClientAccountingInformation	アカウンティング情報

拡張情報を設定するには、以下のようにします。

1. Connection を作成する。
2. java.sql.Connection オブジェクトを com.ibm.db2.jcc.DB2Connection にキャストする。
3. 表 36 に示されているメソッドのいずれかを呼び出す。
4. SQL ステートメントを実行して、情報が DB2 サーバーに送信されるようにする。

以下のコードは、前述のステップを実行して、ユーザー名とワークステーション名を DB2 サーバーに渡します。特定のステートメントの右側にある番号は、前述のステップに対応しています。

```
public class ClientInfoTest {
  public static void main(String[] args) {
    String url = "jdbc:db2://sysmvs1.stl.ibm.com:5021/san_jose";
    try {
      Class.forName("com.ibm.db2.jcc.DB2Driver");
      String user = "db2adm";
      String password = "db2adm";
      Connection con = DriverManager.getConnection(url,         1
        user, password);
      if (conn instanceof DB2Connection) {
        DB2Connection db2conn = (DB2Connection) conn;           2
        db2conn.setDB2ClientUser("Michael L Thompson");         3
        db2conn.setDB2ClientWorkstation("sjwkstn1");
        // Execute SQL to force extended client information to be sent
        // to the server
        conn.prepareStatement("SELECT * FROM SYSIBM.SYSDUMMY1"
          + "WHERE 0 = 1").executeQuery();                      4       }
    } catch (Throwable e) {
        e.printStackTrace();
      }
  }
}
```

図 32. DB2 サーバーへの拡張クライアント情報の送信の例

関連資料:

- 457 ページの『JDBC に対する DB2 Universal JDBC ドライバーの拡張のサマリー』

第 16 章 SQLJ アプリケーション・プログラミング

以下のセクションには、SQLJ アプリケーションの作成に関する情報が含まれています。

基本的な SQLJ アプリケーション・プログラミングの概念

以下のトピックには、SQLJ アプリケーションの作成に関する基本的な情報が含まれています。

SQLJ アプリケーション作成の基本ステップ

SQLJ アプリケーションの作成は、他の言語での SQL アプリケーションの作成と多くの点で共通しています。一般に、以下のことを行う必要があります。

- SQLJ および JDBC メソッドを含む Java™ パッケージをインポートする。
- DB2® 表からのデータ送信またはデータ検索用の変数を宣言する。
- データ・ソースに接続する。
- SQL ステートメントを実行する。
- SQL エラーおよび警告を処理する。
- データ・ソースから切断する。

実行する必要のあるタスクは、他の言語の場合と同様ですが、それらのタスクを実行する方法と順序はいくらか異なります。

352 ページの図 33 は、各タスクを例示する簡単なプログラムです。

```
import sqlj.runtime.*;                                                    1
import java.sql.*;

#sql context EzSqljCtx;                                                   3a
#sql iterator EzSqljNameIter (String LASTNAME);                           4a

public class EzSqlj {
  public static void main(String args[])
    throws SQLException
  {
    EzSqljCtx ctx = null;
    String URLprefix = "jdbc:db2:";
    String url;
    url = new String(URLprefix + args[0]);
                                              // Location name is an input parameter
    String hvmgr="000010";                                                2
    String hvdeptno="A00";
    try {                                                                 3b
      Class.forName("com.ibm.db2.jcc.DB2Driver");
    } catch (Exception e)
    {
       throw new SQLException("Error in EzSqlj: Could not load the driver");
    }
    try
    {
       System.out.println("About to connect using url: " + url);
       Connection con0 = DriverManager.getConnection(url);                3c
                                              // Create a JDBC Connection
       con0.setAutoCommit(false);             // set autocommit OFF
       ctx = new EzSqljCtx(con0);                                         3d

       try
       {
         EzSqljNameIter iter;
         int count=0;

         #sql [ctx] iter =
           {SELECT LASTNAME FROM EMPLOYEE};                               4b
                                              // Create result table of the SELECT
         while (iter.next()) {                                            4c
            System.out.println(iter.LASTNAME()); // Retrieve rows from result table
            count++;
         }
         System.out.println("Retrieved " + count + " rows of data");
       }
```

図 33. 簡単な SQLJ アプリケーション (1/2)

```
      catch( SQLException e )                                              5
      {
         System.out.println ("**** SELECT SQLException...");
         while(e!=null) {
           System.out.println ("Error msg: " + e.getMessage());
           System.out.println ("SQLSTATE: " + e.getSQLState());
           System.out.println ("Error code: " + e.getErrorCode());
           e = e.getNextException(); // Check for chained exceptions
         }
      }
      catch( Exception e )
      {
         System.out.println("**** NON-SQL exception   = " + e);
         e.printStackTrace();
      }
      try
      {
        #sql [ctx]                                                         4d
          {UPDATE DEPARTMENT SET MGRNO=:hvmgr
             WHERE DEPTNO=:hvdeptno};
                                        // Update data for one department  6
        #sql [ctx] {COMMIT};              // Commit the update
      }
      catch( SQLException e )
      {
         System.out.println ("**** UPDATE SQLException...");
         System.out.println ("Error msg: " + e.getMessage() + ".  SQLSTATE=" +
           e.getSQLState() + " Error code=" + e.getErrorCode());
         e.printStackTrace();
      }
      catch( Exception e )
      {
         System.out.println("**** NON-SQL exception   = " + e);
         e.printStackTrace();
      }
      iter.close();                                    // Close the iterator
      ctx.close();                                                         7
  }
  catch(SQLException e)
  {
     System.out.println ("**** SQLException ...");
     System.out.println ("Error msg: " + e.getMessage() + ".  SQLSTATE=" +
       e.getSQLState() + " Error code=" + e.getErrorCode());
     e.printStackTrace();
  }
  catch(Exception e)
  {
    System.out.println ("**** NON-SQL exception = " + e);
    e.printStackTrace();
  }

}
```

図 33. 簡単な SQLJ アプリケーション (2/2)

352 ページの図 33 の注:

1 これらのステートメントは JDBC コア API を含む java.sql パッケージ、および SQLJ API を含む sqlj.runtime パッケージをインポートします。アクセスする必要のある他のパッケージまたはクラスについては、『SQLJ サポート用の Java パッケージ』を参照してください。

2 String 変数の hvmgr と hvdeptno は ホスト変数 です。これらは DB2 ホスト変数に対応しています。詳細については、 『SQLJ アプリケーションでの変数の宣言』を参照してください。

3a、**3b**、**3c**、および **3d** これらのステートメントは、使用可能な 3 つの技法の中の 1 つを使用してデータ・ソースに接続する方法を示しています。詳細については、『SQLJ を使用したデータ・ソースへの接続』を参照してください。

4a、**4b**、**4c**、および **4d**
これらのステートメントは、SQLJ で SQL ステートメントを実行する方法を示しています。ステートメント 4a は、SQLJ で SQL カーソルの宣言に相当するものです。ステートメント 4b と 4c は、 SQL FETCH の実行に相当することを SQLJ で行う 1 つの方法を示しています。ステートメント 4d は、 SQL UPDATE の実行に相当することを SQLJ で行う方法を示しています。詳細については、『SQLJ アプリケーションでの SQL の実行』を参照してください。

5
この `try/catch` ブロックは、SQL エラー処理での `SQLException` クラスの使用を示しています。 SQL エラーの処理の詳細については、 『SQLJ アプリケーションでの SQL エラーの処理』を参照してください。 SQL 警告の処理の詳細については、 『SQLJ アプリケーションでの SQL 警告の処理』を参照してください。

6
これはコメントの例です。 SQLJ プログラムへのコメントの組み込みに関する規則については、 『SQLJ アプリケーションへのコメントの組み込み』を参照してください。

7
このステートメントで、データ・ソースとの接続をクローズします。 『SQLJ アプリケーションでのデータ・ソースへの接続のクローズ』を参照してください。

関連概念:

- 354 ページの『SQLJ サポート用の Java パッケージ』
- 355 ページの『SQLJ アプリケーションでの変数』
- 365 ページの『SQLJ アプリケーションにおける SQL ステートメント』

関連タスク:

- 357 ページの『SQLJ を使用したデータ・ソースへの接続』

SQLJ サポート用の Java パッケージ

SQLJ ステートメントを実行する場合、または SQLJ プログラムで JDBC メソッドを呼び出す場合は、その前に、それらのステートメントをサポートするさまざまな Java™ パッケージのすべてまたは一部にアクセスできるようにしておく必要があります。これは、パッケージまたは特定のクラスをインポートするか、または完全修飾クラス名を使用して実行できます。 SQLJ プログラムには、以下のパッケージまたはクラスが必要になる場合があります。

sqlj.runtime
SQLJ ランタイム API が含まれています。

java.sql
コア JDBC API を含みます。

com.ibm.db2.jcc
JDBC および SQLJ の DB2® 固有のインプリメンテーションが含まれています。

javax.naming
Java Naming and Directory Interface (JNDI) のクラスおよびインターフェースを含みます。これは、`DataSource` をインプリメントするためにしばしば使用されます。

javax.sql
JDBC 2.0 規格の拡張機能を含みます。

関連概念:

- 351 ページの『SQLJ アプリケーション作成の基本ステップ』

SQLJ アプリケーションでの変数

他の言語の DB2® プログラムでは、アプリケーション・プログラムと DB2 の間でデータを受け渡しするためにホスト変数を使用します。 SQLJ プログラムでは、ホスト式 を使用します。ホスト式は、単純な Java™ ID にも、複合式にもすることができます。 SQL ステートメントで使用するホスト式は、すべて先頭をコロンにする必要があります。ホスト式は大文字小文字を区別します。

DB2 Universal JDBC ドライバーの場合の Java ID には、 『Java、JDBC、および SQL のデータ・タイプ』の Java データ・タイプ列にリストされている、いずれのデータ・タイプも使用することができます。イテレーターで指定されるデータ・タイプには、 『Java、JDBC、および SQL のデータ・タイプ』のデータ・タイプ列にあるいずれのタイプも使用することができます。

複合式は、単一値を評価する配列エレメントまたは Java 式です。SQLJ 文節内の複合式は、括弧で囲む必要があります。

以下の例は、ホスト式の使用方法を示しています。

例: Java ID の宣言および SELECT ステートメントでの使用:

この例で、`#sql` で始まるステートメントの機能は、他の言語での SELECT ステートメントの機能と同じです。このステートメントは、従業員番号が 000010 の従業員の姓を Java ID `empname` に割り当てます。

```
String empname;
...
  #sql [ctxt]
  {SELECT LASTNAME INTO :empname FROM EMPLOYEE WHERE EMPNO='000010'};
```

例: Java ID の宣言およびストアード・プロシージャー呼び出しでの使用:

この例で、`#sql` で始まるステートメントの機能は、他の言語での SQL CALL ステートメントの機能と同じです。このステートメントは、Java ID `empno` を、ストアード・プロシージャー `A` の入力パラメーターとして使用します。`empno` の前にある値 `IN` は、`empno` が入力パラメーターであることを指定しています。パラメーターの使用法を示す修飾子 (IN、OUT、または INOUT) は、ストアード・プロシージャーの CREATE PROCEDURE ステートメントで指定したパラメーター定義内の対応する値と一致している必要があります。

```
String empno = "0000010";
...
#sql [ctxt] {CALL A (:IN empno)};
```

例: ホスト ID としての複合式の使用:

この例では、ホスト式として複合式 (((int)yearsEmployed++/5)*500) を使用しています。

```
#sql [ctxt] {UPDATE EMPLOYEE
  SET BONUS=:(((int)yearsEmployed++/5)*500) WHERE EMPNO=:empID};
```

SQLJ は、複合ホスト式を処理するときに、以下のアクションを実行します。

- ホスト式を左から右に評価して、その値を DB2 に割り当てる。
- 後置演算子による操作などの副次作用を、通常の Java 規則に基づいて評価する。ホスト式がすべて完全に評価された後に、それらの値のいずれかが DB2 に渡されます。
- Java 規則を使用して、丸めと切り捨てを行う。

したがって、UPDATE ステートメントを実行する前の yearsEmployed の値が 6 の場合は、 UPDATE ステートメントによって列 BONUS に割り当てられる値は ((int)6/5)*500、つまり 500 になります。 500 が BONUS に割り当てられた後、yearsEmployed の値が増分されます。

変数名に関する制約事項: SQLJ プログラムでは、特別な意味を持つストリングが 2 つあります。以下のストリングを SQLJ プログラムで使用するときは、この制約事項に従ってください。

- ストリング __sJT_ は、SQLJ によって生成される変数名のための予約済み接頭部です。以下のタイプの名前は、先頭を __sJT_ にしないでください。
  - ホスト式の名前。
  - 実行可能 SQL ステートメントを含むブロックで宣言される Java 変数名。
  - 実行可能 SQL ステートメントを含むメソッドのパラメーターの名前。
  - 実行可能 SQL ステートメントを含むクラス内のフィールドの名前、または実行可能 SQL ステートメントを含むサブクラスまたは囲みクラスがあるクラス内のフィールドの名前。
- ストリング _SJ は、SQLJ によって生成されるリソース・ファイルおよびクラスのための予約済み接尾部です。ストリング _SJ は、クラス名および入力ソース・ファイル名では使用しないでください。

関連概念:

- 351 ページの『SQLJ アプリケーション作成の基本ステップ』

関連資料:

- 403 ページの『Java、JDBC、および SQL のデータ・タイプ』

SQLJ アプリケーションでのコメント

プログラムに説明を記入するには、コメントを組み込む必要があります。これを行うには、Java™ コメントを使用します。 Java コメントは、/* */ 、または // で示します。 Java コメントは SQLJ 文節の外側に組み込みます。 Java 言語が許可する場所であればどこにでも組み込むことができます。 SQLJ 文節内で Java コメントを使用できるのは、ホスト式の内側のみです。

関連概念:

- 293 ページの『JDBC アプリケーション作成時の基本的なステップ』

SQLJ を使用したデータ・ソースへの接続

SQLJ アプリケーションでは、他の DB2® アプリケーションの場合のように、SQL ステートメントを実行する前にデータベース・サーバーに接続する必要があります。 SQLJ でも、JDBC の場合のように、データベース・サーバーをデータ・ソース と呼びます。

データ・ソースに接続するには、以下の 5 つの技法のいずれかを使用します。

- JDBC DriverManager インターフェースを使用して、明示的に接続を作成する。これを行う技法は 2 つあります。
- JDBC DataSource インターフェースを使用して、明示的に接続を作成する。これを行う技法は 2 つあります。
- 暗黙的に接続を作成する。

接続技法 1: この技法では、接続を作成するための基本的な手段として JDBC DriverManager を使用します。これを、任意のレベルの JDBC ドライバーとともに使用します。

1. SQLJ *接続宣言文節* を実行する。

   これを行うと、*接続コンテキスト・クラス* が生成されます。最も単純な形式の接続宣言文節は、次のようになります。

   ```
   #sql context context-class-name;
   ```

   生成される接続コンテキスト・クラスの名前が *context-class-name* です。
2. 以下のように Class.forName メソッドを呼び出して、JDBC ドライバーをロードする。
   - DB2 Universal JDBC ドライバーの場合は、次のようにして Class.forName を呼び出す。

     ```
     Class.forName("com.ibm.db2.jcc.DB2Driver");
     ```
   - DB2 JDBC Type 2 ドライバーの場合は、次のようにして Class.forName を呼び出す。

     ```
     Class.forName("COM.ibm.db2.jdbc.app.DB2Driver");
     ```
3. ステップ 1 で作成した接続コンテキスト・クラスのコンストラクターを呼び出す。

   これにより、関連データ・ソースで実行する各 SQL ステートメントで指定する接続コンテキスト・オブジェクトが作成されます。コンストラクター呼び出しステートメントは、以下のいずれかの形式にする必要があります。

   ```
   connection-context-class connection-context-object=
     new connection-context-class(String url, boolean autocommit);

   connection-context-class connection-context-object=
     new connection-context-class(String url, String user,
       String password, boolean autocommit);
   connection-context-class connection-context-object=
     new connection-context-class(String url, Properties info,
       boolean autocommit);
   ```

   各パラメーターの意味は、以下のとおりです。

   url データ・ソースに関連付けられたロケーション名を指定するストリング。こ

の引き数は、 『DriverManager インターフェースと DB2 Universal JDBC ドライバーを使用したデータ・ソースへの接続』で指定されているいずれかの形式です。この形式は、使用する JDBC ドライバーによって異なります。

user **および** *password*
データ・ソースへの接続のためのユーザー ID およびパスワードを指定します (接続先データ・ソースで必要とする場合)。

info
接続のためのドライバーのプロパティーのセットを含むタイプ `java.util.Properties` のオブジェクトを指定します。 Linux、UNIX® および Windows® 用 DB2 JDBC Type 2 ドライバー (DB2 JDBC Type 2 ドライバー) の場合は、 `user` および `password` プロパティーのみを指定します。 DB2 Universal JDBC ドライバーの場合は、 『DB2 Universal JDBC ドライバーのプロパティー』でリストされている任意のプロパティーを指定することができます。

autocommit
各ステートメントの後にデータベース・マネージャーがCOMMIT を発行するかどうかを指定します。指定可能な値は `true` または `false` です。 `false` を指定する場合は、コミット操作を明示的に行う必要があります。

以下のコードは、接続技法 1 を使用してロケーション NEWYORK への接続を作成します。この接続ではユーザー ID とパスワードを必要とし、自動コミットは必要としていません。特定のステートメントの右側にある番号は、前述のステップに対応しています。

```
#sql context Ctx;             // Create connection context class Ctx        1
String userid="dbadm";        // Declare variables for user ID and password
String password="dbadm";
String empname;               // Declare a host variable
...
try {                         // Load the JDBC driver
  Class.forName("com.ibm.db2.jcc.DB2Driver");                              2
}
catch (ClassNotFoundException e) {
   e.printStackTrace();
}
Ctx myConnCtx=                                                             3
  new Ctx("jdbc:db2://sysmvs1.stl.ibm.com:5021/NEWYORK",
  userid,password,false);     // Create connection context object myConnCtx
                              // for the connection to NEWYORK
#sql [myConnCtx] {SELECT LASTNAME INTO :empname FROM EMPLOYEE
   WHERE EMPNO='000010'};
                              // Use myConnCtx for executing an SQL statement
```

図 34. 接続技法 1 を使用したデータ・ソースへの接続

接続技法 2: この技法では、接続の作成に JDBC `DriverManager` インターフェースを使用します。これを、任意のレベルの JDBC ドライバーとともに使用します。

1. SQLJ 接続宣言文節を実行する。

   これは接続技法 1 のステップ 1 (357 ページ) と同じです。

2. ドライバーをロードする。

   これは接続技法 1 のステップ 2 (357 ページ) と同じです。

3. JDBC DriverManager.getConnection メソッドを呼び出す。

   これにより、データ・ソースへの接続のための JDBC 接続オブジェクトが作成されます。 『DriverManager インターフェースと DB2 Universal JDBC ドライバーを使用したデータ・ソースへの接続』で示されている getConnection のいずれかの形式を使用します。

   url、*user*、および *password* パラメーターの意味は、接続技法 1のステップ 3 (357 ページ) での各パラメーターの意味と同じです。

4. ステップ 1 (358 ページ) で作成した接続コンテキスト・クラスのコンストラクターを呼び出す。

   これにより、関連データ・ソースで実行する各 SQL ステートメントで指定する接続コンテキスト・オブジェクトが作成されます。コンストラクター呼び出しステートメントは、以下の形式にする必要があります。

   ```
   connection-context-class connection-context-object=
     new connection-context-class(Connection JDBC-connection-object);
   ```

   JDBC-connection-object パラメーターは、ステップ 3 で作成した Connection オブジェクトです。

以下のコードは、接続技法 2 を使用してロケーション NEWYORK への接続を作成します。この接続ではユーザー ID とパスワードを必要とし、自動コミットは必要としていません。特定のステートメントの右側にある番号は、前述のステップに対応しています。

```
#sql context Ctx;           // Create connection context class Ctx        1
String userid="dbadm";      // Declare variables for user ID and password
String password="dbadm";
String empname;             // Declare a host variable
...
try {                       // Load the JDBC driver
  Class.forName("com.ibm.db2.jcc.DB2Driver");                            2
}
catch (ClassNotFoundException e) {
   e.printStackTrace();
}
Connection jdbccon=                                                      3
  DriverManager.getConnection("jdbc:db2://sysmvs1.stl.ibm.com:5021/NEWYORK",
    userid,password);
                            // Create JDBC connection object jdbccon
jdbccon.setAutoCommit(false); // Do not autocommit                       4
Ctx myConnCtx=new Ctx(jdbccon);                                          5
                            // Create connection context object myConnCtx
                            // for the connection to NEWYORK
#sql [myConnCtx] {SELECT LASTNAME INTO :empname FROM EMPLOYEE
   WHERE EMPNO='000010'};
                            // Use myConnCtx for executing an SQL statement
```

図 35. 接続技法 2 を使用したデータ・ソースへの接続

接続技法 3: この技法では、接続の作成に JDBC DataSource インターフェースを使用します。

1. SQLJ 接続宣言文節を実行する。

   これは接続技法 1 のステップ 1 (357 ページ) と同じです。

2. システム管理者が別のプログラムで DataSource オブジェクトを作成した場合は、以下のようにする。
   a. 接続する必要があるデータ・ソースの論理名を入手する。
   b. 次のステップで使用するコンテキストを作成する。
   c. アプリケーション・プログラムで Java™ Naming and Directory Interface (JNDI) を使用して、論理データ・ソース名に関連付けられた DataSource オブジェクトを取得する。

   これに該当しない場合は、 『DataSource インターフェースを使用したデータ・ソースへの接続』の『同じアプリケーションでの DataSource オブジェクトの作成および使用』の説明に従って、 DataSource オブジェクトを作成してそれにプロパティーを割り当てます。
3. JDBC DataSource.getConnection メソッドを呼び出す。

   これにより、データ・ソースへの接続のための JDBC 接続オブジェクトが作成されます。以下のいずれかの形式の getConnection を使用できます。

   ```
   getConnection();
   getConnection(user, password);
   ```

   user および *password* パラメーターの意味は、接続技法 1 のステップ 3 (357 ページ) での各パラメーターの意味と同じです。
4. デフォルトの自動コミット・モードが適切でない場合は、JDBC Connection.setAutoCommit メソッドを呼び出す。

   これにより、各ステートメント後にデータベース・マネージャーが COMMIT を発行するかどうかを指定します。このメソッドの形式は次のとおりです。

   ```
   setAutoCommit(boolean autocommit);
   ```
5. ステップ 1 (359 ページ) で作成した接続コンテキスト・クラスのコンストラクターを呼び出す。

   これにより、関連データ・ソースで実行する各 SQL ステートメントで指定する接続コンテキスト・オブジェクトが作成されます。コンストラクター呼び出しステートメントは、以下の形式にする必要があります。

   ```
   connection-context-class connection-context-object=
     new connection-context-class(Connection JDBC-connection-object);
   ```

   JDBC-connection-object パラメーターは、ステップ 3 で作成した Connection オブジェクトです。

以下のコードは、接続技法 3 を使用して論理名が jdbc/sampledb のロケーションへの接続を作成します。特定のステートメントの右側にある番号は、前述のステップに対応しています。

```
import java.sql.*;
import javax.naming.*;
import javax.sql.*;
...
#sql context CtxSqlj;       // Create connection context class CtxSqlj   1
Context ctx=new InitialContext();                                         2b
DataSource ds=(DataSource)ctx.lookup("jdbc/sampledb");                    2c
Connection con=ds.getConnection();                                        3
String empname;             // Declare a host variable
...
con.setAutoCommit(false);   // Do not autocommit                          4
CtxSqlj myConnCtx=new CtxSqlj(con);                                       5
                            // Create connection context object myConnCtx
#sql [myConnCtx] {SELECT LASTNAME INTO :empname FROM EMPLOYEE
   WHERE EMPNO='000010'};
                            // Use myConnCtx for executing an SQL statement
```

図 36. 接続技法 3 を使用したデータ・ソースへの接続

接続技法 4 (DB2 Universal JDBC ドライバーのみ): この技法では、接続の作成に JDBC DataSource インターフェースを使用します。この技法では、DataSource が JNDI に登録されていることが**必要**です。

1. 接続する必要があるデータ・ソースの論理名をシステム管理者から入手する。
2. SQLJ 接続宣言文節を実行する。

   このタイプの接続の場合、接続宣言文節は次の形式にする必要があります。

   ```
   #sql public static context context-class-name
    with (dataSource="logical-name");
   ```

   接続コンテキストは、public および static として宣言されていなければなりません。 *logical-name* は、ステップ 1 で入手したデータ・ソース名です。
3. ステップ 2 で作成した接続コンテキスト・クラスのコンストラクターを呼び出す。

   これにより、関連データ・ソースで実行する各 SQL ステートメントで指定する接続コンテキスト・オブジェクトが作成されます。コンストラクター呼び出しステートメントは、以下のいずれかの形式にする必要があります。

   ```
   connection-context-class connection-context-object=
     new connection-context-class();

   connection-context-class connection-context-object=
     new connection-context-class (String user,
       String password);
   ```

   user および *password* パラメーターの意味は、接続技法 1 のステップ 3 (357 ページ) での各パラメーターの意味と同じです。

以下のコードは、接続技法 4 を使用して論理名が jdbc/sampledb のロケーションへの接続を作成します。この接続には ユーザー ID とパスワードが必要です。

```
#sql public static context Ctx
  with (dataSource="jdbc/sampledb");                                    2
                                // Create connection context class Ctx
String userid="dbadm";          // Declare variables for user ID and password
String password="dbadm";

String empname;                 // Declare a host variable
...
Ctx myConnCtx=new Ctx(userid, password);                                3
                                // Create connection context object myConnCtx
                                // for the connection to jdbc/sampledb
#sql [myConnCtx] {SELECT LASTNAME INTO :empname FROM EMPLOYEE
   WHERE EMPNO='000010'};
                                // Use myConnCtx for executing an SQL statement
```

図 37. 接続技法 4 を使用したデータ・ソースへの接続

接続技法 5: この技法では、デフォルト接続を使用してデータ・ソースに接続します。デフォルト接続を使用するには、接続コンテキスト・オブジェクトを指定しないで SQL ステートメントを指定します。この技法を使用する場合は、プログラムで明示的に JDBC インターフェースを使用する場合を除いて、JDBC ドライバーをロードする必要はありません。たとえば、以下のようにします。

```
#sql {SELECT LASTNAME INTO :empname FROM EMPLOYEE
   WHERE EMPNO='000010'};  // Use default connection for
                                    // executing an SQL statement
```

デフォルト接続コンテキストを作成するために、 SQLJ は jdbc/defaultDataSource の JNDI 検索を行います。何も登録されていない場合には、SQLJ がコンテキストにアクセスしようとすると、 NULL コンテキスト例外が出されます。

関連概念:

- 297 ページの『JDBC アプリケーションによるデータ・ソースへの接続の方法』

関連タスク:

- 300 ページの『DriverManager インターフェースと DB2 Universal JDBC ドライバーを使用したデータ・ソースへの接続』
- 303 ページの『DataSource インターフェースを使用したデータ・ソースへの接続』

関連資料:

- 408 ページの『DB2 Universal JDBC ドライバーのプロパティー』

SQLJ トランザクションの分離レベルの設定

SQLJ プログラム内の作業単位の分離レベルを設定するには、SET TRANSACTION ISOLATION LEVEL 文節を使用します。 表 37 に、 SET TRANSACTION ISOLATION LEVEL 文節で指定できる値と、DB2® でのそれに対応する値を示します。

表 37. SQLJ および DB2 での同等の分離レベル

SET TRANSACTION の値	DB2 分離レベル
SERIALIZABLE	反復可能読み取り
REPEATABLE READ	読み取り固定

表 37. SQLJ および DB2 での同等の分離レベル (続き)

SET TRANSACTION の値	DB2 分離レベル
READ COMMITTED	カーソル固定
READ UNCOMMITTED	非コミット読み取り

分離レベルは、SQLJ 接続だけでなく、基礎の JDBC 接続にも影響します。分離レベルを変更できるのは、トランザクションの開始時のみです。

関連概念:

- 「*SQL リファレンス 第 1 巻*」の『分離レベル』

SQLJ トランザクションのコミットまたはロールバック

SQLJ 接続の自動コミットを使用不可にする場合は、コミットまたはロールバックの操作を明示的に行う必要があります。これを行うには、以下のように SQL の COMMIT または ROLLBACK ステートメントを含む実行文節を使用します。

```
#sql [myConnCtx] {COMMIT};
#sql [myConnCtx] {ROLLBACK};
```

関連概念:

- 363 ページの『SQLJ アプリケーションにおけるセーブポイント』

関連タスク:

- 357 ページの『SQLJ を使用したデータ・ソースへの接続』

SQLJ アプリケーションにおけるセーブポイント

SQL セーブポイントは、作業単位内でのある特定のポイント・イン・タイムにおけるデータおよびスキーマの状態を表します。セーブポイントの設定、セーブポイントの解除、セーブポイントが表す状態へのデータとスキーマのリストアを行う SQL ステートメントがあります。

DB2 Universal JDBC ドライバーの使用時では、SQLJ プログラムに任意の形式の SQL SAVEPOINT ステートメントを組み込むことができます。

以下の例は、セーブポイントの設定、セーブポイントへのロールバック、およびセーブポイントの解除を行う方法を示しています。

```
#sql context Ctx;             // Create connection context class Ctx
String empNumVar;
int shoeSizeVar;
...
try {                         // Load the JDBC driver
  Class.forName("com.ibm.db2.jcc.DB2Driver");
}
  catch (ClassNotFoundException e) {
   e.printStackTrace();
}
Connection jdbccon=
  DriverManager.getConnection("jdbc:db2://sysmvs1.stl.ibm.com:5021/NEWYORK",
    userid,password);
                              // Create JDBC connection object jdbccon
jdbccon.setAutoCommit(false); // Do not autocommit
Ctx ctxt=new Ctx(jdbccon);
                              // Create connection context object myConnCtx
                              // for the connection to NEWYORK
#sql [ctxt] {CREATE DISTINCT TYPE SHOESIZE AS INTEGER WITH COMPARISONS};
                              // Create a distinct type
#sql [ctxt] {COMMIT};
                              // Commit the create
#sql [ctxt]
  {CREATE TABLE EMP_SHOE (EMPNO CHAR(6), EMP_SHOE_SIZE SHOESIZE)};
                              // Create table with distinct type
#sql [ctxt] {COMMIT};
                              // Commit the create
#sql [ctxt]
  {INSERT INTO EMP_SHOE VALUES ('000010', 6)};
                              // Insert a row
#sql [ctxt]
  {SAVEPOINT SVPT1 ON ROLLBACK RETAIN CURSORS};
                              // Create a savepoint
...
#sql [ctxt]
  {INSERT INTO EMP_SHOE VALUES ('000020', 10)};
                               // Insert another row
#sql [ctxt] {ROLLBACK TO SAVEPOINT SVPT1};
                               // Roll back work to the point
                               //  after the first insert
...
#sql [ctxt] {RELEASE SAVEPOINT SVPT1};
                               // Release the savepoint
ctx.close();                   // Close the connection context
```

図 38. SQLJ アプリケーションでのセーブポイントの設定、ロールバック、および解除

関連タスク:

- 363 ページの『SQLJ トランザクションのコミットまたはロールバック』

関連資料:

- 「*SQL リファレンス 第 2 巻*」の『ROLLBACK ステートメント』
- 「*SQL リファレンス 第 2 巻*」の『RELEASE SAVEPOINT ステートメント』
- 「*SQL リファレンス 第 2 巻*」の『SAVEPOINT ステートメント』

SQLJ アプリケーションでのデータ・ソースへの接続のクローズ

データ・ソースへの接続が終了したら、データ・ソースへの接続をクローズする必要があります。こうすることで、接続コンテキスト・オブジェクトの DB2® および SQLJ リソースは即時に解放されます。

データ・ソースへの接続をクローズするには、ConnectionContext.close() メソッドを使用します。このメソッドは、データ・ソースへの接続だけでなく、接続コンテキストもクローズします。たとえば、以下のようにします。

```
...
ctx = new EzSqljctx(con0);          // Create a connection context object
                                    // from JDBC connection con0
...                                 // Perform various SQL operations
 EzSqljctx.close();                 // Close the connection context and
                                    // connection to the data source
```

関連タスク:

- 357 ページの『SQLJ を使用したデータ・ソースへの接続』

SQLJ アプリケーションにおける SQL ステートメント

従来の SQL プログラムでは、SQL ステートメントを実行して、表の作成、表に対するデータの挿入、更新、および削除、表からのデータの検索、ストアード・プロシージャーの呼び出し、トランザクションのコミットまたはロールバックを実行します。 SQLJ プログラムでもこれらのステートメントを実行しますが、それを SQLJ *実行可能文節* 内で実行します。実行可能文節の形式は、以下の一般形式のいずれかにします。

```
#sql [connection-context] {sql-statement};
#sql [connection-context,execution-context] {sql-statement};
#sql [execution-context] {sql-statement};
```

実行可能文節では、明示接続コンテキストを必ず 指定する必要があります。ただし唯一の例外として、FETCH ステートメントの場合は明示接続コンテキストを指定しません。特定のケースのみ、実行コンテキストを組み込みます。実行コンテキストが必要な場合については、『SQLJ での SQL ステートメントの実行の制御』を参照してください。

関連概念:

- 356 ページの『SQLJ アプリケーションでのコメント』
- 381 ページの『同じアプリケーションでの SQLJ と JDBC の使用』
- 384 ページの『DB2 Universal JDBC ドライバーを使用した SQLJ アプリケーションでの LOB』
- 391 ページの『SQLJ アプリケーションでのストアード・プロシージャーの複数の結果セットの検索』
- 366 ページの『SQLJ アプリケーションによる DB2 表からのデータの検索方法』

関連タスク:

- 392 ページの『SQLJ アプリケーションでのバッチ更新の実行』
- 379 ページの『SQLJ アプリケーションでのストアード・プロシージャーの呼び出し』
- 363 ページの『SQLJ トランザクションのコミットまたはロールバック』
- 366 ページの『SQLJ アプリケーションでの DB2 オブジェクトの作成および変更』
- 380 ページの『SQLJ アプリケーションでの SQL エラーの処理』
- 362 ページの『SQLJ トランザクションの分離レベルの設定』

- 367 ページの『SQLJ アプリケーションでの名前指定イテレーターの使用』
- 370 ページの『SQLJ アプリケーションでの位置指定イテレーターの使用』
- 373 ページの『SQLJ アプリケーションでの位置指定 UPDATE および DELETE 操作の実行』
- 399 ページの『SQLJ アプリケーションでのスクロール可能イテレーターの使用』
- 380 ページの『SQLJ アプリケーションでの SQL 警告の処理』
- 390 ページの『SQLJ での SQL ステートメントの実行の制御』

関連資料:

- 443 ページの『SQLJ 実行可能文節』

SQLJ アプリケーションでの DB2 オブジェクトの作成および変更

SQLJ 実行可能文節を使用して、以下を行います。

- データ定義ステートメント (CREATE、ALTER、DROP、GRANT、REVOKE) を実行する。
- INSERT、検索済み UPDATE、および検索済み DELETE ステートメントを実行する。

以下の実行可能ステートメントは、 INSERT、検索済み UPDATE、および検索済み DELETE の例です。

```
#sql [myConnCtx] {INSERT INTO DEPARTMENT VALUES
  ("X00","Operations 2","000030","E01",NULL)};
#sql [myConnCtx] {UPDATE DEPARTMENT
  SET MGRNO="000090" WHERE MGRNO="000030"};
#sql [myConnCtx] {DELETE FROM DEPARTMENT
  WHERE DEPTNO="X00"};
```

位置指定 UPDATE および DELETE については、 『SQLJ アプリケーションでの位置指定 UPDATE および DELETE 操作の実行』を参照してください。

関連タスク:

- 373 ページの『SQLJ アプリケーションでの位置指定 UPDATE および DELETE 操作の実行』

SQLJ アプリケーションによる DB2 表からのデータの検索方法

他の言語による DB2® アプリケーションの場合と同様、 SQLJ アプリケーションで DB2 表から単一行を検索する場合も、以下のようにその行のみを含む結果表を定義した WHERE 文節を使用して SELECT INTO ステートメントを作成することができます。

```
#sql [myConnCtx] {SELECT DEPTNO INTO :hvdeptno
  FROM DEPARTMENT WHERE DEPTNAME="OPERATIONS"};
```

ただし、実際に使用する SELECT ステートメントで作成される結果表のほとんどには、多数の行が含まれます。他の言語の DB2 アプリケーションでは、カーソルを使用して結果表の個々の行を選択します。このカーソルはスクロール不可にすることができます。この場合、そのカーソルを使用して行を取り出すときは、カーソルを結果表の始めから終わりまで逐次移動させます。あるいは、カーソルをスクロー

ル可能にすることもできます。この場合、そのカーソルを使用して行を取り出すときは、カーソルを結果表の中で順方向、逆方向、または任意の行に移動させることができます。

SQLJ でカーソルに相当するものが*結果セット・イテレーター* です。カーソルと同様に、結果セット・イテレーターも、スクロール不可またはスクロール可能にすることができます。このトピックでは、スクロール不可イテレーターの使用方法を説明します。スクロール可能イテレーターの使用については、 『SQLJ アプリケーションでのスクロール可能イテレーターの使用』を参照してください。

結果セット・イテレーターは、結果表の行の検索に使用する Java™ オブジェクトです。カーソルとは異なり、結果セット・イテレーターはパラメーターとしてメソッドに渡すことができます。

結果セット・イテレーターを使用する基本的なステップは、以下のとおりです。
1. イテレーターを宣言する。これによってイテレーター・クラスが作成されます。
2. イテレーター・クラスのインスタンスを定義する。
3. SELECT の結果表をイテレーターのインスタンスに割り当てる。
4. 行を検索する。
5. イテレーターをクローズする。

イテレーターには、*位置指定イテレーター* と *名前指定イテレーター* の 2 つのタイプがあります。位置指定イテレーターは、インターフェース `sqlj.runtime.PositionedIterator` の拡張です。位置指定イテレーターは、結果表の列を、結果表の中の位置で識別します。名前指定イテレーターは、インターフェース `sqlj.runtime.NamedIterator` の拡張です。名前指定イテレーターは、結果表の列を、結果表の列名で識別します。

関連タスク:

- 367 ページの『SQLJ アプリケーションでの名前指定イテレーターの使用』
- 370 ページの『SQLJ アプリケーションでの位置指定イテレーターの使用』
- 373 ページの『SQLJ アプリケーションでの位置指定 UPDATE および DELETE 操作の実行』

関連資料:

- 442 ページの『SQLJ イテレーター宣言文節』

SQLJ アプリケーションでの名前指定イテレーターの使用

名前指定イテレーターを使用するステップは、以下のとおりです。

1. イテレーターを宣言する。

   イテレーター宣言文節 を使用して、結果セット・イテレーターを宣言します。これによって、イテレーターと同じ名前のイテレーター・クラスが作成されます。名前指定イテレーターでは、イテレーター宣言文節で以下の情報を指定します。

   - イテレーターの名前。
   - 列名と Java™ データ・タイプのリスト。

- Java クラス宣言の情報。 イテレーターが public か static かといった情報です。
- 属性のセット。イテレーターが保持可能かどうか、列を更新できるかどうかといった属性です。

照会のための名前指定イテレーターを宣言するときは、イテレーターのそれぞれの列に名前を指定します。これらの名前は、照会の結果表の列の名前と一致していなければなりません。イテレーターの列名と結果表の列名とは、大文字小文字のみの相違であれば、一致した名前と見なされます。イテレーター宣言文節から得られる名前指定イテレーター・クラスには、アクセサー・メソッド が含まれます。イテレーターの列ごとに 1 つのアクセサー・メソッドがあります。アクセサー・メソッドの各名前は、対応するイテレーター列名と同じです。アクセサー・メソッドを使用して、結果表の列からデータを検索します。

イテレーターでは、対応する DB2® の列データ・タイプと厳密に一致する Java データ・タイプを指定する必要があります。 Java のデータ・タイプと DB2 のデータ・タイプの最適なマッピングのリストについては、 『Java、JDBC、および SQL のデータ・タイプ』を参照してください。

イテレーターを宣言するには、いくつかの方法があります。ただし、Java クラスが各イテレーターの基礎になっているため、イテレーターを宣言するときは、それを構成するクラスが Java 規則に従うようにする必要があります。たとえば、*with 文節* を含むイテレーターは、public として宣言する必要があります。したがって、イテレーターが public であることが必要な場合、public クラスが可能なところでのみ、それを宣言することができます。以下のリストでは、イテレーターを宣言するいくつかの代替方法を説明します。

- ソース・ファイル自体で public として宣言する。

  この方法では、イテレーター宣言を他のコード・モジュールで使用でき、すべての SQLJ アプリケーションに対して機能するイテレーターを備えることができます。また、同一ソース・ファイルに他の最上位クラスや public クラスがあっても問題はありません。
- 他に最上位クラスの定義があるソース・ファイル内で最上位クラスとして宣言する。

  Java では、コード・モジュール内で可能な public 最上位クラスは 1 つのみです。したがって、イテレーターが with 文節を含むときなど、イテレーターを public として宣言する必要がある場合、そのコード・モジュール内の他のクラスを public として宣言することはできません。
- 別のクラス内に、ネストされた静的クラスとして宣言する。

  この代替方法を使用すると、イテレーター宣言を同一ソース・ファイル内の他のクラス宣言と組み合わせて、イテレーターと他のクラスを public として宣言し、イテレーター・クラスを他のコード・モジュールまたはパッケージから可視にすることができます。ただし、ネスト・クラス外からイテレーターを参照するときは、イテレーター名はネスト・クラスの名前を使用して完全修飾する必要があります。
- 別のクラス内に内部クラスとして宣言する。

この方法でイテレーターを宣言すると、そのインスタンス化をネスト・クラスのインスタンス内でのみ行うことができます。ただし、ファイル内のイテレーターおよび他のクラスを public として宣言することもできます。

イテレーターが内部クラスとして宣言されている場合は、 JDBC ResultSet をイテレーターにキャストすることはできません。ネストされた静的クラスとして宣言されているイテレーターには、この制限は適用されません。イテレーターへの ResultSet のキャストの詳細については、『同じアプリケーションでの SQLJ と JDBC の使用』を参照してください。

2. イテレーター・クラスのインスタンスを作成する。

   名前指定イテレーター・クラスのオブジェクトを宣言して、結果表の行を検索します。

3. SELECT の結果表をイテレーターのインスタンスに割り当てる。

   SELECT の結果表をイテレーターに割り当てるには、SQLJ *代入文節* を使用します。名前指定イテレーターの代入文節の形式は、次のとおりです。

   ```
   #sql context-clause iterator-object={select-statement};
   ```

   詳しくは、『SQLJ 代入文節』 および 『SQLJ コンテキスト文節』を参照してください。

4. 行を検索する。

   これは、アクセサー・メソッドの呼び出しをループで実行することにより行います。アクセサー・メソッドの名前はイテレーターでの対応する列の名前と同じで、パラメーターはありません。アクセサー・メソッドは、結果表の現在行の対応する列からの値を戻します。結果表でカーソルを順方向に移動させるには、NamedIterator.next() メソッドを使用します。

   すべての行を検索したかどうかをテストするには、 next メソッドを呼び出したときに戻される値を検査します。次の行がない場合、next は値が false の boolean を戻します。

5. イテレーターをクローズする。

   NamedIterator.close メソッドを使用してこれを実行します。

以下のコードは、名前指定イテレーターを宣言して使用する方法を示しています。特定のステートメントの右側にある番号は、前述のステップに対応しています。

```
#sql  iterator ByName(String LastName, Date HireDate);                    1
                                // Declare named iterator ByName
{
  ByName nameiter;              // Declare object of ByName class          2
  #sql [ctxt]
 nameiter={SELECT LASTNAME, HIREDATE FROM EMPLOYEE};                       3
                                // Assign the result table of the SELECT
                                // to iterator object nameiter
  while (nameiter.next())       // Move the iterator through the result    4
                                // table and test whether all rows retrieved
  {
    System.out.println( nameiter.LastName() + " was hired on "
      + nameiter.HireDate());   // Use accessor methods LastName and
                                // HireDate to retrieve column values
  }
  nameiter.close();             // Close the iterator                      5
}
```

図 39. 名前指定イテレーターの使用

関連概念:

- 381 ページの『同じアプリケーションでの SQLJ と JDBC の使用』

関連タスク:

- 370 ページの『SQLJ アプリケーションでの位置指定イテレーターの使用』
- 373 ページの『SQLJ アプリケーションでの位置指定 UPDATE および DELETE 操作の実行』

関連資料:

- 403 ページの『Java、JDBC、および SQL のデータ・タイプ』
- 447 ページの『SQLJ 代入文節』
- 444 ページの『SQLJ コンテキスト文節』

SQLJ アプリケーションでの位置指定イテレーターの使用

位置指定イテレーターを使用するステップは、以下のとおりです。

1. イテレーターを宣言する。

   イテレーター宣言文節 を使用して、結果セット・イテレーターを宣言します。これによって、イテレーターと同じ名前と属性のイテレーター・クラスが作成されます。位置指定イテレーターでは、イテレーター宣言文節で以下の情報を指定します。

   - イテレーターの名前。
   - Java™ データ・タイプのリスト。
   - Java クラス宣言の情報。 イテレーターが `public` か `static` かといった情報です。
   - 属性のセット。イテレーターが保持可能かどうか、列を更新できるかどうかといった属性です。

   データ・タイプの宣言は結果表の中の列を表し、これが結果セット・イテレーターの列として参照されます。結果セット・イテレーターの列は、結果表の列に、左から右の順で対応します。たとえば、イテレーター宣言文節にデータ・タイプ

の宣言が 2 つある場合、最初のデータ・タイプ宣言は結果表の最初の列に対応し、 2 番目のデータ・タイプ宣言は結果表の 2 番目の列に対応します。

イテレーターでは、対応する DB2® の列データ・タイプと厳密に一致する Java データ・タイプを指定する必要があります。 Java のデータ・タイプと DB2 のデータ・タイプの最適なマッピングのリストについては、『Java、JDBC、および SQL のデータ・タイプ』を参照してください。

イテレーターを宣言するには、いくつかの方法があります。ただし、Java クラスが各イテレーターの基礎になっているため、イテレーターを宣言するときは、それを構成するクラスが Java 規則に従うようにする必要があります。たとえば、*with 文節* を含むイテレーターは、`public` として宣言する必要があります。したがって、イテレーターが `public` であることが必要な場合、`public` クラスが可能なところでのみ、それを宣言することができます。以下のリストでは、イテレーターを宣言するいくつかの代替方法を説明します。

- ソース・ファイル自体で `public` として宣言する。

  これは、イテレーターを宣言する方法としては最も用途が広いものです。この方法では、イテレーター宣言を他のコード・モジュールで使用でき、すべての SQLJ アプリケーションに対して機能するイテレーターを備えることができます。また、同一ソース・ファイルに他の最上位クラスや `public` クラスがあっても問題はありません。
- 他に最上位クラスの定義があるソース・ファイル内で最上位クラスとして宣言する。

  Java では、コード・モジュール内で可能な public 最上位クラスは 1 つのみです。したがって、イテレーターが with 文節を含むときなど、イテレーターを public として宣言する必要がある場合、そのコード・モジュール内の他のクラスを public として宣言することはできません。
- 別のクラス内に、ネストされた静的クラスとして宣言する。

  この代替方法を使用すると、イテレーター宣言を同一ソース・ファイル内の他のクラス宣言と組み合わせて、イテレーターと他のクラスを public として宣言し、イテレーター・クラスを他のコード・モジュールまたはパッケージから可視にすることができます。ただし、ネスト・クラス外からイテレーターを参照するときは、イテレーター名はネスト・クラスの名前を使用して完全修飾する必要があります。
- 別のクラス内に内部クラスとして宣言する。

  この方法でイテレーターを宣言すると、そのインスタンス化をネスト・クラスのインスタンス内でのみ行うことができます。ただし、ファイル内のイテレーターおよび他のクラスを public として宣言することもできます。

  イテレーターが内部クラスとして宣言されている場合は、 JDBC `ResultSet` をイテレーターにキャストすることはできません。ネストされた静的クラスとして宣言されているイテレーターには、この制限は適用されません。イテレーターへの `ResultSet` のキャストの詳細については、『同じアプリケーションでの SQLJ と JDBC の使用』を参照してください。

2. イテレーター・クラスのインスタンスを作成する。

位置イテレーター・クラスのオブジェクトを宣言して、結果表の行を検索します。

3. SELECT の結果表をイテレーターのインスタンスに割り当てる。

   SELECT の結果表をイテレーターに割り当てるには、SQLJ *代入文節* を使用します。位置指定イテレーターの代入文節の形式は、次のとおりです。

   ```
   #sql context-clause iterator-object={select-statement};
   ```

4. 行を検索する。

   これを行うには、ループにした実行可能文節で FETCH ステートメントを実行します。この FETCH ステートメントは、他の言語の FETCH ステートメントと同様です。

   すべての行を検索したかどうかをテストするには、各 FETCH 後に PositionedIterator.endFetch メソッドを呼び出します。検索する行がなくなったために FETCH が失敗すると、endFetch は 値が true の boolean を戻します。

5. イテレーターをクローズする。

   PositionedIterator.close メソッドを使用して、これを実行します。

以下のコードは、位置指定イテレーターを宣言して使用する方法を示しています。特定のステートメントの右側にある番号は、前述のステップに対応しています。

```
#sql iterator ByPos(String,Date); // Declare positioned iterator ByPos  1
{
  ByPos positer;                  // Declare object of ByPos class    2
  String name = null;             // Declare host variables
  Date hrdate;
  #sql [ctxt] positer =
    {SELECT LASTNAME, HIREDATE FROM EMPLOYEE};                         3
                                  // Assign the result table of the SELECT
                                  // to iterator object positer
  #sql {FETCH :positer INTO :name, :hrdate };                         4
                                  // Retrieve the first row
  while (!positer.endFetch())     // Check whether the FETCH returned a row
  { System.out.println(name + " was hired in " +
      hrdate);
    #sql {FETCH :positer INTO :name, :hrdate };
                                  // Fetch the next row
  }
  positer.close();                // Close the iterator                5
}
```

図 40. 位置指定イテレーターの使用

関連概念:

- 381 ページの『同じアプリケーションでの SQLJ と JDBC の使用』
- 366 ページの『SQLJ アプリケーションによる DB2 表からのデータの検索方法』

関連タスク:

- 367 ページの『SQLJ アプリケーションでの名前指定イテレーターの使用』

関連資料:

- 403 ページの『Java、JDBC、および SQL のデータ・タイプ』

SQLJ アプリケーションでの位置指定 UPDATE および DELETE 操作の実行

他の言語の DB2® アプリケーションの場合と同様、位置指定 UPDATE および DELETE の実行は、結果表の行の検索の拡張機能です。基本的なステップは以下のとおりです。

1. イテレーターを宣言する。

   イテレーターは位置指定または名前指定にすることができます。位置指定 UPDATE または DELETE 操作のためのイテレーターは、更新可能として宣言する必要があります。このためには、宣言に以下の文節を含める必要があります。

   implements sqlj.runtime.ForUpdate
   この文節によって、生成されるイテレーター・クラスに、更新可能イテレーターを使用する場合のメソッドが組み込まれます。位置指定 UPDATE または DELETE 操作を含むプログラムには、この文節が必要です。

   with (updateColumns=*"column-list"***)**
   この文節は、イテレーターが更新する結果表の列の、コンマで区切られたリストを指定します。この文節はオプションです。

   イテレーターは `public` として宣言する必要があります。したがって、`public` イテレーターを同一ファイルまたは異なるファイルで宣言して使用する場合の規則に従う必要があります。

   イテレーターをファイル内でそれ自体で宣言すると、そのイテレーターにアドレス可能で、生成されるクラスをインポートする SQLJ ソース・ファイルであれば、そのイテレーターを使用して、データを検索し、位置指定 UPDATE または DELETE ステートメントを実行することができます。位置指定 UPDATE または DELETE ステートメントの実行のための許可 ID は、そのステートメントを静的に実行するか動的に実行するかによって異なります。ステートメントを静的に実行する場合の許可 ID は、そのステートメントが組み込まれた DB2 のプランまたはパッケージの所有者になります。ステートメントを動的に実行する場合の許可 ID は、そのときの実際の DYNAMICRULES の振る舞いで決定されます。DB2 Universal JDBC ドライバーの場合、振る舞いは常に DYNAMICRULES BIND になります。

2. 接続の自動コミット・モードを使用不可にする。

   自動コミット・モードを使用可能にしていると、位置指定 UPDATE ステートメントを実行するたびに COMMIT 操作が起こり、イテレーターに `with (holdability=true)` 属性が指定されていなければ、イテレーターは破棄されることになります。したがって、イテレーターの使用を終えるまでは自動コミットをオフにして、 COMMIT 操作が起こらないようにする必要があります。各更新操作後に COMMIT が起こるようにする場合、各 COMMIT 操作後にイテレーターが破棄されないようにする代替方法として、イテレーターを `with (holdability=true)` で宣言する方法があります。

3. イテレーター・クラスのインスタンスを作成する。

   このステップは更新不可イテレーターの場合と同じです。

4. SELECT の結果表をイテレーターのインスタンスに割り当てる。

   このステップは更新不可イテレーターの場合と同じです。 SELECT ステートメントに FOR UPDATE 文節を組み込むことはできません。
5. 行を検索して更新する。

   位置指定イテレーターの場合、以下のアクションをループで実行します。

   a. 実行可能文節で FETCH ステートメントを実行して現在行を取得する。
   b. イテレーターが結果表の行を指しているかどうかを、 `PositionedIterator.endFetch` メソッドを呼び出してテストする。
   c. イテレーターが結果表の行を指している場合は、実行可能文節でSQL UPDATE... WHERE CURRENT OF *:iterator-object* ステートメントを実行して、現在行の列を更新する。現在行を削除するには、実行可能文節で SQL DELETE... WHERE CURRENT OF *:iterator-object* ステートメントを実行します。

   名前指定イテレーターの場合、以下のアクションをループで実行します。

   a. `next` メソッドを呼び出して、イテレーターを順方向に移動させる。
   b. イテレーターが結果表の行を指しているかどうかを、 `next` が `true` を戻すかどうかを調べてテストする。
   c. 実行可能文節でSQL UPDATE... WHERE CURRENT OF *:iterator-object* ステートメントを実行して、現在行の列を更新する。現在行を削除するには、実行可能文節で SQL DELETE... WHERE CURRENT OF *:iterator-object* ステートメントを実行します。
6. イテレーターをクローズする。

   `close` メソッドを使用して、これを実行します。

以下のコードは、位置指定イテレーターを宣言して、それを位置指定 UPDATE に使用する方法を示しています。特定のステートメントの右側にある番号は、前述のステップに対応しています。

まず以下のように、1 ファイルで位置指定イテレーター `UpdByPos` を宣言し、列 `SALARY` の更新にそのイテレーターを使用することを指定します。

```
import java.math.*;          // Import this class for BigDecimal data type
#sql public iterator UpdByPos implements sqlj.runtime.ForUpdate       1
  with(updateColumns="SALARY") (String, BigDecimal);
```

図 41. 位置指定 UPDATE のための位置指定イテレーターの宣言

次に別のファイルで、以下のコード・フラグメントで示すように、位置指定 UPDATE に `UpdByPos` を使用します。

```
import sqlj.runtime.*;       // Import files for SQLJ and JDBC APIs
import java.sql.*;
import java.math.*;          // Import this class for BigDecimal data type
import UpdByPos;             // Import the generated iterator class that
                             // was created by the iterator declaration clause
                             // for UpdByName in another file
#sql context HSCtx;          // Create a connnection context class HSCtx
public static void main (String args[])
{
  try {
    Class.forName("com.ibm.db2.jcc.DB2Driver");
  }
  catch (ClassNotFoundException e) {
   e.printStackTrace();
          }
  Connection HSjdbccon=
  DriverManager.getConnection("jdbc:db2:SANJOSE");
                     // Create a JDBC connection object
  HSjdbccon.setAutoCommit(false);
                     // Set autocommit off so automatic commits       2
                     // do not destroy the cursor between updates
  HSCtx myConnCtx=new HSCtx(HSjdbccon);
                     // Create a connection context object
  UpdByPos upditer;  // Declare iterator object of UpdByPos class     3
  String enum;       // Declares host variable to receive EMPNO
  BigDecimal sal;    // and SALARY column values
    #sql [myConnCtx]
    upditer = {SELECT EMPNO, SALARY FROM EMPLOYEE                     4
      WHERE WORKDEPT='D11'};
                     // Assign result table to iterator object
  #sql {FETCH :upditer INTO :enum,:sal};                              5a
                     // Move cursor to next row
  while (!upditer.endFetch())                                         5b
                     // Check if on a row
  {
    #sql [myConnCtx] {UPDATE EMPLOYEE SET SALARY=SALARY*1.05
        WHERE CURRENT OF :upditer};                                   5c
                     // Perform positioned update
    System.out.println("Updating row for " + enum);
    #sql {FETCH :upditer INTO :enum,:sal};
                     // Move cursor to next row
  }
  upditer.close();           // Close the iterator                   6
  #sql [myConnCtx] {COMMIT};
                     // Commit the changes
  myConnCtx.close();         // Close the connection context
}
```

図 42. 位置指定イテレーターを使用した位置指定 UPDATE の実行

以下のコードは、名前指定イテレーターを宣言して、それを位置指定 UPDATE に使用する方法を示しています。特定のステートメントの右側にある番号は、前述のステップに対応しています。

まず以下のように、1 ファイルで名前指定イテレーター UpdByName を宣言し、列 SALARY の更新にそのイテレーターを使用することを指定します。

```
import java.math.*;          // Import this class for BigDecimal data type
#sql public iterator UpdByName implements sqlj.runtime.ForUpdate     1
  with(updateColumns="SALARY") (String EmpNo, BigDecimal Salary);
```

図 43. 位置指定 UPDATE のための名前指定イテレーターの宣言

次に別のファイルで、以下のコード・フラグメントで示すように、位置指定 UPDATE に UpdByName を使用します。

```
import sqlj.runtime.*;       // Import files for SQLJ and JDBC APIs
import java.sql.*;
import java.math.*;          // Import this class for BigDecimal data type
import UpdByName;            // Import the generated iterator class that
                             // was created by the iterator declaration clause
                             // for UpdByName in another file
#sql context HSCtx;          // Create a connnection context class HSCtx
public static void main (String args[])
{
  try {
    Class.forName("com.ibm.db2.jcc.DB2Driver");
  }
  catch (ClassNotFoundException e) {
   e.printStackTrace();
          }
  Connection HSjdbccon=
  DriverManager.getConnection("jdbc:db2:SANJOSE");
                             // Create a JDBC connection object
  HSjdbccon.setAutoCommit(false);
                             // Set autocommit off so automatic commits  2
                             // do not destroy the cursor between updates
  HSCtx myConnCtx=new HSCtx(HSjdbccon);
                             // Create a connection context object
  UpdByName upditer;                                                  3
                             // Declare iterator object of UpdByName class
  String enum;                   // Declare host variable to receive EmpNo
                                 // column values
  #sql [myConnCtx]
    upditer = {SELECT EMPNO, SALARY FROM EMPLOYEE                     4
      WHERE WORKDEPT='D11'};
                                 // Assign result table to iterator object
  while (upditer.next())                                              5a, 5b
                                 // Move cursor to next row and
                                 // check ifon a row
  {
    enum = upditer.EmpNo();  // Get employee number from current row
    #sql [myConnCtx]
      {UPDATE EMPLOYEE SET SALARY=SALARY*1.05
        WHERE CURRENT OF :upditer};                                   5c
                                 // Perform positioned update
    System.out.println("Updating row for " + enum);
  }
  upditer.close();               // Close the iterator                6
  #sql [myConnCtx] {COMMIT};
                                 // Commit the changes
  myConnCtx.close();             // Close the connection context
}
```

図 44. 名前指定イテレーターを使用した位置指定 UPDATE の実行

関連概念:

- 366 ページの『SQLJ アプリケーションによる DB2 表からのデータの検索方法』

- 396 ページの『SQLJ アプリケーションでの、位置指定 UPDATE または DELETE 操作の変数としてのイテレーターの受け渡し』

関連タスク:

- 357 ページの『SQLJ を使用したデータ・ソースへの接続』

SQLJ アプリケーションにおける同じ SQL ステートメントの複数のオープン・イテレーター

DB2 Universal JDBC ドライバーを使用していて、アプリケーションが DB2 UDB for z/OS® バージョン 8 サーバー、またはフィックスパック 4 レベル以降の DB2 UDB for Linux, UNIX®, Windows® サーバーに接続する場合、 SQLJ アプリケーション内の単一の SQL ステートメントで複数のイテレーターを同時にオープンすることができます。この機能により、表に対して 1 つのイテレーターを使用してある操作を実行しながら、同じ表で別のイテレーターを使用して別の操作を実行することができます。

アプリケーションで同時にオープンするイテレーターを使用する場合、 Java™ ヒープでストレージの使用量が増大することを防ぐため、必要としなくなったイテレーターはクローズしてください。

以下の例は、ある表で同じ操作を、単一の SQL ステートメントで同時にイテレーターをオープンしないで実行する方法と、単一の SQL ステートメントで同時にイテレーターをオープンして実行する方法を示しています。これらの例では、以下のイテレーター宣言を使用しています。

```
import java.math.*;
#sql public iterator  MultiIter(String EmpNo, BigDecimal Salary);
```

単一の SQL ステートメントで複数のイテレーターを同時にオープンする機能を使用しないとき、特定の従業員番号に対する従業員および給料の値を選択したい場合は、図 45 に示すように、従業員番号ごとに異なる SQL ステートメントを定義する必要があります。

```
MultiIter iter1 = null;            // Iterator instance for retrieving
                                   // data for first employee
String EmpNo1 = "000100";          // Employee number for first employee
#sql [ctx] iter2 =
  {SELECT EMPNO, SALARY FROM EMPLOYEE WHERE EMPNO = :EmpNo1};
                                   // Assign result table to first iterator
MultiIter iter2 = null;            // Iterator instance for retrieving
                                   // data for second employee
String EmpNo2 = "000200";          // Employee number for second employee
#sql [ctx] iter2 =
  {SELECT EMPNO, SALARY FROM EMPLOYEE WHERE EMPNO = :EmpNo2};
                                   // Assign result table to second iterator
// Process with iter1
// Process with iter2
iter1.close();                     // Close the iterators
iter2.close();
```

図 45. 異なる SQL ステートメントでイテレーターを使用した、表の同時操作

図 46 では、同じ操作を単一の SQL ステートメントで同時に複数のイテレーターをオープンする機能を使用して実行する方法を示しています。

```
...
MultiIter iter1 = openIter("000100"); // Invoke openIter to assign the result table
                                      // (for employee 100) to the first iterator
MultiIter iter2 = openIter("000200"); // Invoke openIter to assign the result
                                      // table to the second iterator
                                      // iter1 stays open when iter2 is opened
// Process with iter1
// Process with iter2
...
iter1.close();                        // Close the iterators
iter2.close();
...
public MultiIter openIter(String EmpNo)
                                      // Method to assign a result table
                                      // to an iterator instance
{
  MultiIter iter;
  #sql [ctxt] iter =
    {SELECT EMPNO, SALARY FROM EMPLOYEE WHERE EMPNO = :EmpNo};
  return iter;                        // Method returns an iterator instance
}
```

図 46. 同じ SQL ステートメントでイテレーターを使用した、表の同時操作

関連概念:

- 366 ページの『SQLJ アプリケーションによる DB2 表からのデータの検索方法』

SQLJ アプリケーションにおけるイテレーターの複数のオープン・インスタンス

イテレーターの複数インスタンスを、単一の SQLJ アプリケーションで同時にオープンすることができます。この能力の応用として、たとえば、ホスト式を使用する単一のイテレーターの複数のインスタンスをオープンすることができます。各インスタンスでは、異なるセットのホスト式の値を使用できます。

以下の例は、単一のイテレーターの 2 つのインスタンスを同時にオープンしているアプリケーションを示しています。

```
...
ResultSet myFunc(String empid)  // Method to open an iterator and get a resultSet
{
  MyIter iter;
  #sql iter = {SELECT * FROM EMPLOYEE WHERE EMPNO = :empid};
  return iter.getResultSet();
}

// An application can call this method to get a resultSet for each
// employee ID. The application can process each resultSet separately.
...
ResultSet rs1 = myFunc("000100");  // Get employee record for employee ID 000100
...
ResultSet rs2 = myFunc("000200");  // Get employee record for employee ID 000200
```

図 47. 単一のアプリケーションでイテレーターの複数のインスタンスをオープンする例

他のイテレーターの場合と同様、ストレージの使用量が増大することを防ぐため、最後に使用した後にイテレーターを必ずクローズする必要があります。

関連概念:

- 366 ページの『SQLJ アプリケーションによる DB2 表からのデータの検索方法』

SQLJ アプリケーションでのストアード・プロシージャーの呼び出し

ストアード・プロシージャーを呼び出すには、 SQL CALL ステートメントを含む実行可能文節を使用します。定数パラメーターまたはホスト ID パラメーターを指定して CALL ステートメントを実行することができます。ストアード・プロシージャーを呼び出す基本的なステップは、以下のとおりです。

1. 入力 (IN または INOUT) パラメーターに値を割り当てる。
2. ストアード・プロシージャーを呼び出す。
3. 出力 (OUT または INOUT) パラメーターを処理する。
4. ストアード・プロシージャーから結果セットが複数戻された場合は、それらの結果セットを検索する。『SQLJ アプリケーションでのストアード・プロシージャーの複数の結果セットの検索』を参照してください。

以下のコードは、3 つの入力パラメーターと 3 つの出力パラメーターを持つストアード・プロシージャーの呼び出しを示しています。特定のステートメントの右側にある番号は、前述のステップに対応しています。

```
String FirstName="TOM";            // Input parameters                  1
String LastName="NARISINST";
String Address="IBM";
int CustNo;                        // Output parameters
String Mark;
String MarkErrorText;
...
#sql [myConnCtx] {CALL ADD_CUSTOMER(:IN FirstName,                       2
                            :IN LastName,
                            :IN Address,
                            :OUT CustNo,
                            :OUT Mark,
                            :OUT MarkErrorText)};
                                  // Call the stored procedure
System.out.println("Output parameters from ADD_CUSTOMER call: ");
System.out.println("Customer number for " + LastName + ": " + CustNo); 3
System.out.println(Mark);
If (MarkErrorText != null)
  System.out.println(" Error messages:" + MarkErrorText);
```

図 48. SQLJ アプリケーションでのストアード・プロシージャーの呼び出し

関連概念:

- 391 ページの『SQLJ アプリケーションでのストアード・プロシージャーの複数の結果セットの検索』

SQLJ アプリケーションでの SQL エラーの処理

SQLJ 文節では、エラー処理に JDBC クラス java.sql.SQLException を使用します。 SQLJ は、以下の状況で SQLException を生成します。

- いずれかの SQL ステートメントが負の SQL エラー・コードを戻したとき。
- SELECT INTO SQL ステートメントが +100 SQL エラー・コードを戻したとき。

SQL エラー・コードの検索には getErrorCode メソッドを、SQLSTATE の検索には getSQLState メソッドを使用することができます。

SQLJ アプリケーションで SQL エラーを処理するには、java.sql.SQLException クラスをインポートし、SQL エラーが発生したときに Java™ エラー処理の try/catch ブロックを使用して、プログラムの流れを変更します。たとえば、以下のようにします。

```
try {
  #sql [ctxt] {SELECT LASTNAME INTO :empname
    FROM EMPLOYEE WHERE EMPNO='000010'};
}
catch(SQLException e) {
  System.out.println("Error code returned: " + e.getErrorCode());
}
```

DB2 Universal JDBC ドライバーを使用して SQLCA を検索することができます。DB2 Universal JDBC ドライバーを使用して SQLCA を検索するコードの作成については、『DB2 Universal JDBC ドライバー使用時の SQLException の処理』を参照してください。

Linux、UNIX®、および Windows® 用 DB2 JDBC Type 2 ドライバー (DB2 JDBC Type 2 ドライバー) の場合は、標準 SQLException を使用して SQL エラー情報を検索します。

関連タスク:

- 315 ページの『DB2 Universal JDBC ドライバー使用時の SQLException の処理』

SQLJ アプリケーションでの SQL 警告の処理

DB2® 警告では、SELECT INTO ステートメントでの +100 SQL エラー・コード以外は、SQLExceptions が出されません。 DB2 警告を処理するには、プログラムで java.sql.SQLWarning クラスにアクセスするようにする必要があります。警告に関する DB2 固有の情報を取り出す場合は、プログラムが com.ibm.db2.jcc.DB2Diagnosable インターフェースおよび com.ibm.db2.jcc.DB2Sqlca クラスにもアクセスできるようにする必要があります。DB2 警告を検査するには、SQL 文節を実行した後に getWarnings メソッドを呼び出します。 getWarnings は、SQL ステートメントが生成した最初の SQLWarning オブジェクトを戻します。後続の SQLWarning オブジェクトは、最初のものにチェーニングされます。

DB2 Universal JDBC ドライバーを使用して SQLWarning オブジェクトから DB2 固有の情報を取り出す方法は、 『DB2 Universal JDBC ドライバー使用時の SQLException の処理』の説明に従ってください。

SQL 文節の getWarnings を実行するには、その SQL 文節の実行コンテキストをあらかじめセットアップしておく必要があります。実行コンテキストのセットアップ方法については、 『SQLJ での SQL ステートメントの実行の制御』を参照してください。以下の例は、実行コンテキストが execCtx の SQL 文節の場合の SQLWarning オブジェクトを取り出す方法を示しています。

```
ExecutionContext execCtx=myConnCtx.getExecutionContext();
                                     // Get default execution context from
                                     // connection context
SQLWarning sqlWarn;
...
#sql [myConnCtx,execCtx] {SELECT LASTNAME INTO :empname
  FROM EMPLOYEE WHERE EMPNO='000010'};
if ((sqlWarn = execCtx.getWarnings()) != null)
System.out.println("SQLWarning " + sqlWarn);
```

関連タスク:

- 315 ページの『DB2 Universal JDBC ドライバー使用時の SQLException の処理』
- 390 ページの『SQLJ での SQL ステートメントの実行の制御』
- 380 ページの『SQLJ アプリケーションでの SQL エラーの処理』

高度な SQLJ アプリケーション・プログラミングの概念

以下のトピックには、SQLJ アプリケーションの作成に関するより高度な情報が含まれています。

同じアプリケーションでの SQLJ と JDBC の使用

SQLJ 文節と JDBC 呼び出しは、単一プログラム内で結合することができます。これを効果的に行うには、以下を実行できることが必要です。

- JDBC Connection を使用して SQLJ ConnectionContext を作成するか、または SQLJ ConnectionContext から JDBC Connection を取得する。
- SQLJ イテレーターを使用して JDBC ResultSet のデータを検索するか、または SQLJ イテレーターから JDBC ResultSet を生成する。

JDBC Connection からの SQLJ ConnectionContext の作成: これを行うには、以下のようにします。

1. SQLJ 接続宣言文節を実行して ConnectionContext クラスを作成する。
2. ドライバーをロードするか、または DataSource インスタンスを取得する。
3. JDBC DriverManager.getConnection または DataSource.getConnection メソッドを呼び出して、JDBC Connection を取得する。
4. Connection を引き数として指定して ConnectionContext コンストラクターを呼び出し、 ConnectionContext オブジェクトを作成する。

SQLJ ConnectionContext からの JDBC Connection の取得: これを行うには、以下のようにします。

1. SQLJ 接続宣言文節を実行して ConnectionContext クラスを作成する。
2. ドライバーをロードするか、または DataSource インスタンスを取得する。

3. ドライバーの URL および他の必要なパラメーターを引き数として指定して ConnectionContext コンストラクターを呼び出し、 ConnectionContext オブジェクトを作成する。
4. JDBC ConnectionContext.getConnection メソッドを呼び出して、JDBC Connection オブジェクトを作成する。

SQLJ 接続の詳細については、『SQLJ を使用したデータ・ソースへの接続』を参照してください。

SQLJ イテレーターを使用した JDBC 結果セットの検索: イテレーター変換ステートメント を使用して、JDBC 結果セットを SQLJ イテレーターとして操作します。イテレーター変換ステートメントの一般的な形式は、次のとおりです。

```
#sql iterator={CAST :result-set};
```

結果セットをイテレーターに正常にキャストできるようにするには、イテレーターは以下の規則に従っている必要があります。

- イテレーターが public として宣言されていなければなりません。
- イテレーターが位置指定イテレーターの場合、結果セットでの列の数がイテレーターでの列の数と一致していなければなりません。さらに、結果セットの各列のデータ・タイプが、イテレーターでの対応する列のデータ・タイプと一致していなければなりません。
- イテレーターが名前指定イテレーターである場合、各アクセサー・メソッドの名前は、結果セット内の列の名前と同じでなければなりません。さらに、アクセサー・メソッドが戻すオブジェクトのデータ・タイプは、結果セット内の対応する列のデータ・タイプと一致していなければなりません。

図 49 に示すコードは、JDBC 呼び出しを使用して照会をビルドおよび実行し、イテレーター変換ステートメントを実行して JDBC 結果セットを SQLJ イテレーターに変換し、そのイテレーターを使用して結果表の行を検索します。

```
#sql public iterator ByName(String LastName, Date HireDate);      1
public void HireDates(ConnectionContext connCtx, String whereClause)
{

  ByName nameiter;           // Declare object of ByName class
  Connection conn=connCtx.getConnection();
                             // Create JDBC connection
  Statement stmt = conn.createStatement();                          2
  String query = "SELECT LASTNAME, HIREDATE FROM EMPLOYEE";
  query+=whereClause;   // Build the query
  ResultSet rs = stmt.executeQuery(query);                          3
  #sql [connCtx] nameiter = {CAST :rs};                             4
  while (nameiter.next())
  {
    System.out.println( nameiter.LastName() + " was hired on "
      + nameiter.HireDate());

  }
  nameiter.close();                                                 5
  stmt.close();
}
```

図 49. JDBC 結果セットの SQLJ イテレーターへの変換

382 ページの図 49 の注:

1 この SQLJ 文節は、名前指定イテレーター・クラス ByName を作成します。このクラスにはアクセサー・メソッドの LastName() と HireDate() が含まれ、これらのメソッドは結果表の列 LASTNAME と HIREDATE のデータを戻します。
2 このステートメントとこれに続く 2 つのステートメントは、 JDBC を使用した動的実行の照会のビルドと準備を行います。
3 このJDBC ステートメントは SELECT ステートメントを実行し、結果表を結果セット rs に割り当てます。
4 このイテレーター変換文節は JDBC ResultSet rs を SQLJ イテレーター nameiter に変換し、これに続くステートメントは nameiter を使用して結果表から値を検索します。
5 nameiter.close() メソッドは、SQLJ イテレーター と JDBC ResultSet rs をクローズします。

SQLJ イテレーターからの JDBC ResultSet の生成: getResultSet メソッドを使用して SQLJ イテレーターから JDBC ResultSet を生成します。すべての SQLJ イテレーターには、getResultSet メソッドが含まれています。イテレーターを結果セットに変換した後の行の取り出しには、結果セットのみを使用する必要があります。

図 50 に示すコードは、照会のための位置指定イテレーターを生成し、そのイテレーターを結果セットに変換して、JDBC メソッドを使用して表から行を取り出します。

```
#sql iterator EmpIter(String, java.sql.Date);
{
...
  EmpIter iter=null;
  #sql [connCtx] iter=
    {SELECT LASTNAME, HIREDATE FROM EMPLOYEE};                  1
  ResultSet rs=iter.getResultSet();                             2
  while (rs.next())                                             3
  { System.out.println(rs.getString(1) + " was hired in " +
      rs.getDate(2));
  }
  rs.close();                                                   4
}
```

図 50. SQLJ イテレーターの JDBC ResultSet への変換

図 50 の注:

1 この SQLJ 文節は SELECT ステートメントを実行し、 SELECT ステートメントの結果テーブルを含むイテレーター・オブジェクトを構成してから、変数 iter にイテレーター・オブジェクトを割り当てます。
2 getResultSet() メソッドは、イテレーター iter を ResultSet rs に変換します。
3 JDBC getString() および getDate() メソッドは、ResultSet から値を検索します。next() メソッドは、カーソルを ResultSet 内の次の行に移動します。
4 rs.close() メソッドは、ResultSet だけでなく SQLJ イテレーターもクローズします。

SQLJ アプリケーションで JDBC ResultSets を使用する場合の規則と制約事項: JDBC 結果セットを含む SQLJ アプリケーションを作成するときは、以下の規則および制約事項に従ってください。

- ResultSet とイテレーターの保持可能性属性が異なる場合は、 ResultSet を SQLJ イテレーターにキャストすることはできません。

  JDBC ResultSet または SQLJ イテレーターは、 COMMIT 操作後もオープンしたままにしておくことができます。 JDBC ResultSet の場合、この特性は、DB2 Universal JDBC ドライバー プロパティー resultSetHoldability で制御します。 SQLJ イテレーターの場合、この特性は、イテレーター宣言の with holdability パラメーターで制御します。保持可能性を持つ ResultSet を、保持可能性を持たない SQLJ イテレーターにキャストすること、また、保持可能性を持たない ResultSet を、保持可能性を持つ SQLJ イテレーターにキャストすることはサポートされていません。
- 生成された ResultSet オブジェクトまたはその基になったイテレーターは、プログラムの終了時にクローズしてください。

  ResultSet オブジェクトが生成されたイテレーター・オブジェクトをクローズすると、 ResultSet オブジェクトもクローズされます。生成された ResultSet オブジェクトをクローズすると、イテレーター・オブジェクトもクローズされます。一般に、最後に使用したオブジェクトをクローズするのが最適の方法です。
- DB2 Universal JDBC ドライバーでは、スクロール可能イテレーターと、スクロール可能および更新可能 ResultSet がサポートされていますが、この場合は以下の制限事項が適用されます。
  - スクロール可能イテレーターは、その基礎となる JDBC ResultSet と同じ制約事項を持ちます。たとえば、スクロール可能 ResultSet は INSERT をサポートしていないため、スクロール可能イテレーターも INSERT をサポートしません。
  - 更新可能ではない JDBC ResultSet を、更新可能 SQLJ イテレーターにキャストすることはできません。

関連タスク:

- 357 ページの『SQLJ を使用したデータ・ソースへの接続』

DB2 Universal JDBC ドライバーを使用した SQLJ アプリケーションでの LOB

DB2 Universal JDBC ドライバーを使用すると、LOB データの Clob または Blob ホスト式への取り出しや、 Clob または Blob ホスト式で CLOB、 BLOB、DBCLOB 列の更新ができます。また、Clob または Blob データ・タイプのイテレーターを宣言して、 CLOB、BLOB、または DBCLOB 列のデータを検索することができます。

LOB データの検索または更新: BLOB 列のデータを検索するには、データ・タイプ Blob または byte[] を含むイテレーターを宣言します。 CLOB または DBCLOB 列のデータを検索するには、対応する列に Clob データ・タイプがあるイテレーターを宣言します。

BLOB 列のデータを更新するには、データ・タイプが Blob のホスト式を使用します。 CLOB 列または DBCLOB 列のデータを更新するには、データ・タイプが Clob のホスト式を使用します。

LOB ロケーターのサポート: DB2 Universal JDBC ドライバーは、LOB ロケーターを使用してデータを検索することができます。 JDBC で LOB ロケーターを使用して、LOB 列からデータを検索するには、 `fullyMaterializeLobData` プロパティーを `false` に設定する必要があります。プロパティーについては、 『DB2® Universal JDBC ドライバーのプロパティー』で説明されています。 `fullyMaterializeLobData` は、両方向スクロール・カーソルを使用して取り出されるストアード・プロシージャーの出力パラメーターまたは LOB には影響しません。 LOB ロケーターのパラメーターを持つストアード・プロシージャーを呼び出すことはできません。両方向スクロール・カーソルで取り出す場合、JDBC は常に LOB ロケーターを使用して LOB 列からデータを検索します。

他の言語の場合と同様、Java アプリケーションにおける LOB ロケーターは、単一のデータベースと関連付けられています。単一の LOB ロケーターを使用して、 2 つの異なるデータベース の間でデータを移動させることはできません。 LOB データを 2 つのデータベース間で移動するには、最初のデータベースにある表から LOB データを検索するときにそれをマテリアライズし、次いでそのデータを 2 番目のデータベースにある表に挿入する必要があります。

関連資料:

- 408 ページの『DB2 Universal JDBC ドライバーのプロパティー』
- 403 ページの『Java、JDBC、および SQL のデータ・タイプ』

SQLJ アプリケーションでの LOB 列データの検索および更新のための Java データ・タイプ

deferPrepares プロパティーが true に設定されている場合、 DB2 Universal JDBC ドライバーがホスト式を含むカスタマイズされていない SQLJ ステートメントを処理すると、ドライバーはデータ・タイプを判別するために余分の処理を行わなければならない場合があります。この追加的な処理はパフォーマンスに影響を及ぼす可能性があります。

LOB 列とともに使用されるパラメーターのデータ・タイプを、 JDBC ドライバーがすぐに判別できない場合、 LOB データ・タイプと互換性のあるパラメーターのデータ・タイプを選択する必要があります。

BLOB 列の入力パラメーター:

BLOB 列に対する入力パラメーターでは、以下の技法のいずれかを使用できます。

- 以下のように、BLOB 列と完全に一致する java.sql.Blob 入力変数を使用します。

```
java.sql.Blob blobData;
#sql {CALL STORPROC(:IN blobData)};
```

  java.sql.Blob 入力変数を使用する前に、java.sql.Blob オブジェクトを作成し、そのオブジェクトにデータを追加しておく必要があります。たとえば、DB2 Universal JDBC ドライバーを使用している場合、 DB2 固有のメソッド `com.ibm.db2.jcc.t2zos.DB2LobFactory.createBlob` を使用して java.sql.Blob オブジェクトを作成し、そのオブジェクトに byte[] データを追加することができます。

```
byte[] byteArray = {0, 1, 2, 3};
java.sql.Blob blobData =
  com.ibm.db2.jcc.t2zos.DB2LobFactory.createBlob(byteArray);
```

- sqlj.runtime.BinaryStream のタイプの入力パラメーターを使用する。
  sqlj.runtime.BinaryStream オブジェクトは、BLOB データ・タイプと互換性があります。この呼び出しでは、入力データの正確な長さを指定する必要があります。

```
java.io.ByteArrayInputStream byteStream =
  new java.io.ByteArrayInputStream(byteData);
int numBytes = byteData.length;
sqlj.runtime.BinaryStream binStream =
  new sqlj.runtime.BinaryStream(byteStream, numBytes);
#sql {CALL STORPROC(:IN binStream)};
```

この技法を入出力パラメーターに使用することはできません。

BLOB 列の出力パラメーター:

BLOB 列に対する出力または入出力パラメーターでは、以下の技法を使用できます。

- 出力パラメーターまたは入出力変数を java.sql.Blob データ・タイプで宣言する。

```
java.sql.Blob blobData = null;
#sql CALL STORPROC (:OUT blobData)};
```

```
java.sql.Blob blobData = null;
#sql CALL STORPROC (:INOUT blobData)};
```

CLOB 列の入力パラメーター:

CLOB 列に対する入力パラメーターでは、以下の技法のいずれかを使用できます。

- 以下のように、CLOB 列と完全に一致する java.sql.CLOB 入力変数を使用します。

```
#sql CALL STORPROC(:IN clobData)};
```

java.sql.Clob 入力変数を使用する前に、java.sql.Clob オブジェクトを作成し、そのオブジェクトにデータを追加しておく必要があります。たとえば、DB2 Universal JDBC ドライバーを使用している場合、 DB2 固有のメソッド com.ibm.db2.jcc.t2zos.DB2LobFactory.createClob を使用して java.sql.Clob オブジェクトを作成し、そのオブジェクトに String データを追加することができます。

```
String stringVal = "Some Data";
java.sql.Clob clobData =
  com.ibm.db2.jcc.t2zos.DB2LobFactory.createClob(stringVal);
```

- 以下のいずれかのタイプのストリーム入力パラメーターを使用する。
  - sqlj.runtime.CharacterStream 入力パラメーターでは、以下のようにします。

```
java.lang.String charData;
java.io.StringReader reader = new java.io.StringReader(charData);
sqlj.runtime.CharacterStream charStream =
  new sqlj.runtime.CharacterStream (reader, charData.length);
#sql {CALL STORPROC(:IN charStream)};
```

  - Unicode UTF-16 データについては、sqlj.runtime.UnicodeStream パラメーターで、以下のようにします。

```
byte[] charDataBytes = charData.getBytes("UnicodeBigUnmarked");
java.io.ByteArrayInputStream byteStream =
  new java.io.ByteArrayInputStream(charDataBytes);
sqlj.runtime.UnicodeStream uniStream =
  new sqlj.runtime.UnicodeStream(byteStream, charDataBytes.length );
#sql {CALL STORPROC(:IN uniStream)};
```

– ASCII データについては、sqlj.runtime.AsciiStream パラメーターで、以下のようにします。

```
byte[] charDataBytes = charData.getBytes("US-ASCII");
java.io.ByteArrayInputStream byteStream =
  new java.io.ByteArrayInputStream (charDataBytes);
sqlj.runtime.AsciiStream asciiStream =
  new sqlj.runtime.AsciiStream (byteStream, charDataBytes.length);
#sql {CALL STORPROC(:IN asciiStream)};
```

これらの呼び出しでは、入力データの正確な長さを指定する必要があります。この技法を入出力パラメーターに使用することはできません。

- java.lang.String 入力パラメーターを使用する。

```
java.lang.String charData;
#sql {CALL STORPROC(:IN charData)};
```

CLOB 列の出力パラメーター:

CLOB 列に対する入出力パラメーターでは、以下の技法のいずれかを使用できます。

- 以下のように、CLOB 列と完全に一致する java.sql.Clob 出力変数を使用します。

```
java.sql.Clob clobData = null;
#sql CALL STORPROC(:OUT clobData)};
```

- java.lang.String 出力変数を使用する。

```
java.lang.String charData = null;
#sql CALL STORPROC(:OUT charData)};
```

この技法は、検索されるデータの長さが 32KB 以下であることが分かっている場合にのみ使用してください。そうでない場合、データが切り捨てられます。

DBCLOB 列に対する出力パラメーター:

ストアード・プロシージャーのための DBCLOB 出力または入出力パラメーターはサポートされていません。

関連概念:

- 384 ページの『DB2 Universal JDBC ドライバーを使用した SQLJ アプリケーションでの LOB』

関連資料:

- 403 ページの『Java、JDBC、および SQL のデータ・タイプ』

DB2 Universal JDBC ドライバーを使用した SQLJ での ROWID

DB2® UDB for z/OS® および DB2 UDB for iSeries™ では、 DB2 表内の列に対して ROWID データ・タイプをサポートしています。 ROWID は、表内の行を固有に識別する値です。

SQLJ プログラムで ROWID を使用する場合、プログラムをカスタマイズする必要があります。

DB2 Universal JDBC ドライバーでは、DB2 固有のクラス com.ibm.db2.jcc.DB2RowID を提供しており、これをイテレーターおよび CALL ステートメント・パラメーターで使用することができます。イテレーターの場合は、byte[] オブジェクト・タイプを使用して ROWID 値を検索することもできます。

図 51 に、イテレーターを使用して ROWID 列から値を選択する例を示します。

```
#sql iterator PosIter(int,String,com.ibm.db2.jcc.DB2RowId);
                                // Declare positioned iterator
                                // for retrieving ITEM_ID (INTEGER),
                                // ITEM_FORMAT (VARCHAR), and ITEM_ROWID (ROWID)
                                // values from table ROWIDTAB
{
  PosIter positrowid;           // Declare object of PosIter class
  com.ibm.db2.jcc.DB2RowId rowid = null;
  int id = 0;
  String i_fmt = null;
                                // Declare host expressions
  #sql [ctxt] positrowid =
    {SELECT ITEM_ID, ITEM_FORMAT, ITEM_ROWID FROM ROWIDTAB
       WHERE ITEM_ID=3};
                                // Assign the result table of the SELECT
                                // to iterator object positrowid
  #sql {FETCH :positrowid INTO :id, :i_fmt, :rowid};
                                // Retrieve the first row
  while (!positrowid.endFetch())
                                // Check whether the FETCH returned a row
    {System.out.println("Item ID " + id + " Item format " +
       i_fmt + " Item ROWID ");
     printBytes(rowid.getBytes());
                                // Use the DB2-only method getBytes to
                                // convert the value to bytes for printing
      #sql {FETCH :positrowid INTO :id, :i_fmt, :rowid};
                                // Retrieve the next row
    }
  positrowid.close();           // Close the iterator
}
```

図 51. ROWID 値を検索するためのイテレーターの使用例

389 ページの図 52 に、3 つの ROWID パラメーター (IN パラメーター、OUT パラメーター、および INOUT パラメーター) を含むストアード・プロシージャーを呼び出す例を示します。

```
com.ibm.db2.jcc.DB2RowId in_rowid = rowid;
com.ibm.db2.jcc.DB2RowId out_rowid = null;
com.ibm.db2.jcc.DB2RowId inout_rowid = rowid;
                                    // Declare an input, output, and
                                    // input/output ROWID parameter
...
#sql [myConnCtx] {CALL SP_ROWID(:IN in_rowid,
                              :OUT out_rowid,
                              :INOUT inout_rowid)};
                                    // Call the stored procedure
System.out.println("Parameter values from SP_ROWID call: ");
System.out.println("Output parameter value ");
printBytes(out_rowid.getBytes());
                                  // Use the DB2-only method getBytes to
                                  // convert the value to bytes for printing
System.out.println("Input/output parameter value ");
printBytes(inout_rowid.getBytes());
```

図 52. ROWID パラメーターを含むストアード・プロシージャーの呼び出しの例

関連資料:

- 403 ページの『Java、JDBC、および SQL のデータ・タイプ』

SQLJ アプリケーションにおける特殊タイプ

DB2® では、特殊タイプとは、組み込み SQL データ・タイプとして内部的に表される、ユーザー定義のデータ・タイプです。特殊タイプは、SQL ステートメント CREATE DISTINCT TYPE を実行して作成します。

SQLJ プログラムでは、実行可能文節で CREATE DISTINCT TYPE ステートメントを使用して、特殊タイプを作成することができます。また、実行可能文節で CREATE TABLE を使用して、このタイプの列を含む表を作成することもできます。該当するタイプの列からデータを検索するか、該当するタイプの列を更新するときには、特殊タイプのベースとなる組み込みタイプに対応するデータ・タイプと共に、Java™ ID を使用します。

以下の例では、INTEGER タイプをベースにした特殊タイプを作成し、該当タイプの列を含む表を作成し、行をその表に挿入し、そして表から行を検索します。

```
String empNumVar;
int shoeSizeVar;
...
#sql [myConnCtx] {CREATE DISTINCT TYPE SHOESIZE AS INTEGER WITH COMPARISONS};
                                          // Create distinct type
#sql [myConnCtx] {COMMIT}; // Commit the create
#sql [myConnCtx] {CREATE TABLE EMP_SHOE
  (EMPNO CHAR(6), EMP_SHOE_SIZE SHOESIZE)};
                                          // Create table using distinct type
#sql [myConnCtx] {COMMIT}; // Commit the create
#sql [myConnCtx] {INSERT INTO EMP_SHOE
  VALUES('000010',6)};                    // Insert a row in the table
#sql [myConnCtx] {COMMIT}; // Commit the INSERT
#sql [myConnCtx] {SELECT EMPNO, EMP_SHOE_SIZE
  INTO :empNumVar, :shoeSizeVar
  FROM EMP_SHOE};                         // Retrieve the row
System.out.println("Employee number: " + empNumVar +
  " Shoe size: " + shoeSizeVar);
```

図 53. 特殊タイプの定義と使用

関連資料:

- 「*SQL リファレンス 第 2 巻*」の『CREATE DISTINCT TYPE ステートメント』

SQLJ での SQL ステートメントの実行の制御

SQLJ ExecutionContext クラスの選択済みメソッドを使用して、 SQL ステートメントの実行を制御またはモニターすることができます。これらのメソッドについては、『選択済み sqlj.runtime クラスおよびインターフェース』で説明しています。

ExecutionContext メソッドを使用するには、以下のステップに従ってください。

1. *実行コンテキスト* を獲得する。

   実行コンテキストを獲得するには、次の 2 つの方法があります。

   - 接続コンテキストからデフォルトの実行コンテキストを獲得する。たとえば、以下のようにします。

     ```
     ExecutionContext execCtx = connCtx.getExecutionContext();
     ```

   - ExecutionContext のコンストラクターを呼び出して、新規実行コンテキストを作成する。たとえば、以下のようにします。

     ```
     ExecutionContext execCtx=new ExecutionContext();
     ```

2. 実行コンテキストを SQL ステートメントと関連付ける。

   これを行うには、SQL ステートメントを含む実行文節で、接続コンテキストの後に実行コンテキストを指定します。たとえば、以下のようにします。

   ```
   #sql [connCtx, execCtx] {DELETE FROM EMPLOYEE WHERE SALARY > 10000};
   ```

3. ExecutionContext メソッドを呼び出す。

   ExecutionContext メソッドの中には、関連した SQL ステートメントを実行する前に適用できるものと、関連した SQL ステートメントを実行した後にのみ適用できるものがあります。

たとえば、DELETE ステートメントによって削除された行数をカウントするには、以下のように DELETE ステートメントを実行した後に getUpdateCount メソッドを使用します。

```
#sql [connCtx, execCtx] {DELETE FROM EMPLOYEE WHERE SALARY > 10000};
System.out.println("Deleted " + execCtx.getUpdateCount() + " rows");
```

関連資料:

- 449 ページの『選択済み sqlj.runtime クラスおよびインターフェース』

SQLJ アプリケーションでのストアード・プロシージャーの複数の結果セットの検索

ストアード・プロシージャーの中には、1 つ以上の結果セットを呼び出し側プログラムに戻すものがあります。このような結果セットから行を検索するには、以下のステップを実行します。

1. ストアード・プロシージャーからの結果セットを検索するための実行コンテキストを獲得する。
2. 実行コンテキストをストアード・プロシージャーの CALL ステートメントと関連付ける。

   この実行コンテキストは、最後の結果セットの検索と処理を終えるまで、他の目的には使用しないでください。
3. 各結果セットに対して、以下を行う。
   a. ExecutionContext メソッド getNextResultSet を使用して、結果セットを検索する。
   b. 結果セットの内容が不明の場合は、ResultSetMetaData メソッドを使用してその情報を取り出す。
   c. SQLJ 結果セット・イテレーターまたは JDBC ResultSet を使用して、結果セットから行を検索する。

結果セットは、そのカーソルがストアード・プロシージャーでオープンしたのと同じ順序で呼び出し側プログラムに戻されます。検索対象の結果セットがそれ以上ない場合、getNextResultSet は NULL 値を戻します。

getNextResultSet には、以下の 2 つの形式があります。

```
getNextResultSet();
getNextResultSet(int current);
```

getNextResultSet の最初の形式を呼び出すと、 SQLJ は現在オープンしている結果セットをクローズして次の結果セットに進みます。 getNextResultSet の 2 番目の形式を呼び出す場合は、次の結果セットに進む前に、現在オープンしている結果セットに対して SQLJ が実行する処理を、以下の *current* の値で指定します。

java.sql.Statement.CLOSE_CURRENT_RESULT
次の ResultSet オブジェクトが戻されるときに、現行の ResultSet オブジェクトをクローズすることを指定します。

java.sql.Statement.KEEP_CURRENT_RESULT
次の ResultSet オブジェクトが戻されるときに、現行の ResultSet オブジェクトをオープンしたままにすることを指定します。

java.sql.Statement.CLOSE_ALL_RESULTS
次の ResultSet オブジェクトが戻されるときに、オープンしている ResultSet オブジェクトをすべてクローズすることを指定します。

getNextResultSet の 2 番目の形式を使用するには、JDK 1.4以降が必要です。

以下のコードは、複数の結果セットを戻すストアード・プロシージャーを呼び出します。この例では、戻される結果セットの数や、それらの結果セットの内容を、呼び出し側は分かっていないと想定しています。 また、autoCommit が false であることも想定しています。 特定のステートメントの右側にある番号は、前述のステップに対応しています。

```
ExecutionContext execCtx=myConnCtx.getExecutionContext();      1
#sql [myConnCtx, execCtx] {CALL MULTRSSP()};                   2
          // MULTRSSP returns multiple result sets
ResultSet rs;
while ((rs = execCtx.getNextResultSet()) != null)              3a
{
  ResultSetMetaData rsmeta=rs.getMetaData();                   3b
  int numcols=rsmeta.getColumnCount();
  while (rs.next())                                            3c
  {
    for (int i=1; i<=numcols; i++)
    {
      String colval=rs.getString(i);
      System.out.println("Column " + i + "value is " + colval);
    }
  }
}
```

図 54. ストアード・プロシージャーからの結果セットの検索

SQLJ アプリケーションでのバッチ更新の実行

DB2 Universal JDBC ドライバーは、SQLJ でのバッチ更新をサポートしています。バッチ更新では、DB2® 表の行を一度に 1 つずつ更新するのではなく、SQLJ によって一連の更新を同時に実行することができます。バッチ更新には、以下のタイプのステートメントを組み込むことができます。

- 検索済み INSERT、UPDATE、または DELETE ステートメント
- CREATE、ALTER、DROP、GRANT、または REVOKE ステートメント
- 入力パラメーターのみを含む CALL ステートメント

JDBC とは異なり、SQLJ では入力パラメーターまたはホスト式がある複数のステートメントを含む異種バッチが可能です。したがって、同一ステートメント、異なるステートメント、入力パラメーターまたはホスト式があるステートメント、入力パラメーターもホスト式もないステートメントのインスタンスを、SQLJ ステートメントの同一バッチとして結合することができます。

ステートメントのバッチの作成、実行、および削除の基本的なステップは、以下のとおりです。

1. 接続の AutoCommit を使用不可にする。
2. 実行コンテキストを獲得する。

バッチで実行するすべてのステートメントは、この実行コンテキストを使用する必要があります。

3. `ExecutionContext.setBatching(true)` メソッドを呼び出して、バッチを作成する。

   ステップ 2 (392 ページ) で作成した実行コンテキストに関連付ける後続のバッチ可能ステートメントは、後から実行するバッチに追加されます。

   同時にはバッチ互換性がないステートメントのセットをバッチ処理する場合は、バッチ互換性があるステートメントのセットごとに実行コンテキストを作成する必要があります。

4. バッチ処理する SQL ステートメントの SQLJ 実行可能文節を組み込む。

   これらの文節には、ステップ 2 (392 ページ) で作成した実行コンテキストを組み込む必要があります。

   SQLJ 実行可能文節に入力パラメーターまたはホスト式がある場合は、入力パラメーターまたはホスト式に異なる値を指定したステートメントを、バッチに複数回組み込むことができます。

   ステートメントが、既存のバッチに追加されたか、新規バッチの最初のステートメントか、またはバッチの内部または外部で実行されたかを判別するには、`ExecutionContext.getUpdateCount` メソッドを呼び出します。このメソッドは、以下のいずれかの値を戻します。

   `ExecutionContext.ADD_BATCH_COUNT`
   これは、ステートメントが既存のバッチに追加された場合に戻される定数です。

   `ExecutionContext.NEW_BATCH_COUNT`
   これは、ステートメントが新規バッチの最初のステートメントだった場合に戻される定数です。

   `ExecutionContext.EXEC_BATCH_COUNT`
   これは、ステートメントがバッチの一部であり、そのバッチが実行された場合に戻される定数です。

   その他の整数
   この値は、ステートメントによって更新された行の数です。この値は、ステートメントがバッチに追加されずに実行された場合に戻されます。

5. バッチを明示的または暗黙的に実行する。

   - `ExecutionContext.executeBatch` メソッドを呼び出して、バッチを明示的に実行する。

     `executeBatch` は、バッチ内の各ステートメントで更新された行数を含む整数の配列を戻します。配列内のエレメントの順序は、バッチにステートメントを追加した順序に対応しています。

   - あるいは、以下の状況ではバッチが暗黙的に実行される。
     – バッチにすでに存在するステートメントと互換性がないバッチ可能ステートメントを、プログラムに組み込んだ場合。この場合、SQLJ はバッチにすでに存在するステートメントを実行し、非互換ステートメントを組み込ん

だ新規バッチが作成されます。 SQLJ は、バッチ内のステートメントと互換性がないステートメントも実行します。

– バッチ可能ではないステートメントをプログラムに組み込んだ場合。この場合は、SQLJ はバッチにすでに存在するステートメントを実行します。SQLJ は、バッチ可能ではないステートメントも実行します。

– `ExecutionContext.setBatchLimit(n)` メソッドを呼び出した後にステートメントをバッチに追加し、そのステートメントのためにバッチ内のステートメントの数が *n* 以上になった場合。 *n* は、以下のいずれかの値にすることができます。

`ExecutionContext.UNLIMITED_BATCH`
この定数は、バッチ可能であっても非互換のステートメント、またはバッチ可能ではないステートメントが SQLJ によって検出されたときのみ、暗黙的実行が行われるようにするものです。この値を設定することは、`setBatchLimit` を呼び出さないことと同じです。

`ExecutionContext.AUTO_BATCH`
この定数は、バッチ内のステートメント数が SQLJ によって設定された数に達したときに暗黙的実行が行われるようにするものです。

正整数
この数のステートメントがバッチに追加されると、SQLJ によってバッチが暗黙的に実行されます。ただし、バッチ可能であっても非互換のステートメント、またはバッチ可能ではないステートメントが SQLJ によって検出されたときは、この数のステートメントがまだ追加されていなくてもバッチが実行されることがあります。

暗黙的に実行されたバッチによって更新された行数を判別するには、`ExecutionContext.getBatchUpdateCounts` メソッドを呼び出します。`getBatchUpdateCounts` は、バッチ内の各ステートメントで更新された行数を含む整数の配列を戻します。配列内のエレメントの順序は、バッチにステートメントを追加した順序に対応しています。各配列エレメントは、以下のいずれかの値になります。

-2 この値は、SQL ステートメントは正常に実行されたが、更新された行数を判別できなかったことを示します。

-3 この値は、SQL ステートメントが失敗したことを示します。

その他の整数
この値は、ステートメントによって更新された行の数です。

6. すべてのステートメントがバッチに追加されたら、バッチ処理を使用不可にする (オプション)。

これを行うには、`ExecutionContext.setBatching(false)` メソッドを呼び出します。バッチ処理を使用不可にしてもバッチを暗黙的または明示的に実行することができますが、バッチにそれ以上のステートメントは追加されません。バッチ処理を使用不可にすることは、バッチがすでに存在するときに、バッチ互換ステートメントをバッチに追加しないで実行したい場合に便利です。

バッチを実行しないでクリアする場合は、`ExecutionContext.cancel` メソッドを呼び出します。

7. バッチが暗黙的に実行された場合は、最後に executeBatch を明示的に実行して、すべてのステートメントが実行済みになるようにします。

バッチ更新の例: 以下のコード・フラグメントでは、バッチ内の UPDATE を実行して、すべての管理職を昇給させます。特定のステートメントの右側にある番号は、前述のステップに対応しています。

```
#sql iterator GetMgr(String);       // Declare positioned iterator
{
  GetMgr deptiter;                  // Declare object of GetMgr class
  String mgrnum = null;             // Declare host variable for manager number
  int raise = 400®;                  // Declare raise amount
  int currentSalary;                // Declare current salary
  String url, username, password;   // Declare url, user ID, password
  ...
  TestContext c1 = new TestContext (url, username, password, false);  1
  ExecutionContext ec = new ExecutionContext();                       2
  ec.setBatching(true);                                               3

  #sql [c1] deptiter =
    {SELECT MGRNO FROM DEPARTMENT};
                                    // Assign the result table of the SELECT
                                    // to iterator object deptiter
  #sql {FETCH :deptiter INTO :mgrnum};
                                    // Retrieve the first manager number
  while (!deptiter.endFetch()) {    // Check whether the FETCH returned a row
    #sql [c1]
      {SELECT SALARY INTO :currentSalary FROM EMPLOYEE
        WHERE EMPNO=:mgrnum};
    #sql [c1, ec]                                                     4
      {UPDATE EMPLOYEE SET SALARY=:(currentSalary+raise)
        WHERE EMPNO=:mgrnum};
    #sql {FETCH :deptiter INTO :mgrnum };
                                    // Fetch the next row
  }
  ec.executeBatch();                                                  5
  ec.setBatching(false);                                              6
  #sql [c1] {COMMIT};
  deptiter.close();                 // Close the iterator
  ec.close();                       // Close the execution context
  c1.close();                       // Close the connection
}
```

図 55. バッチ更新の実行

バッチ内のステートメントを実行中にエラーが発生しても残りのステートメントは実行され、バッチ内のすべてのステートメントの実行後に BatchUpdateException が出されます。 BatchUpdateException の処理方法については、 『JDBC アプリケーションでのバッチ更新の作成』を参照してください。

警告に関する情報を取得するには、executeBatch メソッドを実行したオブジェクトで、 Statement.getWarnings メソッドを使用します。それから、各 SQLWarning オブジェクトのエラー記述、SQLSTATE、エラー・コードを検索できます。

バッチに追加できないステートメントがプログラムに含まれているためにバッチが暗黙的に実行される場合、バッチが実行されてから新規ステートメントが処理されます。バッチの実行中にエラーが発生した場合、そのバッチの実行の要因となったステートメントは実行されません。

推奨: バッチ実行中にエラーが発生したときに実行済みステートメントの変更をコミットするかどうかを制御できるように、バッチ更新の実行時は自動コミットをオフにしてください。

関連タスク:

- 337 ページの『JDBC アプリケーションでのバッチ更新の作成』
- 357 ページの『SQLJ を使用したデータ・ソースへの接続』
- 390 ページの『SQLJ での SQL ステートメントの実行の制御』

関連資料:

- 449 ページの『選択済み sqlj.runtime クラスおよびインターフェース』

SQLJ アプリケーションでの、位置指定 UPDATE または DELETE 操作の変数としてのイテレーターの受け渡し

SQLJ は、メソッド間で iterators を変数として渡すことを許可します。位置指定 UPDATE または DELETE 用に使用されるイテレーターは、実行時にのみ識別されます。同じ SQLJ の位置指定 UPDATE または DELETE ステートメントは、実行時に異なるイテレーターで使用できます。 SQLJ アプリケーションをカスタマイズするときに -staticpositioned に値 YES を指定した場合、 SQLJ カスタマイザーは、位置指定 UPDATE または DELETE ステートメントが静的に実行するように準備します。 この場合、カスタマイザーはどのイテレーターがどの位置指定 UPDATE または DELETE ステートメントに属するかを判別する必要があります。 SQLJ カスタマイザーは、イテレーター・データ・タイプを UPDATE または DELETE ステートメント内のデータ・タイプと突き合わせることによってこれを実行します。ただし、UPDATE または DELETE ステートメント内の表からイテレーター・クラスへのユニークなマッピングが存在しない場合、 SQLJ カスタマイザーはどのイテレーターと UPDATE または DELETE ステートメントとが組になるのかを判別できません。 SQLJ カスタマイザーは、イテレーターと UPDATE または DELETE ステートメントとの任意の対を作成しなければならず、その結果として SQL エラーが生じることもあります。以下のコード・フラグメントは、この点を示しています。

```
#sql iterator GeneralIter ( String );

  public static void main ( String args[] )
  {
...
    GeneralIter iter1 = null;
    #sql [ctxt] iter1 = { SELECT CHAR_COL1 FROM TABLE1 };

    GeneralIter iter2 = null;
    #sql [ctxt] iter2 = { SELECT CHAR_COL2 FROM TABLE2 };
...

    doUpdate ( iter1 );
  }

  public static void doUpdate ( GeneralIter iter )
  {
    #sql [ctxt] { UPDATE TABLE1 ... WHERE CURRENT OF :iter };
  }
```

図 56. 正常に実行される静的な位置指定 UPDATE

この例では、1 つのイテレーターだけが定義されています。そのイテレーターの 2 つのインスタンスが定義され、それぞれが、異なる表からデータを検索する、異なる SELECT ステートメントに関連付けられます。イテレーターはメソッド doUpdate に変数として渡されるため、位置指定 UPDATE にどのイテレーター・インスタンスが使用されるかは、ランタイムまで分かりません。 DB2® バインド処理は、DB2 プランをバインドする際に、最初のイテレーター・インスタンス iter1 を使用します。ランタイムには、図 56 に示すように、iter1 がdoUpdate メソッドに渡された場合、 iter1 および UPDATE ステートメントの両方が TABLE1を使用するので、UPDATE は正常に実行されます。

このプログラムが図 57 に示すようにいくらか異なる方法で作成された場合、プログラムが有効に見えても、DB2 バインドは失敗します。

```
#sql iterator GeneralIter ( String );

  public static void main ( String args[] )
  {
...
    GeneralIter iter2 = null;
    #sql [ctxt] iter2 = { SELECT CHAR_COL2 FROM TABLE2 };
    GeneralIter iter1 = null;
    #sql [ctxt] iter1 = { SELECT CHAR_COL1 FROM TABLE1 };

...

    doUpdate ( iter1 );
  }

  public static void doUpdate ( GeneralIter iter )
  {
    #sql [ctxt] { UPDATE TABLE1 ... WHERE CURRENT OF :iter };
  }
```

図 57. バインド時に失敗する、静的な位置指定 UPDATE

この場合、iter2 がプログラム内で最初に現れるので、 DB2 バインド処理は iter2 を位置指定 UPDATE と関連付けます。 DB2 がプランをプログラムにバイ

ンドするとき、 iter2 は TABLE2 を使用し、 UPDATE は TABLE1 を使用するので、バインドは SQLCODE -509 を出して失敗します。ただし、プログラムが正常にバインドすることを許可され、 iter1 を doUpdate メソッドに受け渡すと、プログラムは正常に実行します。

397 ページの図 57 のようなプログラムで、バインド時のエラーを避けるには、DB2 BIND オプション SQLERROR(CONTINUE) を使用します。ただし、この手法には、プログラム内の SQL エラーには関係なく DB2 がパッケージを作成するという欠点もあります。より優れた手法は、位置指定 UPDATE または DELETE とイテレーター・クラスとの間に 1 対 1 のマッピングが存在するようにプログラムを記述することです。 図 58 は、これを行う方法を示しています。

```
#sql iterator Table2Iter(String);
#sql iterator Table1Iter(String);
  public static void main ( String args[] )
  {
...
    Table2Iter iter2 = null;
    #sql [ctxt] iter2 = { SELECT CHAR_COL2 FROM TABLE2 };

    Table1Iter iter1 = null;
    #sql [ctxt] iter1 = { SELECT CHAR_COL1 FROM TABLE1 };
...

    doUpdate(iter1);

  }

 public static void doUpdate ( Table1Iter iter )
  {
    ...
    #sql [ctxt] { UPDATE TABLE1 ... WHERE CURRENT OF :iter };
    ...
  }
 public static void doUpdate ( Table2Iter iter )
  {
    ...
    #sql [ctxt] { UPDATE TABLE2 ... WHERE CURRENT OF :iter };
    ...
  }
```

図 58. イテレーターの順序に関係なく正常に実行される、静的な位置指定 UPDATE

このコーディングの方法では、各イテレーター・クラスは 1 つの表にのみ関連付けられています。そのため、DB2 バインド処理は常に位置指定 UPDATE ステートメントを、有効なイテレーターと関連付けます。

関連タスク:

- 373 ページの『SQLJ アプリケーションでの位置指定 UPDATE および DELETE 操作の実行』

関連資料:

- 「コマンド・リファレンス」の『db2sqljcustomize - DB2 SQLJ プロファイル・カスタマイザー・コマンド』

SQLJ アプリケーションでのスクロール可能イテレーターの使用

結果表を順方向に 1 行ずつ移動することに加えて、逆方向に移動したり特定の行に直接移動したりすることもできます。 DB2 Universal JDBC ドライバーでは、これを実行することができます。

順方向、逆方向、あるいは特定の行に移動できるイテレーターを、*スクロール可能イテレーター* と呼びます。 SQLJ でのスクロール可能イテレーターは、 SCROLL と宣言された DB2® カーソルの結果表に相当します。

両方向スクロール・カーソルのように、スクロール可能イテレーターには、*インセンシティブ* または*センシティブ* を指定できます。センシティブ・スクロール可能イテレーターには、*静的* または*動的* を指定することができます。インセンシティブの場合、イテレーターをオープンした後の基礎表の変更はイテレーターからは見えません。インセンシティブ・イテレーターは読み取り専用です。センシティブの場合、イテレーターまたは他のプロセスが行う基礎表の変更は、イテレーターから見ることができます。

スクロール可能イテレーターが静的の場合、イテレーターをオープンした後も、結果表のサイズや結果表の行の順序は変わりません。つまり、結果表に挿入を行うことはできません。そのため、結果表の行を削除すると削除ホールが発生します。結果表の行を更新したためにその行が結果表から外れると、更新ホールが発生します。ホールから取り出すと `SQLException` になります。

スクロール可能イテレーターが動的の場合、イテレーターをオープンした後も、結果表のサイズや結果表の行の順序を変えることができます。アプリケーションの同一プロセスで実行された INSERT および DELETE ステートメントで挿入または削除された行は、即時に可視になります。アプリケーションの他のプロセスで実行された INSERT および DELETE ステートメントで挿入または削除された行は、その変更がコミットされた後に可視になります。

スクロール可能イテレーターを作成して使用するには、以下のステップに従う必要があります。

1. 以下の文節を含むイテレーター宣言文節を指定する。
   - `implements sqlj.runtime.Scrollable`

     これで、イテレーターがスクロール可能であることを示します。
   - `with (sensitivity=INSENSITIVE|SENSITIVE)` または `with (sensitivity=SENSITIVE, dynamic=true|false)`

     `sensitivity=INSENSITIVE|SENSITIVE` は、基礎表での更新または削除操作をイテレーターに可視にするかどうかを示します。 sensitivity のデフォルトは `INSENSITIVE` です。

     `dynamic=true|false` は、イテレーターをオープンした後に結果表のサイズまたは結果表の行の順序を変更できるかどうかを示します。 dynamic のデフォルト値は `false` です。

イテレーターは名前指定または位置指定イテレーターにすることができます。たとえば、以下のイテレーター宣言文節は、位置指定、センシティブ、動的、スクロール可能イテレーターを宣言します。

```
#sql public iterator ByPos
  implements sqlj.runtime.Scrollable
  with (sensitivity=SENSITIVE, dynamic=true)  (String);
```

たとえば、以下のイテレーター宣言文節は、名前指定、インセンシティブ、スクロール可能イテレーターを宣言します。

```
#sql public iterator ByName
  implements sqlj.runtime.Scrollable
  with (sensitivity=INSENSITIVE) (String EmpNo);
```

制約事項: スクロール可能イテレーターを使用する場合、 DB2 UDB for Linux, UNIX, Windows サーバー上の表からは以下のデータ・タイプの列を選択することができません。

- LONG VARCHAR
- LONG VARGRAPHIC
- DATALINK
- BLOB
- CLOB
- このリストにあるデータ・タイプのいずれかを基にした特殊タイプ。
- 構造化タイプ。

2. イテレーター・オブジェクトを作成する。これはこのイテレーター・クラスのインスタンスです。
3. SQLJ ランタイム環境に対して初期フェッチ方向を設定する場合は、`setFetchDirection(int direction)` メソッドを使用する。 *direction* は `FETCH_FORWARD` または `FETCH_REVERSE` にします。 `setFetchDirection` を呼び出さない場合、フェッチ方向は `FETCH_FORWARD` になります。
4. アクセスしたい行ごとに、以下を行う。
   - 名前指定イテレーターの場合は、以下のステップを実行します。
     a. 表 38 にリストするいずれかのメソッドを使用して、カーソルの位置を指定する。

表 38. 両方向スクロール・カーソルの位置指定のための sqlj.runtime.Scrollable メソッド

メソッド	カーソルの位置
`first()`	結果表の最初の行。
`last()`	結果表の最後の行。
`previous()`[1]	結果表の直前の行。
`next()`	結果表の次の行。
`absolute(int n)`[2]	*n*>0 の場合、結果表の行 *n*。 *n*<0 の場合、 *m* を結果表の行数として、結果表の行 *m+n+1*。
`relative(int n)`[3]	*n*>0 の場合、現在行から *n* 行後の行。 *n*<0 の場合、現在行から *n* 行前の行。 *n*=0 の場合、現在行。
`afterLast()`	結果表の最後の行の後。
`beforeFirst()`	結果表の最初の行の前。

表 38. 両方向スクロール・カーソルの位置指定のための sqlj.runtime.Scrollable メソッド (続き)

メソッド	カーソルの位置

注:

1. カーソルが結果表の最後の行の後にある場合、このメソッドによるカーソルの位置は最後の行になります。
2. *n* の絶対値が結果表の行数よりも大きい場合、このメソッドによるカーソルの位置は、 *n* が正のときは最後の行の後に、*n* が負のときは最初の行の前になります。
3. *m* を結果表の行数、*x* を結果表での現在行番号とします。 *n*>0 で、*x*+*n*>*m* の場合、イテレーターは最後の行の後に置かれます。 *n*<0 で *x*+*n*<1 の場合、イテレーターは最初の行の前に置かれます。

   b. 現行カーソル位置を知る必要があるときは、 `getRow`、`isFirst`、`isLast`、`isBeforeFirst`、または `isAfterLast` メソッドを使用してその情報を取得する。現行フェッチ方向を知る必要がある場合は、`getFetchDirection` メソッドを呼び出します。

   c. アクセサー・メソッドを使用して、結果表の現在行を検索する。

   d. イテレーターまたは他の手段による更新または削除操作が結果表で分かる場合は、 `getWarnings` メソッドを呼び出して、現在行がホールかどうかを検査する。

- 位置指定イテレーターの場合は、以下のステップを実行します。

  a. フェッチ・オリエンテーション文節を含む FETCH ステートメントを使用してイテレーターの位置を指定し、結果表の現在行を検索する。表 39 に、カーソルの位置指定に使用できる文節を示します。

表 39. 両方向スクロール・カーソルの位置指定のための FETCH 文節

メソッド	カーソルの位置
FIRST	結果表の最初の行。
LAST	結果表の最後の行。
PRIOR[1]	結果表の直前の行。
NEXT	結果表の次の行。
ABSOLUTE(*n*)[2]	*n*>0 の場合、結果表の行 *n*。 *n*<0 の場合、 *m* を結果表の行数として、結果表の行 *m*+*n*+*1*。
RELATIVE(*n*)[3]	*n*>0 の場合、現在行から *n* 行後の行。 *n*<0 の場合、現在行から *n* 行前の行。 *n*=0 の場合、現在行。
AFTER[4]	結果表の最後の行の後。
BEFORE[4]	結果表の最初の行の前。

表 39. 両方向スクロール・カーソルの位置指定のための FETCH 文節 (続き)

メソッド	カーソルの位置

注:

1. カーソルが結果表の最後の行の後にある場合、このメソッドによるカーソルの位置は最後の行になります。
2. *n* の絶対値が結果表の行数よりも大きい場合、このメソッドによるカーソルの位置は、 *n* が正のときは最後の行の後に、*n* が負のときは最初の行の前になります。
3. *m* を結果表の行数、*x* を結果表での現在行番号とします。 *n*>0 で、*x*+*n*>*m* の場合、イテレーターは最後の行の後に置かれます。 *n*<0 で *x*+*n*<1 の場合、イテレーターは最初の行の前に置かれます。
4. ホスト式に値は割り当てられません。

   b. イテレーターまたは他の手段による更新または削除操作が結果表で分かる場合は、 `getWarnings` メソッドを呼び出して、現在行がホールかどうかを検査する。

5. `close` メソッドを呼び出して、イテレーターをクローズする。

たとえば、以下のコードは、名前指定イテレーターを使用して、従業員表のすべての行の従業員番号と姓を逆順に検索する方法を示しています。特定のステートメントの右側にある番号は、前述のステップに対応しています。

```
#sql  iterator ScrollIter implements sqlj.runtime.Scrollable          1
  (String EmpNo, String LastName);
{
  ScrollIter scrliter;                                                2
  #sql [ctxt]
  scrliter={SELECT EMPNO, LASTNAME FROM EMPLOYEE};
  scrliter.afterLast();
  while (scrliter.previous())                                         4a
  {
    System.out.println(scrliter.EmpNo() + " "                         4c
      + scrliter.LastName());
  }
  scrliter.close();                                                   5
}
```

図 59. スクロール可能イテレーターの使用

関連概念:

- 366 ページの『SQLJ アプリケーションによる DB2 表からのデータの検索方法』

関連タスク:

- 367 ページの『SQLJ アプリケーションでの名前指定イテレーターの使用』
- 370 ページの『SQLJ アプリケーションでの位置指定イテレーターの使用』

第 17 章 JDBC および SQLJ リファレンス

以下のセクションには、 JDBC メソッドおよび SQLJ 文節についての参照情報が含まれています。

Java、JDBC、および SQL のデータ・タイプ

以下の表は、Java データ・タイプから DB2 UDB for Linux、UNIX、Windows システムでの JDBC および SQL へのマッピングを要約しています。

表 40 は、 JDBC プログラムの `PreparedStatement.setXXX` または `ResultSet.updateXXX` メソッド、および SQLJ プログラムでの入力ホスト式での、Java データ・タイプから DB2 データ・タイプへのマッピングを要約しています。複数の Java データ・タイプがリストされている場合、最初のデータ・タイプが推奨されるデータ・タイプです。

表 40. DB2 表を更新するための、Java データ・タイプから DB2 データ・タイプへのマッピング

Java データ・タイプ	SQL データ・タイプ
short、boolean[1]、byte[1]	SMALLINT
int、java.lang.Integer	INTEGER
long、java.lang.Long	DECIMAL(19,0)[2]
long、java.lang.Long	BIGINT[3]
float、java.lang.Float	REAL
double、java.lang.Double	DOUBLE
java.math.BigDecimal	DECIMAL(*p*,*s*)[4]
java.lang.String	CHAR(*n*)[5]
java.lang.String	GRAPHIC(*m*)[6]
java.lang.String	VARCHAR(*n*)[7]
java.lang.String	VARGRAPHIC(*m*)[8]
java.lang.String	CLOB(*n*)[9]
byte[]	CHAR(*n*) FOR BIT DATA[5]
byte[]	VARCHAR(*n*) FOR BIT DATA[7]
byte[]	BLOB(*n*)[9,10]
byte[]	ROWID
java.sql.Blob	BLOB(*n*)[10]
java.sql.Clob	CLOB(*n*)[10]
java.sql.Clob	DBCLOB(*m*)[11]
java.sql.Date	DATE
java.sql.Time	TIME
java.sql.Timestamp	TIMESTAMP
java.io.ByteArrayInputStream	BLOB(*n*)[10]
java.io.StringReader	CLOB(*n*)[10]

表 40. DB2 表を更新するための、Java データ・タイプから DB2 データ・タイプへのマッピング (続き)

Java データ・タイプ	SQL データ・タイプ
java.io.ByteArrayInputStream	CLOB(*n*)[10]
com.ibm.db2.jcc.DB2RowID	ROWID
java.net.URL	DATALINK[12]

注:

1. DB2 には、 Java の boolean または byte データ・タイプに正確に対応するものがありませんが、最適なものは SMALLINT です。
2. OS/390 または z/OS 環境の DB2 UDB には、 Java の long または java.lang.Long データ・タイプに正確に対応するものがありませんが、最適なものは DECIMAL(19,0) です。
3. BIGINT SQL タイプは、 DB2 UDB for Linux、UNIX、Windows でのみ使用可能です。
4. *p* は小数部の精度であり、 *s* は DB2 列の位取りです。

   金融アプリケーションは、java.math.BigDecimal 列が DECIMAL 列にマップするように設計してください。DECIMAL 列の精度および位取りを知っている場合、 java.math.BigDecimal 変数のデータを使用して DECIMAL 列のデータを更新すると、他のデータ・タイプの組み合わせを使用する場合よりも、精度とパフォーマンスが向上します。
5. *n*<=254。
6. *m*<=127。
7. *n*<=32672。
8. *m*<=16336。
9. このマッピングが有効なのは、DB2 が列のデータ・タイプを判別できる場合だけです。
10. *n*<=2147483647。
11. *m*<=1073741823。
12. DATALINK データ・タイプをサポートするのは、 Linux、UNIX、および Windows 用 DB2 JDBC Type 2 ドライバーだけです。

表 41 は、 JDBC プログラムの `ResultSet.getXXX` メソッド、および SQLJ プログラムのイテレーターでの、 DB2 データ・タイプから Java データ・タイプへのマッピングを要約しています。この表には、 `ResultSet.getObject` を使用して検索される Java 数値ラッパーのオブジェクト・タイプはリストされていません。

表 41. DB2 表からデータを検索するための、DB2 データ・タイプから Java データ・タイプへのマッピング

SQL データ・タイプ	推奨される Java データ・タイプまたは Java オブジェクト・タイプ	サポートされる他の Java データ・タイプ
SMALLINT	short	byte、int、 long、 float、 double、 java.math.BigDecimal、 boolean、 java.lang.String
INTEGER	int	short、 byte、 long、 float、 double、 java.math.BigDecimal、 boolean、 java.lang.String
BIGINT[1]	long	int、 short、 byte、 float、 double、 java.math.BigDecimal、 boolean、 java.lang.String

表 41. DB2 表からデータを検索するための、DB2 データ・タイプから Java データ・タイプへのマッピング (続き)

SQL データ・タイプ	推奨される Java データ・タイプまたは Java オブジェクト・タイプ	サポートされる他の Java データ・タイプ
REAL	float	long、 int、 short、 byte、 double、 java.math.BigDecimal、 boolean、 java.lang.String
DOUBLE	double	long、 int、 short、 byte、 float、 java.math.BigDecimal、 boolean、 java.lang.String
CHAR(*n*)	java.lang.String	long、 int、 short、 byte、 float、 double、 java.math.BigDecimal、 boolean、 java.sql.Date、 java.sql.Time、 java.sql.Timestamp、 java.io.InputStream、 java.io.Reader
VARCHAR(*n*)	java.lang.String	long、 int、 short、 byte、 float、 double、 java.math.BigDecimal、 boolean、 java.sql.Date、 java.sql.Time、 java.sql.Timestamp、 java.io.InputStream、 java.io.Reader
CHAR(*n*) FOR BIT DATA	byte[]	java.lang.String、 java.io.InputStream、 java.io.Reader
VARCHAR(*n*) FOR BIT DATA	byte[]	java.lang.String、 java.io.InputStream、 java.io.Reader
GRAPHIC(*m*)	java.lang.String	long、 int、 short、 byte、 float、 double、 java.math.BigDecimal、 boolean、 java.sql.Date、 java.sql.Time、 java.sql.Timestamp、 java.io.InputStream、 java.io.Reader
VARGRAPHIC(*m*)	java.lang.String	long、 int、 short、 byte、 float、 double、 java.math.BigDecimal、 boolean、 java.sql.Date、 java.sql.Time、 java.sql.Timestamp、 java.io.InputStream、 java.io.Reader
CLOB(*n*)	java.sql.Clob	java.lang.String
BLOB(*n*)	java.sql.Blob	byte[][3]
DBCLOB(*m*)	厳密な対応なし。 java.sql.Clob を使用。	
ROWID	com.ibm.db2.jcc.DB2RowID	byte[]
DATE	java.sql.Date	java.sql.String、java.sql.Timestamp
TIME	java.sql.Time	java.sql.String、java.sql.Timestamp
TIMESTAMP	java.sql.Timestamp	java.sql.String、 java.sql.Date、 java.sql.Time、 java.sql.Timestamp
DATALINK	java.net.URL[4]	

表 41. DB2 表からデータを検索するための、DB2 データ・タイプから Java データ・タイプへのマッピング (続き)

SQL データ・タイプ	推奨される Java データ・タイプまたは Java オブジェクト・タイプ	サポートされる他の Java データ・タイプ

注:

1. BIGINT SQL タイプは、 DB2 UDB for Linux、UNIX、Windows でのみ使用可能です。
2. 金融アプリケーションは、 DECIMAL 列が java.math.BigDecimal 列にマップするように設計してください。 DECIMAL 列の精度および位取りを知っている場合、その列からデータを検索して java.math.BigDecimal 変数に入れると、他のデータ・タイプの組み合わせを使用する場合よりも、精度とパフォーマンスは向上します。
3. このマッピングが有効なのは、DB2 が列のデータ・タイプを判別できる場合だけです。
4. DATALINK データ・タイプをサポートするのは、 Linux、UNIX、および Windows 用 DB2 JDBC Type 2 ドライバーだけです。

表 42 は、ユーザー定義関数およびストアード・プロシージャー・パラメーターのための、 Java データ・タイプから JDBC データ・タイプおよび DB2 データ・タイプへのマッピングを要約しています。 Java データ・タイプから JDBC データ・タイプへのマッピングは、 JDBC プログラムの `CallableStatement.registerOutParameter` メソッドのためのものです。 Java データ・タイプから DB2 データ・タイプへのマッピングは、ストアード・プロシージャーのパラメーターまたはユーザー定義関数の呼び出しのためのものです。

複数の Java データ・タイプが表 42 にリストされている場合、最初のデータ・タイプが**推奨される**データ・タイプです。

表 42. ストアード・プロシージャーおよびユーザー定義関数を呼び出すための、Java、JDBC、および SQL データ・タイプのマッピング

Java データ・タイプ	JDBC データ・タイプ	SQL データ・タイプ
boolean[1]	BIT	SMALLINT
byte[1]	TINYINT	SMALLINT
short、java.lang.Integer	SMALLINT	SMALLINT
int、java.lang.Integer	INTEGER	INTEGER
long	BIGINT	BIGINT[2]
float、java.lang.Float	REAL	REAL
float、java.lang.Float	FLOAT	REAL
double、java.lang.Double	DOUBLE	DOUBLE
java.math.BigDecimal	NUMERIC	DECIMAL
java.math.BigDecimal	DECIMAL	DECIMAL
java.lang.String	CHAR	CHAR
java.lang.String	CHAR	GRAPHIC
java.lang.String	VARCHAR	VARCHAR
java.lang.String	VARCHAR	VARGRAPHIC
java.lang.String	LONGVARCHAR	VARCHAR
java.lang.String	VARCHAR	CLOB(*n*)
java.lang.String	LONGVARCHAR	CLOB(*n*)
java.lang.String	CLOB	CLOB(*n*)

表 42. ストアード・プロシージャーおよびユーザー定義関数を呼び出すための、Java、JDBC、および SQL データ・タイプのマッピング (続き)

Java データ・タイプ	JDBC データ・タイプ	SQL データ・タイプ
byte[]	BINARY	CHAR FOR BIT DATA
byte[]	VARBINARY	VARCHAR FOR BIT DATA
byte[]	LONGVARBINARY	VARCHAR FOR BIT DATA
byte[]	VARBINARY	BLOB(*n*)[3]
byte[]	LONGVARBINARY	BLOB(*n*)[3]
java.sql.Date	DATE	DATE
java.sql.Time	TIME	TIME
java.sql.Timestamp	TIMESTAMP	TIMESTAMP
java.sql.Blob	BLOB	BLOB
java.sql.Clob	CLOB	CLOB
java.sql.Clob	CLOB	DBCLOB
java.io.ByteArrayInputStream	なし	BLOB(*n*)
java.io.StringReader	なし	CLOB(*n*)
java.io.ByteArrayInputStream	なし	CLOB(*n*)
com.ibm.db2.jcc.DB2RowID	com.ibm.db2.jcc.DB2Types.ROWID	ROWID
java.net.URL	DATALINK	DATALINK[4]

注:

1. SMALLINT パラメーターを使用して定義したストアード・プロシージャーまたはユーザー定義関数は、 boolean または byte パラメーターを使用して呼び出すことができます。ただし、この方法は推奨されていません。
2. BIGINT SQL タイプは、 DB2 UDB for Linux、UNIX、Windows サーバーでのみ使用可能です。 DB2 UDB バージョン 8.1 クライアントから DB2 UDB バージョン 7 サーバーに接続している Java アプリケーションでは、 BIGINT 値の検索に `CallableStatement.getObject` メソッドが使用された場合、 `java.math.BigDecimal` オブジェクトが戻されます。
3. このマッピングが有効なのは、DB2 が列のデータ・タイプを判別できる場合だけです。
4. DATALINK データ・タイプをサポートするのは、 Linux、UNIX、および Windows 用 DB2 JDBC Type 2 ドライバーだけです。

408 ページの表 43 は、 CREATE PROCEDURE または CREATE FUNCTION ステートメントの SQL パラメーター・データ・タイプから、対応する Java ストアード・プロシージャーまたはユーザー定義関数メソッドへのマッピングを要約しています。

DB2 UDB for Linux、UNIX、Windows では、 1 つの SQL データ・タイプに対して複数の Java データ・タイプがリストされている場合、**最初の** Java データ・タイプだけが有効になります。

OS/390 または z/OS 環境の DB2 UDB では、複数の Java データ・タイプがリストされている場合、最初のデータ・タイプ以外のデータ・タイプをメソッド・パラメーターとして使用するのであれば、メソッド・パラメーターの Java データ・タイプを指定する CREATE PROCEDURE または CREATE FUNCTION の EXTERNAL 文節に、メソッド・シグニチャーを組み込む必要があります。

表 43. CREATE PROCEDURE または CREATE FUNCTION ステートメントの SQL データ・タイプから、対応する Java ストアード・プロシージャーまたはユーザー定義関数プログラムへのマッピング

CREATE PROCEDURE または CREATE FUNCTION の SQL データ・タイプ	Java ストアード・プロシージャーまたはユーザー定義関数メソッドのデータ・タイプ
SMALLINT	short、java.lang.Integer
INTEGER	int、java.lang.Integer
BIGINT[1]	long
REAL	float、java.lang.Float
DOUBLE	double、java.lang.Double
DECIMAL	java.math.BigDecimal
CHAR	java.lang.String
GRAPHIC	java.lang.String
VARCHAR	java.lang.String
VARGRAPHIC	java.lang.String
CHAR FOR BIT DATA	byte[]
VARCHAR FOR BIT DATA	byte[]
DATE	java.sql.Date
TIME	java.sql.Time
TIMESTAMP	java.sql.Timestamp
BLOB	java.sql.Blob
CLOB	java.sql.Clob
DBCLOB	java.sql.Clob
ROWID	com.ibm.db2.jcc.DB2Types.ROWID
DATALINK	java.net.URL[2]

注:

1. BIGINT SQL タイプは、 DB2 UDB for Linux、UNIX、Windows サーバーでのみ使用可能です。
2. DATALINK データ・タイプをサポートするのは、 Linux、UNIX、および Windows 用 DB2 JDBC Type 2 ドライバーだけです。

関連資料:

- 470 ページの『DB2 Universal JDBC ドライバーと他の DB2 JDBC ドライバーの JDBC との相違点』

DB2 Universal JDBC ドライバーのプロパティー

特定のデータ・ソースへの接続をどのように作成するかを定義するプロパティー。特に説明のない限り、プロパティーを DataSource オブジェクトまたは Connection オブジェクトに設定できます。プロパティーは以下の方法のいずれかで設定できます。

- setXXX メソッドの使用

  プロパティーは、 com.ibm.db2.jcc.DB2BaseDataSource から継承される、以下の DB2 固有のインプリメンテーションに適用可能です。
  - com.ibm.db2.jcc.DB2SimpleDataSource

- com.ibm.db2.jcc.DB2DataSource
- com.ibm.db2.jcc.DB2ConnectionPoolDataSource
- com.ibm.db2.jcc.DB2XADataSource

プロパティー名およびデータ・タイプのサマリーについては、『JDBC に対する DB2 Universal JDBC ドライバーの拡張のサマリー』を参照してください。

- DriverManager.getConnection 呼び出しの *info* パラメーター内の java.util.Properties 値。 『DB2 Universal JDBC ドライバーでの DriverManager インターフェースの使用によるデータ・ソースへの接続』を参照してください。
- DriverManager.getConnection 呼び出しでの *url* パラメーター内の java.lang.String 値。 『DriverManager インターフェースと DB2 Universal JDBC ドライバーを使用したデータ・ソースへの接続』を参照してください。

プロパティーは以下の通りです。

activeServerListJNDIName
代替サーバー情報の JNDI リポジトリー内の、 DB2ActiveServerList インスタンスへの JNDI 参照を識別します。 activeServerListJNDIName の値がヌルでない場合、その値で参照される DB2ActiveServerList インスタンスで指定されている代替サーバーに、接続をフェイルオーバーできます。 activeServerListJNDIName がヌルの場合、 JNDI リポジトリーからの代替サーバー情報の使用による、接続のフェイルオーバーは行われません。

clientAccountingInformation
接続の現在のクライアントに関するアカウンティング情報を指定します。この情報は、クライアントのアカウンティングを目的としています。この値は、接続中に変更することもできます。このプロパティーのデータ・タイプは String です。 DB2 UDB for Linux、for UNIX、および for Windows サーバーの場合、最大長は 255 バイトです。 Java 空ストリング ("") はこの値として有効ですが、Java null 値は無効です。

clientApplicationInformation
接続の現在のクライアントに関するアプリケーション情報を指定します。この情報は、クライアントのアカウンティングを目的としています。この値は、接続中に変更することもできます。このプロパティーのデータ・タイプは String です。 DB2 UDB for Linux、for UNIX、および for Windows サーバーの場合、最大長は 255 バイトです。 Java 空ストリング ("") はこの値として有効ですが、Java null 値は無効です。

clientUser
接続のための現行クライアントのユーザー名を指定します。この情報は、クライアントのアカウンティングを目的としています。 JDBC 接続ユーザー名とは異なり、この値は接続中に変更することもできます。 DB2 UDB for Linux、for UNIX、および for Windows サーバーの場合、最大長は 255 バイトです。

clientWorkstation
接続対象の現行クライアントのワークステーション名を指定します。この情報は、クライアントのアカウンティングを目的としています。この値は、接続中に変更することもできます。このプロパティーのデータ・タイプは String です。

DB2 UDB for Linux、for UNIX、および for Windows サーバーの場合、最大長は 255 バイトです。 Java 空ストリング ("") はこの値として有効ですが、Java `null` 値は無効です。

cliSchema
アプリケーションが `DatabaseMetaData` メソッドを呼び出す際に検索される、DB2 シャドー・カタログ表またはビューのスキーマを指定します。

currentFunctionPath
JDBC プログラム内にある SQL ステートメント内の非修飾の表データ・タイプ名および関数名を解決するために使用される、 SQL パスを指定します。このプロパティーのデータ・タイプは String です。 DB2 UDB for Linux、for UNIX、および for Windows サーバーの場合、最大長は 254 バイトです。値は、スキーマ名のコンマで区切られたリストになります。それらの名前は、通常 ID または区切り ID にできます。

currentLockTimeout
ロックをすぐに取得できない場合に、 DB2 UDB for Linux、for UNIX、および for Windows サーバーがそのロックを無期限に待機するか、またはロックに指定した秒数待機するかを指示します。このプロパティーのデータ・タイプは int です。ゼロの値は、待機しないことを意味します。 -1 の値は、無期限に待機することを意味します。正の整数は、ロックを待機する秒数を示します。

currentPackagePath
サーバー上のコレクションのコンマで区切られたリストを指定します。 DB2 サーバーは、これらのコレクションから DB2 Universal JDBC ドライバー用 DB2 パッケージを検索します。

currentPackagePath および currentPackageSet プロパティーの優先順位規則は、DB2 CURRENT PACKAGESET および CURRENT PACKAGE PATH 特殊レジスターの優先順位規則に従います。

currentPackageSet
DB2 Universal JDBC ドライバー用 DB2 パッケージを検索するための、コレクション ID を指定します。このプロパティーのデータ・タイプは String です。デフォルトは NULLID です。 currentPackageSet が設定されている場合、その値は jdbcCollection の値をオーバーライドします。

DB2binder ユーティリティーを複数回実行することにより、データベース・サーバーで DB2 Universal JDBC ドライバーの複数のインスタンスをインストールできます。 DB2binder ユーティリティーの -collection オプションを使用すると、各 DB2 Universal JDBC ドライバー・インスタンスのコレクション ID をインストーラーが指定します。接続に DB2 Universal JDBC ドライバーのインスタンスを選択するには、いずれかの DB2 Universal JDBC ドライバー・インスタンスのコレクション ID に一致する currentPackageSet 値を指定します。

currentPackagePath および currentPackageSet プロパティーの優先順位規則は、DB2 CURRENT PACKAGESET および CURRENT PACKAGE PATH 特殊レジスターの優先順位規則に従います。

currentSchema
動的に作成された SQL ステートメントで、非修飾データベース・オブジェクトを修飾するのに使用される、デフォルト・スキーマを指定します。このプロパティーのこの値は、 DB2 UDB for z/OS サーバー以外のサーバーでの CURRENT

SCHEMA 特殊レジスター内の値を設定します。 DB2 UDB for z/OS サーバーには、このプロパティーを設定しないでください。

currentSQLID

下記のことを指定します。

- 動的に作成された CREATE、GRANT、および REVOKE SQL ステートメントに対する許可検査で使用される許可 ID。
- 動的に発行された CREATE ステートメントによって作成される、表スペース、データベース、ストレージ・グループ、またはシノニムの所有者。
- 動的 SQL ステートメントで指定される、すべての表、ビュー、別名、および索引名の暗黙修飾子。

currentSQLID は、DB2 UDB for z/OS サーバーの CURRENT SQLID 特殊レジスター内の値を設定します。 currentSQLID プロパティーが設定されていない場合、デフォルトのスキーマ名は、CURRENT SQLID 特殊レジスター内の値です。

cursorSensitivity

JDBC ResultSet の java.sql.ResultSet.TYPE_SCROLL_SENSITIVE 値を、基礎 DB2 カーソルの SENSITIVE DYNAMIC 属性または SENSITIVE STATIC 属性にマップするかどうかを指定します。可能な値は TYPE_SCROLL_SENSITIVE_STATIC および TYPE_SCROLL_SENSITIVE_DYNAMIC です。デフォルトは TYPE_SCROLL_SENSITIVE_STATIC です。

このプロパティーは、センシティブ動的両方向スクロール・カーソルをサポートしていないデータベース・サーバーでは無視されます。

databaseName

データベース・サーバーの名前を指定します。この名前は、接続 URL のデータベース 部分として使用されます。名前は、Universal Type 4 Connectivity か Universal Type 2 Connectivity のどちらが使用されるかによって異なります。

Universal Type 4 Connectivity の場合:

- 接続先が DB2 for z/OS サーバーの場合、 databaseName 値は、インストール中に定義される DB2 ロケーション名です。この値の中の文字はすべて大文字でなければなりません。サーバーで以下の SQL ステートメントを実行することにより、ロケーション名を判別できます。

  ```
  SELECT CURRENT SERVER FROM SYSIBM.SYSDUMMY1;
  ```

- 接続先が DB2 UDB for Linux、for UNIX、および for Windows サーバーの場合、 databaseName 値は、インストール中に定義されるデータベース名です。
- 接続先が IBM Cloudscape サーバーの場合、 databaseName 値は、データベースを含むファイルの完全修飾名です。この名前は、二重引用符 (") で囲まなければなりません。たとえば、次のようになります。

  ```
  "c:/databases/testdb"
  ```

このプロパティーが設定されていない場合、ローカル・サイトへの接続が行われます。

Universal Type 2 Connectivity の場合:

- serverName 接続プロパティーの値がヌルの場合、 databaseName 値は、インストール中に定義されるデータベース名になります。 serverName プロパティーの値がヌルではない場合、databaseName 値はデータベース別名になります。

deferPrepares

ランタイムまで準備操作を据え置くかどうかを指定します。このプロパティーのデータ・タイプ は boolean です。 Universal Type 4 Connectivity でのデフォルトは `true` です。このプロパティーは Universal Type 2 Connectivity には適用されません。

準備操作を据え置くと、ネットワークの遅延を減らすことができます。ただし、準備操作を据え置く場合、その入力データ・タイプが DB2 表列のタイプと一致しているかどうか確認する必要があります。

description

データ・ソースの説明。このプロパティーのデータ・タイプは String です。

driverType

`DataSource` インターフェースでは、接続に使用するドライバーを判別します。このプロパティーのデータ・タイプは int です。有効な値は 2 または 4 です。 2 がデフォルトです。

fullyMaterializeLobData

FETCH 操作のときにドライバーが LOB ロケーターを検索するかどうかを示します。このプロパティーのデータ・タイプ は boolean です。値が `true` の場合、行が取り出されると LOB データは JDBC ドライバー内で完全にマテリアライズされます。この値が `false` の場合、LOB データはストリームされます。ドライバーは内部でロケーターを使用し、必要に応じて LOB データをチャンクとして検索します。大量のデータが含まれている LOB を検索する場合、この値を `false` に設定するように強くお勧めします。デフォルトは `true` です。

このプロパティーは、ストアード・プロシージャーのパラメーターや、両方向スクロール・カーソルを使用して取り出される LOB には影響しません。 LOB ストアード・プロシージャー・パラメーターは常に完全にマテリアライズされます。 LOB ロケーターは、両方向スクロール・カーソルを使用して取り出されるデータには常に使用されます。

gssCredential

Kerberos セキュリティーを使用するデータ・ソースの場合に、他のプリンシパルから渡される委任証明書を指定します。このプロパティーのデータ・タイプは org.ietf.jgss.GSSCredential です。委任証明書は、たとえばクライアントが WebSphere に接続し、次いでそれが DB2 に接続するなどの、複数層での環境で使用されます。 `GSSContext.getDelegCred` メソッドを呼び出すことにより、このプロパティーの値をクライアントから取得します。 `GSSContext` は IBM Java Generic Security Service (GSS) API の一部です。このプロパティーを設定する場合、 Mechanism および KerberosServerPrincipal プロパティーも設定しなければなりません。

このプロパティーを適用できるのは Universal Type 4 Connectivity だけです。

DB2 Universal JDBC ドライバーでの Kerberos セキュリティーの使用について詳細は、 『DB2 Universal JDBC ドライバー使用時の Kerberos セキュリティーの使用』を参照してください。

jdbcCollection
ランタイムに DB2 Universal JDBC ドライバーのインスタンスによって使用される、パッケージのコレクション ID を指定します。 jdbcCollection のデータ・タイプは String です。デフォルトは NULLID です。

このプロパティーは、DB2Binder -collection オプションとともに使用されます。DB2Binder ユーティリティーは、jdbcCollection 値と一致する -collection 値を使用して、サーバーで前もって DB2 Universal JDBC ドライバー・パッケージをバインドしていなければなりません。

jdbcCollection 設定は、SQLJ アプリケーションで使用されるコレクションの判別は行いません。 SQLJ に関しては、SQLJ カスタマイザーの -collection オプションでコレクションが判別されます。

jdbcCollection は DB2 UDB for z/OS での Universal Type 2 Connectivity には適用されません。

kerberosServerPrincipal
Kerberos セキュリティーを使用するデータ・ソースの場合、 Kerberos Key Distribution Center (KDC) への登録時に、データ・ソースに使用される名前を指定します。このプロパティーのデータ・タイプは String です。

このプロパティーを適用できるのは Universal Type 4 Connectivity だけです。

loginTimeout
データ・ソースへの接続またはそのデータ・ソースへの SQL 要求の最大待ち時間 (秒)。 loginTimeout で指定された秒数が経過すると、ドライバーはデータ・ソースへの接続をクローズします。このプロパティーのデータ・タイプは int です。デフォルトは 0 です。値 0 は、タイムアウト値がデフォルトのシステム・タイムアウト値であることを意味します。 z/OS または OS/390 環境の DB2 UDB での Universal Type 2 Connectivity の場合、このプロパティーはサポートされていません。

logWriter
`DataSource` オブジェクトのすべてのロギングおよびトレース・メッセージが出力される文字出力ストリーム。このプロパティーのデータ・タイプは `java.io.PrinterWriter` です。デフォルト値はヌルです。この場合、`DataSource` のロギングもトレースも出力されません。

password
接続の確立に使用するパスワード。このプロパティーのデータ・タイプは String です。 `DataSource` インターフェースを使用して接続を確立するときは、`DataSource.getConnection` メソッドを次の形式で呼び出すと、このプロパティー値をオーバーライドすることができます。

```
getConnection(user, password);
```

portNumber
DRDA® サーバーが要求を listen するポート番号。このプロパティーのデータ・タイプは int です。

readOnly
接続を読み取り専用にするかどうかを指定します。このプロパティーのデータ・タイプ は boolean です。デフォルトは `false` です。

resultSetHoldability
コミット操作後もカーソルをオープンしたままにするかどうかを指定します。このプロパティーのデータ・タイプは int です。有効な値は com.ibm.db2.jcc.DB2BaseDataSource.HOLD_CURSORS_OVER_COMMIT または com.ibm.db2.jcc.DB2BaseDataSource.CLOSE_CURSORS_AT_COMMIT です。これらの値は、JDBC 3.0 で定義される ResultSet.HOLD_CURSORS_OVER_COMMIT および ResultSet.CLOSE_CURSORS_AT_COMMIT 定数と同じです。

retrieveMessagesFromServerOnGetMessage
JDBC SQLException.getMessage 呼び出しによって、メッセージ・テキストを検索してエラーを探す DB2 UDB for OS/390 or z/OS ストアード・プロシージャーを、 DB2 Universal JDBC ドライバーが呼び出すようにするかどうかを指定します。このプロパティーのデータ・タイプは boolean です。デフォルトは false です。これは、メッセージ・テキスト全体がクライアントに戻されるわけではないことを意味します。このプロパティーを true に設定する代わりに、アプリケーションで DB2 限定の DB2Sqlca.getMessage メソッドを使用することもできます。いずれの技法でも、ストアード・プロシージャーが呼び出されて作業単位が開始されます。

securityMechanism
DRDA セキュリティー・メカニズムを指定します。このプロパティーのデータ・タイプは int です。可能な値は以下のとおりです。

CLEAR_TEXT_PASSWORD_SECURITY
ユーザー ID およびパスワード

USER_ONLY_SECURITY
ユーザー ID のみ

ENCRYPTED_PASSWORD_SECURITY
ユーザー ID、暗号化されたパスワード

ENCRYPTED_USER_AND_PASSWORD_SECURITY
暗号化されたユーザー ID およびパスワード

KERBEROS_SECURITY
Kerberos

このプロパティーが指定されている場合、指定されているセキュリティー・メカニズムが使用される唯一のメカニズムです。セキュリティー・メカニズムが接続でサポートされていない場合、例外が出されます。

このプロパティーに値が指定されていない場合、リクエスターは使用可能な最も安全なセキュア・メカニズムを使用して接続を試行します。サーバーがセキュリティー・メカニズムをサポートしていないために、接続を確立できない場合、サーバーは代替選択肢のリストをリクエスターに戻します。リクエスターはそれらのセキュリティー・メカニズムのそれぞれを、それらのいずれかで接続を確立できるまで試行します。代替選択肢がない場合、または代替選択肢がすべて失敗した場合は、例外が出されます。

serverName
データ・ソースのホスト名または TCP/IP アドレス。このプロパティーのデータ・タイプは String です。

traceFile

DB2 Universal JDBC ドライバーがトレース情報を書き込むファイルの名前を指定します。このプロパティーのデータ・タイプは String です。 traceFile プロパティーは、ファイルに出力トレース・ストリームを送信する logWriter プロパティーの代わりに使用できます。

traceFileAppend

traceFile プロパティーによって指定されるファイルへ付加するか、あるいは上書きするかを指定します。このプロパティーのデータ・タイプ は boolean です。デフォルトは false です。これは、traceFile プロパティーによって指定されるファイルが上書きされることを意味します。

traceLevel

何をトレースするかを指定します。このプロパティーのデータ・タイプは int です。

traceLevel プロパティーでは、以下のトレースを 1 つ以上指定できます。

- com.ibm.db2.jcc.DB2BaseDataSource.TRACE_NONE
- com.ibm.db2.jcc.DB2BaseDataSource.TRACE_CONNECTION_CALLS
- com.ibm.db2.jcc.DB2BaseDataSource.TRACE_STATEMENT_CALLS
- com.ibm.db2.jcc.DB2BaseDataSource.TRACE_RESULT_SET_CALLS
- com.ibm.db2.jcc.DB2BaseDataSource.TRACE_DRIVER_CONFIGURATION
- com.ibm.db2.jcc.DB2BaseDataSource.TRACE_CONNECTS
- com.ibm.db2.jcc.DB2BaseDataSource.TRACE_DRDA_FLOWS
- com.ibm.db2.jcc.DB2BaseDataSource.TRACE_RESULT_SET_META_DATA
- com.ibm.db2.jcc.DB2BaseDataSource.TRACE_PARAMETER_META_DATA
- com.ibm.db2.jcc.DB2BaseDataSource.TRACE_DIAGNOSTICS
- com.ibm.db2.jcc.DB2BaseDataSource.TRACE_SQLJ
- com.ibm.db2.jcc.DB2BaseDataSource.TRACE_XA_CALLS (DB2 UDB for Linux、for UNIX、for Windows での Universal Type 2 Connectivity のみ)
- com.ibm.db2.jcc.DB2BaseDataSource.TRACE_ALL

複数のトレースを指定するには、以下の技法のいずれかを使用します。

- 2 つ以上のトレース値にビット単位 OR (|) 演算子を使用します。たとえば、DRDA フローおよび接続呼び出しをトレースするには、traceLevel に以下の値を指定します。

  ```
  TRACE_DRDA_FLOWS|TRACE_CONNECTION_CALLS
  ```

- ビット単位補数 (~) 演算子を演算子とトレース値とを使用して、特定のトレース以外のすべてを指定する。たとえば、DRDA フロー以外のすべてをトレースする場合、traceLevel に以下の値を指定します。

  ```
  ~TRACE_DRDA_FLOWS
  ```

user

接続の確立に使用するユーザー ID。このプロパティーのデータ・タイプは String です。 DataSource インターフェースを使用して接続を確立するときは、DataSource.getConnection メソッドを次の形式で呼び出すと、このプロパティー値をオーバーライドすることができます。

```
getConnection(user, password);
```

関連概念:

- 488 ページの『DB2 Universal JDBC ドライバー使用時のセキュリティー』
- 321 ページの『DB2 Universal JDBC ドライバーがある JDBC アプリケーションでの LOB』

関連タスク:

- 303 ページの『DataSource インターフェースを使用したデータ・ソースへの接続』
- 300 ページの『DriverManager インターフェースと DB2 Universal JDBC ドライバーを使用したデータ・ソースへの接続』

関連資料:

- 470 ページの『DB2 Universal JDBC ドライバーと他の DB2 JDBC ドライバーの JDBC との相違点』
- 457 ページの『JDBC に対する DB2 Universal JDBC ドライバーの拡張のサマリー』

JDBC API 用ドライバー・サポートの比較

以下の表では、JDBC インターフェースをリストしており、それらがどのドライバーによってサポートされるかを示しています。ドライバーおよびサポートされるプラットフォームは、以下のとおりです。

表 44. DB2 UDB の JDBC ドライバー

JDBC ドライバー名	関連する DB2 UDB
DB2 Universal JDBC ドライバー	DB2 UDB for Linux、UNIX、Windows または DB2 UDB for z/OS
OS/390 用 JDBC/SQLJ 2.0 ドライバー	DB2 UDB for z/OS
Linux、UNIX、および Windows 用 DB2 JDBC Type 2 ドライバー (DB2 JDBC Type 2 ドライバー) (推奨されていません)	DB2 UDB for Linux, UNIX and Windows

表 45. Array メソッドの DB2 JDBC サポート

JDBC メソッド	DB2 Universal JDBC ドライバーのサポート	OS/390 用 JDBC/SQLJ 2.0 ドライバーのサポート	Linux、UNIX、および Windows 用の DB2 JDBC タイプ 2 ドライバーのサポート
getArray	なし	なし	なし
getBaseType	なし	なし	なし
getBaseTypeName	なし	なし	なし
getResultSet	なし	なし	なし

表 46. *BatchUpdateException* メソッドの *DB2 JDBC* サポート

JDBC メソッド	DB2 Universal JDBC ドライバーのサポート	OS/390 用 JDBC/SQLJ 2.0 ドライバーのサポート	Linux、UNIX、および Windows 用の DB2 JDBC タイプ 2 ドライバーのサポート
java.lang.Exception から継承されたメソッド	あり	あり	あり
getUpdateCounts	あり	あり	あり

表 47. *Blob* メソッドの *DB2 JDBC* サポート

JDBC メソッド	DB2 Universal JDBC ドライバーのサポート	OS/390 用 JDBC/SQLJ 2.0 ドライバーのサポート	Linux、UNIX、および Windows 用の DB2 JDBC タイプ 2 ドライバーのサポート
getBinaryStream	あり	あり	あり
getBytes	あり	あり	あり
length	あり	あり	あり
position	あり	あり	あり
setBinaryStream[1]	あり	なし	なし
setBytes[1]	あり	なし	なし
truncate[1]	あり	なし	なし

注:

1. これは JDBC 3.0 のメソッドです。

表 48. *CallableStatement* メソッドの *DB2 JDBC* サポート

JDBC メソッド	DB2 Universal JDBC ドライバーのサポート	OS/390 用 JDBC/SQLJ 2.0 ドライバーのサポート	Linux、UNIX、および Windows 用の DB2 JDBC タイプ 2 ドライバーのサポート
java.sql.Statement から継承されたメソッド	あり	あり	あり
java.sql.PreparedStatement から継承されたメソッド	あり	あり	あり
getArray	なし	なし	なし
getBigDecimal	あり	あり	あり
getBlob	あり	あり	あり
getBoolean	あり	あり	あり
getByte	あり	あり	あり
getBytes	あり	あり	あり
getClob	あり	あり	あり
getDate	あり	あり	あり
getDouble	あり	あり	あり
getFloat	あり	あり	あり
getInt	あり	あり	あり

表 48. *CallableStatement* メソッドの *DB2 JDBC* サポート *(続き)*

JDBC メソッド	DB2 Universal JDBC ドライバーのサポート	OS/390 用 JDBC/SQLJ 2.0 ドライバーのサポート	Linux、UNIX、および Windows 用の DB2 JDBC タイプ 2 ドライバーのサポート
getLong	あり	あり	あり
getObject	あり[1]	あり[1]	あり[1]
getRef	なし	なし	なし
getShort	あり	あり	あり
getString	あり	あり	あり
getTime	あり	あり	あり
getTimestamp	あり	あり	あり
registerOutParameter[2]	あり	あり	あり
wasNull	あり	あり	あり

注:

1. `getObject` メソッドの以下の形式は、サポートされていません。

   `getObject(int parameterIndex, java.util.Map map)`

2. `registerOutParameter` メソッドの以下の形式は、サポートされていません。

   `registerOutParameter(int parameterIndex, int jdbcType, String typeName)`

表 49. *Clob* メソッドの *DB2 JDBC* サポート

JDBC メソッド	DB2 Universal JDBC ドライバーのサポート	OS/390 用 JDBC/SQLJ 2.0 ドライバーのサポート	Linux、UNIX、および Windows 用の DB2 JDBC タイプ 2 ドライバーのサポート
getAsciiStream	あり	あり	あり
getCharacterStream	あり	あり	あり
getSubString	あり	あり	あり
length	あり	あり	あり
position	あり	あり	あり
setAsciiStream[1]	あり	なし	なし
setCharacterStream[1]	あり	なし	なし
setString[1]	あり	なし	なし
truncate[1]	あり	なし	なし

注:

1. これは JDBC 3.0 のメソッドです。

表 50. *Connection* メソッドの *DB2 JDBC* サポート

JDBC メソッド	DB2 Universal JDBC ドライバーのサポート	OS/390 用 JDBC/SQLJ 2.0 ドライバーのサポート	Linux、UNIX、および Windows 用の DB2 JDBC タイプ 2 ドライバーのサポート
clearWarnings	あり	あり	あり
close	あり	あり	あり
commit	あり	あり	あり
createStatement	あり[1]	あり [2]	あり
getAutoCommit	あり	あり	あり
getCatalog	あり	あり	あり
getMetaData	あり	あり	あり
getTransactionIsolation	あり	あり	あり
getTypeMap	なし	なし	なし
getWarnings	あり	あり	あり
isClosed	あり	あり	あり
isReadOnly	あり	あり	あり
nativeSQL	あり	あり	あり
prepareCall	あり	あり[3]	あり
prepareStatement	あり[4]	あり	あり
releaseSavepoint	あり[5]	なし	なし
rollback	あり	あり[6]	あり[6]
setAutoCommit	あり	あり	あり
setCatalog	あり	あり	あり
setReadOnly	あり[7]	あり[7]	あり
setSavepoint	あり[5]	なし	なし
setTransactionIsolation	あり	あり	あり
setTypeMap	なし	なし	なし

表 50. *Connection* メソッドの *DB2 JDBC* サポート *(続き)*

JDBC メソッド	DB2 Universal JDBC ドライバーのサポート	OS/390 用 JDBC/SQLJ 2.0 ドライバーのサポート	Linux、UNIX、および Windows 用の DB2 JDBC タイプ 2 ドライバーのサポート

注:

1. `createStatement` ステートメントの JDBC 2.0 形式に加えて、 `createStatement` の以下の JDBC 3.0 形式がサポートされています。

   ```
   createStatement(int resultSetType,
    int resultSetConcurrency,
    int resultSetHoldability)
   ```

2. `createStatement` の以下の形式、 `TYPE_FORWARD_ONLY` の *resultSetType* 値、および `CONCUR_READ_ONLY` の *resultSetConcurrency* 値がサポートされています。

   ```
   createStatement(int resultSetType, int resultSetConcurrency)
   ```

3. `prepareCall` の以下の形式は、サポートされていません。

   ```
   prepareCall(String sql, int resultSetType, int resultSetConcurrency)
   ```

4. `prepareStatement` の他の形式に加えて、 DB2 Universal JDBC ドライバーは以下の JDBC 3.0 形式をサポートしています。

   ```
   prepareStatement(String sql, int autoGeneratedKeys)
   ```

5. これは JDBC 3.0 のメソッドです。
6. JDBC 3.0 `rollback(Savepoint savepoint)` メソッドは、サポートされていません。
7. このドライバーは、この設定値を使用しません。 DB2 Universal JDBC ドライバーでは、 `Connection` または `DataSource` オブジェクトの `readOnly` プロパティーによって、接続を読み取り専用に設定できます。

表 51. *ConnectionEvent* メソッドの *DB2 JDBC* サポート

JDBC メソッド	DB2 Universal JDBC ドライバーのサポート	OS/390 用 JDBC/SQLJ 2.0 ドライバーのサポート	Linux、UNIX、および Windows 用の DB2 JDBC タイプ 2 ドライバーのサポート
java.util.EventObject から継承されたメソッド	あり	あり	あり
getSQLException	あり	あり	あり

表 52. *ConnectionEventListener* メソッドの *DB2 JDBC* サポート

JDBC メソッド	DB2 Universal JDBC ドライバーのサポート	OS/390 用 JDBC/SQLJ 2.0 ドライバーのサポート	Linux、UNIX、および Windows 用の DB2 JDBC タイプ 2 ドライバーのサポート
connectionClosed	あり	あり	あり
connectionErrorOccurred	あり	あり	あり

表 53. *ConnectionPoolDataSource* メソッドの *DB2 JDBC* サポート

JDBC メソッド	DB2 Universal JDBC ドライバーのサポート	OS/390 用 JDBC/SQLJ 2.0 ドライバーのサポート	Linux、UNIX、および Windows 用の DB2 JDBC タイプ 2 ドライバーのサポート
getLoginTimeout	あり	あり	あり
getLogWriter	あり	あり	あり
getPooledConnection	あり	あり	あり
setLoginTimeout	あり[1]	あり	あり
setLogWriter	あり	あり	あり

注:

1. このメソッドは、 OS/390 または z/OS 環境の DB2 UDB での Universal Type 2 Connectivity ではサポートされていません。

表 54. *DatabaseMetaData* メソッドの *DB2 JDBC* サポート

JDBC メソッド	DB2 Universal JDBC ドライバーのサポート	OS/390 用 JDBC/SQLJ 2.0 ドライバーのサポート	Linux、UNIX、および Windows 用の DB2 JDBC タイプ 2 ドライバーのサポート
allProceduresAreCallable	あり	あり	あり
allTablesAreSelectable	あり	あり	あり
dataDefinitionCausesTransactionCommit	あり	あり	あり
dataDefinitionIgnoredInTransactions	あり	あり	あり
deletesAreDetected	あり	あり	あり
doesMaxRowSizeIncludeBlobs	あり	あり	あり
getAttributes	あり	なし	なし
getBestRowIdentifier	あり	あり	あり
getCatalogs	あり	あり	あり
getCatalogSeparator	あり	あり	あり
getCatalogTerm	あり	あり	あり
getColumnPrivileges	あり	あり	あり
getColumns	あり[1]	あり	あり
getConnection	あり	あり	あり
getCrossReference	あり	あり	あり
getDatabaseMajorVersion	あり	なし	なし
getDatabaseMinorVersion	あり	なし	なし
getDatabaseProductName	あり	あり	あり
getDatabaseProductVersion	あり	あり	あり
getDefaultTransactionIsolation	あり	あり	あり
getDriverMajorVersion	あり	あり	あり
getDriverMinorVersion	あり	あり	あり
getDriverName	あり	あり	あり

表 54. *DatabaseMetaData* メソッドの *DB2 JDBC* サポート *(続き)*

JDBC メソッド	DB2 Universal JDBC ドライバーのサポート	OS/390 用 JDBC/SQLJ 2.0 ドライバーのサポート	Linux、UNIX、および Windows 用の DB2 JDBC タイプ 2 ドライバーのサポート
getDriverVersion	あり	あり	あり
getExportedKeys	あり	あり	あり
getExtraNameCharacters	あり	あり	あり
getIdentifierQuoteString	あり	あり	あり
getImportedKeys	あり	あり	あり
getIndexInfo	あり	あり	あり
getJDBCMajorVersion	あり	なし	なし
getJDBCMinorVersion	あり	なし	なし
getMaxBinaryLiteralLength	あり	あり	あり
getMaxCatalogNameLength	あり	あり	あり
getMaxCharLiteralLength	あり	あり	あり
getMaxColumnNameLength	あり	あり	あり
getMaxColumnsInGroupBy	あり	あり	あり
getMaxColumnsInIndex	あり	あり	あり
getMaxColumnsInOrderBy	あり	あり	あり
getMaxColumnsInSelect	あり	あり	あり
getMaxColumnsInTable	あり	あり	あり
getMaxConnections	あり	あり	あり
getMaxCursorNameLength	あり	あり	あり
getMaxIndexLength	あり	あり	あり
getMaxProcedureNameLength	あり	あり	あり
getMaxRowSize	あり	あり	あり
getMaxSchemaNameLength	あり	あり	あり
getMaxStatementLength	あり	あり	あり
getMaxStatements	あり	あり	あり
getMaxTableNameLength	あり	あり	あり
getMaxTablesInSelect	あり	あり	あり
getMaxUserNameLength	あり	あり	あり
getNumericFunctions	あり	あり	あり
getPrimaryKeys	あり	あり	あり
getProcedureColumns	あり	あり	あり
getProcedures	あり	あり	あり
getProcedureTerm	あり	あり	あり
getResultSetHoldability	あり	なし	なし
getSchemas	あり[1]	あり	あり
getSchemaTerm	あり	あり	あり

表 54. DatabaseMetaData メソッドの DB2 JDBC サポート (続き)

JDBC メソッド	DB2 Universal JDBC ドライバーのサポート	OS/390 用 JDBC/SQLJ 2.0 ドライバーのサポート	Linux、UNIX、および Windows 用の DB2 JDBC タイプ 2 ドライバーのサポート
getSearchStringEscape	あり	あり	あり
getSQLKeywords	あり	あり	あり
getSQLStateType	あり	なし	なし
getStringFunctions	あり	あり	あり
getSuperTables	あり [2]	なし	なし
getSuperTypes	あり [2]	なし	なし
getSystemFunctions	あり	あり	あり
getTablePrivileges	あり	あり	あり
getTables	あり [1]	あり	あり
getTableTypes	あり	あり	あり
getTimeDateFunctions	あり	あり	あり
getTypeInfo	あり	あり	あり
getUDTs	なし	なし	あり [2]
getURL	あり	あり	あり
getUserName	あり	あり	あり
getVersionColumns	あり	あり	あり
insertsAreDetected	あり	あり	あり
isCatalogAtStart	あり	あり	あり
isReadOnly	あり	あり	あり
nullPlusNonNullIsNull	あり	あり	あり
nullsAreSortedAtEnd	あり	あり	あり
nullsAreSortedAtStart	あり	あり	あり
nullsAreSortedHigh	あり	あり	あり
nullsAreSortedLow	あり	あり	あり
othersDeletesAreVisible	あり	あり	あり
othersInsertsAreVisible	あり	あり	あり
othersUpdatesAreVisible	あり	あり	あり
ownDeletesAreVisible	あり	あり	あり
ownInsertsAreVisible	あり	あり	あり
ownUpdatesAreVisible	あり	あり	あり
storesLowerCaseIdentifiers	あり	あり	あり
storesLowerCaseQuotedIdentifiers	あり	あり	あり
storesMixedCaseIdentifiers	あり	あり	あり
storesMixedCaseQuotedIdentifiers	あり	あり	あり
storesUpperCaseIdentifiers	あり	あり	あり
storesUpperCaseQuotedIdentifiers	あり	あり	あり

表 54. *DatabaseMetaData* メソッドの *DB2 JDBC* サポート *(続き)*

JDBC メソッド	DB2 Universal JDBC ドライバーのサポート	OS/390 用 JDBC/SQLJ 2.0 ドライバーのサポート	Linux、UNIX、および Windows 用の DB2 JDBC タイプ 2 ドライバーのサポート
supportsAlterTableWithAddColumn	あり	あり	あり
supportsAlterTableWithDropColumn	あり	あり	あり
supportsANSI92EntryLevelSQL	あり	あり	あり
supportsANSI92FullSQL	あり	あり	あり
supportsANSI92IntermediateSQL	あり	あり	あり
supportsBatchUpdates	あり	あり	あり
supportsCatalogsInDataManipulation	あり	あり	あり
supportsCatalogsInIndexDefinitions	あり	あり	あり
supportsCatalogsInPrivilegeDefinitions	あり	あり	あり
supportsCatalogsInProcedureCalls	あり	あり	あり
supportsCatalogsInTableDefinitions	あり	あり	あり
SupportsColumnAliasing	あり	あり	あり
supportsConvert	あり	あり	あり
supportsCoreSQLGrammar	あり	あり	あり
supportsCorrelatedSubqueries	あり	あり	あり
supportsDataDefinitionAndDataManipulationTransactions	あり	あり	あり
supportsDataManipulationTransactionsOnly	あり	あり	あり
supportsDifferentTableCorrelationNames	あり	あり	あり
supportsExpressionsInOrderBy	あり	あり	あり
supportsExtendedSQLGrammar	あり	あり	あり
supportsFullOuterJoins	あり	あり	あり
supportsGetGeneratedKeys	あり	なし	なし
supportsGroupBy	あり	あり	あり
supportsGroupByBeyondSelect	あり	あり	あり
supportsGroupByUnrelated	あり	あり	あり
supportsIntegrityEnhancementFacility	あり	あり	あり
supportsLikeEscapeClause	あり	あり	あり
supportsLimitedOuterJoins	あり	あり	あり
supportsMinimumSQLGrammar	あり	あり	あり
supportsMixedCaseIdentifiers	あり	あり	あり
supportsMixedCaseQuotedIdentifiers	あり	あり	あり
supportsMultipleOpenResults	あり	あり	なし
supportsMultipleResultSets	あり	あり	あり
supportsMultipleTransactions	あり	あり	あり
supportsNamedParameters	あり	なし	なし
supportsNonNullableColumns	あり	あり	あり

表 54. DatabaseMetaData メソッドの DB2 JDBC サポート (続き)

JDBC メソッド	DB2 Universal JDBC ドライバーのサポート	OS/390 用 JDBC/SQLJ 2.0 ドライバーのサポート	Linux、UNIX、および Windows 用の DB2 JDBC タイプ 2 ドライバーのサポート
supportsOpenCursorsAcross Commit	あり	あり	あり
supportsOpenCursorsAcross Rollback	あり	あり	あり
supportsOpenStatementsAcrossCommit	あり	あり	あり
supportsOpenStatementsAcrossRollback	あり	あり	あり
supportsOrderByUnrelated	あり	あり	あり
supportsOuterJoins	あり	あり	あり
supportsPositionedDelete	あり	あり	あり
supportsPositionedUpdate	あり	あり	あり
supportsResultSetConcurrency	あり	あり	あり
supportsResultSetHoldability	あり	なし	なし
supportsResultSetType	あり	あり	あり
supportsSavepoints	あり	なし	なし
supportsSchemasInDataManipulation	あり	あり	あり
supportsSchemasInIndexDefinitions	あり	あり	あり
supportsSchemasInPrivilegeDefinitions	あり	あり	あり
supportsSchemasInProcedureCalls	あり	あり	あり
supportsSchemasInTableDefinitions	あり	あり	あり
supportsSelectForUpdate	あり	あり	あり
supportsStoredProcedures	あり	あり	あり
supportsSubqueriesInComparisons	あり	あり	あり
supportsSubqueriesInExists	あり	あり	あり
supportsSubqueriesInIns	あり	あり	あり
supportsSubqueriesInQuantifieds	あり	あり	あり
supportsSuperTables	あり	なし	なし
supportsSuperTypes	あり	なし	なし
supportsTableCorrelationNames	あり	あり	あり
supportsTransactionIsolationLevel	あり	あり	あり
supportsTransactions	あり	あり	あり
supportsUnion	あり	あり	あり
supportsUnionAll	あり	あり	あり
updatesAreDetected	あり	あり	あり
usesLocalFilePerTable	あり	あり	あり
usesLocalFiles	あり	あり	あり

注:

1. このメソッドの JDBC 3.0 バージョンはサポートされています。
2. メソッドは実行可能ですが、空の `ResultSet` が戻されます。

表 55. *DataSource* メソッドの *DB2 JDBC* サポート

JDBC メソッド	DB2 Universal JDBC ドライバーのサポート	OS/390 用 JDBC/SQLJ 2.0 ドライバーのサポート	Linux、UNIX、および Windows 用の DB2 JDBC タイプ 2 ドライバーのサポート
getConnection	あり	あり	あり
getLoginTimeout	あり	あり	あり[1]
getLogWriter	あり	あり	あり
setLoginTimeout	あり[2]	あり	あり[1]
setLogWriter	あり	あり	あり

注:

1. DB2 JDBC Type 2 ドライバーでは、この設定を使用しません。
2. このメソッドは、 OS/390 または z/OS 環境の DB2 UDB での Universal Type 2 Connectivity ではサポートされていません。

表 56. *DataTruncation* メソッドの *DB2 JDBC* サポート

JDBC メソッド	DB2 Universal JDBC ドライバーのサポート	OS/390 用 JDBC/SQLJ 2.0 ドライバーのサポート	Linux、UNIX、および Windows 用の DB2 JDBC タイプ 2 ドライバーのサポート
java.lang.Throwable から継承されたメソッド	あり	あり	あり
java.sql.SQLException から継承されたメソッド	あり	あり	あり
java.sql.SQLWarning から継承されたメソッド	あり	あり	あり
getDataSize	あり	あり	あり
getIndex	あり	あり	あり
getParameter	あり	あり	あり
getRead	あり	あり	あり
getTransferSize	あり	あり	あり

表 57. *Driver* メソッドの *DB2 JDBC* サポート

JDBC メソッド	DB2 Universal JDBC ドライバーのサポート	OS/390 用 JDBC/SQLJ 2.0 ドライバーのサポート	Linux、UNIX、および Windows 用の DB2 JDBC タイプ 2 ドライバーのサポート
acceptsURL	あり	あり	あり
connect	あり	あり	あり
getMajorVersion	あり	あり	あり
getMinorVersion	あり	あり	あり
getPropertyInfo	あり	あり	あり
jdbcCompliant	あり	あり	あり

表 58. *DriverManager* メソッドの *DB2 JDBC* サポート

JDBC メソッド	DB2 Universal JDBC ドライバーのサポート	OS/390 用 JDBC/SQLJ 2.0 ドライバーのサポート	Linux、UNIX、および Windows 用の DB2 JDBC タイプ 2 ドライバーのサポート
deregisterDriver	あり	あり	あり
getConnection	あり	あり	あり
getDriver	あり	あり	あり
getDrivers	あり	あり	あり
getLoginTimeout	あり	あり	あり[1]
getLogStream	あり	あり	あり
getLogWriter	あり	あり	あり
println	あり	あり	あり
registerDriver	あり	あり	あり
setLoginTimeout	あり[2]	あり	あり[1]
setLogStream	あり	あり	あり
setLogWriter	あり	あり	あり

注:

1. DB2 JDBC Type 2 ドライバーでは、この設定を使用しません。
2. このメソッドは、 OS/390 または z/OS 環境の DB2 UDB での Universal Type 2 Connectivity ではサポートされていません。

表 59. *ParameterMetaData* メソッドの *DB2 JDBC* サポート

JDBC メソッド	DB2 Universal JDBC ドライバーのサポート	OS/390 用 JDBC/SQLJ 2.0 ドライバーのサポート	Linux、UNIX、および Windows 用の DB2 JDBC タイプ 2 ドライバーのサポート
getParameterClassName	なし	なし	なし
getParameterCount	あり	なし	なし
getParameterMode	あり	なし	なし
getParameterType	あり	なし	なし
getParameterTypeName	あり	なし	なし
getPrecision	あり	なし	なし
getScale	あり	なし	なし
isNullable	あり	なし	なし
isSigned	あり	なし	なし

表 60. *PooledConnection* メソッドの *DB2 JDBC* サポート

JDBC メソッド	DB2 Universal JDBC ドライバーのサポート	OS/390 用 JDBC/SQLJ 2.0 ドライバーのサポート	Linux、UNIX、および Windows 用の DB2 JDBC タイプ 2 ドライバーのサポート
addConnectionEventListener	あり	あり	あり
close	あり	あり	あり

表 60. *PooledConnection* メソッドの *DB2 JDBC* サポート *(続き)*

JDBC メソッド	DB2 Universal JDBC ドライバーのサポート	OS/390 用 JDBC/SQLJ 2.0 ドライバーのサポート	Linux、UNIX、および Windows 用の DB2 JDBC タイプ 2 ドライバーのサポート
getConnection	あり	あり	あり
removeConnectionEventListener	あり	あり	あり

表 61. *PreparedStatement* メソッドの *DB2 JDBC* サポート

JDBC メソッド	DB2 Universal JDBC ドライバーのサポート	OS/390 用 JDBC/SQLJ 2.0 ドライバーのサポート	Linux、UNIX、および Windows 用の DB2 JDBC タイプ 2 ドライバーのサポート
java.sql.Statement から継承されたメソッド	あり	あり	あり
addBatch	あり	あり	あり
clearParameters	あり	あり	あり
execute	あり	あり	あり
executeQuery	あり	あり	あり
executeUpdate	あり	あり	あり
getMetaData	あり	あり	あり
setArray	なし	なし	なし
setAsciiStream	あり	あり	あり
setBigDecimal	あり	あり	あり
setBinaryStream	あり	あり	あり
setBlob	あり	あり	あり
setBoolean	あり	あり	あり
setByte	あり	あり	あり
setBytes	あり	あり	あり
setCharacterStream	あり	あり	あり
setClob	あり	あり	あり
setDate	あり	あり[1]	あり
setDouble	あり	あり	あり
setFloat	あり	あり	あり
setInt	あり	あり	あり
setLong	あり	あり	あり
setNull	あり [2]	あり [2]	あり [2]
setObject	あり	あり	あり
setRef	なし	なし	なし
setShort	あり	あり	あり
setString	あり[3]	あり[3]	あり[3]
setTime	あり[4]	あり[4]	あり
setTimestamp	あり[5]	あり[5]	あり

表 61. *PreparedStatement* メソッドの *DB2 JDBC* サポート *(続き)*

JDBC メソッド	DB2 Universal JDBC ドライバーのサポート	OS/390 用 JDBC/SQLJ 2.0 ドライバーのサポート	Linux、UNIX、および Windows 用の DB2 JDBC タイプ 2 ドライバーのサポート
setUnicodeStream	あり	あり	あり
setURL	あり	なし	あり

注:

1. `setDate` の以下の形式は、サポートされていません。

   `setDate(int parameterIndex, java.sql.Date x, java.util.Calendar cal)`
2. `setNull` の以下の形式は、サポートされていません。

   `setNull(int parameterIndex, int jdbcType, String typeName)`
3. 列に FOR BIT DATA 属性が指定されているか、またはデータ・タイプが BLOB の場合は、 setString はサポートされません。
4. `setTime` の以下の形式は、サポートされていません。

   `setTime(int parameterIndex, java.sql.Time x, java.util.Calendar cal)`
5. `setTimestamp` の以下の形式は、サポートされていません。

   `setTimestamp(int parameterIndex, java.sql.Timestamp x, java.util.Calendar cal)`

表 62. *Ref* メソッドの *DB2 JDBC* サポート

JDBC メソッド	DB2 Universal JDBC ドライバーのサポート	OS/390 用 JDBC/SQLJ 2.0 ドライバーのサポート	Linux、UNIX、および Windows 用の DB2 JDBC タイプ 2 ドライバーのサポート
get BaseTypeName	なし	なし	なし

表 63. *ResultSet* メソッドの *DB2 JDBC* サポート

JDBC メソッド	DB2 Universal JDBC ドライバーのサポート	OS/390 用 JDBC/SQLJ 2.0 ドライバーのサポート	Linux、UNIX、および Windows 用の DB2 JDBC タイプ 2 ドライバーのサポート
absolute	あり	なし	あり
afterLast	あり	なし	あり
beforeFirst	あり	なし	あり
cancelRowUpdates	あり	なし	なし
clearWarnings	あり	あり	あり
close	あり	あり	あり
deleteRow	あり	なし	なし
findColumn	あり	あり	あり
first	あり	なし	あり
getArray	なし	なし	なし
getAsciiStream	あり	あり	あり

表 63. *ResultSet* メソッドの *DB2 JDBC* サポート *(続き)*

JDBC メソッド	DB2 Universal JDBC ドライバーのサポート	OS/390 用 JDBC/SQLJ 2.0 ドライバーのサポート	Linux、UNIX、および Windows 用の DB2 JDBC タイプ 2 ドライバーのサポート
getBigDecimal	あり	あり	あり
getBinaryStream	あり[1]	あり[1]	あり
getBlob	あり	あり	あり
getBoolean	あり	あり	あり
getByte	あり	あり	あり
getBytes	あり	あり	あり
getCharacterStream	あり	あり	あり
getClob	あり	あり	あり
getConcurrency	あり	あり	あり
getCursorName	あり	あり	あり
getDate	あり	あり [2]	あり
getDouble	あり	あり	あり
getFetchDirection	あり	あり	あり
getFetchSize	あり	あり	あり
getFloat	あり	あり	あり
getInt	あり	あり	あり
getLong	あり	あり	あり
getMetaData	あり	あり	あり
getObject	あり[3]	あり[3]	あり[3]
getRef	なし	なし	なし
getRow	あり	なし	あり
getShort	あり	あり	あり
getStatement	あり	あり	あり
getString	あり	あり	あり
getTime	あり	あり[4]	あり
getTimestamp	あり	あり[5]	あり
getType	あり	あり	あり
getUnicodeStream	あり	あり	あり
getURL	あり	なし	あり
getWarnings	あり	あり	あり
insertRow	なし	なし	なし
isAfterLast	あり	なし	あり
isBeforeFirst	あり	なし	あり
isFirst	あり	なし	あり
isLast	あり	なし	あり
last	あり	なし	あり
moveToCurrentRow	あり	なし	なし

表 63. *ResultSet* メソッドの *DB2 JDBC* サポート *(続き)*

JDBC メソッド	DB2 Universal JDBC ドライバーのサポート	OS/390 用 JDBC/SQLJ 2.0 ドライバーのサポート	Linux、UNIX、および Windows 用の DB2 JDBC タイプ 2 ドライバーのサポート
moveToInsertRow	なし	なし	なし
next	あり	あり	あり
previous	あり	なし	あり
refreshRow	あり	なし	なし
relative	あり	なし	あり
rowDeleted	あり	なし	なし
rowInserted	なし	なし	なし
rowUpdated	あり	なし	なし
setFetchDirection	あり	あり[6]	あり
setFetchSize	あり	あり	あり
updateAsciiStream	あり	なし	なし
updateBigDecimal	あり	なし	なし
updateBinaryStream	あり	なし	なし
updateBoolean	あり	なし	なし
updateByte	あり	なし	なし
updateBytes	あり	なし	なし
updateCharacterStream	あり	なし	なし
updateDate	あり	なし	なし
updateDouble	あり	なし	なし
updateFloat	あり	なし	なし
updateInt	あり	なし	なし
updateLong	あり	なし	なし
updateNull	あり	なし	なし
updateObject	あり	なし	なし
updateRow	あり	なし	なし
updateShort	あり	なし	なし
updateString	あり	なし	なし
updateTime	あり	なし	なし
updateTimestamp	あり	なし	なし
wasNull	あり	あり	あり

表 63. *ResultSet* メソッドの *DB2 JDBC* サポート *(続き)*

JDBC メソッド	DB2 Universal JDBC ドライバーのサポート	OS/390 用 JDBC/SQLJ 2.0 ドライバーのサポート	Linux、UNIX、および Windows 用の DB2 JDBC タイプ 2 ドライバーのサポート

注:

1. `getBinaryStream` は、CLOB 列ではサポートされません。
2. `getDate` の以下の形式は、サポートされていません。

   ```
   getDate(int columnIndex, java.util.Calendar cal)
   getDate(String columnName, java.util.Calendar cal)
   ```

3. `getObject` メソッドの以下の形式は、サポートされていません。

   ```
   getObject(int parameterIndex, java.util.Map  map)
   ```

4. `getTime` の以下の形式は、サポートされていません。

   ```
   getTime(int columnIndex, java.util.Calendar cal)
   getTime(String columnName, java.util.Calendar cal)
   ```

5. `getTimestamp` の以下の形式は、サポートされていません。

   ```
   getTimestamp(int columnIndex, java.util.Calendar cal)
   getTimestamp(String columnName, java.util.Calendar cal)
   ```

6. *direction* が `ResultSet.FETCH_FORWARD` である場合にのみサポートされます。

表 64. *ResultSetMetaData* メソッドの *DB2 JDBC* サポート

JDBC メソッド	DB2 Universal JDBC ドライバーのサポート	OS/390 用 JDBC/SQLJ 2.0 ドライバーのサポート	Linux、UNIX、および Windows 用の DB2 JDBC タイプ 2 ドライバーのサポート
getCatalogName	あり	あり	あり
getColumnClassName	なし	なし	あり
getColumnCount	あり	あり	あり
getColumnDisplaySize	あり	あり	あり
getColumnLabel	あり	あり	あり
getColumnName	あり	あり	あり
getColumnType	あり	あり	あり
getColumnTypeName	あり	あり	あり
getPrecision	あり	あり	あり
getScale	あり	あり	あり
getSchemaName	あり	あり	あり
getTableName	あり	あり	あり
isAutoIncrement	あり	あり	あり
isCaseSensitive	あり	あり	あり
isCurrency	あり	あり	あり
isDefinitelyWritable	あり	あり	あり
isNullable	あり	あり	あり
isReadOnly	あり	あり	あり
isSearchable	あり	あり	あり

表 64. *ResultSetMetaData* メソッドの *DB2 JDBC* サポート *(続き)*

JDBC メソッド	DB2 Universal JDBC ドライバーのサポート	OS/390 用 JDBC/SQLJ 2.0 ドライバーのサポート	Linux、UNIX、および Windows 用の DB2 JDBC タイプ 2 ドライバーのサポート
isSigned	あり	あり	あり
isWritable	あり	あり	あり

表 65. *SQLData* メソッドの *DB2 JDBC* サポート

JDBC メソッド	DB2 Universal JDBC ドライバーのサポート	OS/390 用 JDBC/SQLJ 2.0 ドライバーのサポート	Linux、UNIX、および Windows 用の DB2 JDBC タイプ 2 ドライバーのサポート
getSQLTypeName	なし	なし	なし
readSQL	なし	なし	なし
writeSQL	なし	なし	なし

表 66. *SQLException* メソッドの *DB2 JDBC* サポート

JDBC メソッド	DB2 Universal JDBC ドライバーのサポート	OS/390 用 JDBC/SQLJ 2.0 ドライバーのサポート	Linux、UNIX、および Windows 用の DB2 JDBC タイプ 2 ドライバーのサポート
java.lang.Exception から継承されたメソッド	あり	あり	あり
getSQLState	あり	あり	あり
getErrorCode	あり	あり	あり
getNextException	あり	あり	あり
setNextException	あり	あり	あり

表 67. *SQLInput* メソッドの *DB2 JDBC* サポート

JDBC メソッド	DB2 Universal JDBC ドライバーのサポート	OS/390 用 JDBC/SQLJ 2.0 ドライバーのサポート	Linux、UNIX、および Windows 用の DB2 JDBC タイプ 2 ドライバーのサポート
readArray	なし	なし	なし
readAsciiStream	なし	なし	なし
readBigDecimal	なし	なし	なし
readBinaryStream	なし	なし	なし
readBlob	なし	なし	なし
readBoolean	なし	なし	なし
readByte	なし	なし	なし
readBytes	なし	なし	なし
readCharacterStream	なし	なし	なし
readClob	なし	なし	なし

表 67. *SQLInput* メソッドの *DB2 JDBC* サポート *(続き)*

JDBC メソッド	DB2 Universal JDBC ドライバーのサポート	OS/390 用 JDBC/SQLJ 2.0 ドライバーのサポート	Linux、UNIX、および Windows 用の DB2 JDBC タイプ 2 ドライバーのサポート
readDate	なし	なし	なし
readDouble	なし	なし	なし
readFloat	なし	なし	なし
readInt	なし	なし	なし
readLong	なし	なし	なし
readObject	なし	なし	なし
readRef	なし	なし	なし
readShort	なし	なし	なし
readString	なし	なし	なし
readTime	なし	なし	なし
readTimestamp	なし	なし	なし
wasNull	なし	なし	なし

表 68. *SQLOutput* メソッドの *DB2 JDBC* サポート

JDBC メソッド	DB2 Universal JDBC ドライバーのサポート	OS/390 用 JDBC/SQLJ 2.0 ドライバーのサポート	Linux、UNIX、および Windows 用の DB2 JDBC タイプ 2 ドライバーのサポート
writeArray	なし	なし	なし
writeAsciiStream	なし	なし	なし
writeBigDecimal	なし	なし	なし
writeBinaryStream	なし	なし	なし
writeBlob	なし	なし	なし
writeBoolean	なし	なし	なし
writeByte	なし	なし	なし
writeBytes	なし	なし	なし
writeCharacterStream	なし	なし	なし
writeClob	なし	なし	なし
writeDate	なし	なし	なし
writeDouble	なし	なし	なし
writeFloat	なし	なし	なし
writeInt	なし	なし	なし
writeLong	なし	なし	なし
writeObject	なし	なし	なし
writeRef	なし	なし	なし
writeShort	なし	なし	なし
writeString	なし	なし	なし
writeStruct	なし	なし	なし

表 68. *SQLOutput* メソッドの *DB2 JDBC* サポート *(続き)*

JDBC メソッド	DB2 Universal JDBC ドライバーのサポート	OS/390 用 JDBC/SQLJ 2.0 ドライバーのサポート	Linux、UNIX、および Windows 用の DB2 JDBC タイプ 2 ドライバーのサポート
writeTime	なし	なし	なし
writeTimestamp	なし	なし	なし

表 69. *Statement* メソッドの *DB2 JDBC* サポート

JDBC メソッド	DB2 Universal JDBC ドライバーのサポート	OS/390 用 JDBC/SQLJ 2.0 ドライバーのサポート	Linux、UNIX、および Windows 用の DB2 JDBC タイプ 2 ドライバーのサポート
addBatch	あり	あり	あり
cancel	あり[1]	なし	あり
clearBatch	あり	あり	あり
clearWarnings	あり	あり	あり
close	あり	あり	あり
execute	あり[2]	あり	あり
executeBatch	あり	あり	あり
executeQuery	あり	あり	あり
executeUpdate	あり[2]	あり	あり
getConnection	あり	なし	あり
getFetchDirection	あり	なし	あり
getFetchSize	あり	なし	あり
getGeneratedKeys	あり	なし	なし
getMaxFieldSize	あり	あり	あり
getMaxRows	あり	あり	あり
getMoreResults	あり[3]	あり	あり
getQueryTimeout	あり[1]	あり	あり
getResultSet	あり	あり	あり
getResultSetConcurrency	あり	あり	あり
getResultSetType	あり	あり	あり
getUpdateCount[4]	あり	あり	あり
getWarnings	あり	あり	あり
setCursorName	あり	あり	あり
setEscapeProcessing	あり	あり	あり
setFetchDirection	あり	あり	あり
setFetchSize	あり	なし	あり
setMaxFieldSize	あり	あり	あり
setMaxRows	あり	あり	あり
setQueryTimeout	あり[5]	あり[5]	あり

表 69. *Statement* メソッドの *DB2 JDBC* サポート *(続き)*

JDBC メソッド	DB2 Universal JDBC ドライバーのサポート	OS/390 用 JDBC/SQLJ 2.0 ドライバーのサポート	Linux、UNIX、および Windows 用の DB2 JDBC タイプ 2 ドライバーのサポート

注:

1. このメソッドは、 OS/390 または z/OS 環境での Universal Type 2 Connectivity ではサポートされていません。
2. `execute` または `executeUpdate` の他の形式に加えて、 DB2 Universal JDBC ドライバーは以下の JDBC 3.0 形式をサポートしています。

   ```
   executeUpdate(String sql, int autoGeneratedKeys)
   execute(String sql, int autoGeneratedKeys)
   ```

3. `getMoreResults()` に加えて、 DB2 Universal JDBC ドライバーは以下の JDBC 3.0 形式をサポートしています。
   - `getMoreResults(java.sql.Statement.CLOSE_CURRENT_RESULT)`
   - `getMoreResults(java.sql.Statement.KEEP_CURRENT_RESULT)`
   - `getMoreResults(java.sql.Statement.CLOSE_ALL_RESULTS)`
4. ストアード・プロシージャー `ResultSet` ではサポートされていません。
5. *seconds* 値が 0 の場合にのみサポートされます。

表 70. *Struct* メソッドの *DB2 JDBC* サポート

JDBC メソッド	DB2 Universal JDBC ドライバーのサポート	OS/390 用 JDBC/SQLJ 2.0 ドライバーのサポート	Linux、UNIX、および Windows 用の DB2 JDBC タイプ 2 ドライバーのサポート
getSQLTypeName	なし	なし	なし
getAttributes	なし	なし	なし

表 71. *XAConnection* メソッドの *DB2 JDBC* サポート

JDBC メソッド	DB2 Universal JDBC ドライバーのサポート	OS/390 用 JDBC/SQLJ 2.0 ドライバーのサポート	Linux、UNIX、および Windows 用の DB2 JDBC タイプ 2 ドライバーのサポート
javax.sql.PooledConnection から継承されたメソッド	あり[1]	なし	あり
getXAResource	あり[1]	なし	あり

注:

1. このメソッドは、 DB2 UDB for Linux、for UNIX、および for Windows サーバーへの DB2 Universal JDBC Driver type 2 connectivity の場合と、 DB2 UDB for z/OS サーバーへの DB2 Universal JDBC Driver type 4 connectivity の場合にサポートされます。

表 72. *XADataSource* メソッドの *DB2 JDBC* サポート

JDBC メソッド	DB2 Universal JDBC ドライバーのサポート	OS/390 用 JDBC/SQLJ 2.0 ドライバーのサポート	Linux、UNIX、および Windows 用の DB2 JDBC タイプ 2 ドライバーのサポート
getLoginTimeout	あり	なし	あり
getLogWriter	あり	なし	あり

表 72. *XADataSource* メソッドの *DB2 JDBC* サポート *(続き)*

JDBC メソッド	DB2 Universal JDBC ドライバーのサポート	OS/390 用 JDBC/SQLJ 2.0 ドライバーのサポート	Linux、UNIX、および Windows 用の DB2 JDBC タイプ 2 ドライバーのサポート
getXAConnection	あり	なし	あり
setLoginTimeout	あり	なし	あり
setLogWriter	あり	なし	あり

関連資料:

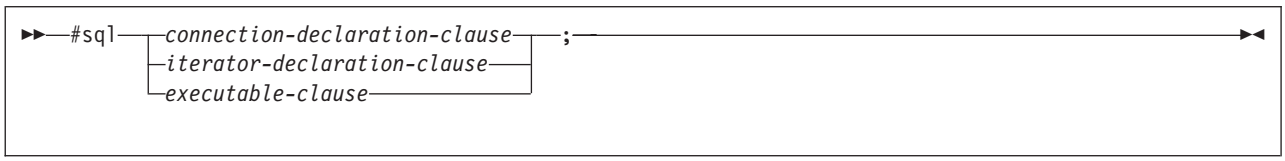
- 470 ページの『DB2 Universal JDBC ドライバーと他の DB2 JDBC ドライバーの JDBC との相違点』

SQLJ ステートメントのリファレンス

以下のセクションには、 SQLJ 文節の構文についての情報が含まれています。

SQLJ 文節

SQL プログラム内の SQLJ ステートメントは、SQLJ 文節内にあります。SQLJ 文節の一般的な構文は、以下のとおりです。



SQLJ 文節内のキーワードは、それらのキーワードが実行可能文節内の SQL ステートメントの一部である場合を除いて、大文字小文字が区別されます。

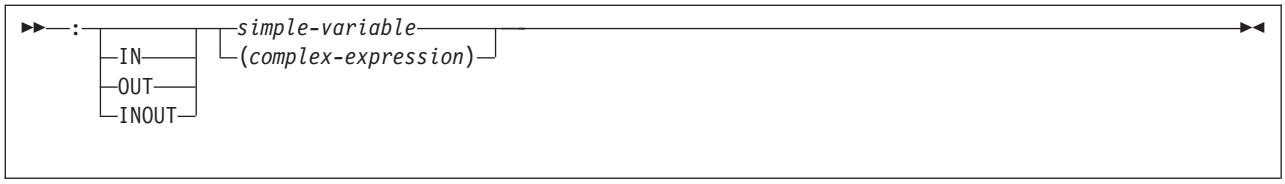
関連資料:

- 441 ページの『SQLJ 接続宣言文節』
- 443 ページの『SQLJ 実行可能文節』
- 442 ページの『SQLJ イテレーター宣言文節』

SQLJ ホスト式

ホスト式は、SQLJ アプリケーション・プログラム内で SQLJ 文節によって参照される Java 変数または式です。

構文:



説明:

: 続く変数または式がホスト式であることを示します。変数または式の直前にコロンを付ける必要があります。

IN|OUT|INOUT
プロシージャー呼び出しでパラメーターとして使用されるホスト式について、そのパラメーターがストアード・プロシージャーにデータを提供する (IN) のか、ストアード・プロシージャーからデータを検索する (OUT) のか、またはその両方 (INOUT) なのかを示します。デフォルトは IN です。

simple-variable
Java の修飾なしの ID を指定します。

complex-expression
単一値の結果となる Java 式を指定します。

使用上の注意:

- 複合式は、括弧で囲む必要があります。
- 静的 SQL ステートメント内でホスト式を配置可能な場所では、ANSI/ISO 規則が適用されます。

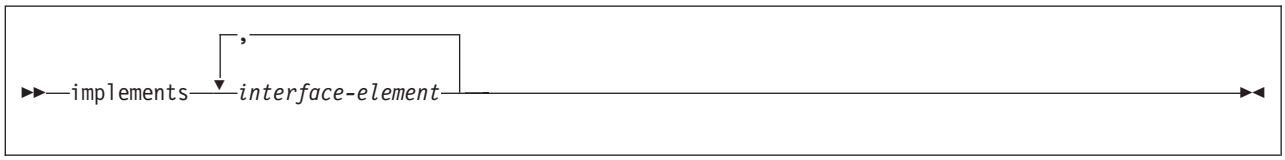
関連概念:

- 355 ページの『SQLJ アプリケーションでの変数』

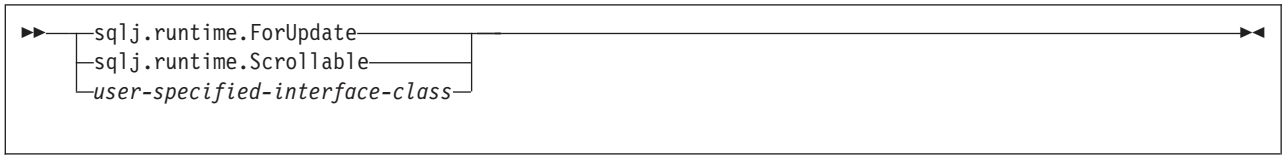
SQLJ インプリメント文節

インプリメント文節は、1 つ以上のクラスを Java インターフェースから派生します。

構文:



interface-element:



説明:

interface-element

ユーザー定義の Java インターフェース、SQLJ インターフェース `sqlj.runtime.ForUpdate`、または SQLJ インターフェース `sqlj.runtime.Scrollable` を指定します。

位置指定 UPDATE または位置指定 DELETE 操作のイテレーターを宣言するときは、 `sqlj.runtime.ForUpdate` をインプリメントする必要があります。SQLJ で位置指定 UPDATE および DELETE 操作を実行する方法については、『SQLJ アプリケーションでの位置指定 UPDATE および DELETE 操作の実行』を参照してください。

スクロール可能イテレーターを宣言するときは、`sqlj.runtime.Scrollable` をインプリメントする必要があります。スクロール可能イテレーターについては、『SQLJ アプリケーションでのスクロール可能イテレーターの使用』を参照してください。

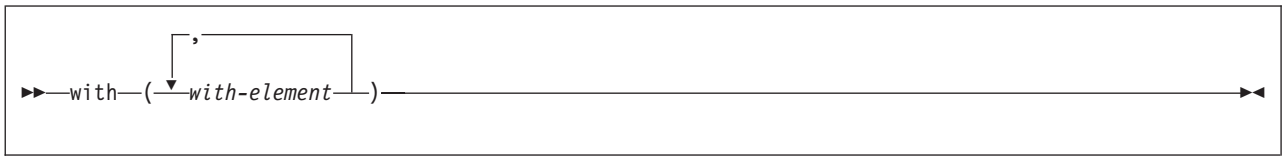
関連タスク:

- 373 ページの『SQLJ アプリケーションでの位置指定 UPDATE および DELETE 操作の実行』
- 399 ページの『SQLJ アプリケーションでのスクロール可能イテレーターの使用』

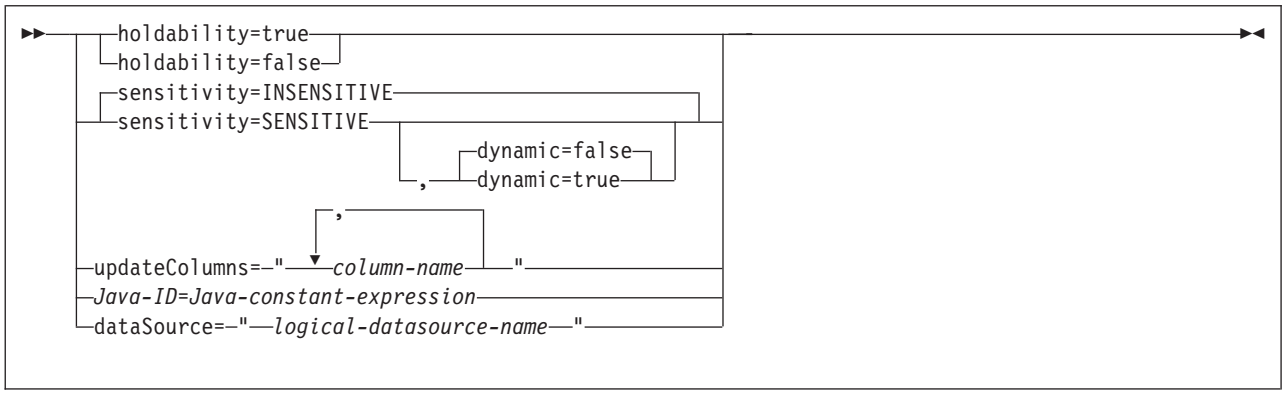
SQLJ with 文節

with 文節は、イテレーターまたは接続コンテキストに対して 1 つ以上の属性を指定します。

構文:



with-element:



説明:

holdability
イテレーターについて、COMMIT の実行後にイテレーターが表内での位置を保持できるかどうかを指定します。 holdability の値は、`true` または `false` でなければなりません。

sensitivity
イテレーターについて、オープンした後に基礎表に対して加えられた変更をイテレーターから見ることができるかどうかを指定します。値は INSENSITIVE または SENSITIVE でなければなりません。デフォルトは INSENSITIVE です。

dynamic
sensitivity=SENSITIVE として定義されたイテレーターについて、以下が当てはまるかどうかを指定します。

- アプリケーションが、イテレーターがある位置指定 UPDATE および DELETE ステートメントを実行するとき、それらの変更をイテレーターから見ることができる。
- アプリケーションが、アプリケーション内で、ただしイテレーターの外で、INSERT、UPDATE、および DELETE ステートメントを実行するとき、それらの変更をイテレーターから見ることができる。

dynamic の値は、`true` または `false` でなければなりません。デフォルトは `false` です。

dynamic の値が `true` の場合、データ・ソースが動的な両方向スクロール・カーソルをサポートしている必要があります。

updateColumns
イテレーターについて、イテレーターが位置指定 UPDATE ステートメントに使用されるときに変更される列を指定します。 updateColumns の値は、コンマで区切られた列名を含むリテラル・ストリングでなければなりません。

column-name
イテレーターについて、イテレーターを使用して更新する結果表の列を指定します。

Java-ID
イテレーターまたは接続コンテキストについて、イテレーターまたは接続コンテキストのユーザー定義属性を識別する Java 変数を指定します。
Java-constant-expression の値も、ユーザー定義です。

dataSource
接続コンテキストについて、アプリケーションの接続先となるデータ・ソースを表す、個別に作成された `DataSource` オブジェクトの論理名を指定します。このオプションは DB2 Universal JDBC ドライバーの場合のみ使用できます。

使用上の注意:

- with エレメントの左側の値は、その with 文節内でユニークでなければなりません。
- イテレーター宣言文節の with エレメントに updateColumns を指定する場合、イテレーター宣言文節には sqlj.runtime.ForUpdate インターフェースを指定するインプリメント文節も含める必要があります。

- SQLJ プログラムをカスタマイズしない場合、JDBC ドライバーはwith 文節内の holdability の値を無視します。その代わり、ドライバーは holdability に JDBC ドライバーの設定を使用します。

関連概念:

- 381 ページの『同じアプリケーションでの SQLJ と JDBC の使用』

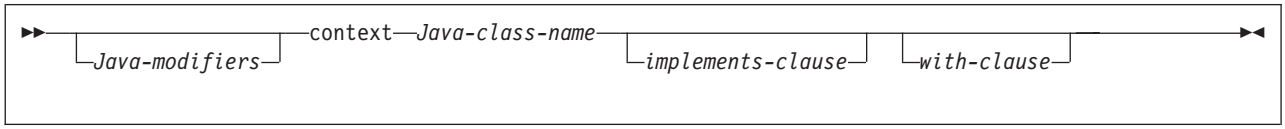
関連タスク:

- 357 ページの『SQLJ を使用したデータ・ソースへの接続』
- 373 ページの『SQLJ アプリケーションでの位置指定 UPDATE および DELETE 操作の実行』
- 399 ページの『SQLJ アプリケーションでのスクロール可能イテレーターの使用』

SQLJ 接続宣言文節

接続宣言文節は、SQLJ アプリケーション・プログラム内のデータ・ソースに対する接続を宣言します。

構文:



説明:

Java-modifiers
Java クラス宣言に有効な修飾子 (static、public、private、protected など) を指定します。

Java-class-name
有効な Java ID を指定します。プログラム準備処理の際に、SQLJ はこの ID を名前とする接続コンテキスト・クラスを生成します。

implements-clause
この文節については、『SQLJ インプリメント文節』を参照してください。接続宣言文節で、インプリメント文節が参照するインターフェース・クラスはユーザー定義のインターフェース・クラスでなければなりません。

with-clause
この文節については、『SQLJ with 文節』を参照してください。

使用上の注意:

- SQLJ は、指定した接続宣言文節ごとに接続クラス宣言を生成します。SQLJ データ・ソース接続は、それら生成された接続クラスのオブジェクトです。
- Java プログラム内の Java クラス定義を配置可能な場所であればどこにでも、接続宣言文節を指定することができます。

関連タスク:

- 357 ページの『SQLJ を使用したデータ・ソースへの接続』

関連資料:

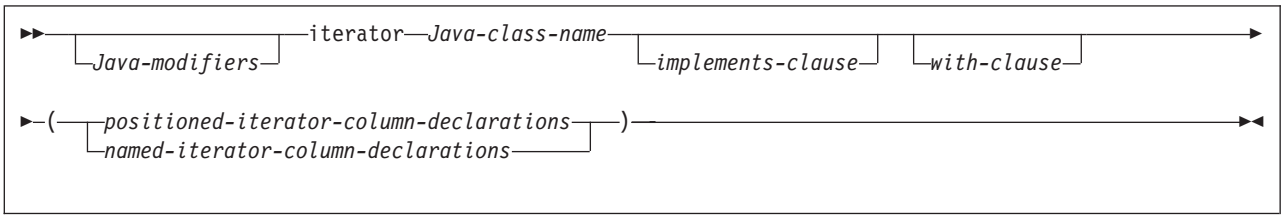
- 438 ページの『SQLJ インプリメント文節』
- 439 ページの『SQLJ with 文節』

SQLJ イテレーター宣言文節

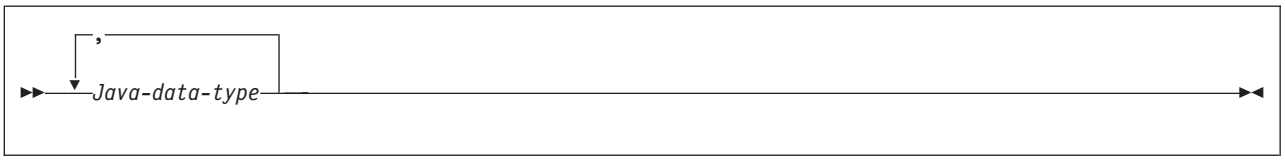
イテレーター宣言文節は、SQLJ アプリケーション・プログラム内で、位置指定イテレーター・クラスまたは名前指定イテレーター・クラスを宣言します。イテレーターには、照会からの結果表が含まれます。指定したイテレーター宣言文節ごとに、SQLJ はイテレーター・クラスを生成します。イテレーターは、イテレーター・クラスのオブジェクトです。

イテレーター宣言文節には、位置指定イテレーターの形式および名前指定イテレーターの形式があります。これら 2 種類のイテレーターは別個のものであり、異なるインターフェースによってインプリメントされた互換性のない Java タイプです。

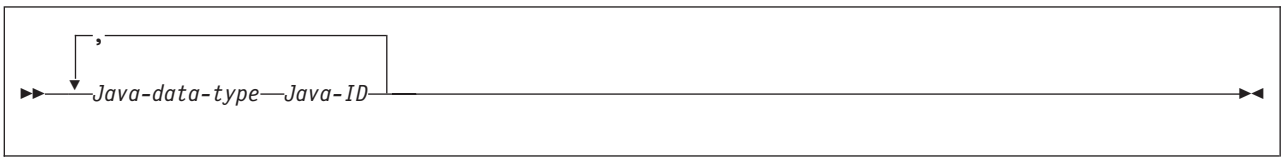
構文:



positioned-iterator-column declarations:



named-iterator-column-declarations:



説明:

Java-modifiers
Java クラス宣言に有効な修飾子 (static、public、private、protected など)。

Java-class-name
有効な Java ID。プログラム準備処理の際に、SQLJ はこの ID を名前とするイテレーター・クラスを作成します。

implements-clause
この文節については、『SQLJ インプリメント文節』を参照してください。位置指定 UPDATE または位置指定 DELETE 操作のイテレーターを宣言するイテレーター宣言文節では、インプリメント文節が、インターフェース

sqlj.runtime.ForUpdate を指定する必要があります。スクロール可能イテレーターを宣言するイテレーター宣言文節では、インプリメント文節が、インターフェース sqlj.runtime.Scrollable を指定する必要があります。

with-clause
この文節については、『SQLJ with 文節』を参照してください。

positioned-iterator-column-declarations
位置指定イテレーター内の列のデータ・タイプである、Java データ・タイプのリストを指定します。リスト内のデータ・タイプは、コンマで区切る必要があります。位置指定イテレーター宣言内のデータ・タイプの順序は、結果表内の列の順序と同じです。一連のプロファイルのカスタマイズ中にオンライン検査が成功するように、イテレーターの列のデータ・タイプは、結果表の列のデータ・タイプと互換性がなければなりません。互換性のあるデータ・タイプのリストについては、『Java、JDBC、および SQL のデータ・タイプ』を参照してください。

named-iterator-column-declarations
名前指定イテレーター内の列のデータ・タイプおよび名前である、 Java データ・タイプおよび Java ID のリストを指定します。データ・タイプと名前の対は、コンマで区切る必要があります。イテレーターの列の名前は、大文字小文字の違いを除いて、結果表の列の名前と同じでなければなりません。一連のプロファイルのカスタマイズ中にオンライン検査が成功するように、イテレーターの列のデータ・タイプは、結果表の列のデータ・タイプと互換性がなければなりません。互換性のあるデータ・タイプのリストについては、『Java、JDBC、および SQL のデータ・タイプ』を参照してください。

使用上の注意:

- イテレーター宣言文節は、Java クラス宣言が配置可能な場所であれば、Java プログラムのどこにでも配置できます。
- 名前指定イテレーター宣言に Java データ・タイプおよび Java ID の複数の対が含まれる場合、リスト内のすべての Java ID はユニークでなければなりません。

関連タスク:

- 357 ページの『SQLJ を使用したデータ・ソースへの接続』

関連資料:

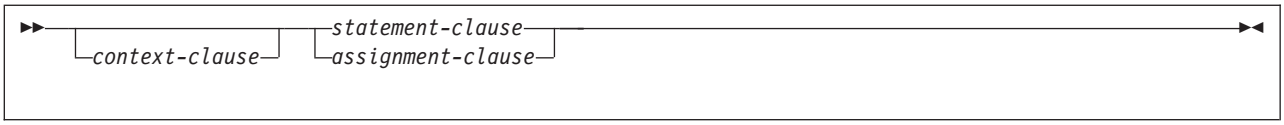
- 438 ページの『SQLJ インプリメント文節』
- 439 ページの『SQLJ with 文節』

SQLJ 実行可能文節

実行可能文節には、SQL ステートメントまたは代入ステートメントが含まれます。代入ステートメントは、SQL 操作の結果を Java 変数に代入します。

このトピックでは、一般的な形式の実行可能文節について説明します。

構文:



使用上の注意:

- 実行可能文節は、Java ステートメントが配置可能な場所であれば、 Java プログラムのどこにでも配置できます。
- SQLJ はクラス `java.sql.SQLException` を介して、実行可能文節から負の SQL コードを報告します。

  実行可能文節の実行中に SQLJ がランタイム例外を出した場合、タイプ OUT または INOUT のホスト式の値が未定義です。

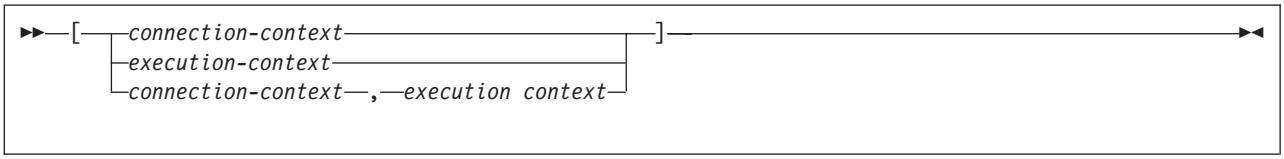
関連資料:

- 447 ページの『SQLJ 代入文節』
- 444 ページの『SQLJ コンテキスト文節』
- 445 ページの『SQLJ ステートメント文節』

SQLJ コンテキスト文節

コンテキスト文節は、接続コンテキスト、実行文節、またはその両方を指定します。 接続コンテキストは、データ・ソースに接続するために使用します。実行コンテキストは、SQL ステートメントの実行をモニターおよび変更するために使用します。

構文:



説明:

connection-context
SQLJ プログラム内で既に宣言済みの有効な Java ID を指定します。その ID は、SQLJ が接続宣言文節のために生成する、接続コンテキスト・クラスのインスタンスとして宣言されている必要があります。

execution-context
SQLJ プログラム内で既に宣言済みの有効な Java ID を指定します。その ID は、クラス `sqlj.runtime.ExecutionContext` のインスタンスとして宣言されている必要があります。

使用上の注意:

- 接続コンテキストを実行可能文節で指定しない場合、SQLJ はデフォルトの接続コンテキストを使用します。
- 実行コンテキストを指定しない場合、SQLJ は実行コンテキストをステートメントの接続コンテキストから取得します。

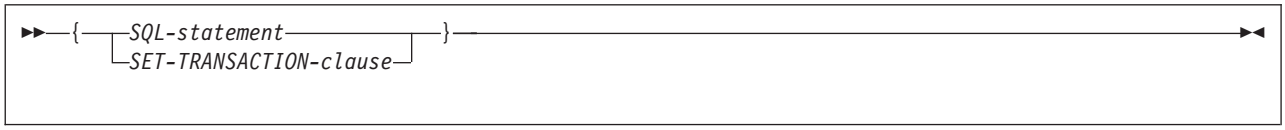
関連タスク:

- 357 ページの『SQLJ を使用したデータ・ソースへの接続』
- 390 ページの『SQLJ での SQL ステートメントの実行の制御』

SQLJ ステートメント文節

ステートメント文節には、SQL ステートメントまたは SET TRANSACTION 文節が含まれます。

構文:



説明:

SQL-statement
ステートメント文節には、表 73 に示す DB2 UDB for Linux, UNIX, Windows SQL ステートメントを含めることができます。

SET-TRANSACTION-clause
プログラム内の SQL ステートメントの分離レベル、および接続のアクセス・モードを設定します。 SET TRANSACTION 文節は、1992 年の ANSI/ISO SQL 標準で記述され、 SQL の一部のインプリメンテーションでサポートされている、SET TRANSACTION ステートメントに相当します。詳しくは、SQLJ SET-TRANSACTION 文節を参照してください。

表 73. SQLJ ステートメント文節での有効な SQL ステートメント

ALTER DATABASE
ALTER FUNCTION
ALTER INDEX
ALTER PROCEDURE
ALTER STOGROUP
ALTER TABLE
ALTER TABLESPACE
CALL
COMMENT ON
COMMIT
CREATE ALIAS
CREATE DATABASE
CREATE DISTINCT TYPE
CREATE FUNCTION
CREATE GLOBAL TEMPORARY TABLE
CREATE INDEX
CREATE PROCEDURE
CREATE STOGROUP
CREATE SYNONYM
CREATE TABLE
CREATE TABLESPACE
CREATE TRIGGER
CREATE VIEW
DECLARE GLOBAL TEMPORARY TABLE
DELETE
DROP ALIAS

表 73. SQLJ ステートメント文節での有効な SQL ステートメント (続き)

DROP DATABASE
DROP DISTINCT TYPE
DROP FUNCTION
DROP INDEX
DROP PACKAGE
DROP PROCEDURE
DROP STOGROUP
DROP SYNONYM
DROP TABLE
DROP TABLESPACE
DROP TRIGGER
DROP VIEW
FETCH
GRANT
INSERT
LOCK TABLE
REVOKE
ROLLBACK
SAVEPOINT
SELECT INTO
SET CURRENT DEFAULT TRANSFORM GROUP
SET CURRENT DEGREE
SET CURRENT EXPLAIN MODE
SET CURRENT EXPLAIN SNAPSHOT
SET CURRENT ISOLATION
SET CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION
SET CURRENT OPTIMIZATION HINT
SET CURRENT PACKAGESET (USER はサポートされない)
SET CURRENT PRECISION
SET CURRENT QUERY OPTIMIZATION
SET CURRENT REFRESH AGE
SET CURRENT SCHEMA
SET PATH
UPDATE

使用上の注意:

- SQLJ は、位置指定および検索済み DELETE と UPDATE 操作の両方をサポートします。
- FETCH ステートメント、位置指定 DELETE ステートメント、または位置指定 UPDATE ステートメントでは、イテレーターを使用して結果表の行を参照する必要があります。

関連タスク:

- 362 ページの『SQLJ トランザクションの分離レベルの設定』

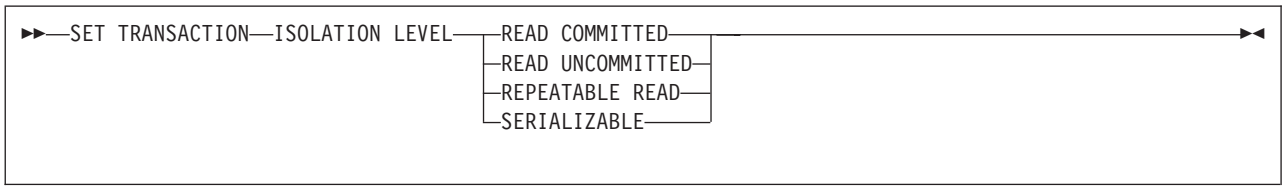
関連資料:

- 446 ページの『SQLJ SET-TRANSACTION 文節』

SQLJ SET-TRANSACTION 文節

SET TRANSACTION 文節は、現行の作業単位の分離レベルを設定します。

構文:



説明:

ISOLATION LEVEL
以下の分離レベルのいずれかを指定します。

READ COMMITTED
現行の DB2 分離レベルがカーソル固定であることを指定します。

READ UNCOMMITTED
現行の DB2 分離レベルが非コミット読み取りであることを指定します。

REPEATABLE READ
現行の DB2 分離レベルが読み取り固定であることを指定します。

SERIALIZABLE
現行の DB2 分離レベルが反復可能読み取りであることを指定します。

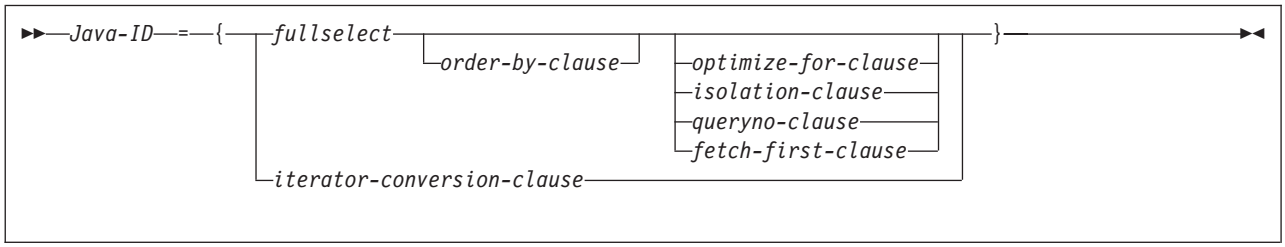
使用上の注意:

SET TRANSACTION を実行できるのは、トランザクションの開始時だけです。

SQLJ 代入文節

代入文節は、SQL 操作の結果を Java 変数に代入します。

構文:



説明:

Java-ID
イテレーター・クラスのインスタンスとして以前に宣言されたイテレーターを示します。

fullselect
結果表を生成します。

iterator-conversion-clause
この文節については、 『SQLJ イテレーター変換文節』を参照してください。

使用上の注意:

- *Java-ID* で識別されるオブジェクトが位置指定イテレーターの場合、結果セットの列の数がイテレーターの列の数と一致していなければなりません。さらに、結果セット内の各列のデータ・タイプは、イテレーター内の対応する列のデータ・タイプと互換性がなければなりません。互換性のある Java および SQL データ・タイプのリストについては、 『Java、JDBC、および SQL のデータ・タイプ』を参照してください。
- *Java-ID* で識別されるオブジェクトが名前指定イテレーターである場合、各アクセサー・メソッドの名前は、大文字小文字の違いを除いて、結果セット内の列の名前と同じでなければなりません。さらに、アクセサー・メソッドが戻すオブジェクトのデータ・タイプは、結果セット内の対応する列のデータ・タイプと互換性がなければなりません。
- 代入文節は、Java 代入ステートメントが配置可能な場所であれば、Java プログラムのどこにでも配置できます。ただし、Java 代入式が配置可能な場所には代入文節を配置できません。たとえば、ステートメントの制御リストには代入文節を指定できません。

関連概念:

- 381 ページの『同じアプリケーションでの SQLJ と JDBC の使用』

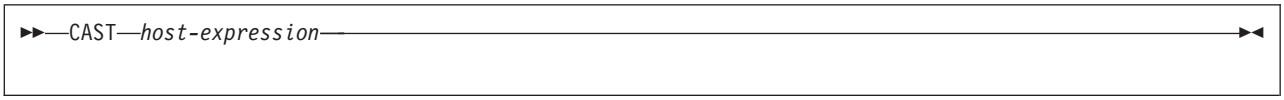
関連資料:

- 「*SQL リファレンス 第 1 巻*」の『全選択』
- 「*SQL リファレンス 第 1 巻*」の『Select-statement』
- 448 ページの『SQLJ イテレーター変換文節』

SQLJ イテレーター変換文節

イテレーター変換文節は、JDBC `ResultSet` をイテレーターに変換します。

構文:



説明:

host-expression
SQLJ イテレーターに変換する JDBC `ResultSet` を示します。

使用上の注意:

- JDBC `ResultSet` が変換される先のイテレーターが位置指定イテレーターの場合、`ResultSet` での列の数がイテレーターでの列の数と一致していなければなりません。さらに、`ResultSet` 内の各列のデータ・タイプは、イテレーター内の対応する列のデータ・タイプと互換性がなければなりません。
- イテレーターが名前指定イテレーターである場合、各アクセサー・メソッドの名前は、大文字小文字の違いを除いて、`ResultSet` 内の列の名前と同じでなければなりません。さらに、アクセサー・メソッドが戻すオブジェクトのデータ・タイプは、`ResultSet` 内の対応する列のデータ・タイプと互換性がなければなりません。

- イテレーター変換文節によって変換されたイテレーターがクローズされると、イテレーターの生成元である ResultSet もクローズされます。

関連概念:

- 381 ページの『同じアプリケーションでの SQLJ と JDBC の使用』

選択済み sqlj.runtime クラスおよびインターフェース

sqlj.runtime パッケージは、SQLJ が使用するランタイム・クラスおよびインターフェースを定義しています。このトピックでは、以下について説明します。

- SQLJ アプリケーション・プログラムで呼び出すことができるメソッドを含む sqlj.runtime の各クラス
- SQLJ アプリケーション・プログラムにインプリメントが必要になる場合のある各インターフェース

sqlj.runtime.ExecutionContext クラス:

sqlj.runtime.ExecutionContext クラスは、実行コンテキスト用に定義されます。実行コンテキストを使用して、SQL ステートメントの実行を制御することができます。実行コンテキストを宣言してその実行コンテキストのインスタンスを作成した後に、以下のメソッドを使用できます。

executeBatch

形式:

```
public synchronized int[] executeBatch()
```

ペンディングのステートメント・バッチを実行して、更新カウントの配列を戻します。ペンディングのステートメント・バッチがない場合は、NULL が戻されます。このメソッドが呼び出されると、その呼び出しの結果として例外が出される場合でも、ステートメント・バッチはクリアされます。

getBatchLimit

形式:

```
synchronized public int getBatchLimit()
```

現行のバッチ制限を戻します。これは、バッチが暗黙的に実行される前にそのバッチに追加されるステートメントの数です。

getBatchUpdateCounts

形式:

```
public synchronized int[] getBatchUpdateCounts()
```

バッチを正常に実行した各ステートメントによって更新された行数を含む配列を戻します。バッチ内のステートメントで正常に完了したものがない場合は、NULL を戻します。

getMaxFieldSize

形式:

```
public int getMaxFieldSize()
```

指定の実行コンテキストを使用する、照会の文字またはバイナリー列に対して戻される、最大バイト数を戻します。値 0 は、最大バイト数は無制限であることを示します。

getMaxRows

形式:

```
public int getMaxRows()
```

指定の実行コンテキストを使用する、照会に対して戻される行の最大数を戻します。値 0 は、最大行数は無制限であることを示します。

getNextResultSet

形式:

```
public ResultSet getNextResultSet()
public ResultSet getNextResultSet(int current)
```

ストアード・プロシージャー呼び出しの後で、ストアード・プロシージャーから結果セットを戻します。値 NULL は、戻す結果セットがないことを示します。

getNextResultSet() を呼び出すと、SQLJ は現在開いている結果セットをクローズして、次の結果セットに進みます。 getNextResultSet(int *current*) を呼び出す場合は、次の結果セットに進む前に、現在オープンしている結果セットに対して SQLJ が実行する処理を、以下の *current* の値で指定します。

java.sql.Statement.CLOSE_CURRENT_RESULT
次の ResultSet オブジェクトが戻されるときに、現行の ResultSet オブジェクトをクローズすることを指定します。

java.sql.Statement.KEEP_CURRENT_RESULT
次の ResultSet オブジェクトが戻されるときに、現行の ResultSet オブジェクトをオープンしたままにすることを指定します。

java.sql.Statement.CLOSE_ALL_RESULTS
次の ResultSet オブジェクトが戻されるときに、オープンしている ResultSet オブジェクトをすべてクローズすることを指定します。

getNextResultSet(int *current*) には、JDK 1.4 以降が必要です。

getUpdateCount

形式:

```
public abstract int getUpdateCount() throws SQLException
```

以下が戻されます。

ExecutionContext.ADD_BATCH_COUNT
ステートメントが既存のバッチに追加された場合。

ExecutionContext.NEW_BATCH_COUNT
ステートメントが新規バッチの最初のステートメントだった場合。

ExecutionContext.EXEC_BATCH_COUNT
ステートメントがバッチの一部であり、そのバッチが実行された場合。

その他の整数
ステートメントがバッチに追加されずに実行された場合。この値は、ステートメントによって更新された行の数です。

getWarnings

形式:

```
public SQLWarning getWarnings()
```

このコンテキストを使用して実行された最後の SQL 操作によって報告された、最初の警告を戻します。後続の警告は、最初の警告にチェーニングされます。

このメソッドを使用して、正の SQLCODE を検索します。

isBatching

形式:

```
public synchronized boolean isBatching()
```

バッチ処理が使用可能であれば、`true` を戻します。バッチ処理が使用不可であれば、`false` を戻します。

setBatching

形式:

```
public synchronized void setBatching(boolean)
```

バッチ処理を使用可能または使用不可にします。

setBatchLimit

形式:

```
public synchronized void setBatchLimit(int)
```

バッチが暗黙的に実行される前にそのバッチに追加されるステートメントの最大数を設定します。入力パラメーターに使用できる値は、以下のとおりです。

ExecutionContext.UNLIMITED_BATCH
バッチ可能であっても非互換のステートメント、またはバッチ可能ではないステートメントが SQLJ によって検出されたときのみ、暗黙的実行が行われることを示します。この値を設定することは、`setBatchLimit` を呼び出さないことと同じです。

ExecutionContext.AUTO_BATCH
バッチ内のステートメント数が SQLJ によって設定された数に達したときに暗黙的実行が行われることを示します。

正整数
SQLJ がバッチを暗黙的に実行する前にバッチに追加されるステートメントの数。バッチ可能であっても非互換のステートメント、またはバッチ可能ではないステートメントが SQLJ によって検出されたときは、この数のステートメントがまだ追加されていなくてもバッチが実行されることがあります。

setMaxFieldSize

形式:

```
public void setMaxFieldSize(int max)
```

指定の実行コンテキストを使用する、照会の文字またはバイナリー列に対して戻される、最大バイト数を指定します。デフォルトは 0 です。これは、最大バイト数が無制限であることを示します。

setMaxRows

形式:

```
public void setMaxRows(int max)
```

指定の実行コンテキストを使用する、照会に対して戻される行の最大数を指定します。デフォルトは 0 です。これは、最大行数が無制限であることを示します。

sqlj.runtime.ConnectionContext インターフェース:

sqlj.runtime.ConnectionContext は、接続宣言文節を実行して、それにより接続コンテキスト・クラスを作成したときに、SQLJ がインプリメントするインターフェースです。

たとえば、Ctx という名前の接続を宣言するとします。その場合、以下のメソッドを使用してデフォルトのコンテキストを判別または変更することができます。

getDefaultContext

形式:

```
public static Ctx getDefaultContext()
```

Ctx クラスのデフォルトの接続コンテキスト・オブジェクトを戻します。

SetDefaultContext

形式:

```
public static void Ctx setDefaultContext(Ctx default-context)
```

Ctx クラスのデフォルトの接続コンテキスト・オブジェクトを設定します。

sqlj.runtime.ForUpdate インターフェース:

sqlj.runtime.ForUpdate インターフェースを、位置指定 UPDATE または DELETE 操作のためにインプリメントします。 sqlj.runtime.ForUpdate は、SQLJ イテレーター宣言文節にインプリメントします。

sqlj.runtime.NamedIterator インターフェース:

sqlj.runtime.NamedIterator は、名前指定イテレーターを宣言するときに SQLJ がインプリメントするインターフェースです。名前指定イテレーターのインスタンスを宣言すると、SQLJ は期待される結果表の各列にアクセサー・メソッドを作成します。アクセサー・メソッドは、結果表の列からデータを戻します。アクセサー・メソッドの名前は、名前指定イテレーターの対応する列の名前と同じです。

アクセサー・メソッドに加えて、SQLJ は SQLJ アプリケーションで呼び出すことのできる以下のメソッドを生成します。

close

形式:

```
public abstract void close() throws SQLException
```

イテレーターが使用するデータベース・リソースを解放します。

isClosed

形式:

```
public abstract boolean isClosed() throws SQLException
```

close メソッドが呼び出されている場合に、値 true を戻します。

next

形式:

```
public abstract boolean next() throws SQLException
```

イテレーターを次の行に進めます。 next メソッドのインスタンスが初めて呼び出される前に、イテレーターは結果表の最初の行の前に置かれます。next は、次の行が使用可能なときは true を戻し、すべての行が検索済みの場合は値 false を戻します。

sqlj.runtime.PositionedIterator インターフェース:

sqlj.runtime.PositionedIterator は、位置指定イテレーターを宣言するときに SQLJ がインプリメントするインターフェースです。位置指定イテレーターのインスタンスを宣言して作成した後に、以下のメソッドを使用できます。

endFetch

形式:

```
public abstract boolean endFetch() throws SQLException
```

イテレーターが行に置かれていないときに、値 true を戻します。

sqlj.runtime.ResultSetIterator インターフェース:

sqlj.runtime.ResultSetIterator は、イテレーターを宣言するときに SQLJ がインプリメントするインターフェースです。イテレーターを宣言して作成した後に、以下のメソッドを使用できます。

clearWarnings

形式:

```
public abstract void clearWarnings() throws SQLException
```

新規の警告がこのイテレーターに対して報告されるまで、NULL を戻します。

close

形式:

```
public abstract void close() throws SQLException
```

イテレーターが使用するデータベース・リソースを解放します。

getResultSet

形式:

```
public abstract ResultSet getResultSet() throws SQLException
```

SQLJ イテレーターを JDBC 結果セットで表したものを戻します。

getWarnings

形式:

```
public abstract SQLWarning getWarnings() throws SQLException
```

呼び出しによりこのイテレーターについて報告された、最初の警告を戻します。後続のイテレーター警告は、この SQLWarning にチェーニングされます。警告のチェーンは、新規の行の読み取りごとにクリアされます。

isClosed

形式:

```
public abstract boolean isClosed() throws SQLException
```

close メソッドが呼び出されている場合に、値 true を戻します。

next

形式:

```
public abstract boolean next() throws SQLException
```

イテレーターを次の行に進めます。 next メソッドのインスタンスが初めて呼び出される前に、イテレーターは結果表の最初の行の前に置かれます。next は、次の行が使用可能なときは true を戻し、すべての行が検索済みの場合は値 false を戻します。

sqlj.runtime.Scrollable インターフェース:

sqlj.runtime.Scrollable は、スクロール可能イテレーターを宣言するときにインプリメントするインターフェースです。 sqlj.runtime.Scrollable メソッドを使用して、結果表内を移動し、結果表内での位置を確認します。

absolute(int)

形式:

```
public abstract boolean absolute (int n) throws SQLException
```

イテレーターを指定行に移動させます。

n>0 の場合、イテレーターを結果表の行 *n* に置きます。 *n*<0 で、*m* が結果表に含まれる行数の場合、イテレーターを結果表の行 *m+n+1* に置きます。

n の絶対値が結果表の行数よりも大きい場合、 *n* が正のときは最後の行の後に置き、 *n* が負のときは最初の行の前に置きます。

Absolute(1) は first() と同じです。Absolute(-1) は last() と同じです。

イテレーターが行の上にある場合、true を戻します。そうでない場合は、false を戻します。

afterLast()

形式:

```
public abstract void afterLast() throws SQLException
```

イテレーターを結果表の最後の行の後に移動させます。

beforeFirst()

形式:

```
public abstract void beforeFirst() throws SQLException
```

イテレーターを結果表の最初の行の前に移動させます。

first()

形式:

```
public abstract boolean first() throws SQLException
```

イテレーターを結果表の最初の行に移動させます。

イテレーターが行の上にある場合、true を戻します。そうでない場合は、false を戻します。

getFetchDirection()

形式:

```
public abstract int getFetchDirection ( ) throws SQLException
```

イテレーターのフェッチ方向を戻します。 可能な値は以下のとおりです。

sqlj.runtime.ResultSetIterator.FETCH_FORWARD
行は最初から最後に向けて、順方向に処理されます。

sqlj.runtime.ResultSetIterator.FETCH_REVERSE
行は最後から最初に向けて、逆方向に処理されます。

sqlj.runtime.ResultSetIterator.FETCH_UNKNOWN
処理の順序は不明です。

isAfterLast()

形式:

```
public abstract boolean isAfterLast() throws SQLException
```

イテレーターが結果表の最後の行の後にある場合、true を戻します。そうでない場合は、false を戻します。

isBeforeFirst()

形式:

```
public abstract boolean isBeforeFirst() throws SQLException
```

イテレーターが結果表の最初の行の前にある場合、true を戻します。そうでない場合は、false を戻します。

isFirst()

形式:

```
public abstract boolean isFirst() throws SQLException
```

イテレーターが結果表の最初の行にある場合、true を戻します。そうでない場合は、false を戻します。

isLast()

形式:

```
public abstract boolean isLast() throws SQLException
```

イテレーターが結果表の最後の行にある場合、true を戻します。そうでない場合は、false を戻します。

last()

形式:

```
public abstract boolean last() throws SQLException
```

イテレーターを結果表の最後の行に移動させます。

イテレーターが行の上にある場合、true を戻します。そうでない場合は、false を戻します。

previous()

形式:

```
public abstract boolean previous() throws SQLException
```

イテレーターを結果表の直前の行に移動させます。

イテレーターが行の上にある場合、true を戻します。そうでない場合は、false を戻します。

relative(int)

形式:

```
public abstract boolean relative(int n) throws SQLException
```

n>0 の場合、イテレーターを現在行から *n* 行後の行に置きます。 *n*<0 の場合、イテレーターを現在行から *n* 行前の行に置きます。 *n*=0 の場合、イテレーターを現在行に置きます。

このメソッドを使用するには、カーソルが結果表の有効な行になければなりません。カーソルが最初の行の前または最後の行の後にある場合は、このメソッドによって SQLException が出されます。

m を結果表の行数、*x* を結果表での現在行番号とします。 *n*>0 で、*x*+*n*>*m* の場合、イテレーターは最後の行の後に置かれます。 *n*<0 で *x*+*n*<1 の場合、イテレーターは最初の行の前に置かれます。

イテレーターが行の上にある場合、true を戻します。そうでない場合は、false を戻します。

setFetchDirection(int)

形式:

```
public abstract void setFetchDirection (int) throws SQLException
```

SQLJ ランタイム環境に対して、このイテレーター・オブジェクトの行が処理される方向を設定します。可能な値は以下のとおりです。

sqlj.runtime.ResultSetIterator.FETCH_FORWARD
行は最初から最後に向けて、順方向に処理されます。

sqlj.runtime.ResultSetIterator.FETCH_REVERSE
行は最後から最初に向けて、逆方向に処理されます。

sqlj.runtime.ResultSetIterator.FETCH_UNKNOWN
処理の順序は不明です。

関連タスク:

- 392 ページの『SQLJ アプリケーションでのバッチ更新の実行』
- 357 ページの『SQLJ を使用したデータ・ソースへの接続』
- 367 ページの『SQLJ アプリケーションでの名前指定イテレーターの使用』

- 370 ページの『SQLJ アプリケーションでの位置指定イテレーターの使用』
- 373 ページの『SQLJ アプリケーションでの位置指定 UPDATE および DELETE 操作の実行』
- 399 ページの『SQLJ アプリケーションでのスクロール可能イテレーターの使用』
- 380 ページの『SQLJ アプリケーションでの SQL 警告の処理』
- 390 ページの『SQLJ での SQL ステートメントの実行の制御』

DB2 Universal JDBC ドライバーの参照情報

以下のセクションには、DB2 Universal JDBC ドライバーに特有の情報が含まれています。

JDBC に対する DB2 Universal JDBC ドライバーの拡張のサマリー

このトピックでは、DB2 Universal JDBC ドライバーに固有の JDBC API について説明します。

このトピックで説明するメソッドのいずれかを使用するには、関連する標準 JDBC クラスのインスタンスを、 DB2 のみのクラスのインスタンスにキャストしなければなりません。以下に例を示します。

```
javax.sql.DataSource ds =
 new com.ibm.db2.jcc.DB2SimpleDataSource();
((com.ibm.db2.jcc.DB2BaseDataSource) ds).setServerName("sysmvs1.stl.ibm.com");
```

DB2ActiveServerList クラス:

com.ibm.db2.jcc.DB2ActiveServerList クラスは、 java.io.Serializable および javax.naming.Referenceable インターフェースをインプリメントします。

DB2ActiveServerList メソッド:

getAlternatePortNumber

形式:

```
public int[] getAlternatePortNumber()
```

代替 DB2 UDB サーバーに関連するポート番号を検索します。

getAlternateServerName

形式:

```
public String[] getAlternateServerName()
```

代替 DB2 UDB サーバーの名前を含む配列を検索します。これらの値は、IP アドレスまたは DNS サーバー名です。

setAlternatePortNumber

形式:

```
public void setAlternatePortNumber(int[] alternatePortNumberList)
```

代替 DB2 UDB サーバーに関連するポート番号を設定します。

setAlternateServerName

形式:

```
public void setAlternateServerName(String[] alternateServer)
```

DB2 UDB サーバーの代替サーバー名を設定します。これらの値は、IP アドレスまたは DNS サーバー名です。

DB2BaseDataSource クラス:

com.ibm.db2.jcc.DB2BaseDataSource クラスは、 javax.sql.DataSource、javax.sql.ConnectionPoolDataSource、および javax.sql.XADataSource のすべての DB2 固有のインプリメンテーションのための抽象データ・ソース親クラスです。

DB2BaseDataSource のプロパティー:

以下のプロパティーは、DB2 Universal JDBC ドライバーに対してのみ定義されます。これらのプロパティーの説明については、 『DB2 Universal JDBC ドライバーのプロパティー』を参照してください。

これらの各プロパティーには、プロパティーの値を設定する setXXX メソッド、および値を検索する getXXX メソッドがあります。 setXXX メソッドの形式は以下のとおりです。

```
void setProperty-name(data-type property-value)
```

getXXX メソッドの形式は以下のとおりです。

```
data-type getProperty-name()
```

Property-name は、先頭文字が大文字になっている非修飾プロパティー名です。

表 74 では、 DB2 Universal JDBC ドライバーのプロパティーとそれらのデータ・タイプをリストしています。

表 74. DB2 Universal JDBC ドライバーのプロパティーとそれらのデータ・タイプ

プロパティー名	データ・タイプ
com.ibm.db2.jcc.DB2BaseDataSource.activeServerListJNDIName	String
com.ibm.db2.jcc.DB2BaseDataSource.clientAccountingInformation	String
com.ibm.db2.jcc.DB2BaseDataSource.clientApplicationInformation	String
com.ibm.db2.jcc.DB2BaseDataSource.clientUser	String
com.ibm.db2.jcc.DB2BaseDataSource.clientWorkstation	String
com.ibm.db2.jcc.DB2BaseDataSource.cliSchema	String
com.ibm.db2.jcc.DB2BaseDataSource.currentFunctionPath	String
com.ibm.db2.jcc.DB2BaseDataSource.currentLockTimeout	int
com.ibm.db2.jcc.DB2BaseDataSource.currentPackagePath	String
com.ibm.db2.jcc.DB2BaseDataSource.cursorSensitivity	int
com.ibm.db2.jcc.DB2BaseDataSource.currentSchema	String
com.ibm.db2.jcc.DB2BaseDataSource.currentSQLID	String

表 74. DB2 Universal JDBC ドライバーのプロパティーとそれらのデータ・タイプ (続き)

プロパティー名	データ・タイプ
com.ibm.db2.jcc.DB2BaseDataSource.currentSQLID	String
com.ibm.db2.jcc.DB2BaseDataSource.databaseName	String
com.ibm.db2.jcc.DB2BaseDataSource.deferPrepares	boolean
com.ibm.db2.jcc.DB2BaseDataSource.description	String
com.ibm.db2.jcc.DB2BaseDataSource.driverType	int
com.ibm.db2.jcc.DB2BaseDataSource.fullyMaterializeLobData	boolean
com.ibm.db2.jcc.DB2BaseDataSource.gssCredential	オブジェクト
com.ibm.db2.jcc.DB2BaseDataSource.jdbcCollection	String
com.ibm.db2.jcc.DB2BaseDataSource.keepDynamic	int
com.ibm.db2.jcc.DB2BaseDataSource.kerberosServerPrincipal	String
com.ibm.db2.jcc.DB2BaseDataSource.logWriter	PrintWriter
com.ibm.db2.jcc.DB2BaseDataSource.portNumber	int
com.ibm.db2.jcc.DB2BaseDataSource.resultSetHoldability	int
com.ibm.db2.jcc.DB2BaseDataSource.securityMechanism	int
com.ibm.db2.jcc.DB2BaseDataSource.serverName	String
com.ibm.db2.jcc.DB2BaseDataSource.readOnly	boolean
com.ibm.db2.jcc.DB2BaseDataSource.traceFile	String
com.ibm.db2.jcc.DB2BaseDataSource.traceLevel	int
com.ibm.db2.jcc.DB2BaseDataSource.user	String

DB2BaseDataSource メソッド:

DB2BaseDataSource プロパティーの `getXXX` および `setXXX` メソッドに加えて、以下のメソッドが DB2 Universal JDBC ドライバーのためだけに定義されています。

getReference

形式:

```
public javax.naming.Reference getReference()
  throws javax.naming.NamingException
```

`DataSource` オブジェクトの `Reference` を検索します。 `Reference` については、以下の URL にある JNDI 文書から `javax.naming.Referenceable` についての説明を参照してください。

```
http://java.sun.com/products/jndi/docs.html
```

DB2Connection インターフェース:

`com.ibm.db2.jcc.DB2Connection` インターフェースは `java.sql.Connection` を拡張するものです。

DB2Connection メソッド:

以下のメソッドは、DB2 Universal JDBC ドライバーに対してのみ定義されます。

getDB2ClientAccountingInformation

形式:

```
public String getDB2ClientAccountingInformation()
    throws SQLException
```

現在のクライアントのアカウンティング情報を戻します。

getDB2ClientApplicationInformation

形式:

```
public String getDB2ClientApplicationInformation()
    throws SQLException
```

現在のクライアントのアプリケーション情報を戻します。

getDB2ClientUser

形式:

```
public String getDB2ClientUser()
    throws SQLException
```

接続のための現在のクライアントのユーザー名を戻します。この名前は、JDBC 接続のユーザー値ではありません。

getDB2ClientWorkstation

形式:

```
public String getDB2ClientWorkstation()
    throws SQLException
```

現在のクライアントの現在のクライアント・ワークステーション名を戻します。

getDB2CurrentPackagePath

形式:

```
public String getDB2CurrentPackagePath()
    throws SQLException
```

DB2 Universal JDBC ドライバー・パッケージが検索される DB2 パッケージ・コレクションのリストを戻します。

getDB2CurrentPackageSet

形式:

```
public String getDB2CurrentPackageSet()
    throws SQLException
```

接続のためのコレクション ID を戻します。

getDB2SystemMonitor

形式:

```
public abstract DB2SystemMonitor getDB2SystemMonitor()
    throws SQLException
```

接続のシステム・モニター・オブジェクトを戻します。 DB2 Universal JDBC ドライバー接続ごとに 1 つのシステム・モニターを使用できます。詳しくは、466 ページの『DB2SystemMonitor インターフェース』を参照してください。

getJccLogWriter

形式:

```
public PrintWriter getJccLogWriter()
    throws SQLException
```

DB2 Universal JDBC ドライバー・トレースの現在のトレースの宛先を戻します。

setDB2ClientAccountingInformation

形式:

```
public void setDB2ClientAccountingInformation(String info)
    throws SQLException
```

接続のアカウンティング情報を指定します。この情報は、クライアントのアカウンティングを目的としています。この値は、接続中に変更することもできます。

パラメーターの記述:

info
ユーザー指定のアカウンティング情報。最大長は、サーバーによって異なります。 DB2 UDB for Linux、for UNIX、および for Windows サーバーの場合、最大長は 255 バイトです。 Java 空ストリング ("") はこのパラメーター値として有効ですが、 Java `null` 値は無効です。

setDB2ClientApplicationInformation

形式:

```
public void setDB2ClientApplicationInformation(String info)
    throws SQLException
```

接続のアプリケーション情報を指定します。この情報は、クライアントのアカウンティングを目的としています。この値は、接続中に変更することもできます。

パラメーターの記述:

info
ユーザー指定のアプリケーション情報。最大長は、サーバーによって異なります。 DB2 UDB for Linux、for UNIX、および for Windows サーバーの場合、最大長は 255 バイトです。 Java 空ストリング ("") はこのパラメーター値として有効ですが、 Java `null` 値は無効です。

setDB2ClientUser

形式:

```
public void setDB2ClientUser(String user)
    throws SQLException
```

接続のための現行クライアントのユーザー名を指定します。この名前はクライアントのアカウンティングを目的としており、JDBC 接続のユーザー値ではありません。 JDBC 接続のユーザーの場合とは異なり、現在のクライアントのユーザー名は接続時に変更することができます。

パラメーターの記述:

user
現在のクライアントのユーザー ID。最大長は、サーバーによって異なります。 DB2 UDB for Linux、for UNIX、および for Windows サーバーの場合、最大長は 255 バイトです。 Java 空ストリング ("") はこのパラメーター値として有効ですが、 Java `null` 値は無効です。

setDB2ClientWorkstation

形式:

```
public void setDB2ClientWorkstation(String name)
    throws SQLException
```

接続のための現在のクライアントのワークステーション名を指定します。この名前は、クライアントのアカウンティングを目的としています。現在のクライアントのワークステーション名は、接続時に変更することができます。

パラメーターの記述:

name
現在のクライアントのワークステーション名。最大長は、サーバーによって異なります。 DB2 UDB for Linux、for UNIX、および for Windows サーバーの場合、最大長は 255 バイトです。 Java 空ストリング ("") はこのパラメーター値として有効ですが、 Java null 値は無効です。

setDB2CurrentPackagePath

形式:

```
public void setDB2CurrentPackagePath(String packagePath)
    throws SQLException
```

DB2 が DB2 Universal JDBC ドライバー DB2 パッケージを検索するコレクション ID のリストを指定します。

パラメーターの記述:

packagePath
コレクション ID の、コンマで区切られたリスト。

setDB2CurrentPackageSet

形式:

```
public void setDB2CurrentPackageSet(String packageSet)
    throws SQLException
```

接続のためのコレクション ID を指定します。この値を設定すると、接続に使用される DB2 Universal JDBC ドライバー・インスタンスのコレクション ID も設定されます。

パラメーターの記述:

packageSet
接続のためのコレクション ID。 packageSet 値の最大長は 18 バイトです。このメソッドは、 SQL SET CURRENT PACKAGESET ステートメントの実行の代替方法としてプログラム内で呼び出すことができます。

setJccLogWriter

形式:

```
public void setJccLogWriter(PrintWriter logWriter)
    throws SQLException

public void setJccLogWriter(PrintWriter logWriter, int traceLevel)
    throws SQLException
```

DB2 Universal JDBC ドライバー・トレースを有効または無効にします。または、アクティブな接続中にトレースの宛先を変更します。

パラメーターの説明:

logWriter
DB2 Universal JDBC ドライバーがトレース出力を書き込む、タイプ java.io.PrintWriter のオブジェクト。トレースをオフにするには、*logWriter* の値を null に設定します。

traceLevel
収集するトレースのタイプを指定します。有効な値については、 『DB2 Universal JDBC ドライバーのプロパティー』の traceLevel プロパティーの説明を参照してください。

DB2DatabaseMetaData インターフェース:

com.ibm.db2.jcc.DB2DatabaseMetaData インターフェースは、java.sql.DatabaseMetaData インターフェースを拡張します。

DB2DatabaseMetaData メソッド:

以下のメソッドは、DB2 Universal JDBC ドライバーに対してのみ定義されます。

DB2Diagnosable インターフェース:

com.ibm.db2.jcc.DB2Diagnosable インターフェース DB2 SQLException から DB2 診断を取得するための手段を提供します。

DB2Diagnosable メソッド:

以下のメソッドは、DB2 Universal JDBC ドライバーに対してのみ定義されます。

getSqlca
形式:

```
public DB2Sqlca getSqlca()
```

DB2Sqlca オブジェクトを、 DB2 Universal JDBC ドライバーで生成された java.sql.Exception から戻します。

getThrowable
形式:

```
public Throwable getThrowable()
```

java.lang.Throwable オブジェクトを、 DB2 Universal JDBC ドライバーで生成された java.sql.Exception から戻します。

printTrace
形式:

```
static public void printTrace(java.io.PrintWriter printWriter,
  String header)
```

DB2 Universal JDBC ドライバーで java.sql.Exception が出された後に、診断情報をプリントします。

パラメーターの説明:

printWriter
診断情報の宛先

header
出力の先頭にプリントされるユーザー定義の情報。

DB2ExceptionFormatter クラス:

com.ibm.db2.jcc.DB2ExceptionFormatter クラスには、診断情報をストリームにプリントするメソッドが含まれています。

DB2ExceptionFormatter メソッド:

以下のメソッドは、DB2 Universal JDBC ドライバーに対してのみ定義されます。

printTrace
形式:

```
static public void printTrace(java.sql.SQLException sqlException,
  java.io.PrintWriter printWriter, String header)

static public void printTrace(DB2Sqlca sqlca,
  java.io.PrintWriter printWriter, String header)

static public void printTrace(java.lang.Throwable throwable,
  java.io.PrintWriter printWriter, String header)
```

例外が出された後に、診断情報をプリントします。

パラメーターの説明:

sqlException|sqlca|throwable
直前の JDBC または Java 操作の際に出された例外。

printWriter
診断情報の宛先

header
出力の先頭にプリントされるユーザー定義の情報。

DB2RowID インターフェース:

com.ibm.db2.jcc.DB2RowID クラスは、 DB2 ROWID データ・タイプとともに使用する Java オブジェクトを宣言するのに使用します。

DB2RowID メソッド:

以下のメソッドは、DB2 Universal JDBC ドライバーに対してのみ定義されます。

getBytes
形式:

```
public byte[] getBytes()
```

com.ibm.jcc.DB2RowID オブジェクトをバイトに変換します。

DB2SimpleDataSource クラス:

com.ibm.db2.jcc.DB2SimpleDataSource クラスは、 DataBaseDataSource クラスを拡張します。 DataBaseDataSource オブジェクトは、接続プールや分散トランザクションをサポートしません。これには、DB2BaseDataSource クラスに含まれるすべてのプロパティーとメソッドが含まれています。さらに、DB2SimpleDataSource には以下の DB2 Universal JDBC ドライバーのみのプロパティーが含まれます。

DB2SimpleDataSource のプロパティー:

以下のプロパティーは、DB2 Universal JDBC ドライバーに対してのみ定義されます。このプロパティーについては、 『DB2 Universal JDBC ドライバーのプロパティー』を参照してください。

String com.ibm.db2.jcc.DB2SimpleDataSource.password

DB2SimpleDataSource メソッド:

以下のメソッドは、DB2 Universal JDBC ドライバーに対してのみ定義されます。

setPassword

形式:

```
public void setPassword(String password)
```

DB2SimpleDataSource オブジェクトのパスワードを設定します。対応する getPassword メソッドはありません。したがって、暗号化解除のためにパスワードを検索する方法がないので、パスワードを暗号化することはできません。

DB2Sqlca クラス:

com.ibm.db2.jcc.DB2Sqlca クラスは、DB2 SQLCA をカプセル化したものです。

DB2Sqlca メソッド:

以下のメソッドは、DB2 Universal JDBC ドライバーに対してのみ定義されます。

getMessage

形式:

```
public abstract String getMessage()
```

エラー・メッセージ・テキストを戻します。

getSqlCode

形式:

```
public abstract int getSqlCode()
```

SQL エラー・コード値を戻します。

getSqlErrd

形式:

```
public abstract int[] getSqlErrd()
```

それぞれのエレメントに SQLCA SQLERRD を含む配列を戻します。

getSqlErrmc

形式:

```
public abstract String getSqlErrmc()
```

スペースで区切られた複数の SQLCA SQLERRMC 値を含むストリングを戻します。

getSqlErrmcTokens

形式:

```
public abstract String[] getSqlErrmcTokens()
```

それぞれのエレメントに SQLCA SQLERRMC トークンを含む配列を戻します。

getSqlErrd

形式:

```
public abstract int[] getSqlErrd()
```

それぞれのエレメントに SQLCA SQLERRP 値を含む配列を戻します。

getSqlErrp

形式:

```
public abstract String getSqlErrp()
```

SQLCA SQLERRP 値を戻します。

getSqlState

形式:

```
public abstract String getSqlState()
```

SQLCA SQLSTATE 値を戻します。

getSqlWarn

形式:

```
public abstract char[] getSqlWarn()
```

それぞれのエレメントに SQLCA SQLWARN 値を含む配列を戻します。

DB2SystemMonitor インターフェース:

com.ibm.db2.jcc.DB2SystemMonitor インターフェースは、接続に関するシステム・モニター・データを収集するために使用します。接続ごとに 1 つの DB2SystemMonitor インスタンスを含めることができます。

DB2SystemMonitor フィールド:

以下のフィールドは、DB2 Universal JDBC ドライバーに対してのみ定義されます。

public final static int RESET_TIMES
public final static int ACCUMULATE_TIMES

これらの値は、 DB2SystemMonitor.start メソッドの引き数です。 RESET_TIMES はモニターを開始する前に時間カウンターをゼロに設定します。 ACCUMULATE_TIMES は時間カウンターをゼロに設定しません。

DB2SystemMonitor メソッド:

以下のメソッドは、DB2 Universal JDBC ドライバーに対してのみ定義されます。

enable

形式:

```
public void enable(boolean on)
  throws java.sql.SQLException
```

接続に関連するシステム・モニターを有効にします。このメソッドをモニター中に呼び出すことはできません。 enable が呼び出されると、すべての時間がリセットされます。

getApplicationTimeMillis

形式:

```
public long getApplicationTimeMillis()
  throws java.sql.SQLException
```

アプリケーション、JDBC ドライバー、ネットワーク I/O、および DB2 サーバーの経過時間の合計を戻します。時間の単位はミリ秒です。

モニター対象の経過時間のインターバルとは、 JDBC ドライバー処理における以下の時点の間の期間 (ミリ秒) のことです。

インターバルの始点
start が呼び出される時。

インターバルの終点
stop が呼び出される時。

システム・モニターが無効の場合、getApplicationTimeMillis は 0 を戻します。最初に stop メソッドを呼び出さずに、このメソッドを呼び出すと、SQLException になります。

getCoreDriverTimeMicros

形式:

```
public long getCoreDriverTimeMicros()
  throws java.sql.SQLException
```

システム・モニターが有効な間に収集された、モニター対象の API の経過時間の合計を戻します。時間の単位はマイクロ秒です。

モニター対象の API とは、処理時間が収集される JDBC ドライバー・メソッドのことです。一般的に、経過時間がモニターされるのは、ネットワーク I/O または DB2 サーバー対話が行われる可能性のある API に限ります。たとえば、 PreparedStatement.setXXX メソッドおよび ResultSet.getXXX メソッドはモニターされません。

モニター対象の API の経過時間には、メソッド呼び出しの際にドライバーでかかった時間の合計が含まれます。この時間には、すべてのネットワーク I/O 時間および DB2 サーバーの経過時間が含まれます。

モニター対象の API の経過時間のインターバルとは、 JDBC ドライバー処理における以下の時点の間の期間 (マイクロ秒) のことです。

インターバルの始点
アプリケーションによってモニター対象の API が呼び出された時。

インターバルの終点
モニター対象の API が制御をアプリケーションに戻す直前。

システム・モニターが無効な場合、getCoreDriverTimeMicros は 0 を戻します。最初に stop メソッドを呼び出さずにこのメソッドを呼び出すか、または、基礎となる JVM がマイクロ秒でのレポート時間をサポートしていない場合に、このメソッドを呼び出すと、SQLException になります。

getNetworkIOTimeMicros

形式:

```
public long getNetworkIOTimeMicros()
  throws java.sql.SQLException
```

システム・モニターが有効な間に収集された、ネットワーク I/O の経過時間の合計を戻します。時間の単位はマイクロ秒です。

ネットワーク I/O の経過時間には、ネットワーク I/O ストリームへの DRDA データの書き込み、およびネットワーク I/O ストリームからの DRDA データの読み取りの時間が含まれます。ネットワーク I/O の経過時間のインターバルとは、 JDBC ドライバーで以下の操作を実行する際の時間インターバルのことです。

- TCP/IP コマンドを発行して、DRDA メッセージを DB2 サーバーに送信する。この時間インターバルとは、ネットワーク I/O ストリームへの書き込みおよびフラッシュが実行される時点の直前と直後の間の期間 (マイクロ秒) のことです。
- TCP/IP コマンドを発行して、DB2 サーバーから DRDA 応答メッセージを受信する。この時間インターバルとは、ネットワーク I/O ストリームでの読み取りが実行される時点の直前と直後の間の期間 (マイクロ秒) のことです。

ネットワーク I/O 時間インターバルは、コミットおよびロールバックでのメッセージの送信など、すべての送信および受信操作に関してキャプチャーされます。

優先順位の低い SQL 要求により DB2 サーバーでの CPU のディスパッチングが遅れると、ネットワーク I/O の待機に費やした時間が影響を受ける場合があります。ネットワーク I/O の時間インターバルには、DB2 サーバーの経過時間も含まれます。

システム・モニターが無効な場合、getNetworkIOTimeMicros は 0 を戻します。最初に stop メソッドを呼び出さずにこのメソッドを呼び出すか、または、基礎となる JVM がマイクロ秒でのレポート時間をサポートしていない場合に、このメソッドを呼び出すと、SQLException になります。

getServerTimeMicros

形式:

```
public long getServerTimeMicros()
  throws java.sql.SQLException
```

システム・モニターが有効な間に収集された、報告されたすべての DB2 サーバーの経過時間の合計を戻します。時間の単位はマイクロ秒です。

DB2 サーバーは、以下の条件で経過時間を報告します。

- 経過時間データのクライアントへの戻しをサーバーがサポートしている。
- モニターできる操作をサーバーが実行している。たとえば、 DB2 サーバーの経過時間はコミットまたはロールバックでは戻されません。

DB2 サーバーの経過時間は、要求データ・ストリームの構文解析、コマンドの処理、サーバーでの応答データ・ストリームの生成のための経過時間として定義されます。データ・ストリームを受信または送信するためのネットワーク時間は含まれません。

DB2 サーバーの経過時間のインターバルとは、サーバー処理における以下の時点の間の期間 (マイクロ秒) のことです。

インターバルの始点
JDBC ドライバーから受信される TCP/IP メッセージを処理するために、オペレーティング・システムが DB2 をディスパッチする時。

インターバルの終点
応答メッセージをクライアントに戻すための TCP/IP コマンドを発行する準備を DB2 が整えた時。

システム・モニターが無効な場合、getServerTimeMicros は 0 を戻します。最初に stop メソッドを呼び出さずに、このメソッドを呼び出すと、SQLException になります。

start

形式:

```
public void start (int lapMode)
  throws java.sql.SQLException
```

システム・モニターが有効な場合、 start は接続のためのシステム・モニター・データの収集を開始します。 *lapMode* の有効な値は RESET_TIMES または ACCUMULATE_TIMES です。

システム・モニターを無効にしてこのメソッドを呼び出すと、何も行われません。 stop 呼び出しで中断させずにこのメソッドを複数回呼び出すと、SQLException になります。

stop

形式:

```
public void stop()
  throws java.sql.SQLException
```

システム・モニターが有効な場合、 stop は接続のためのシステム・モニター・データの収集を終了します。モニターが停止された後で、 DB2SystemMonitor の getXXX メソッドにより、モニターされる時間を取得できます。

システム・モニターを無効にしてこのメソッドを呼び出すと、何も行われません。最初に start を呼び出さずにこのメソッドを呼び出すか、 start で中断させずにこのメソッドを複数回呼び出すと、SQLException になります。

関連タスク:

- 303 ページの『DataSource インターフェースを使用したデータ・ソースへの接続』
- 300 ページの『DriverManager インターフェースと DB2 Universal JDBC ドライバーを使用したデータ・ソースへの接続』
- 315 ページの『DB2 Universal JDBC ドライバー使用時の SQLException の処理』

関連資料:

- 「*SQL リファレンス 第 1 巻*」の『SQLCA (SQL 連絡域)』
- 408 ページの『DB2 Universal JDBC ドライバーのプロパティー』

DB2 Universal JDBC ドライバーと他の DB2 JDBC ドライバーの JDBC との相違点

DB2 Universal JDBC ドライバーは、 Linux、UNIX、および Windows 用 DB2 JDBC Type 2 ドライバー (DB2 JDBC Type 2 ドライバー) と、以下の点で異なります。

サポートされるメソッド:

DB2 Universal JDBC ドライバーは、他のドライバーがサポートしていない多数の JDBC メソッドをサポートしていますが、一方では他のドライバーがサポートしていてもサポートしていないメソッドがいくつかあります。詳細については、『JDBC API 用ドライバー・サポートの比較』を参照してください。

スクロール可能および更新可能 ResultSet のサポート:

DB2 Universal JDBC ドライバーは、スクロール可能および更新可能 `ResultSet` をサポートしています。

DB2 JDBC Type 2 ドライバーは、スクロール可能 `ResultSet` をサポートしていますが、更新可能 `ResultSet` はサポートしていません。

URL 構文の相違点:

`DriverManager.getConnection` メソッドの *url* パラメーターの構文は、ドライバーごとに異なります。詳細については、以下のトピックを参照してください。

- 『DriverManager インターフェースと DB2 Universal JDBC ドライバーを使用したデータ・ソースへの接続』
- 『DriverManager インターフェースと DB2 JDBC Type 2 ドライバーを使用したデータ・ソースへの接続』

ドライバー・エラーで戻されるエラー・コードおよび SQLSTATE との相違点:

DB2 Universal JDBC ドライバーは、他のドライバーとは異なり、内部エラーの場合に既存の SQLCODE または SQLSTATE を使用しません。 『DB2 Universal JDBC ドライバーが発行するエラー・コード』および DB2 Universal JDBC ドライバーが発行する SQLSTATE を参照してください。

z/OS 用 JDBC/SQLJ ドライバーは、内部エラーが発生すると ODBC SQLSTATE を戻します。

戻されるエラー・メッセージ・テキストの数:

DB2 Universal JDBC ドライバーでは、 retrieveMessagesFromServerOnGetMessage プロパティーを true に設定していなければ、 SQLException.getMessage() を実行したときに形式設定メッセージ・テキストは戻されません。

DB2 JDBC Type 2 ドライバーでは、 SQLException.getMessage() を実行すると、形式設定メッセージ・テキストが戻されます。

セキュリティー・メカニズム:

JDBC ドライバーのセキュリティー・メカニズムはさまざまです。

DB2 Universal JDBC ドライバーのセキュリティー・メカニズムについては、『DB2 Universal JDBC ドライバー使用時のセキュリティー』を参照してください。

DB2 JDBC Type 2 ドライバーのセキュリティー・メカニズムについては、 『DB2 JDBC Type 2 ドライバー使用時のセキュリティー』を参照してください。

読み取り専用接続のサポート:

DB2 Universal JDBC ドライバーでは、 Connection または DataSource オブジェクトの readOnly プロパティーで接続を読み取り専用にすることができます。

DB2 JDBC Type 2 ドライバーでは、接続を読み取り専用にするかどうかを Connection.setReadOnly 値を使用して決定します。ただし、 Connection.setReadOnly(true) に設定しても、接続が読み取り専用になることが保証されるわけではありません。

BIT DATA 列に対する ResultSet.getString から戻される結果:

DB2 Universal JDBC ドライバーは、 CHAR FOR BIT DATA または VARCHAR FOR BIT DATA 列に対する ResultSet.getString 呼び出しのデータを、小文字 16 進数ストリングで戻します。

DB2 JDBC Type 2 ドライバーは、データを大文字 16 進数ストリングで戻します。

どの行にも影響を与えない executeUpdate 呼び出しの結果:

executeUpdate 呼び出しがどの行にも影響を与えないときは、 DB2 Universal JDBC ドライバーは SQLWarning を生成します。

DB2 JDBC Type 2 ドライバーの場合は SQLWarning を生成しないで戻します。

TIMESTAMP 列に対する getDate または getTime 呼び出しの結果:

DB2 Universal JDBC ドライバーは、 TIMESTAMP 列に対して getDate または getTime 呼び出しが行われたときに SQLWarning を生成しません。

DB2 JDBC Type 2 ドライバーは、 TIMESTAMP 列に対して getDate または getTime 呼び出しが行われると SQLWarning を生成します。

長さが不一致の PreparedStatement.setXXXStream に対して例外が出されるタイミング:

PreparedStatement.setBinaryStream、 PreparedStatement.setCharacterStream、または PreparedStatement.setUnicodeStream メソッドを使用するときは、 *length* パラメーターの値が入力ストリーム内のバイト数に一致していなければなりません。

バイト数が一致していない場合、 DB2 Universal JDBC ドライバーは後続の PreparedStatement.executeUpdate メソッドが実行されるまで例外を出しません。そのため、DB2 Universal JDBC ドライバーの場合は、長さが一致していなくても、一部のデータがサーバーに送信される可能性があります。そのデータは、サーバーによって切り捨てまたは埋め込みが行われます。呼び出し側アプリケーションは、ロールバック要求を発行して、切り捨てまたは埋め込みが行われたデータが含まれるデータベース更新を取り消す必要があります。

DB2 JDBC Type 2 ドライバーは、PreparedStatement.setBinaryStream、PreparedStatement.setCharacterStream、または PreparedStatement.setUnicodeStream メソッドが実行された後に例外を出します。

PreparedStatement.setXXXStream のデフォルトのマッピング:

DB2 Universal JDBC ドライバーでは、PreparedStatement.setBinaryStream、PreparedStatement.setCharacterStream、または PreparedStatement.setUnicodeStream メソッドを使用したときに、ターゲット列のデータ・タイプに関する情報がない場合は、入力データは BLOB または CLOB データ・タイプにマップされます。

DB2 JDBC Type 2 ドライバーの場合は、入力データは VARCHAR FOR BIT DATA または VARCHAR データ・タイプにマップされます。

文字変換の実行方法:

クライアントとサーバー間で文字データが転送されるとき、そのデータは受信側が処理できる形式に変換されなければなりません。

DB2 Universal JDBC ドライバーの場合、データベース・サーバーからクライアントに送信される文字データは、 Java の標準装備文字コンバーターを使用して変換されます。 DB2 Universal JDBC ドライバーがサポートする変換は、その基礎になる JRE 実装でサポートされているものに限定されます。

DB2 Universal JDBC ドライバーを使用するクライアントは、データを Unicode でデータベース・サーバーに送信します。

DB2 JDBC Type 2 ドライバーの場合は、文字変換は DB2 サーバーが変換をサポートしている場合に行うことができます。

入力パラメーターの暗黙的または明示的なデータ・タイプ変換:

PreparedStatement.setXXX メソッドを実行し、 setXXX メソッドの結果のデータ・タイプが、パラメーター値が割り当てられた表列のデータ・タイプに一致しない場合は、データ・タイプの変換が行われない限りドライバーはエラーを戻します。

DB2 Universal JDBC ドライバーでは、ターゲットのデータ・タイプが分かっていて、 deferPrepares 接続プロパティーが false に適切に設定されていれば、正しい SQL データ・タイプへの変換が暗黙的に行われます。このときの暗黙値は、setXXX 呼び出しでの明示値をオーバーライドします。 deferPrepares 接続プロパティーが true に設定されている場合は、 PreparedStatement.setObject メソッドを使用して、パラメーターを正しい SQL データ・タイプに変換する必要があります。

DB2 JDBC Type 2 ドライバーの場合、パラメーターのデータ・タイプがそのデフォルトの SQL データ・タイプに一致しないときは、PreparedStatement.setObject メソッドを使用して、パラメーターを正しい SQL データ・タイプに変換する必要があります。

入力パラメーターの String から BINARY への変換のサポート:

DB2 Universal JDBC ドライバーは、*x* がタイプ String のオブジェクトの場合の、以下の形式の PreparedStatement.setObject 呼び出しはサポートしません。

```
setObject(parameterIndex, x, java.sqlTypes.BINARY)
```

DB2 JDBC Type 2 ドライバーは、このタイプの呼び出しをサポートしています。ドライバーは *x* の値を 16 進数ストリングとして解釈します。

10 進位取りが不一致の場合の PreparedStatement.setObject の結果:

DB2 Universal JDBC ドライバーでは、10 進入力パラメーターで PreparedStatement.setObject を呼び出したときに、入力パラメーターの位取りがターゲット列の位取りより大きい場合、ドライバーは入力値の末尾桁を切り捨ててからその値を列に割り当てます。

DB2 JDBC Type 2 ドライバーは、入力値の末尾桁を丸めた後にその値を列に割り当てます。

入力パラメーターの java.lang.Character データ・タイプからの変換のサポート:

以下の形式の PreparedStatement.setObject の場合、DB2 Universal JDBC ドライバーは、 *x* を JDBC データ・タイプに変換するときに、以下の Java オブジェクトから JDBC データ・タイプへの標準データ・タイプのマッピングをサポートしています。

```
setObject(parameterIndex, x)
```

DB2 JDBC Type 2 ドライバーは、 java.lang.Character から CHAR への *x* の非標準マッピングをサポートしています。

文字列に対する ResultSet.getBinaryStream のサポート:

DB2 Universal JDBC ドライバーが文字列を表す引き数を持つ ResultSet.getBinaryStream をサポートするのは、その列が FOR BIT DATA 属性を持つ場合のみです。

DB2 JDBC Type 2 ドライバーの場合は、 ResultSet.getBinaryStream の引き数が文字列のときに、その列が FOR BIT DATA 属性を持っている必要はありません。

バイナリー列に対して ResultSet.getBinaryStream から戻されるデータ:

DB2 Universal JDBC ドライバーでは、バイナリー列に対して ResultSet.getBinaryStream を実行した場合、戻されるデータは小文字の 16 進数字のペアの形式になります。

DB2 JDBC Type 2 ドライバーでは、バイナリー列に対して ResultSet.getBinaryStream を実行した場合、戻されるデータは大文字の 16 進数字のペアの形式になります。

Boolean 入力タイプと CHAR ターゲット・タイプで setObject を使用したときの結果:

DB2 Universal JDBC ドライバーでは、PreparedStatement.setObject(*parameterIndex*,*x*,CHAR) を実行するときに *x* が Boolean の場合、表列には値「0」または「1」が入ります。

DB2 JDBC Type 2 ドライバーでは、ストリング「false」または「true」が表列に挿入されます。表列の長さは少なくとも 5 でなければなりません。

getBoolean を使用して CHAR 列の値を検索したときの結果:

DB2 Universal JDBC ドライバーでは、 ResultSet.getBoolean または CallableStatement.getBoolean を実行して CHAR 列から Boolean 値を取り出す場合、その列に値「false」または「0」が含まれていると、値 false が戻されます。列にその他の値が含まれているときは、true が戻されます。

DB2 JDBC Type 2 ドライバーでは、 ResultSet.getBoolean または CallableStatement.getBoolean を実行して CHAR 列から Boolean 値を取り出す場合、その列に値「true」または「1」が含まれていると、値 true が戻されます。列にその他の値が含まれているときは、false が戻されます。

クローズされたカーソルに対して ResultSet.next() を実行したときの結果:

DB2 Universal JDBC ドライバーでは、クローズされたカーソルに対して ResultSet.next() を実行すると、 SQLException が出されます。これは JDBC の規格に合致しています。

DB2 JDBC Type 2 ドライバーでは、クローズされたカーソルに対して ResultSet.next() を実行すると、値 false が戻されて例外が出されます。

DatabaseMetaData 呼び出しで NULL 引き数を指定したときの結果:

DB2 Universal JDBC ドライバーでは、 DatabaseMetaData メソッド呼び出しで引き数に null を指定できるのは、 JDBC 仕様で null が許可されているところだけです。それ以外のところでは、例外が出されます。

DB2 JDBC Type 2 ドライバーでは、 null はその引き数を検索の絞り込みに使用しないことを意味します。

DATALINK のサポート:

DB2 Universal JDBC ドライバーは DATALINK SQL タイプをサポートしていません。

DB2 JDBC Type 2 ドライバーは、以下の形式のメソッド呼び出しで DATALINK タイプをサポートしています。

- PreparedStatement.setObject(*parameterIndex*, *x*, DB2Constants.DATALINK)
- PreparedStatement.setObject(*parameterIndex*, *x*, java.sql.Types.DATALINK) (Java 1.4 以降)
- PreparedStatement.setURL(*parameterIndex*, java.net.URL)
- PreparedStatement.setObject(*parameterIndex*, java.net.URL)
- PreparedStatement.setObject(*parameterIndex*, java.net.URL, java.sql.Types.DATALINK) (Java 1.4 以降)
- ResultSet.getString (DATALINK 列)
- ResultSet.getURL (DATALINK 列)

メソッドの引き数の大文字への変換:

DB2 Universal JDBC ドライバーは、メソッド呼び出しのどの引き数も大文字に変換しません。

DB2 JDBC Type 2 ドライバーは、Statement.setCursorName 呼び出しの引き数を大文字に変換します。カーソル名が大文字に変換されないようにするには、カーソル名の先頭と末尾に文字 ¥" を付けます。たとえば、以下のようにします。

```
Statement.setCursorName("¥"mycursor¥"");
```

タイム・スタンプのエスケープ文節のサポート:

DB2 Universal JDBC ドライバーは、TIME のエスケープ文節として、以下の標準形式をサポートしています。

```
{t 'hh:mm:ss'}
```

DB2 JDBC Type 2 ドライバーは、標準形式に加えて、以下の形式の TIME エスケープ文節をサポートしています。

```
{ts 'hh:mm:ss'}
```

CALL ステートメントのパラメーターとしてのリテラルまたは式の使用:

DB2 Universal JDBC ドライバーは、 CALL ステートメントのパラメーターとしてパラメーター・マーカーのみの使用をサポートしています。

DB2 JDBC Type 2 ドライバーは、 CALL ステートメントのパラメーターとしてリテラルとパラメーターの使用をサポートしています。

ステートメント・バッチへの CALL ステートメントの組み込み:

DB2 Universal JDBC ドライバー は、ステートメント・バッチでの CALL ステートメントをサポートしています。

DB2 UDB Type 2 ドライバーは、ステートメント・バッチでの CALL ステートメントをサポートしていません。

SQL ステートメントのテキストからの余分な文字の除去:

DB2 Universal JDBC ドライバーは、SQL ステートメントのテキストをデータベース・サーバーに渡す前に、そこからスペース、タブ、改行文字などの空白文字を除去しません。

DB2 UDB Type 2 ドライバーは、SQL ステートメントのテキストから空白文字を除去してから、そのテキストをデータベース・サーバーに渡します。

PreparedStatement.executeBatch の実行結果:

PreparedStatement.executeBatch ステートメントを DB2 Universal JDBC ドライバーの使用時に実行すると、ドライバーは更新のカウントを `int` の配列で戻します。配列の各エレメントには、バッチ内のステートメントによって更新された行数が含まれます。

PreparedStatement.executeBatch ステートメントを DB2 UDB Type 2 ドライバーの使用時に実行すると、ドライバーは更新カウントを判別できないために、更新カウントごとに -3 を戻します。

コンパウンド SQL のサポート:

DB2 Universal JDBC ドライバーは、コンパウンド SQL ブロックをサポートしていません。

DB2 UDB Type 2 ドライバーは、コンパウンド SQL ブロックをサポートしています。 `PreparedStatement.executeUpdate` または `Statement.executeUpdate` 呼び出しを使用して、コンパウンド SQL ブロックを実行することができます。

バッチ更新でパラメーターを設定しない場合の結果:

DB2 Universal JDBC ドライバーは、パラメーターが設定されていない場合は、`PreparedStatement.addBatch` 呼び出しの後に例外を出します。

DB2 UDB Type 2 ドライバーは、バッチ内のどのステートメントにもパラメーターが設定されていない場合は、 `PreparedStatement.executeBatch` 呼び出しの後に例外を出します。

アンカタログ・ストアード・プロシージャーを呼び出す機能:

DB2 Universal JDBC ドライバーでは、DB2 カタログで定義されていないストアード・プロシージャーを呼び出すことはできません。

DB2 UDB Type 2 ドライバーでは、 DB2 カタログで定義されていないストアード・プロシージャーを呼び出すことができます。

関連概念:

- 488 ページの『DB2 Universal JDBC ドライバー使用時のセキュリティー』
- 321 ページの『DB2 Universal JDBC ドライバーがある JDBC アプリケーションでの LOB』
- 487 ページの『DB2 JDBC Type 2 ドライバー使用時のセキュリティー』

関連タスク:

- 337 ページの『JDBC アプリケーションでのバッチ更新の作成』

- 313 ページの『CallableStatement メソッドを使用したストアード・プロシージャーの呼び出し』
- 303 ページの『DataSource インターフェースを使用したデータ・ソースへの接続』
- 300 ページの『DriverManager インターフェースと DB2 Universal JDBC ドライバーを使用したデータ・ソースへの接続』
- 315 ページの『DB2 Universal JDBC ドライバー使用時の SQLException の処理』
- 310 ページの『DB2 表のデータを更新するための PreparedStatement.executeUpdate メソッドの使用』
- 342 ページの『JDBC アプリケーションでの ResultSet の更新可能性、スクロール可能性、および保持可能性の指定』
- 308 ページの『DB2 オブジェクトを作成および変更するための Statement.executeUpdate メソッドの使用』

関連資料:

- 408 ページの『DB2 Universal JDBC ドライバーのプロパティー』
- 479 ページの『DB2 Universal JDBC ドライバーが発行するエラー・コード』
- 479 ページの『DB2 Universal JDBC ドライバーが発行する SQLSTATE』
- 416 ページの『JDBC API 用ドライバー・サポートの比較』
- 403 ページの『Java、JDBC、および SQL のデータ・タイプ』

DB2 Universal JDBC ドライバーと他の DB2 JDBC ドライバーとの間の SQLJ の相違

DB2 Universal JDBC ドライバーの SQLJ サポートは、他の DB2 JDBC ドライバーの SQLJ サポートと以下の領域で異なります。

逐次プロファイルでの相違点:

DB2 JDBC Type 2 ドライバーと DB2 Universal JDBC ドライバーは、 SQLJ 変換プログラムおよび SQLJ カスタマイザー・ユーティリティーを実行する際に、異なるバイナリー・コードを生成します。そのため、 DB2 JDBC Type 2 ドライバー `sqlj` および `db2profc` ユーティリティーを使用して、変換およびカスタマイズした SQLJ アプリケーションは、 DB2 Universal JDBC ドライバーの使用時に実行されません。 ***DB2 Universal JDBC ドライバーの使用時にそれらの SQLJ アプリケーションを実行するには、その前に DB2 Universal JDBC ドライバー `sqlj` および `db2sqljcustomize` ユーティリティーを使用して、アプリケーションを再変換および再カスタマイズしなければなりません。*** アプリケーションを変更していない場合でも、そうしなければなりません。

SQL VALUES サポート:

DB2 JDBC Type 2 ドライバーは SQLJ ステートメント文節での SQL VALUES ステートメントをサポートしていますが、 DB2 Universal JDBC ドライバーはサポートしていません。そのため、VALUES ステートメントを含む SQLJ ステートメントを変更する必要があります。

例: SQLJ プログラムに以下のステートメントが含まれているとします。

```
#sql [ctxt] hv = {VALUES (MY_ROUTINE(1))};
```

DB2 Universal JDBC ドライバーでは、そのステートメントを以下のように変更する必要があります。

```
#sql [ctxt] {SELECT MY_ROUTINE(1) INTO :hv FROM SYSIBM.SYSDUMMY1};
```

コンパウンド SQL ステートメントのサポート:

DB2 JDBC Type 2 ドライバーは SQLJ ステートメント文節でのコンパウンド SQL ステートメントをサポートしていますが、 DB2 Universal JDBC ドライバーはサポートしていません。そのため、BEGIN COMPOUND および END COMPOUND を伴う SQLJ ステートメントを含む、 SQLJ アプリケーションを変更する必要があります。コンパウンド・ステートメントを使用してバッチ更新を行う場合、代わりに、SQLJ バッチ更新プログラミング・インターフェースを使用できます。

接続技法の相違点:

使用可能な接続技法、およびそれらの接続技法に使用されるドライバー名と URL は、ドライバーによって異なります。詳細については、『SQLJ を使用したデータ・ソースへの接続』を参照してください。

スクロール可能および更新可能なイテレーターのサポート:

DB2 Universal JDBC ドライバーのある SQLJ は、スクロール可能および更新可能なイテレーターをサポートします。

Linux、UNIX、および Windows 用 DB2 JDBC Type 2 ドライバー (DB2 JDBC Type 2 ドライバー) は、両方向スクロール・カーソルをサポートしますが、更新可能なイテレーターはサポートしません。

WebSphere Application Server での SQL ステートメントの動的実行:

DB2 Universal JDBC ドライバーでは、5.0.1 よりも前の WebSphere Application Server のバージョンを使用している場合、 SQLJ プログラムをカスタマイズするかどうかに関係なく、 SQLJ プログラム内のすべての SQL ステートメントが動的に実行されます。 For WebSphere Application Server バージョン 5.0.1 以降で、 SQLJ プログラムをカスタマイズする場合、SQL ステートメントは静的に実行されます。

関連概念:

- 488 ページの『DB2 Universal JDBC ドライバー使用時のセキュリティー』
- 321 ページの『DB2 Universal JDBC ドライバーがある JDBC アプリケーションでの LOB』
- 487 ページの『DB2 JDBC Type 2 ドライバー使用時のセキュリティー』

関連タスク:

- 337 ページの『JDBC アプリケーションでのバッチ更新の作成』
- 313 ページの『CallableStatement メソッドを使用したストアード・プロシージャーの呼び出し』
- 303 ページの『DataSource インターフェースを使用したデータ・ソースへの接続』

- 300 ページの『DriverManager インターフェースと DB2 Universal JDBC ドライバーを使用したデータ・ソースへの接続』
- 315 ページの『DB2 Universal JDBC ドライバー使用時の SQLException の処理』
- 310 ページの『DB2 表のデータを更新するための PreparedStatement.executeUpdate メソッドの使用』
- 342 ページの『JDBC アプリケーションでの ResultSet の更新可能性、スクロール可能性、および保持可能性の指定』
- 308 ページの『DB2 オブジェクトを作成および変更するための Statement.executeUpdate メソッドの使用』

関連資料:

- 408 ページの『DB2 Universal JDBC ドライバーのプロパティー』
- 479 ページの『DB2 Universal JDBC ドライバーが発行するエラー・コード』
- 479 ページの『DB2 Universal JDBC ドライバーが発行する SQLSTATE』
- 416 ページの『JDBC API 用ドライバー・サポートの比較』
- 403 ページの『Java、JDBC、および SQL のデータ・タイプ』

DB2 Universal JDBC ドライバーが発行するエラー・コード

+4200 から +4299、+4450 から +4499、-4200 から -4299、および -4450 から -4499 の範囲のエラー・コードは、 DB2 Universal JDBC ドライバー用に予約されています。現在のところ、DB2 Universal JDBC ドライバーは以下のエラー・コードを発行します。

-4200 XA 環境内のグローバル・トランザクションに含まれていたアプリケーションが、無効なコミットまたはロールバックを発行しました。

-4498 フェイルオーバーまたはフェイルバックが発生し、トランザクションが失敗しました。

-4499 致命的エラーが発生したため、切断されました。

-99999
まだエラー・コードのないエラーを、 DB2 Universal JDBC ドライバーが出しました。

関連タスク:

- 315 ページの『DB2 Universal JDBC ドライバー使用時の SQLException の処理』
- 380 ページの『SQLJ アプリケーションでの SQL エラーの処理』

関連資料:

- 470 ページの『DB2 Universal JDBC ドライバーと他の DB2 JDBC ドライバーの JDBC との相違点』

DB2 Universal JDBC ドライバーが発行する SQLSTATE

46600 から 466ZZ の範囲の SQLSTATE は、DB2 Universal JDBC ドライバー用に予約済みです。現在のところ、 DB2 Universal JDBC ドライバーは DRDA エラー以外の内部エラーに対して NULL の SQLSTATE 値を戻します。 DRDA エラーに対しては、以下の SQLSTATE が発行されます。

08004 アプリケーション・サーバーが、接続の確立を拒否しました。

22021 文字がコード化文字セットにありません。

24501 識別されたカーソルがオープンしていません。

2D521 SQL COMMIT または ROLLBACK が、現在の操作環境では無効です。

58008 分散プロトコル・エラーのため、実行が失敗しました。このエラーは、後続の DDM コマンドまたは SQL ステートメントの正常な実行には影響しません。

58009 会話の割り振り解除の原因となる分散プロトコル・エラーのため、実行が失敗しました。

58010 分散プロトコル・エラーのため、実行が失敗しました。このエラーは、後続の DDM コマンドまたは SQL ステートメントの正常な実行に影響を与えます。

58014 DDM コマンドはサポートされていません。

58015 DDM オブジェクトはサポートされていません。

58016 DDM パラメーターはサポートされていません。

58017 DDM パラメーターの値がサポートされていません。

関連タスク:

- 315 ページの『DB2 Universal JDBC ドライバー使用時の SQLException の処理』
- 380 ページの『SQLJ アプリケーションでの SQL エラーの処理』

関連資料:

- 470 ページの『DB2 Universal JDBC ドライバーと他の DB2 JDBC ドライバーの JDBC との相違点』

第 18 章 JDBC ドライバーのインストール

以降のセクションには、JDBC ドライバーのインストールに関する情報が含まれています。

DB2 Universal JDBC ドライバーのインストール

DB2 UDB for Linux、UNIX、および Windows のいずれかの製品のインストール中に JDBC サポートを選択すると、インストール・プログラムが、いくつかの DB2 Universal JDBC ドライバー のインストール・ステップを実行します。 DB2 UDB for Windows バージョン 8 のインストール・プログラムは、以下のタスクを実行します。

- db2jcc.jar ファイルおよび sqlj.zip ファイルをインストールし、これらをシステム CLASSPATH に追加します。
- Universal Type 2 Connectivity に必要とされるファイル db2jcct2.dll を sqllib¥bin ディレクトリーにインストールします。

DB2 UDB for UNIX バージョン 8 のインストール・プログラムは、以下のタスクを実行します。

- db2jcc.jar ファイルおよび sqlj.zip ファイルをインストールします。
- db2jcc.jar ファイルおよび sqlj.zip ファイルを、 db2profile (Bourne または Korn シェルの場合) または db2cshrc (C シェルの場合) スクリプトの CLASSPATH ステートメントに追加します。
- Universal Type 2 Connectivity に必要とされるファイル libdb2jcct2.so (HP-UX の場合は libdb2jcct2.sl) を sqllib/lib ディレクトリーにインストールします。

DB2 Universal JDBC ドライバーをインストールするには、以下の追加のステップを実行する必要があります。

前提条件:

- JDK 1.3.1 以降

  JDBC 3.0 フィーチャーには、最低でも JDK 1.4 が必要です。

- TCP/IP

  次の場合は、サーバーで TCP/IP 通信用の構成を行う必要があります。

  – JDBC または SQLJ アプリケーションが Universal Type 4 Connectivity を使用する。

  – JDBC または SQLJ アプリケーションが Universal Type 2 Connectivity を使用し、接続 URL にサーバー およびポート を指定する。

- DB2 UDB for z/OS or OS/390 のストアード・プロシージャー

  JDBC または SQLJ アプリケーションが DB2 UDB for z/OS or OS/390 サーバーに接続する場合は、 DB2 カタログ情報の検索、トレース、およびエラー・メッセージのフォーマットをサポートするために、いくつかのストアード・プロシ

ージャーがそのサーバー上にインストールされている必要があります。それらのストアード・プロシージャーは以下のとおりです。
- SQLCOLPRIVILEGES
- SQLCOLUMNS
- SQLFOREIGNKEYS
- SQLGETTYPEINFO
- SQLPRIMARYKEYS
- SQLPROCEDURECOLS
- SQLPROCEDURES
- SQLSPECIALCOLUMNS
- SQLSTATISTICS
- SQLTABLEPRIVILEGES
- SQLTABLES
- SQLUDTS
- SQLCAMESSAGE

これらのストアード・プロシージャーは、 DB2 UDB for z/OS バージョン 8 製品に付属して出荷されています。これらのストアード・プロシージャーは、以下の DB2 UDB for OS/390 and z/OS バージョン 7 または DB2 UDB for OS/390 バージョン 6 の PTF に入れて出荷されています。

表 75. DB2 for OS/390 and z/OS の PTF

DB2 for OS/390 and z/OS のバージョン	PTF 番号
バージョン 6	UQ72081 および UQ72082
バージョン 7	UQ72083

DB2 UDB for z/OS システム管理者に、これらのストアード・プロシージャーがインストールされているかどうか尋ねてください。

- iSeries サーバーの Unicode サポート

  SQLJ または JDBC プログラムが Universal Type 4 Connectivity を使用して DB2 UDB for iSeries サーバーに接続する場合は、 OS/400 オペレーティング・システムが、 Unicode UTF-8 コード化スキームをサポートしていなければなりません。次の表には、Unicode UTF-8 のサポートのために必要な OS/400 PTF をリストしています。

表 76. Unicode UTF-8 のサポートのための OS/400 PTF

OS/400 のバージョン	PTF 番号
V5R3 以降	なし (サポートは組み込まれている)
V5R2	SI06541、 SI06796、 SI07557、 SI07564、 SI07565、 SI07566、および SI07567
V5R1	SI06308、 SI06300、 SI06301、 SI06302、 SI06305、 SI06307、および SI05872

- HP-UX クライアントおよびサーバーの Java サポート

  HP-UX サーバーに関して: DB2 Universal JDBC ドライバーは、 HP-UX のデフォルトの文字セットである Roman8 のデータベースをサポートしません。そのた

め、DB2 Universal JDBC ドライバーを使用してアクセスする予定の HP-UX サーバー上にデータベースを作成する際は、異なる文字セットを用いて作成する必要があります。

HP-UX クライアントおよびサーバーに関して: HP-UX システムの Java 環境において、 DB2 Universal JDBC ドライバーでアプリケーションを実行するためには、特殊なセットアップが必要です。

詳細は、 『HP-UX Java 環境のセットアップ』を参照してください。

手順:

注:

DB2 Universal JDBC ドライバー をインストールするには、次のようにします。

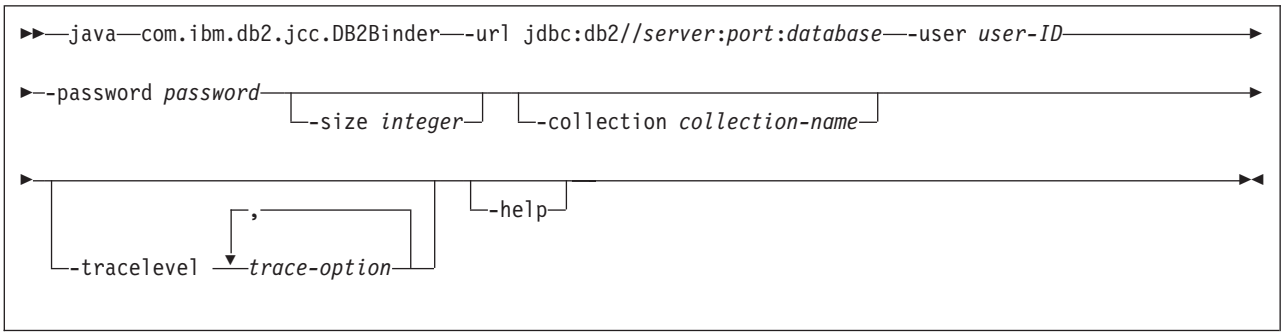
1. Java アプリケーションの CLASSPATH に、 db2jcc.jar を、その他の JDBC ドライバー用の JAR ファイル名の前に入れます。
2. Kerberos セキュリティーを使用することを計画している場合は、 Java アプリケーションの CLASSPATH に以下のファイルを入れます。
   - ibmjceprovider.jar
   - ibmjcefw.jar
   - ibmjlog.jar
   - US_export_policy.jar
   - Local_policy.jar
   - ibmjgssprovider.jar
   - jaas.jar
   - ibmjceprovider.jar
   - ibmjcefw.jar
   - ibmjlog.jar
   - US_export_policy.jar
   - Local_policy.jar
3. Java アプリケーションの CLASSPATH に、サーバーへの接続を許可する 1 つ以上のライセンス・ファイル名を入れます。これらのライセンス・ファイルは、表 77 にリストしています。

表 77. DB2 Universal JDBC ドライバーのライセンス・ファイル

ライセンス・ファイル	ライセンス・ファイルが接続を許可するサーバー	ライセンス・ファイルが組み込まれている製品
db2jcc_license_c.jar	Cloudscape	Cloudscape Network Server
db2jcc_license_cu.jar	Cloudscape DB2 UDB for Linux, UNIX and Windows	DB2 UDB for Linux, UNIX and Windows
db2jcc_license_cisuz.jar	Cloudscape DB2 UDB for Linux, UNIX and Windows DB2 UDB for z/OS または DB2 UDB for OS/390 and z/OS DB2 UDB for iSeries DB2 Server for VSE & VM	DB2 Connect

4. SQLJ の使用を計画している場合は、Java アプリケーションの CLASSPATH に sqlj.zip を追加してください。古いバージョンの sqlj.zip および runtime.zip は、CLASSPATH から除去してください。
5. DB2 UDB for z/OS または OS/390 サーバーに接続することを計画している場合は、 com.ibm.db2.jcc.DB2Binder ユーティリティーを使用して、サーバーで DB2 Universal JDBC ドライバーによって使用される DB2 パッケージをバインドします。

DB2Binder の構文:



DB2Binder オプションの説明:

-url
JCC パッケージをバインドするデータ・ソースを指定します。 -url 値の可変部は、以下のとおりです。

server
DB2 サブシステムがある MVS システムのドメイン名または IP アドレス。

port
DB2 サブシステムに割り当てられている TCP/IP サーバー・ポート番号。デフォルトは 446 です。

database
SYSIBM.LOCATIONS カタログ表で定義されている、DB2 サブシステムのロケーション名。

-user
パッケージをバインドするユーザー ID を指定します。このユーザーは、パッケージに対する BIND 権限を持っている必要があります。

-size
4 つの DB2 分離レベルごと、および 2 つの保留可能性 (holdability) の値ごとに、 DB2binder がバインドする DB2 パッケージの数を指定します。
DB2 Universal JDBC ドライバーはこれらのパッケージを使用して動的 SQL を処理します。さらに、DB2binder は、DB2 Universal JDBC ドライバーが静的 SQL 用に使用する単一のパッケージもバインドします。したがって、DB2binder がバインドするパッケージ数の合計は、次のようになります。

```
4*2*integer+1
```

integer のデフォルト値は 3 です。

-collection

DB2 Universal JDBC ドライバーのインスタンスが使用するパッケージのコレクション ID を指定します。デフォルトは NULLID です。 DB2binder はこの値を大文字に変換します。

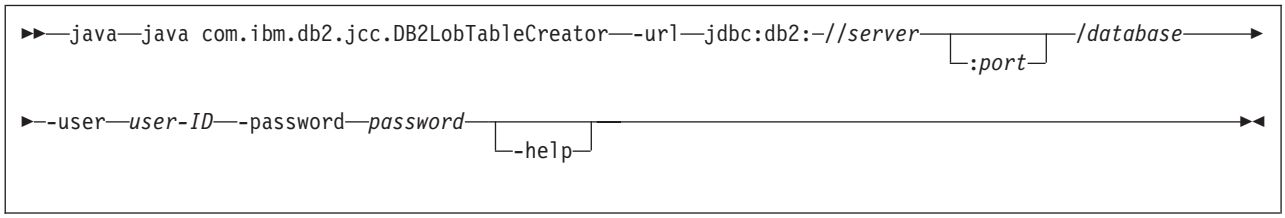
com.ibm.db2.jcc.DB2Binder を複数回実行して、-collection に毎回異なる値を指定すれば、単一のロケーションで JCC パッケージ・セットのインスタンスを複数作成することができます。ランタイムに、DB2 Universal JDBC ドライバーのコピーは、 currentPackageSet プロパティーに -collection の値と一致する値を設定することによって選択します。 currentPackageSet プロパティーについては、 『DB2 Universal JDBC ドライバーのプロパティー』を参照してください。

-tracelevel

DB2Binder の実行時に何をトレースするかを指定します。使用可能なオプションについては、 『DB2 Universal JDBC ドライバーのプロパティー』にある traceLevel プロパティーの説明を参照してください。

6. LOB ロケーターを使用して DB2 UDB for z/OS または OS/390 サーバー上の DB2 表の DBCLOB 列にアクセスすることを計画している場合は、これらの各サーバー上で com.ibm.db2.jcc.DB2LobTableCreator ユーティリティーを実行して、 LOB ロケーターを取り出すために必要な表を作成します。

DB2LobTableCreator の構文:



DB2LobTableCreator オプションの説明:

-url

DB2LobTableCreator を実行するデータ・ソースを指定します。 -url 値の可変部は、以下のとおりです。

jdbc:db2:

接続が DB2 UDB ファミリー内のサーバーへ行われることを示しています。

server

データベース・サーバーのドメイン・ネームまたは IP アドレス。

port

データベース・サーバーに割り当てられる TCP/IP サーバーのポート番号。これは、0 から 65535 の間の整数です。デフォルトは 446 です。

database

データベース・サーバーの名前。

database は、インストール中に定義される DB2 ロケーション名です。この値の中の文字はすべて大文字でなければなりません。サーバーで以下の SQL ステートメントを実行することにより、ロケーション名を判別できます。

```
SELECT CURRENT SERVER FROM SYSIBM.SYSDUMMY1;
```

-user
DB2LobTableCreator を実行するユーザー ID を指定します。このユーザーは、DSNATPDB データベース内に表を作成する権限を持っていなければなりません。

-password
ユーザー ID のパスワードを指定します。

-help
DB2LobTableCreator ユーティリティーで、サポートされる各オプションの説明を表示するように指定します。 -help と共に他のオプションが指定されても、それらは無視されます。

関連タスク:

- 「*アプリケーション開発ガイド アプリケーションの構築および実行*」の『HP-UX Java 環境のセットアップ』
- 「*インストールおよび構成 補足*」の『DB2 インスタンスの TCP/IP 通信の構成』
- 「*インストールおよび構成 補足*」の『TCP/IP 通信のためのサーバー上のデータベース・マネージャー構成ファイルの更新』
- 「*インストールおよび構成 補足*」の『TCP/IP 通信のためのサーバー上のサービス・ファイルの更新』

関連資料:

- 408 ページの『DB2 Universal JDBC ドライバーのプロパティー』

第 19 章 JDBC および SQLJ セキュリティー

以下に続くセクションには、JDBC ドライバー使用時に使用可能なセキュリティー・メカニズムについての情報が含まれています。

DB2 JDBC Type 2 ドライバー使用時のセキュリティー

Linux、UNIX®、および Windows® 用 DB2® JDBC Type 2 ドライバー (DB2 JDBC Type 2 ドライバー) は、ユーザー ID およびパスワード・セキュリティーをサポートしています。ユーザー ID とパスワードを設定するか、あるいはそれらを両方とも設定しないようにしなければなりません。ユーザー ID とパスワードを設定しない場合、ドライバーは、オペレーティング・システムに現在ログオンしているユーザーのユーザー ID とパスワードを使用します。

JDBC 接続で、ユーザー ID およびパスワード・セキュリティーを指定する場合、以下のいずれかの技法を使用します。

DriverManager インターフェースの場合: DriverManager.getConnection 呼び出しで、ユーザー ID およびパスワードを直接指定することができます。たとえば、以下のようにします。

```
import java.sql.*;          // JDBC base
...
String id = "db2adm";       // Set user ID
Sring pw = "db2adm";        // Set password
String url = "jdbc:db2:toronto";
                            // Set URL for the data source
Connection con = DriverManager.getConnection(url, id, pw);
                            // Create connection
```

別の方法として、 Properties オブジェクトに user および password プロパティーを設定し、 Properties オブジェクトをパラメーターとして組み込んだフォームの getConnection メソッドを呼び出して、ユーザー ID およびパスワードを設定できます。たとえば、以下のようにします。

```
import java.sql.*;                          // JDBC base
import COM.ibm.db2.jdbc.*;                  // DB2 implementation of JDBC 2.0
...
Properties properties = new java.util.Properties();
                                            // Create Properties object
properties.put("user", "db2adm");           // Set user ID for the connection
properties.put("password", "db2adm");       // Set password for the connection
String url = "jdbc:db2:toronto";
                                            // Set URL for the data source
Connection con = DriverManager.getConnection(url, properties);
                                            // Create connection
```

DataSource インターフェースの場合: DataSource.getConnection 呼び出しで、ユーザー ID およびパスワードを直接指定することができます。たとえば、以下のようにします。

```
import java.sql.*;                          // JDBC base
import COM.ibm.db2.jdbc.*;                  // DB2 implementation of JDBC 2.0
...
```

```
Context ctx=new InitialContext();         // Create context for JNDI
DataSource ds=(DataSource)ctx.lookup("jdbc/sampledb");
                                          // Get DataSource object
String id = "db2adm";                     // Set user ID
Sring pw = "db2adm";                      // Set password
Connection con = ds.getConnection(id, pw);
                                          // Create connection
```

DataSource オブジェクトを作成してデプロイする場合、 DataSource オブジェクトの作成後に、 DataSource.setUser および DataSource.setPassword メソッドを呼び出して、ユーザー ID およびパスワードを設定できます。たとえば、以下のようにします。

```
import java.sql.*;                        // JDBC base
import COM.ibm.db2.jdbc.*;                // DB2 implementation of JDBC 2.0
...
DB2DataSource db2ds = new DB2DataSource();
                                            // Create DataSource object
db2ds.setDatabaseName("toronto");           // Set location
db2ds.setUser("db2adm");                    // Set user ID
db2ds.setPassword("db2adm");                // Set password
```

関連概念:

- 298 ページの『DB2 アプリケーションによる DriverManager インターフェースと DB2 JDBC Type 2 ドライバーを使用したデータ・ソースへの接続』

関連タスク:

- 303 ページの『DataSource インターフェースを使用したデータ・ソースへの接続』
- 345 ページの『DataSource オブジェクトの作成およびデプロイ』

DB2 Universal JDBC ドライバー使用時のセキュリティー

DB2 Universal JDBC ドライバーを使用するときには、 securityMechanism プロパティーに値を指定して、セキュリティー・メカニズムを選択します。このプロパティーは、以下のいずれかの方法で設定できます。

- DriverManager インターフェースを使用している場合、 java.util.Properties パラメーターを組み込んだフォームの getConnection メソッドを呼び出す前に、java.util.Properties オブジェクトに securityMechanism を設定します。
- DataSource インターフェースを使用しており、独自の DataSource オブジェクトを作成してデプロイしている場合、 DataSource オブジェクトの作成後に DataSource.setSecurityMechanism メソッドを呼び出します。

表 78 には、 DB2 Universal JDBC ドライバーがサポートするセキュリティー・メカニズムと、各セキュリティー・メカニズムを指定するために securityMechanism プロパティーに指定する必要のある値がリストされています。デフォルトのセキュリティー・メカニズムは、ユーザー ID とパスワードのメカニズムです。

表 78. DB2 Universal JDBC ドライバーでサポートされているセキュリティー・メカニズム

セキュリティー・メカニズム	securityMechanism プロパティー値
ユーザー ID およびパスワード	DB2BaseDataSource.CLEAR_TEXT_PASSWORD_SECURITY
ユーザー ID のみ	DB2BaseDataSource.USER_ONLY_SECURITY

表 78. DB2 Universal JDBC ドライバーでサポートされているセキュリティー・メカニズム (続き)

セキュリティー・メカニズム	securityMechanism プロパティー値
ユーザー ID および暗号化されたパスワード	DB2BaseDataSource.ENCRYPTED_PASSWORD_SECURITY
暗号化されたユーザー ID および暗号化されたパスワード	DB2BaseDataSource.ENCRYPTED_USER_AND_PASSWORD_SECURITY
Kerberos[1]	DB2BaseDataSource.KERBEROS_SECURITY

注:

1. Universal Type 4 Connectivity でのみ使用可能。

関連概念:

- 492 ページの『DB2 Universal JDBC ドライバー使用時の暗号化されたユーザー ID セキュリティーまたは暗号化されたパスワード・セキュリティー』
- 493 ページの『DB2 Universal JDBC ドライバー使用時の Kerberos セキュリティーの使用』
- 491 ページの『DB2 Universal JDBC ドライバー使用時のユーザー ID 専用セキュリティーの使用』
- 489 ページの『DB2 Universal JDBC ドライバー使用時のユーザー ID およびパスワード・セキュリティーの使用』

関連資料:

- 408 ページの『DB2 Universal JDBC ドライバーのプロパティー』

DB2 Universal JDBC ドライバー使用時のユーザー ID およびパスワード・セキュリティーの使用

JDBC 接続で、ユーザー ID およびパスワード・セキュリティーを指定する場合、以下のいずれかの技法を使用します。

DriverManager インターフェースの場合: DriverManager.getConnection 呼び出しで、ユーザー ID およびパスワードを直接指定することができます。たとえば、以下のようにします。

```
import java.sql.*;          // JDBC base
...
String id = "db2adm";       // Set user ID
String pw = "db2adm";       // Set password
String url = "jdbc:db2://mvs1.sj.ibm.com:5021/san_jose";
                            // Set URL for the data source
Connection con = DriverManager.getConnection(url, id, pw);
                            // Create connection
```

別の方法は、ユーザー ID とパスワードを URL ストリングに直接設定することです。たとえば、以下のようにします。

```
import java.sql.*;          // JDBC base
...
String url =
  "jdbc:db2://mvs1.sj.ibm.com:5021/san_jose:user=db2adm;password=db2adm;";
```

```
                          // Set URL for the data source
Connection con = DriverManager.getConnection(url);
                          // Create connection
```

別の方法として、 Properties オブジェクトに user および password プロパティーを設定し、 Properties オブジェクトをパラメーターとして組み込んだフォームの getConnection メソッドを呼び出して、ユーザー ID およびパスワードを設定できます。任意で、securityMechanism プロパティーを設定して、ユーザー ID およびパスワード・セキュリティーを使用していることを示すことができます。たとえば、以下のようにします。

```
import java.sql.*;                        // JDBC base
import com.ibm.db2.jcc.*;                  // DB2® implementation of JDBC 2.0
...
Properties properties = new java.util.Properties();
                                          // Create Properties object
properties.put("user", "db2adm");         // Set user ID for the connection
properties.put("password", "db2adm");     // Set password for the connection
properties.put("securityMechanism",
  new String("" + com.ibm.db2.jcc.DB2BaseDataSource.CLEAR_TEXT_PASSWORD_SECURITY +
  ""));
                                          // Set security mechanism to
                                          // user ID and password
String url = "jdbc:db2://mvs1.sj.ibm.com:5021/san_jose";
                                          // Set URL for the data source
Connection con = DriverManager.getConnection(url, properties);
                                          // Create connection
```

DataSource インターフェースの場合: DataSource.getConnection 呼び出しで、ユーザー ID およびパスワードを直接指定することができます。たとえば、以下のようにします。

```
import java.sql.*;                        // JDBC base
import com.ibm.db2.jcc.*;                 // DB2 implementation of JDBC 2.0
...
Context ctx=new InitialContext();         // Create context for JNDI
DataSource ds=(DataSource)ctx.lookup("jdbc/sampledb");
                                          // Get DataSource object
String id = "db2adm";                     // Set user ID
String pw = "db2adm";                      // Set password
Connection con = ds.getConnection(id, pw);
                                          // Create connection
```

DataSource オブジェクトを作成してデプロイする場合、 DataSource オブジェクトの作成後に、 DataSource.setUser および DataSource.setPassword メソッドを呼び出して、ユーザー ID およびパスワードを設定できます。任意で、DataSource.setSecurityMechanism メソッド・プロパティーを呼び出して、ユーザー ID およびパスワード・セキュリティーを使用していることを示すことができます。たとえば、以下のようにします。

```
...
com.ibm.db2.jcc.DB2SimpleDataSource db2ds =    // Create DB2SimpleDataSource object
 new com.ibm.db2.jcc.DB2SimpleDataSource();
db2ds.setDriverType(4);                         // Set driver type
db2ds.setDatabaseName("san_jose");              // Set location
db2ds.setServerName("mvs1.sj.ibm.com");         // Set server name
db2ds.setPortNumber(5021);                      // Set port number
db2ds.setUser("db2adm");                        // Set user ID
db2ds.setPassword("db2adm");                    // Set password
db2ds.setSecurityMechanism(
  com.ibm.db2.jcc.DB2BaseDataSource.CLEAR_TEXT_PASSWORD_SECURITY);
                                                // Set security mechanism to
                                                // user ID and password
```

関連タスク:

- 303 ページの『DataSource インターフェースを使用したデータ・ソースへの接続』
- 345 ページの『DataSource オブジェクトの作成およびデプロイ』
- 300 ページの『DriverManager インターフェースと DB2 Universal JDBC ドライバーを使用したデータ・ソースへの接続』

関連資料:

- 408 ページの『DB2 Universal JDBC ドライバーのプロパティー』

DB2 Universal JDBC ドライバー使用時のユーザー ID 専用セキュリティーの使用

JDBC 接続で、ユーザー ID セキュリティーを指定する場合、以下のいずれかの技法を使用します。

DriverManager インターフェースの場合: Properties オブジェクトに、 user および securityMechanism プロパティーを設定し、 Properties オブジェクトをパラメーターとして組み込んだフォームの getConnection メソッドを呼び出して、ユーザー ID およびセキュリティーのメカニズムを設定します。たとえば、以下のようにします。

```
import java.sql.*;                          // JDBC base
import com.ibm.db2.jcc.*;                   // DB2® implementation of JDBC 2.0
...
Properties properties = new Properties(); // Create a Properties object
properties.put("user", "db2adm");         // Set user ID for the connection
properties.put("securityMechanism",
  new String("" + com.ibm.db2.jcc.DB2BaseDataSource.USER_ONLY_SECURITY + ""));
                                          // Set security mechanism to
                                          // user ID only
String url = "jdbc:db2://mvs1.sj.ibm.com:5021/san_jose";
                                          // Set URL for the data source
Connection con = DriverManager.getConnection(url, properties);
                                          // Create the connection
```

DataSource インターフェースの場合: DataSource オブジェクトを作成してデプロイする場合、 DataSource オブジェクトの作成後に、 DataSource.setUser および DataSource.setSecurityMechanism メソッドを呼び出して、ユーザー ID およびセキュリティーのメカニズムを設定します。たとえば、以下のようにします。

```
import java.sql.*;                          // JDBC base
import com.ibm.db2.jcc.*;                   // DB2 implementation of JDBC 2.0
...
com.ibm.db2.jcc.DB2SimpleDataSource db2ds =
 new com.ibm.db2.jcc.DB2SimpleDataSource();
                                          // Create DB2SimpleDataSource object
db2ds.setDriverType(4);                   // Set the driver type
db2ds.setDatabaseName("san_jose");        // Set the location
db2ds.setServerName("mvs1.sj.ibm.com");   // Set the server name
db2ds.setPortNumber(5021);                // Set the port number
db2ds.setUser("db2adm");                  // Set the user ID
db2ds.setSecurityMechanism(
  com.ibm.db2.jcc.DB2BaseDataSource.USER_ONLY_SECURITY);
                                          // Set security mechanism to
                                          // user ID only
```

DB2 Universal JDBC ドライバー使用時の暗号化されたユーザー ID セキュリティーまたは暗号化されたパスワード・セキュリティー

DB2® for z/OS™ サーバーにアクセスするときに、暗号化されたユーザー ID セキュリティーまたは暗号化されたパスワード・セキュリティーを使用する場合、Java™ Cryptography Extension、IBMJCE for z/OS をサーバーで使用可能にする必要があります。 Java Cryptography Extension は、 IBM® Developer Kit for OS/390® Java 2 Technology Edition または IBM Developer Kit for z/OS Java 2 Technology Edition の一部です。 IBMJCE を使用可能にする方法の詳細については、次の URL の Web をご覧ください。

```
http://www.ibm.com/servers/eserver/zseries/software/java/aboutj2.html
```

JDBC 接続で、暗号化されたユーザー ID または暗号化されたパスワード・セキュリティーを指定する場合、以下のいずれかの技法を使用します。

DriverManager インターフェースの場合: Properties オブジェクトに、 user、password、および securityMechanism プロパティーを設定し、 Properties オブジェクトをパラメーターとして組み込んだフォームの getConnection メソッドを呼び出して、ユーザー ID、パスワード、およびセキュリティーのメカニズムを設定します。例えば、次のようなコードを使用して、ユーザー ID および暗号化されたパスワード・セキュリティー・メカニズムを設定します。

```
import java.sql.*;                          // JDBC base
import com.ibm.db2.jcc.*;                   // DB2 implementation of JDBC 2.0
...
Properties properties = new Properties(); // Create a Properties object
properties.put("user", "db2adm");           // Set user ID for the connection
properties.put("password", "db2adm");       // Set password for the connection
properties.put("securityMechanism",
  new String("" + com.ibm.db2.jcc.DB2BaseDataSource.ENCRYPTED_PASSWORD_SECURITY +
  ""));
                                            // Set security mechanism to
                                            // user ID and encrypted password
String url = "jdbc:db2://mvs1.sj.ibm.com:5021/san_jose";
                                            // Set URL for the data source
Connection con = DriverManager.getConnection(url, properties);
                                            // Create the connection
```

DataSource インターフェースの場合:DataSource オブジェクトを作成してデプロイする場合、 DataSource オブジェクトの作成後に、 DataSource.setUser、DataSource.setPassword、および DataSource.setSecurityMechanism メソッドを呼び出して、ユーザー ID、パスワード、およびセキュリティーのメカニズムを設定することができます。例えば、次のようなコードを使用して、暗号化されたユーザー ID および暗号化されたパスワード・セキュリティー・メカニズムを設定します。

```
import java.sql.*;                          // JDBC base
import com.ibm.db2.jcc.*;                   // DB2 implementation of JDBC 2.0
...
com.ibm.db2.jcc.DB2SimpleDataSource db2ds =
 new com.ibm.db2.jcc.DB2SimpleDataSource();
                                            // Create the DataSource object
db2ds.setDriverType(4);                     // Set the driver type
db2ds.setDatabaseName("san_jose");          // Set the location
db2ds.setServerName("mvs1.sj.ibm.com");
                                            // Set the server name
db2ds.setPortNumber(5021);                  // Set the port number
db2ds.setUser("db2adm");                    // Set the user ID
db2ds.setPassword("db2adm");                // Set the password
```

```
db2ds.setSecurityMechanism(
  com.ibm.db2.jcc.DB2BaseDataSource.ENCRYPTED_PASSWORD_SECURITY);
                                          // Set security mechanism to
                                          // encrypted user ID and password
```

関連タスク:

- 303 ページの『DataSource インターフェースを使用したデータ・ソースへの接続』
- 345 ページの『DataSource オブジェクトの作成およびデプロイ』
- 300 ページの『DriverManager インターフェースと DB2 Universal JDBC ドライバーを使用したデータ・ソースへの接続』

関連資料:

- 408 ページの『DB2 Universal JDBC ドライバーのプロパティー』

DB2 Universal JDBC ドライバー使用時の Kerberos セキュリティーの使用

Kerberos セキュリティーは Universal Type 4 Connectivity でのみ使用可能です。

DB2® for z/OS™ サーバーにアクセスする際に Kerberos セキュリティーを使用する場合、以下の製品またはその同等品をインストールして構成する必要があります。

- SecureWay® Security Server for z/OS および OS/390®
- OS/390 SecureWay Security Server Network Authentication and Privacy Service (これは OS/390 SecureWay Security Server のコンポーネントです)

  これは Kerberos バージョン 5 の IBM® OS/390 インプリメンテーションです。

  詳細については、「*OS/390 SecureWay Server Network Authentication and Privacy Service Administration*」を参照してください。

IBM Developer Kit for OS/390、 Java™ 2 Technology Edition、または IBM Developer Kit for z/OS、 Java 2 Technology Edition の以下のコンポーネントを使用可能にする必要もあります。

- Java Cryptography Extension (IBMJCE) for OS/390
- IBM Java Generic Security Service (IBMJGSS)
- Java Authentication and Authorization Service (JAAS) for OS/390

これらのコンポーネントを使用可能にする方法の詳細については、次の URL の Web をご覧ください。

```
http://www.ibm.com/servers/eserver/zseries/software/java/aboutj2.html
```

接続に Kerberos セキュリティーを指定する方法は、以下の 3 つがあります。

- ユーザー ID とパスワードを使用する
- ユーザー ID とパスワードを使用しない
- 委任証明書を使用する

Kerberos セキュリティーとユーザー ID およびパスワードの使用:

この場合、Kerberos は、指定のユーザー ID とパスワードを使用して発券許可証 (TGT) を入手し、それによって DB2 サーバーでユーザーを認証します。

user、password、kerberosServerPrincipal、および securityMechanism プロパティーを設定する必要があります。 kerberosServerPrincipal プロパティーは、クライアントが登録されている領域の Kerberos サーバーのアドレスを指定します。

DriverManager インターフェースの場合: Properties オブジェクトに、user、password、 kerberosServerPrincipal、および securityMechanism プロパティーを設定し、 Properties オブジェクトをパラメーターとして組み込んだフォームの getConnection メソッドを呼び出して、ユーザー ID、パスワード、Kerberos サーバー、およびセキュリティーのメカニズムを設定します。例えば、次のようなコードを使用して、ユーザー ID およびパスワードで Kerberos セキュリティー・メカニズムを設定します。

```
import java.sql.*;                          // JDBC base
import com.ibm.db2.jcc.*;                   // DB2 implementation of JDBC 2.0
...
Properties properties = new Properties(); // Create a Properties object
properties.put("user", "db2adm");           // Set user ID for the connection
properties.put("password", "db2adm");       // Set password for the connection
properties.put("kerberosServerPrincipal", "kdcsrv1.sj.ibm.com");
                                            // Set the Kerberos server
properties.put("securityMechanism",
  new String("" + com.ibm.db2.jcc.DB2BaseDataSource.KERBEROS_SECURITY + ""));
                                            // Set security mechanism to
                                            // Kerberos
String url = "jdbc:db2://mvs1.sj.ibm.com:5021/san_jose";
                                            // Set URL for the data source
Connection con = DriverManager.getConnection(url, properties);
                                            // Create the connection
```

DataSource インターフェースの場合: DataSource オブジェクトを作成してデプロイする場合、 DataSource オブジェクトの作成後に、 DataSource.setKerberosServerPrincipal および DataSource.setSecurityMechanism メソッドを呼び出して、 Kerberos サーバーおよびセキュリティーのメカニズムを設定します。たとえば、以下のようにします。

```
import java.sql.*;                          // JDBC base
import com.ibm.db2.jcc.*;                   // DB2 implementation of JDBC 2.0
...
com.ibm.db2.jcc.DB2SimpleDataSource db2ds =
 new com.ibm.db2.jcc.DB2SimpleDataSource();
                                            // Create the DataSource object
db2ds.setDriverType(4);                     // Set the driver type
db2ds.setDatabaseName("san_jose");          // Set the location
db2ds.setUser("db2adm");                    // Set the user
db2ds.setPassword("db2adm");                // Set the password
db2ds.setServerName("mvs1.sj.ibm.com");
                                            // Set the server name
db2ds.setPortNumber(5021);                  // Set the port number
db2ds.setKerberosServerPrincipal("kdcsrv1.sj.ibm.com");
                                            // Set the Kerberos server
db2ds.setSecurityMechanism(
  com.ibm.db2.jcc.DB2BaseDataSource.KERBEROS_SECURITY);
                                            // Set security mechanism to
                                            // Kerberos
```

ユーザー ID またはパスワードを使用しない Kerberos セキュリティーの使用:

このケースでは、 Kerberos デフォルト証明書キャッシュには発券許可証 (TGT) が含まれていなければならず、それによって DB2 サーバーでユーザーを認証します。

kerberosServerPrincipal および securityMechanism プロパティーを設定する必要があります。

DriverManager インターフェースの場合: Properties オブジェクトに、kerberosServerPrincipal および securityMechanism プロパティーを設定し、Properties オブジェクトをパラメーターとして組み込んだフォームの getConnection メソッドを呼び出して、 Kerberos サーバーおよびセキュリティーのメカニズムを設定します。例えば、次のようなコードを使用して、ユーザー ID およびパスワードを使用せずに Kerberos セキュリティー・メカニズムを設定します。

```
import java.sql.*;                          // JDBC base
import com.ibm.db2.jcc.*;                   // DB2 implementation of JDBC 2.0
...
Properties properties = new Properties(); // Create a Properties object
properties.put("kerberosServerPrincipal", "kdcsrv1.sj.ibm.com");
                                            // Set the Kerberos server
properties.put("securityMechanism",
  new String("" + com.ibm.db2.jcc.DB2BaseDataSource.KERBEROS_SECURITY + ""));
                                            // Set security mechanism to
                                            // Kerberos
String url = "jdbc:db2://mvs1.sj.ibm.com:5021/san_jose";
                            // Set URL for the data source
Connection con = DriverManager.getConnection(url, properties);
                                            // Create the connection
```

DataSource インターフェースの場合: DataSource オブジェクトを作成してデプロイする場合、 DataSource オブジェクトの作成後に、DataSource.setKerberosServerPrincipal および DataSource.setSecurityMechanism メソッドを呼び出して、 Kerberos サーバーおよびセキュリティーのメカニズムを設定します。たとえば、以下のようにします。

```
import java.sql.*;                          // JDBC base
import com.ibm.db2.jcc.*;                   // DB2 implementation of JDBC 2.0
...
DB2DataSource db2ds = new com.ibm.db2.jcc.DB2SimpleDataSource();
                                            // Create the DataSource object
db2ds.setDriverType(4);                     // Set the driver type
db2ds.setDatabaseName("san_jose");          // Set the location
db2ds.setServerName("mvs1.sj.ibm.com");
                                            // Set the server name
db2ds.setPortNumber(5021);                  // Set the port number
db2ds.setKerberosServerPrincipal("kdcsrv1.sj.ibm.com");
                                            // Set the Kerberos server
db2ds.setSecurityMechanism(
  com.ibm.db2.jcc.DB2BaseDataSource.KERBEROS_SECURITY);
                                            // Set security mechanism to
                                            // Kerberos
```

別のプリンシパルからの委任証明書を使用した Kerberos セキュリティーの使用:

この場合、別のプリンシパルから渡される委任証明書を使用して、DB2 サーバーで認証します。

kerberosServerPrincipal、gssCredential、および securityMechanism プロパティーを設定する必要があります。

DriverManager インターフェースの場合: Properties オブジェクトに、kerberosServerPrincipal および securityMechanism プロパティーを設定し、Kerberos サーバー、委任証明書、およびセキュリティーのメカニズムを設定します。 gssCredential プロパティーはストリングではないため、 Properties.put メ

ソッドを使用して設定することはできません。代わりに、
DB2BaseDataSource.setGSSCredential メソッドを使用します。その後、Properties オブジェクトをパラメーターとして組み込んだフォームの getConnection メソッドを呼び出します。例えば、次のようなコードを使用して、ユーザー ID およびパスワードを使用せずに Kerberos セキュリティー・メカニズムを設定します。

```
import java.sql.*;                          // JDBC base
import com.ibm.db2.jcc.*;                   // DB2 implementation of JDBC 2.0
...
Properties properties = new Properties(); // Create a Properties object
properties.put("kerberosServerPrincipal", "kdcsrv1.sj.ibm.com");
                                            // Set the Kerberos server
properties.put("gssCredential",delegatedCredential);
                                            // Set the delegated credential
properties.put("securityMechanism",
  new String("" + com.ibm.db2.jcc.DB2BaseDataSource.KERBEROS_SECURITY + ""));

                                            // Set security mechanism to
                                            // Kerberos
String url = "jdbc:db2://mvs1.sj.ibm.com:5021/san_jose";
                                            // Set URL for the data source
Connection con = DriverManager.getConnection(url, properties);
                                            // Create the connection
```

DataSource インターフェースの場合: DataSource オブジェクトを作成してデプロイする場合、 DataSource オブジェクトの作成後に、
DataSource.setKerberosServerPrincipal、DataSource.setGssCredential、および DataSource.setSecurityMechanism メソッドを呼び出して、 Kerberos サーバー、委任証明書、およびセキュリティーのメカニズムを設定します。たとえば、以下のようにします。

```
DB2DataSource db2ds = new com.ibm.db2.jcc.DB2SimpleDataSource();
                                            // Create the DataSource object
db2ds.setDriverType(4);                     // Set the driver type
db2ds.setDatabaseName("san_jose");          // Set the location
db2ds.setServerName("mvs1.sj.ibm.com");     // Set the server name
db2ds.setPortNumber(5021);                  // Set the port number
db2ds.setKerberosServerPrincipal("kdcsrv1.sj.ibm.com");
                                            // Set the Kerberos server
db2ds.setGssCredential(delegatedCredential);
                                            // Set the delegated credential
db2ds.setSecurityMechanism(
  com.ibm.db2.jcc.DB2BaseDataSource.KERBEROS_SECURITY);
                                            // Set security mechanism to
                                            // Kerberos
```

関連タスク:

- 303 ページの『DataSource インターフェースを使用したデータ・ソースへの接続』
- 345 ページの『DataSource オブジェクトの作成およびデプロイ』
- 300 ページの『DriverManager インターフェースと DB2 Universal JDBC ドライバーを使用したデータ・ソースへの接続』

関連資料:

- 408 ページの『DB2 Universal JDBC ドライバーのプロパティー』

第 20 章 JDBC および SQLJ 問題の診断

以降のセクションでは、JDBC および SQLJ の問題の診断に関する情報が含まれています。

DB2 Universal JDBC ドライバーでの JDBC および SQLJ 問題の診断

以降のセクションでは、 DB2 Universal JDBC ドライバーでの JDBC および SQLJ の問題の診断に関する情報が含まれています。

DB2 Universal JDBC ドライバーでの JDBC および SQLJ 問題の診断

DB2 Universal JDBC ドライバーでの SQLJ または JDBC 問題を診断するためのデータを入手するには、トレース・データを収集して、そのトレース・データを形式設定するユーティリティーを実行します。トレースおよび診断ユーティリティーは、 IBM® ソフトウェア・サポートの指示の下でのみ実行してください。

アプリケーションが DB2® UDB for z/OS™ or OS/390® サーバーに接続する場合、トレース・データを収集する前に、そのサーバーに多数のストアード・プロシージャーをインストールする必要があります。それらのストアード・プロシージャーは、いくつかの `DatabaseMetaData` 呼び出しにも使用されます。それらのストアード・プロシージャーは以下のとおりです。

- SQLCOLPRIVILEGES
- SQLCOLUMNS
- SQLFOREIGNKEYS
- SQLGETTYPEINFO
- SQLPRIMARYKEYS
- SQLPROCEDURECOLS
- SQLPROCEDURES
- SQLSPECIALCOLUMNS
- SQLSTATISTICS
- SQLTABLEPRIVILEGES
- SQLTABLES
- SQLUDTS
- SQLCAMESSAGE

DB2 UDB for OS/390 and z/OS バージョン 7 または DB2 UDB for OS/390 バージョン 6 では、ストアード・プロシージャーは PTF に付属しています。これらの PTF は、以下の PTF 番号を使って、通常のサービス・チャネルを通して注文することができます。

表 79. DB2 Universal Database for z/OS and OS/390 のPTF

DB2 Universal Database for z/OS and OS/390 のバージョン	PTF 番号
バージョン 6	UQ72081 および UQ72082
バージョン 7	UQ72083

DB2 UDB for z/OS システム管理者に、これらのストアード・プロシージャーがインストールされているかどうか尋ねてください。

JDBC トレース・データの収集:

以下のいずれかの手順を使用して、トレースを開始します。

手順 1:

1. DataSource インターフェースを使用してデータ・ソースに接続する場合、DB2BaseDataSource.setTraceLevel メソッドを呼び出して、必要なトレースのタイプを設定します。デフォルトのトレース・レベルは TRACE_ALL です。複数のトレース・タイプの指定方法の詳細については、 『DB2 Universal JDBC ドライバーのプロパティー』を参照してください。
2. DB2BaseDataSource.setJccLogWriter メソッドを呼び出して、トレース宛先を指定し、トレースをオンにします。

手順 2:

DataSource インターフェースを使用してデータ・ソースに接続する場合、javax.sql.DataSource.setLogWriter メソッドを呼び出して、トレースをオンにします。このメソッドでは、TRACE_ALL が使用できる唯一のトレース・レベルになります。

DriverManager インターフェースを使用してデータ・ソースに接続する場合、以下の手順に従ってトレースを開始します。

1. 必要なトレース・タイプの *info* パラメーターまたは *url* パラメーターに traceLevel プロパティーを設定して、 DriverManager.getConnection メソッドを呼び出します。デフォルトのトレース・レベルは TRACE_ALL です。複数のトレース・タイプの指定方法の詳細については、 『DB2 Universal JDBC ドライバーのプロパティー』を参照してください。
2. DriverManager.setLogWriter メソッドを呼び出して、トレース宛先を指定し、トレースをオンにします。

接続が確立されたら、トレースをオフにするかもう一度オンにするか、トレース宛先を変更するか、または DB2Connection.setJccLogWriter メソッドでトレース・レベルを変更できます。トレースをオフにするには、logWriter 値を null に設定します。

logWriter プロパティーは、タイプ java.io.PrintWriter のオブジェクトです。ご使用のアプリケーションで java.io.PrintWriter オブジェクトを処理できない場合、 traceFile プロパティーを使用して、トレース出力の宛先を指定できます。traceFile プロパティーを使用するには、 logWriter プロパティーを null に設定し、 traceFile プロパティーを、ドライバーがトレース・データを書き込むファイ

ル名に設定します。このファイルと、ファイルが存在するディレクトリーは、書き込み可能でなければなりません。ファイルがすでに存在している場合、ドライバーはそれを上書きします。

手順 3: DriverManager を使用している場合、ドライバーをロードするときに、URL の一部として traceFile および traceLevel プロパティーを指定します。たとえば、以下のようにします。

```
String url = "jdbc:db2://sysmvs1.stl.ibm.com:5021/san_jose" +
 ":traceFile=/u/db2p/jcctrace;" +
 "traceLevel=com.ibm.db2.jcc.DB2BaseDataSource.TRACE_DRDA_FLOWS;";
```

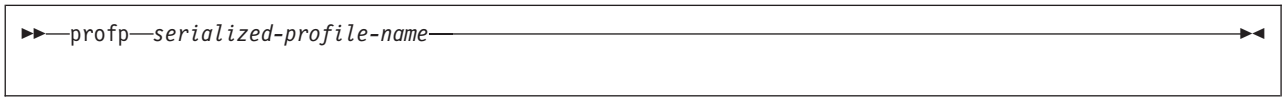
トレースのプログラム例: DB2 Universal JDBC ドライバーの使用時のトレースについての完全なプログラム例は、 『DB2 Universal JDBC ドライバー使用時のトレースの例』を参照してください。

SQLJ トレース・データの収集:

SQLJ カスタマイズ時またはバインド処理中にトレース・データを収集して問題を診断するには、 db2sqljcustomize または db2sqljbind バインド・ユーティリティーの実行時に、 -tracelevel および -tracefile オプションを指定します。

SQLJ 逐次プロファイルについての情報の形式設定:

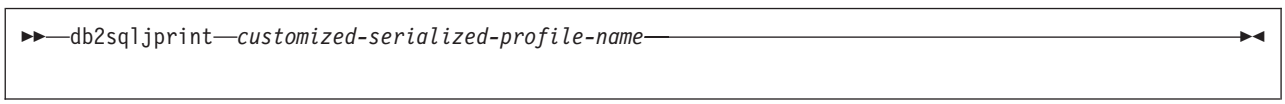
profp ユーティリティーは、逐次プロファイルの各 SQLJ 文節についての情報を形式設定します。 profp ユーティリティーの形式は、以下のとおりです。



profp ユーティリティーは、エラーが発生する接続の逐次プロファイルに対して実行します。例外が出されたら、Java™ スタック・トレースが生成されます。スタック・トレースから例外が出されたときは、どの逐次プロファイルが使用中だったのかを判別できます。

SQLJ カスタマイズ逐次プロファイルについての情報の形式設定:

db2sqljprint ユーティリティーは、 DB2 Universal JDBC ドライバー用にカスタマイズされた、逐次プロファイルの各 SQLJ 文節についての情報を形式設定します。 db2sqljprint ユーティリティーの形式は、以下のとおりです。



db2sqljprint ユーティリティーは、エラーが発生する接続のカスタマイズ逐次プロファイルに対して実行します。

関連概念:

- 500 ページの『DB2 Universal JDBC ドライバー使用時のトレースの例』

関連資料:

- 「コマンド・リファレンス」の『db2sqljcustomize - DB2 SQLJ プロファイル・カスタマイザー・コマンド』
- 「コマンド・リファレンス」の『db2sqljbind - DB2 SQLJ プロファイル・バインド・プログラム・コマンド』
- 408 ページの『DB2 Universal JDBC ドライバーのプロパティー』

DB2 Universal JDBC ドライバー使用時のトレースの例

次の例は、接続を確立し、 DB2 Universal JDBC ドライバーの使用時にトレース・データを収集して表示するためのクラスを示しています。このクラスには、 DriverManager インターフェース用のメソッドと、 DataSource インターフェース用のメソッドが含まれています。

```
public class TraceExample
{

  public static void main(String[] args)
  {
    sampleConnectUsingSimpleDataSource();
    sampleConnectWithURLUsingDriverManager();
  }

  private static void sampleConnectUsingSimpleDataSource()
  {
    java.sql.Connection c = null;
    java.io.PrintWriter printWriter =
     new java.io.PrintWriter(System.out, true);
                                      // Prints to console, true means
                                      // auto-flush so you don't lose trace
  try {
      javax.sql.DataSource ds =
       new com.ibm.db2.jcc.DB2SimpleDataSource();
      ((com.ibm.db2.jcc.DB2BaseDataSource) ds).setServerName("sysmvs1.stl.ibm.com");
      ((com.ibm.db2.jcc.DB2BaseDataSource) ds).setPortNumber(5021);
      ((com.ibm.db2.jcc.DB2BaseDataSource) ds).setDatabaseName("san_jose");
      ((com.ibm.db2.jcc.DB2BaseDataSource) ds).setDriverType(4);

     ds.setLogWriter(printWriter);    // This turns on tracing

     // Refine the level of tracing detail
     ((com.ibm.db2.jcc.DB2BaseDataSource) ds).
      setTraceLevel(com.ibm.db2.jcc.DB2SimpleDataSource.TRACE_CONNECTS |
      com.ibm.db2.jcc.DB2SimpleDataSource.TRACE_DRDA_FLOWS);

     // This connection request is traced using trace level
     // TRACE_CONNECTS | TRACE_DRDA_FLOWS
     c = ds.getConnection("myname", "mypass");

     // Change the trace level to TRACE_ALL
     // for all subsequent requests on the connection
     ((com.ibm.db2.jcc.DB2Connection) c).setJccLogWriter(printWriter,
      com.ibm.db2.jcc.DB2BaseDataSource.TRACE_ALL);
```

図 60. DB2 Universal JDBC ドライバーの使用時のトレースの例 (1/5)

```
    // The following INSERT is traced using trace level TRACE_ALL
    java.sql.Statement s1 = c.createStatement();
    s1.executeUpdate("INSERT INTO sampleTable(sampleColumn) VALUES(1)");
    s1.close();

    // This code disables all tracing on the connection
    ((com.ibm.db2.jcc.DB2Connection) c).setJccLogWriter(null);

    // The following INSERT statement is not traced
    java.sql.Statement s2 = c.createStatement();
    s2.executeUpdate("INSERT INTO sampleTable(sampleColumn) VALUES(1)");
    s2.close();

       c.close();
   }
   catch(java.sql.SQLException e) {
    com.ibm.db2.jcc.DB2ExceptionFormatter.printTrace(e,
     printWriter, "[TraceExample]");
   }
   finally {
     cleanup(c, printWriter);
     printWriter.flush();
   }
 }

 // If the code ran successfully, the connection should
 // already be closed. Check whether the connection is closed.
 // If so, just return.
 // If a failure occurred, try to roll back and close the connection.

 private static void cleanup(java.sql.Connection c,
  java.io.PrintWriter printWriter)
 {
   if(c == null) return;

   try {
    if(c.isClosed()) {
      printWriter.println("[TraceExample] " +
        "The connection was successfully closed");
       return;
    }

    // If we get to here, something has gone wrong.
    // Roll back and close the connection.
    printWriter.println("[TraceExample] Rolling back the connection");
    try {
      c.rollback();
    }
```

図 60. DB2 Universal JDBC ドライバーの使用時のトレースの例 (2/5)

```
          catch(java.sql.SQLException e) {
       printWriter.println("[TraceExample] " +
        "Trapped the following java.sql.SQLException while trying to roll back:");
       com.ibm.db2.jcc.DB2ExceptionFormatter.printTrace(e, printWriter,
      "[TraceExample]");
       printWriter.println("[TraceExample] " +
        "Unable to roll back the connection");
      }
      catch(java.lang.Throwable e) {
        printWriter.println("[TraceExample] Trapped the " +
         "following java.lang.Throwable while trying to roll back:");
        com.ibm.db2.jcc.DB2ExceptionFormatter.printTrace(e,
         printWriter, "[TraceExample]");
        printWriter.println("[TraceExample] Unable to " +
         "roll back the connection");
      }

      // Close the connection
      printWriter.println("[TraceExample] Closing the connection");
      try {
        c.close();
      }
      catch(java.sql.SQLException e) {
        printWriter.println("[TraceExample] Exception while " +
         "trying to close the connection");
        printWriter.println("[TraceExample] Deadlocks could " +
         "occur if the connection is not closed.");
        com.ibm.db2.jcc.DB2ExceptionFormatter.printTrace(e, printWriter,
         "[TraceExample]");
      }
      catch(java.lang.Throwable e) {
        printWriter.println("[TraceExample] Throwable caught " +
         "while trying to close the connection");
        printWriter.println("[TraceExample] Deadlocks could " +
         "occur if the connection is not closed.");
        com.ibm.db2.jcc.DB2ExceptionFormatter.printTrace(e, printWriter,
         "[TraceExample]");
      }
    }
    catch(java.lang.Throwable e) {
      printWriter.println("[TraceExample] Unable to " +
       "force the connection to close");
      printWriter.println("[TraceExample] Deadlocks " +
       "could occur if the connection is not closed.");
     com.ibm.db2.jcc.DB2ExceptionFormatter.printTrace(e, printWriter,
       "[TraceExample]");
    }
  }
```

図 60. DB2 Universal JDBC ドライバーの使用時のトレースの例 (3/5)

```
private static void sampleConnectWithURLUsingDriverManager()
{
  java.sql.Connection c = null;

  // This time, send the printWriter to a file.
  java.io.PrintWriter printWriter = null;
  try {
    printWriter =
      new java.io.PrintWriter(
        new java.io.BufferedOutputStream(
          new java.io.FileOutputStream("/temp/driverLog.txt"), 4096), true);
  }
  catch(java.io.FileNotFoundException e) {
    java.lang.System.err.println("Unable to establish a print writer for trace");
    java.lang.System.err.flush();
      return;
  }

  try {
    Class.forName("com.ibm.db2.jcc.DB2Driver");
  }
  catch(ClassNotFoundException e) {
    printWriter.println("[TraceExample] Universal Type 4 Connectivity " +
     "is not in the application classpath. Unable to load driver.");
    printWriter.flush();
    return;
  }

  // This URL describes the target data source for Type 4 connectivity.
  // The traceLevel property is established through the URL syntax,
  // and driver tracing is directed to file "/temp/driverLog.txt"
  String databaseURL =
   "jdbc:db2://sysmvs1.stl.ibm.com:5021" +
   "/sample:traceFile=/temp/driverLog.txt;traceLevel=" +
   "(com.ibm.db2.jcc.DB2BaseDataSource.TRACE_DRDA_FLOWS " +
   "| com.ibm.db2.jcc.DB2BaseDataSource.TRACE_CONNECTS);";

  // Set other properties
  java.util.Properties properties = new java.util.Properties();
  properties.setProperty("user", "myname");
  properties.setProperty("password", "mypass");
```

図 60. DB2 Universal JDBC ドライバーの使用時のトレースの例 (4/5)

```
    try {
     // This connection request is traced using trace level
     // TRACE_CONNECTS | TRACE_DRDA_FLOWS
     c = java.sql.DriverManager.getConnection(databaseURL, properties);

     // Change the trace level for all subsequent requests
     // on the connection to TRACE_ALL
     ((com.ibm.db2.jcc.DB2Connection) c).setJccLogWriter(printWriter,
      com.ibm.db2.jcc.DB2BaseDataSource.TRACE_ALL);

     // The following INSERT is traced using trace level TRACE_ALL
     java.sql.Statement s1 = c.createStatement();
     s1.executeUpdate("INSERT INTO sampleTable(sampleColumn) VALUES(1)");
     s1.close();

     // Disable all tracing on the connection
     ((com.ibm.db2.jcc.DB2Connection) c).setJccLogWriter(null);

      // The following SQL insert code is not traced
      java.sql.Statement s2 = c.createStatement();
      s2.executeUpdate("insert into sampleTable(sampleColumn) values(1)");
      s2.close();

      c.close();
    }
    catch(java.sql.SQLException e) {
     com.ibm.db2.jcc.DB2ExceptionFormatter.printTrace(e, printWriter,
       "[TraceExample]");
    }
    finally {
      cleanup(c, printWriter);
      printWriter.flush();
    }
  }
}
```

図 60. DB2 Universal JDBC ドライバーの使用時のトレースの例 (5/5)

関連タスク:

- 303 ページの『DataSource インターフェースを使用したデータ・ソースへの接続』
- 300 ページの『DriverManager インターフェースと DB2 Universal JDBC ドライバーを使用したデータ・ソースへの接続』

関連資料:

- 408 ページの『DB2 Universal JDBC ドライバーのプロパティー』

DB2 JDBC Type 2 ドライバーでの JDBC および SQLJ 問題の診断

以降のセクションでは、 Linux、UNIX、および Windows 用の DB2 JDBC Type 2 ドライバー (DB2 JDBC Type 2 ドライバー) での JDBC および SQLJ の問題の診断に関する情報が含まれています。

CLI/ODBC/JDBC トレース機能

このトピックでは、以下の対象について説明します。

- 505 ページの『DB2 CLI および DB2 JDBC トレースの構成』
- 506 ページの『DB2 CLI トレース・オプションおよび db2cli.ini ファイル』

- 507 ページの『DB2 JDBC トレース・オプションおよび db2cli.ini ファイル』
- 509 ページの『DB2 CLI ドライバー・トレースと ODBC Driver Manager トレースの比較』
- 509 ページの『DB2 CLI ドライバー、CLI-based Legacy Type 2 JDBC Driver、および DB2 トレース』
- 510 ページの『DB2 CLI および DB2 JDBC トレース、および CLI または Java ストアード・プロシージャー』

DB2 CLIおよび CLI-based Legacy Type 2 JDBC Driver for Linux, UNIX®, and Windows® には、広範囲のトレース機能が備えられています。デフォルトでは、これらの機能は使用不可になっており、付加的なコンピューター・リソースを使用しません。これらのトレース機能を使用可能にすると、アプリケーションが適切なドライバー (DB2 CLI または CLI-based Legacy Type 2 JDBC Driver) にアクセスしたときに、 1 つ以上のテキスト・ログ・ファイルが生成されます。これらのログ・ファイルには、以下のものに関する詳細情報が記録されています。

- CLI または JDBC 関数がアプリケーションによって呼び出された順序
- CLI または JDBC 関数との間でやり取りされた入出力パラメーターの内容
- CLI または JDBC 関数によって生成された戻りコードおよびエラーまたは警告メッセージ

DB2 CLI および DB2® JDBC トレース・ファイルを分析することにより、アプリケーション開発者にさまざまな利益をもたらします。まず、プログラム・ロジックおよびパラメーター初期化に関する微妙なエラーが、しばしばトレースで明確になります。 2 番目に、DB2 CLI および DB2 JDBC トレースから、アクセス先のアプリケーションやデータベースをより良く調整する方法が分かる場合があります。たとえば、DB2 CLI トレースで、ある表が特定の属性セットに基づいて何度も照会されていることが示されている場合は、それらの属性に対応する索引を表で作成することによって、アプリケーションのパフォーマンスを向上させることができます。最後に、DB2 CLI および DB2 JDBC トレース・ファイルの分析は、サード・パーティーのアプリケーションやインターフェースがどのように動作しているかをアプリケーション開発者が理解するのに役立ちます。

DB2 CLI および DB2 JDBC トレースの構成:

DB2 CLI および DB2 JDBC トレース機能の構成パラメーターは、いずれも DB2 CLI 構成ファイル `db2cli.ini` から読み取られます。デフォルトでは、このファイルは Windows プラットフォームでは `¥sqllib` パスにあり、 UNIX プラットフォームでは `/sqllib/cfg` パスにあります。このデフォルト・パスは、DB2CLIINIPATH 環境変数を設定することによってオーバーライドできます。Windows プラットフォームでは、ユーザー定義のデータ・ソースが ODBC Driver Manager によって定義されている場合に、付加的な `db2cli.ini` ファイルがユーザーのプロファイル (またはホーム) ディレクトリーに存在することがあります。この `db2cli.ini` ファイルは、デフォルト・ファイルをオーバーライドします。

現在の `db2cli.ini` トレース構成パラメーターをコマンド行プロセッサーから表示するには、以下のコマンドを発行します。

```
db2 GET CLI CFG FOR SECTION COMMON
```

以下の 3 つの方法で db2cli.ini ファイルを変更すれば、DB2 CLI および DB2 JDBC トレース機能を構成できます。

- DB2 構成アシスタント (使用可能な場合) を使用する
- テキスト・エディターを使用して、db2cli.ini ファイルを手動で編集する
- UPDATE CLI CFG コマンドをコマンド行プロセッサーから発行する

たとえば、以下のコマンドをコマンド行プロセッサーから発行すると、db2cli.ini ファイルが更新され、JDBC トレース機能が使用可能になります。

```
db2 UPDATE CLI CFG FOR SECTION COMMON USING jdbctrace 1
```

注:

1. 通常、DB2 CLI および DB2 JDBC トレース構成オプションは、アプリケーションが初期化されるときにのみ、db2cli.ini 構成ファイルから読み取られます。ただし、特殊な db2cli.ini トレース・オプションである TraceRefreshInterval を使用すれば、特定の DB2 CLI トレース・オプションが db2cli.ini ファイルから再読み取りされるインターバルを指定できます。
2. DB2 CLI トレース機能は、 SQL_ATTR_TRACE および SQL_ATTR_TRACEFILE 環境属性を設定することにより、動的に構成することもできます。これらの設定は、db2cli.ini ファイルに含まれている設定をオーバーライドします。

重要: 必要でなければ、DB2 CLI および DB2 JDBC トレース機能を使用不可にしてください。不必要なトレースを行うと、アプリケーションのパフォーマンスが低下し、不要なトレース・ログ・ファイルが生成される場合があります。 DB2 では、生成されたトレース・ファイルは削除されず、新しいトレース情報は既存のトレース・ファイルに付加されます。

DB2 CLI トレース・オプションおよび db2cli.ini ファイル:

DB2 CLI ドライバーを使用するアプリケーションが実行を開始すると、このドライバーは、db2cli.ini ファイルの [COMMON] セクションにトレース機能オプションがないかどうかをチェックします。これらのトレース・オプションは、db2cli.ini ファイルの [COMMON] セクションで特定の値に設定された特定のトレース・キーワードです。

注: DB2 CLI トレース・キーワードは db2cli.ini ファイルの [COMMON] セクションに存在するので、それらの値は DB2 CLI ドライバーを介したすべてのデータベース接続に適用されます。

定義可能な DB2 CLI トレース・キーワードは以下のとおりです。

- Trace
- TraceComm
- TraceErrImmediate
- TraceFileName
- TraceFlush
- TraceFlushOnError
- TraceLocks
- TracePathName

- TracePIDList
- TracePIDTID
- TraceRefreshInterval
- TraceStmtOnly
- TraceTime
- TraceTimeStamp

注: DB2 CLI トレース・キーワードは、TraceRefreshInterval キーワードが設定されていない限り、アプリケーションの初期化時に db2cli.ini ファイルから一度だけ読み取られます。このキーワードが設定されていると、指定されたインターバルで Trace および TracePIDList キーワードが db2cli.ini ファイルから再読み取りされ、適切であれば、現在実行中のアプリケーションに適用されます。

これらの DB2 CLI キーワードおよび値を使用する db2cli.ini ファイル・トレース構成の例を以下に示します。

```
[COMMON]
trace=1
TraceFileName=¥temp¥clitrace.txt
TraceFlush=1
```

注:

1. CLI トレース・キーワードでは、大文字小文字の区別がありません。ただし、パスおよびファイル名のキーワード値は、一部のオペレーティング・システム (UNIX など) で大文字小文字の区別がある場合があります。
2. db2cli.ini にある DB2 CLI トレース・キーワードか関連値が無効な場合、DB2 CLI トレース機能はそれを無視し、そのトレース・キーワードにデフォルト値を使用します。

DB2 JDBC トレース・オプションおよび db2cli.ini ファイル:

CLI-based Legacy Type 2 JDBC Driver を使用するアプリケーションが実行を開始すると、このドライバーも、db2cli.ini ファイルにトレース機能オプションがないかどうかをチェックします。 DB2 CLI トレース・オプションと同様、DB2 JDBC トレース・オプションは、 db2cli.ini ファイルの [COMMON] セクションで、キーワード/値の対として指定されます。

注: DB2 JDBC トレース・キーワードは db2cli.ini ファイルの [COMMON] セクションに存在するので、それらの値は CLI-based Legacy Type 2 JDBC Driver を介したすべてのデータベース接続に適用されます。

定義可能な DB2 JDBC トレース・キーワードは以下のとおりです。

- JDBCTrace
- JDBCTracePathName
- JDBCTraceFlush

JDBCTrace = 0 | 1

JDBCTrace キーワードは、他の DB2 JDBC トレース・キーワードがプログラム実行に影響するかどうかを制御します。 JDBCTrace をデフォルト値の

0 に設定すると、DB2 JDBC トレース機能が使用不可になります。JDBCTrace を 1 に設定すると、この機能が使用可能になります。

それだけでは、JDBCTrace キーワードは、JDBCTracePathName キーワードも指定されていない限り、ほとんど効果がなく、トレース出力も生成しません。

JDBCTracePathName = <fully_qualified_trace_path_name>

JDBCTracePathName の値は、すべての DB2 JDBC トレース情報が書き込まれるディレクトリーの完全修飾パスです。 DB2 JDBC トレース機能は、JDBC アプリケーションが CLI-based Legacy Type 2 JDBC Driver を使用して実行されるたびに、新しいトレース・ログ・ファイルを生成しようとします。マルチスレッド・アプリケーションの場合は、各スレッドごとに別個のトレース・ログ・ファイルが生成されます。トレース・ログ・ファイルの名前は、アプリケーション・プロセス ID、スレッド・シーケンス番号、およびスレッド識別ストリングを連結したものにより、自動的に付けられます。 DB2 JDBC トレース出力ログ・ファイルが書き込まれるデフォルト・パス名は存在しません。

JDBCTraceFlush = 0 | 1

JDBCTraceFlush キーワードは、トレース情報が DB2 JDBC トレース・ログ・ファイルに書き込まれる頻度を指定します。デフォルトでは、JDBCTraceFlush は 0 に設定されており、各 DB2 JDBC トレース・ログ・ファイルは、トレース対象アプリケーションまたはスレッドが正常終了するまでオープンされたままになっています。アプリケーションが異常終了すると、トレース・ログ・ファイルに書き込まれていない一部のトレース情報が失われる可能性があります。

DB2 JDBC トレース・ログ・ファイルに書き込まれるトレース情報の保全性と完全性を保証するため、 JDBCTraceFlush キーワードを 1 に設定できます。各トレース項目がトレース・ログ・ファイルに書き込まれると、DB2 JDBC ドライバーはこのファイルをクローズしてから再オープンし、新しいトレース項目をこのファイルの末尾に付加します。これにより、トレース情報が失われないことが保証されます。

注: *DB2 JDBC ログ・ファイルのクローズおよび再オープン操作が行われるたびに有効な入出力オーバーヘッドが発生するため、パフォーマンスがかなり低下する可能性があります。*

これらの DB2 JDBC キーワードおよび値を使用する `db2cli.ini` ファイル・トレース構成の例を以下に示します。

```
[COMMON]
jdbctrace=1
JdbcTracePathName=¥temp¥jdbctrace¥
JDBCTraceFlush=1
```

注:

1. JDBC トレース・キーワードでは、大文字小文字の区別がありません。ただし、パスおよびファイル名のキーワード値は、一部のオペレーティング・システム (UNIX など) で大文字小文字の区別がある場合があります。

2. db2cli.ini にある DB2 JDBC トレース・キーワードか関連値が無効な場合、DB2 JDBC トレース機能はそれを無視し、そのトレース・キーワードにデフォルト値を使用します。
3. DB2 JDBC トレースを使用可能にしても、DB2 CLI トレースは使用可能になりません。一部の CLI-based Legacy Type 2 JDBC Driver は、DB2 CLI ドライバーを使用してデータベースにアクセスします。現在、Java™ 開発者は DB2 CLI トレースを使用可能にすることにより、自分のアプリケーションがさまざまなソフトウェア層を介してデータベースと対話する方法に関する追加情報を得ることができます。 DB2 JDBC および DB2 CLI トレース・オプションは相互に依存しておらず、db2cli.ini ファイルの [COMMON] セクションで、任意の順序で指定できます。

DB2 CLI ドライバー・トレースと ODBC Driver Manager トレースの比較:

ODBC Driver Manager トレースと DB2 CLI ドライバー・トレースの違いを理解することは重要です。 ODBC Driver Manager トレースは、ODBC アプリケーションが ODBC Driver Manager に対して行った ODBC 関数呼び出しを示します。対照的に、DB2 CLI ドライバー・トレースは、ODBC Driver Manager がアプリケーションに代わって DB2 CLI ドライバーに対して行った関数呼び出しを示します。

ODBC Driver Manager は、一部の関数呼び出しをアプリケーションから DB2 CLI ドライバーに直接転送することがあります。しかし、ODBC Driver Manager は、ドライバーへの一部の関数呼び出しの転送を遅らせたり回避したりすることもあります。また、ODBC Driver Manager は、DB2 CLI ドライバーに呼び出しを転送する前に、アプリケーション関数引き数を変更したり、アプリケーション関数を他の関数にマップしたりすることもあります。

アプリケーション関数呼び出しに ODBC ドライバーが干渉する理由には、以下のものがあります。

- ODBC 3.0 では使用すべきでなくなった ODBC 2.0 関数を使用して作成されたアプリケーションでは、以前の関数が新しい関数にマップされます。
- ODBC 3.0 で使用すべきでなくなった ODBC 2.0 関数引き数は、等価の ODBC 3.0 引き数にマップされます。
- Microsoft® カーソル・ライブラリーでは、SQLExtendedFetch() などの呼び出しが、同じ終了結果を実現するように、SQLFetch() や他のサポート関数への複数の呼び出しにマップされます。
- ODBC Driver Manager の接続プールでは、通常、SQLDisconnect() 要求が据え置かれます (または、接続が再利用されたときに回避されます)。

これらおよび他の理由により、アプリケーション開発者は ODBC マネージャー・トレースを、DB2 CLI ドライバー・トレースを補完するものとして活用できます。

ODBC Driver Manager トレースの取り込みと解釈について詳しくは、ODBC Driver Manager の資料を参照してください。 Windows プラットフォームでは、Microsoft ODBC 3.0 Software Development Kit and Programmer's Reference を参照してください。これはオンライン (http://www.msdn.microsoft.com/) でも使用可能です。

DB2 CLI ドライバー、CLI-based Legacy Type 2 JDBC Driver、および DB2 トレース:

CLI-based Legacy Type 2 JDBC Driver は、内部で DB2 CLI ドライバーを使用してデータベースにアクセスします。たとえば、Java の getConnection() メソッドは、内部で CLI-based Legacy Type 2 JDBC Driver によって DB2 CLI SQLConnect() 関数にマップされます。その結果、Java 開発者は DB2 CLI トレースを、DB2 JDBC トレースを補完するものとして活用できます。

DB2 CLI ドライバーは、多くの内部関数および DB2 固有の関数を使用して作業を行います。これらの内部関数および DB2 固有の関数は、DB2 トレースに記録されます。 DB2 トレースは、IBM® サービスによる問題判別および解決を支援するためにのみ存在しているので、アプリケーション開発者にとって役に立ちません。

DB2 CLI および DB2 JDBC トレース、および CLI または Java ストアード・プロシージャー:

どのワークステーション・プラットフォームでも、DB2 CLI および DB2 JDBC トレース機能を使用すれば、DB2 CLI および DB2 JDBC ストアード・プロシージャーをトレースできます。

これ以前のセクションに載せられている DB2 CLI および DB2 JDBC トレース情報および指示は汎用的なものであり、アプリケーションとストアード・プロシージャーの両方に等しく当てはまります。しかし、データベース・サーバーのクライアントである (また、通常はデータベース・サーバーとは別個のマシンで実行される) アプリケーションとは異なり、ストアード・プロシージャーはデータベース・サーバーで実行されます。したがって、DB2 CLI または DB2 JDBC ストアード・プロシージャーをトレースするときは、以下の付加的なステップを行う必要があります。

- トレース・キーワード・オプションが、DB2 サーバーに存在する db2cli.ini ファイルで指定されていることを確かめる。
- TraceRefreshInterval キーワードがゼロ以外の正の値に設定されていない場合は、データベースの始動時 (つまり、db2start コマンドが発行されるとき) より前に、すべてのキーワードが正しく構成されていることを確かめる。データベース・サーバーが稼働しているときにトレース設定を変更すると、予測不能な結果が生じることがあります。たとえば、サーバーが稼働しているときに TracePathName が変更されると、次にストアード・プロシージャーが実行されるときに、一部のトレース・ファイルは新しいパスに書き込まれ、他のファイルは元のパスに書き込まれることがあります。整合性を保証するため、Trace または TracePIDList 以外のトレース・キーワードが変更されたときは、サーバーを再始動してください。

関連概念:

- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 1 巻*」の『db2cli.ini 初期設定ファイル』
- 511 ページの『CLI および JDBC トレース・ファイル』

関連資料:

- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLSetEnvAttr 関数 (CLI) - 環境属性の設定』
- 「コマンド・リファレンス」の『db2trc - トレース・コマンド』
- 「コマンド・リファレンス」の『GET CLI CONFIGURATION コマンド』
- 「コマンド・リファレンス」の『UPDATE CLI CONFIGURATION コマンド』

- 「*管理ガイド: パフォーマンス*」の『その他の変数』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 1 巻*」の『CLI/ODBC 構成キーワード (カテゴリー別)』

CLI および JDBC トレース・ファイル

DB2® CLI および DB2 JDBC ドライバーにアクセスするアプリケーションは、DB2 CLI および DB2 JDBC トレース機能を利用できます。これらのユーティリティーは、DB2 CLI または DB2 JDBC ドライバーにより行われたすべての機能呼び出しを、問題判別で利用するログ・ファイルに記録します。このトピックでは、トレース機能により生成されたこれらのログ・ファイルにアクセスする方法と解釈する方法について説明します。

- 『CLI および JDBC トレース・ファイルの場所』
- 512 ページの『CLI トレース・ファイルの解釈』
- 516 ページの『JDBC トレース・ファイルの解釈』

CLI および JDBC トレース・ファイルの場所:

TraceFileName キーワードが `db2cli.ini` ファイルで使用されて完全修飾ファイル名が指定されている場合、 DB2 CLI トレース・ログ・ファイルは指定された場所に作成されます。 DB2 CLI トレース・ログ・ファイル名として相対ファイル名が指定される場合は、そのファイルの場所は、オペレーティング・システムがアプリケーションの現行パスとして認識する場所に依存します。

注: アプリケーションを実行するユーザーに、指定されているパスにトレース・ログ・ファイルを書き込む十分な権限がない場合、ファイルは生成されませんが、警告にもエラーにもなりません。

TracePathName および JDBCTracePathName キーワードのうち少なくともどちらかが `db2cli.ini` ファイルで使用されて完全修飾ディレクトリーが指定されている場合、 DB2 CLI トレース・ログ・ファイルは指定された場所に作成されます。 2 つのトレース・ディレクトリーの少なくとも 1 つが相対ディレクトリーで指定される場合は、その場所は、オペレーティング・システムがアプリケーションの現行パスとして認識する場所に基づいて決定されます。

注: アプリケーションを実行するユーザーに、指定されているパスにトレース・ファイルを書き込む十分な権限がない場合、ファイルは生成されませんが、警告にもエラーにもなりません。指定されたトレース・パスが存在しない場合、そのパスは作成されません。

TracePathName および JDBCTracePathName キーワードが設定されているときには、 DB2 CLI および DB2 JDBC トレース機能は自動的にアプリケーションのプロセス ID とスレッド・シーケンス番号を使用してトレース・ログ・ファイルに名前を付けます。たとえば、スレッドが 3 つあるアプリケーションの DB2 CLI トレースは次のような DB2 CLI トレース・ログ・ファイルを生成します。`100390.0`、`100390.1`、 `100390.2`。

同様に、スレッドが 2 つある Java™ アプリケーションの DB2 JDBC トレースは、次のような JDBC トレース・ログ・ファイルを生成します。`7960main.trc`、`7960Thread-1.trc`。

注: トレース・ディレクトリーに古いトレース・ログ・ファイルと新しいトレース・ログ・ファイルの両方がある場合、ファイルのタイム・スタンプ情報を使用して最新のトレース・ファイルを見付けることができます。

DB2 CLI または DB2 JDBC トレース出力ファイルが作成されていないように思える場合、次のことを行ってください。

- トレース構成キーワードが db2cli.ini ファイルに正しくセットされていることを検査する。コマンド行プロセッサーから db2 GET CLI CFG FOR SECTION COMMON コマンドを発行すれば、簡単に検査することができます。
- db2cli.ini ファイルを更新した後、アプリケーションが確実に再始動するようにする。特に、DB2 CLI および DB2 JDBC トレース機能はアプリケーションの始動時に初期化されます。いったん初期化されると、 DB2 JDBC トレース機能は再構成できません。 DB2 CLI トレース機能は、ランタイムでも再構成できますが、アプリケーションの始動前に TraceRefreshInterval キーワードが適切に指定されている場合に限ります。

  注: Trace および TracePIDList DB2 CLI キーワードだけはランタイムに再構成できます。*他の DB2 CLI キーワード (TraceRefreshInterval を含む) を変更しても、アプリケーションを再始動しなければ反映されません。*
- TraceRefreshInterval キーワードをアプリケーションの始動前に指定し、Trace キーワードが 0 に初期設定されている場合、 DB2 CLI トレース機能が Trace キーワードの値を再度読むために十分な時間を取るようにしてください。
- TracePathName および JDBCTracePathName キーワードのうち少なくとも 1 つが使用され、トレース・ディレクトリーが指定されている場合、アプリケーションの開始前にそれらのディレクトリーが存在していることを確かめてください。
- アプリケーションに指定されたトレース・ログ・ファイルまたはトレース・ディレクトリーに対する書き込みアクセス権限があることを確かめてください。
- DB2CLIINIPATH 環境変数をチェックしてください。セットされている場合、DB2 CLI および DB2 JDBC トレース機能は db2cli.ini ファイルがこの変数で指定されている場所にあるものと想定します。
- アプリケーションが DB2 CLI ドライバーとのインターフェースに ODBC を使用している場合、SQLConnect()、 SQLDriverConnect() または SQLBrowseConnect() 関数のうちのどれかが正常に呼び出されていることを確認してください。データベース接続が正常に行われるまで、DB2 CLI トレース・ログ・ファイルには何も書き込まれません。

CLI トレース・ファイルの解釈:

DB2 CLI トレースの先頭には常に、トレースを生成したアプリケーションのプロセス ID、トレースの開始時刻、ローカル DB2 ビルド・レベルや DB2 CLI ドライバーのバージョンを示すヘッダーがあります。たとえば、次のようになります。

```
1   [ Process: 1227, Thread: 1024 ]
2   [ Date, Time:           01-27-2002 13:46:07.535211 ]
3   [ Product:              QDB2/LINUX 7.1.0 ]
4   [ Level Identifier:     02010105 ]
5   [ CLI Driver Version:   07.01.0000 ]
6   [ Informational Tokens: "DB2 v7.1.0","n000510","" ]
```

注: このセクションで使用するトレースの例では、トレースの左側に行番号を追加しています。これらの行番号は説明の助けとして追加したもので、実際の DB2 CLI トレースにはありません。

トレース・ヘッダーのすぐ次には、通常、環境と接続ハンドルの割り振りと初期化に関連したいくつかのトレース項目があります。たとえば、次のようになります。

```
7   SQLAllocEnv( phEnv=&bffff684 )
8       ---> Time elapsed - +9.200000E-004 seconds

9   SQLAllocEnv( phEnv=0:1 )
10      <--- SQL_SUCCESS   Time elapsed - +7.500000E-004 seconds

11  SQLAllocConnect( hEnv=0:1, phDbc=&bffff680 )
12      ---> Time elapsed - +2.334000E-003 seconds

13  SQLAllocConnect( phDbc=0:1 )
14      <--- SQL_SUCCESS   Time elapsed - +5.280000E-004 seconds

15  SQLSetConnectOption( hDbc=0:1, fOption=SQL_ATTR_AUTOCOMMIT, vParam=0 )
16      ---> Time elapsed - +2.301000E-003 seconds

17  SQLSetConnectOption( )
18      <--- SQL_SUCCESS   Time elapsed - +3.150000E-004 seconds

19  SQLConnect( hDbc=0:1, szDSN="SAMPLE", cbDSN=-3, szUID="", cbUID=-3,
                          szAuthStr="", cbAuthStr=-3 )
20      ---> Time elapsed - +7.000000E-005 seconds
21  ( DBMS NAME="DB2/LINUX", Version="07.01.0000", Fixpack="0x22010105" )

22  SQLConnect( )
23      <--- SQL_SUCCESS   Time elapsed - +5.209880E-001 seconds
24  ( DSN=""SAMPLE"" )

25  ( UID=" " )

26  ( PWD="*" )
```

上記のトレース例に、それぞれの DB2 CLI 関数呼び出しごとに 2 つの項目がある (たとえば、19-21 行と 22-26 行に SQLConnect() 関数呼び出しに対する項目があります) ことに注意してください。DB2 CLI トレースでは常にこうなります。最初の項目は関数呼び出しに渡された入力パラメーターを示し、2 番目の項目は関数の出力パラメーター値とアプリケーションへの戻りコードを示します。

上記のトレース例では SQLAllocEnv() 関数が正常に環境ハンドルを割り振った (phEnv=0:1) ことが 9 行目に示されています。ハンドルはその後、13 行目で正常にデータベース接続ハンドルを割り振った (phDbc=0:1) SQLAllocConnect() 関数に渡されています。次に、15 行目で SQLSetConnectOption() 関数が phDbc=0:1 接続の SQL_ATTR_AUTOCOMMIT 属性を SQL_AUTOCOMMIT_OFF (vParam=0) にセットするために使用されています。最後に、19 行目で SQLConnect() が呼び出され、ターゲット・データベース (SAMPLE) に接続しています。

21 行目の SQLConnect() 関数の入力トレース項目に含められているのは、ターゲット・データベース・サーバーのビルドおよびフィックスパックのレベルです。このトレース項目に表れている他の情報には、入力接続ストリング・キーワードおよびクライアントとサーバーのコード・ページが含まれます。たとえば、次の情報も SQLConnect() トレース項目にあったとします。

```
( Application Codepage=819, Database  Codepage=819,
  Char Send/Recv Codepage=819, Graphic Send/Recv Codepage=819,
  Application Char Codepage=819, Application Graphic Codepage=819 )
```

これは、アプリケーションとデータベース・サーバーが同じコード・ページ (819) を使用していることを意味しています。

SQLConnect() 関数の戻りトレース項目には、重要な接続情報 (上記のトレース例では 24-26 行目) も含まれています。戻り項目に表示される可能性がある追加情報には、接続に適用されるすべての PATCH1 または PATCH2 キーワード値が含まれます。たとえば、PATCH2=27,28 が db2cli.ini ファイルの COMMON セクションに指定されている場合、次の行も SQLConnect() 戻り項目に表示されます。

```
( PATCH2="27,28" )
```

環境と接続に関連したトレース項目の次は、ステートメント関連のトレース項目です。たとえば、次のようになります。

```
27  SQLAllocStmt( hDbc=0:1, phStmt=&bffff684 )
28      ---> Time elapsed - +1.868000E-003 seconds

29  SQLAllocStmt( phStmt=1:1 )
30      <--- SQL_SUCCESS   Time elapsed - +6.890000E-004 seconds

31  SQLExecDirect( hStmt=1:1, pszSqlStr="CREATE TABLE GREETING (MSG
                           VARCHAR(10))", cbSqlStr=-3 )
32      ---> Time elapsed - +2.863000E-003 seconds
33  ( StmtOut="CREATE TABLE GREETING (MSG VARCHAR(10))" )

34  SQLExecDirect( )
35      <--- SQL_SUCCESS   Time elapsed - +2.387800E-002 seconds
```

上記のトレース例では、29 行目で、データベース接続ハンドル (phDbc=0:1) が使用されてステートメント・ハンドル (phStmt=1:1) が割り振られています。それから 31 行目で、準備されていない SQL ステートメントがそのステートメント・ハンドルで実行されています。TraceComm=1 キーワードが db2cli.ini ファイルでセットされている場合、 SQLExecDirect() 関数呼び出しトレース項目に次のような追加のクライアント/サーバー通信情報が見られます。

```
SQLExecDirect( hStmt=1:1, pszSqlStr="CREATE TABLE GREETING (MSG
                           VARCHAR(10))", cbSqlStr=-3 )
    ---> Time elapsed - +2.876000E-003 seconds
( StmtOut="CREATE TABLE GREETING (MSG VARCHAR(10))" )

    sqlccsend( ulBytes - 232 )
    sqlccsend( Handle - 1084869448 )
    sqlccsend( ) - rc - 0, time elapsed - +1.150000E-004
    sqlccrecv( )
    sqlccrecv( ulBytes - 163 ) - rc - 0, time elapsed - +2.243800E-002

SQLExecDirect( )
    <--- SQL_SUCCESS   Time elapsed - +2.384900E-002 seconds
```

このトレース項目にある追加の sqlccsend() および sqlccrecv() 関数呼び出し情報に注意してください。 sqlccsend() 呼び出し情報では、クライアントからサーバーに送られたデータ量、伝送にかかった時間、およびその伝送が成功した (0 = SQL_SUCCESS) ことが示されています。sqlccrecv() 呼び出し情報には、クライアントがサーバーからの応答を待った時間と応答に含まれていたデータ量が示されています。

しばしば、DB2 CLI トレースには複数のステートメント・ハンドルがあります。ステートメント・ハンドル ID のクローズに注意すると、トレースに表示されている他のすべてのステートメント・ハンドルに関係なく、あるステートメント・ハンドルの実行パスを容易に追えます。

DB2 CLI トレースに表示されるステートメント実行パスは、通常は上記の例よりももっと複雑です。たとえば、次のようになります。

```
36  SQLAllocStmt( hDbc=0:1, phStmt=&bffff684 )
37      ---> Time elapsed - +1.532000E-003 seconds

38  SQLAllocStmt( phStmt=1:2 )
39      <--- SQL_SUCCESS   Time elapsed - +6.820000E-004 seconds

40  SQLPrepare( hStmt=1:2, pszSqlStr="INSERT INTO GREETING VALUES ( ? )",
                cbSqlStr=-3 )
41      ---> Time elapsed - +2.733000E-003 seconds
42  ( StmtOut="INSERT INTO GREETING VALUES ( ? )" )

43  SQLPrepare( )
44      <--- SQL_SUCCESS   Time elapsed - +9.150000E-004 seconds

45  SQLBindParameter( hStmt=1:2, iPar=1, fParamType=SQL_PARAM_INPUT,
                      fCType=SQL_C_CHAR, fSQLType=SQL_CHAR, cbColDef=14,
                      ibScale=0, rgbValue=&080eca70, cbValueMax=15,
                      pcbValue=&080eca4c )
46      ---> Time elapsed - +4.091000E-003 seconds

47  SQLBindParameter( )
48      <--- SQL_SUCCESS   Time elapsed - +6.780000E-004 seconds

49  SQLExecute( hStmt=1:2 )
50      ---> Time elapsed - +1.337000E-003 seconds
51  ( iPar=1, fCType=SQL_C_CHAR, rgbValue="Hello World!!!", pcbValue=14,
      piIndicatorPtr=14 )

52  SQLExecute( )
53      <--- SQL_ERROR   Time elapsed - +5.951000E-003 seconds
```

上記のトレース例では、38 行目で、データベース接続ハンドル (phDbc=0:1) が使用されて 2 番目のステートメント・ハンドル (phStmt=1:2) が割り振られています。40 行目で、1 つのパラメーター・マーカーのある SQL ステートメントがそのステートメント・ハンドルで準備されています。次に 45 行目で、適切な SQL タイプ (SQL_CHAR) の入力パラメーター (iPar=1) がそのパラメーター・マーカーにバインドされています。最後に、49 行目でステートメントが実行されています。入力パラメーター (rgbValue=″Hello World!!!″, pcbValue=14) の内容と長さの両方が、トレースの 51 行目に表示されていることに注意してください。

SQLExecute() 関数が 52 行目で失敗しています。アプリケーションが、SQLError() のような診断 DB2 CLI 関数を呼び出して失敗の原因を診断する場合、その原因はトレースに表示されます。たとえば、次のようになります。

```
54  SQLError( hEnv=0:1, hDbc=0:1, hStmt=1:2, pszSqlState=&bffff680,
              pfNativeError=&bfffee78, pszErrorMsg=&bffff280,
              cbErrorMsgMax=1024, pcbErrorMsg=&bfffee76 )
55      ---> Time elapsed - +1.512000E-003 seconds

56  SQLError( pszSqlState="22001", pfNativeError=-302, pszErrorMsg="[IBM][CLI
       Driver][DB2/LINUX] SQL0302N  The value of a host variable in the EXECUTE
```

```
       or OPEN statement is too large for its corresponding use.
       SQLSTATE=22001", pcbErrorMsg=157 )
57      <--- SQL_SUCCESS   Time elapsed - +8.060000E-004 seconds
```

56 行目で戻されているエラー・メッセージには、生成された DB2 のネイティブ・エラー・コード (SQL0302N)、そのコードに対応する sqlstate (SQLSTATE=22001) およびエラーの要旨が含まれています。この例では、エラーの原因は明らかです。31 行目で VARCHAR(10) として定義されている列に 49 行目でアプリケーションが 14 文字のストリングを挿入しようとしています。

アプリケーションが SQLError() のような診断機能を呼び出して DB2 CLI 関数の警告やエラー・コードに応答しなかったとしても、警告またはエラー・メッセージは DB2 CLI トレースに書き込まれます。しかし、そのメッセージの位置は、実際にエラーが発生したところの近くにない可能性があります。さらに、エラーまたは警告メッセージがアプリケーションにより検索されなかったことがトレースに示されます。たとえば、検索されなかった場合、次のように、上記の例のエラー・メッセージはもっと後の関連のないと思われる DB2 CLI 関数呼び出しになるまで出力されないかもしれません。

```
SQLDisconnect( hDbc=0:1 )
    ---> Time elapsed - +1.501000E-003 seconds
    sqlccsend( ulBytes - 72 )
    sqlccsend( Handle - 1084869448 )
    sqlccsend( ) - rc - 0, time elapsed - +1.080000E-004
    sqlccrecv( )
    sqlccrecv( ulBytes - 27 ) - rc - 0, time elapsed - +1.717950E-001
( Unretrieved error message="SQL0302N  The value of a host variable in the
  EXECUTE or OPEN statement is too large for its corresponding use.
  SQLSTATE=22001" )

SQLDisconnect( )
    <--- SQL_SUCCESS   Time elapsed - +1.734130E-001 seconds
```

DB2 CLI トレースの最後の部分は、トレースの前の部分で割り振ったデータベース接続や環境ハンドルの解放を示します。たとえば、次のようになります。

```
58  SQLTransact( hEnv=0:1, hDbc=0:1, fType=SQL_ROLLBACK )
59      ---> Time elapsed - +6.085000E-003 seconds
60  ( ROLLBACK=0 )

61  SQLTransact( )
        <--- SQL_SUCCESS   Time elapsed - +2.220750E-001 seconds

62  SQLDisconnect( hDbc=0:1 )
63      ---> Time elapsed - +1.511000E-003 seconds

64  SQLDisconnect( )
65      <--- SQL_SUCCESS   Time elapsed - +1.531340E-001 seconds

66  SQLFreeConnect( hDbc=0:1 )
67      ---> Time elapsed - +2.389000E-003 seconds

68  SQLFreeConnect( )
69      <--- SQL_SUCCESS   Time elapsed - +3.140000E-004 seconds

70  SQLFreeEnv( hEnv=0:1 )
71      ---> Time elapsed - +1.129000E-003 seconds

72  SQLFreeEnv( )
73      <--- SQL_SUCCESS   Time elapsed - +2.870000E-004 seconds
```

JDBC トレース・ファイルの解釈:

DB2 JDBC トレースの先頭には、常に、重要な環境変数設定値、JDK または JRE レベル、DB2 JDBC ドライバー・レベル、DB2 ビルド・レベルなどの重要なシステム情報をリストするヘッダーがあります。たとえば、次のようになります。

```
1   =============================================================
2   |    Trace beginning on 2002-1-28 7:21:0.19
3   =============================================================

4   System Properties:
5   ------------------
6   user.language = en
7   java.home = c:¥Program Files¥SQLLIB¥java¥jdk¥bin¥..
8   java.vendor.url.bug =
9   awt.toolkit = sun.awt.windows.WToolkit
10  file.encoding.pkg = sun.io
11  java.version = 1.1.8
12  file.separator = ¥
13  line.separator =
14  user.region = US
15  file.encoding = Cp1252
16  java.compiler = ibmjitc
17  java.vendor = IBM® Corporation
18  user.timezone = EST
19  user.name = db2user
20  os.arch = x86
21  java.fullversion = JDK 1.1.8 IBM build n118p-19991124 (JIT ibmjitc
                       V3.5-IBMJDK1.1-19991124)
22  os.name = Windows® NT
23  java.vendor.url = http://www.ibm.com/
24  user.dir = c:¥Program Files¥SQLLIB¥samples¥java
25  java.class.path =
          .:C:¥Program Files¥SQLLIB¥lib;C:¥Program Files¥SQLLIB¥java;
          C:¥Program Files¥SQLLIB¥java¥jdk¥bin¥
26  java.class.version = 45.3
27  os.version = 5.0
28  path.separator = ;
29  user.home = C:¥home¥db2user
30  -----------------------------------------
```

注: このセクションで使用するトレースの例では、トレースの左側に行番号を追加しています。これらの行番号は説明の助けとして追加したもので、実際の DB2 JDBC トレースにはありません。

トレース・ヘッダーのすぐ次には、通常、JDBC 環境の初期化とデータベース接続の確立に関連したいくつかのトレース項目があります。たとえば、次のようになります。

```
31  jdbc.app.DB2Driver -> DB2Driver() (2002-1-28 7:21:0.29)
32  | Loaded db2jdbc from java.library.path
33  jdbc.app.DB2Driver <- DB2Driver() [Time Elapsed = 0.01]

34  DB2Driver - connect(jdbc:db2:sample)

35  jdbc.app.DB2ConnectionTrace -> connect( sample, info, db2driver, 0, false )
                  (2002-1-28 7:21:0.59)
36  | 10: connectionHandle = 1
37  jdbc.app.DB2ConnectionTrace <- connect() [Time Elapsed = 0.16]

38  jdbc.app.DB2ConnectionTrace -> DB2Connection (2002-1-28 7:21:0.219)
39  | source = sample
40  | Connection handle = 1
41  jdbc.app.DB2ConnectionTrace <- DB2Connection
```

上記のトレース例では、31 行目で、DB2 JDBC ドライバーのロード要求がなされています。この要求が正常に戻ったことが、33 行目で報告されています。

DB2 JDBC トレース機能は、特定の Java クラスを使用してトレース情報をキャプチャーします。上記のトレース例では、それらのトレース・クラスの 1 つである DB2ConnectionTrace が、35-37 行目と 38-41 行目の 2 つのトレース項目を生成しています。

35 行目には、connect() メソッドの呼び出しとそのメソッド呼び出しの入力パラメーターが示されています。 37 行目は connect() メソッド呼び出しが正常に戻ったことを示し、36 行目にはその呼び出しの出力パラメーター (Connection handle = 1) が示されています。

JDBC トレースでは、接続関連の項目、ステートメント関連の項目が続きます。たとえば、次のようになります。

```
42  jdbc.app.DB2ConnectionTrace -> createStatement() (2002-1-28 7:21:0.219)
43  | Connection handle = 1
44  | jdbc.app.DB2StatementTrace -> DB2Statement( con, 1003, 1007 )
          (2002-1-28 7:21:0.229)
45  | jdbc.app.DB2StatementTrace <- DB2Statement() [Time Elapsed = 0.0]
46  | jdbc.app.DB2StatementTrace -> DB2Statement (2002-1-28 7:21:0.229)
47  | | Statement handle = 1:1
48  | jdbc.app.DB2StatementTrace <- DB2Statement
49  jdbc.app.DB2ConnectionTrace <- createStatement - Time Elapsed = 0.01

50  jdbc.app.DB2StatementTrace -> executeQuery(SELECT * FROM EMPLOYEE WHERE
                                  empno = 000010) (2002-1-28 7:21:0.269)
51  | Statement handle = 1:1
52  | jdbc.app.DB2StatementTrace -> execute2( SELECT * FROM EMPLOYEE WHERE
                                    empno = 000010 ) (2002-1-28 7:21:0.269)
52  | | jdbc.DB2Exception -> DB2Exception() (2002-1-28 7:21:0.729)
53  | | | 10: SQLError = [IBM][CLI Driver][DB2/NT] SQL0401N  The data types of
              the operands for the operation "=" are not compatible.
              SQLSTATE=42818
54  | | |     SQLState = 42818
55  | | |     SQLNativeCode = -401
56  | | |     LineNumber = 0
57  | | |     SQLerrmc = =
58  | | jdbc.DB2Exception <- DB2Exception() [Time Elapsed = 0.0]
59  | jdbc.app.DB2StatementTrace <- executeQuery - Time Elapsed = 0.0
```

42 行目と 43 行目では、JDBC createStatement() メソッドが接続ハンドル 1 で呼び出されたことを DB2ConnectionTrace クラスが報告しています。もう 1 つの DB2 JDBCトレース機能クラス DB2StatementTrace による報告に伴い、そのメソッド内では内部メソッド DB2Statement() が呼び出されています。 この内部メソッド呼び出しは、トレース項目内では「ネスト」されているように表示されることに注意してください。47-49 行目には、メソッドが正常に戻り、ステートメント・ハンドル 1:1 が割り振られたことが示されています。

50 行目では、SQL 照会メソッド呼び出しがステートメント 1:1 で行われていますが、その呼び出しは 52 行目で例外を生成しています。53 行目で報告されているエラー・メッセージには、生成された DB2 のネイティブ・エラー・コード (SQL0401N)、そのコードに対応する sqlstate (SQLSTATE=42818) およびエラーの要旨が含まれています。この例では、EMPLOYEE.EMPNO 列は CHAR(6) として定義されているので、照会では整数値でない値が想定されます。

関連概念:

- 504 ページの『CLI/ODBC/JDBC トレース機能』

関連資料:

- 「*管理ガイド: パフォーマンス*」の『その他の変数』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 1 巻*」の『Trace CLI/ODBC 構成キーワード』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 1 巻*」の『TraceComm CLI/ODBC 構成キーワード』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 1 巻*」の『TraceFileName CLI/ODBC 構成キーワード』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 1 巻*」の『TracePathName CLI/ODBC 構成キーワード』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 1 巻*」の『TracePIDList CLI/ODBC 構成キーワード』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 1 巻*」の『TraceRefreshInterval CLI/ODBC 構成キーワード』

第 21 章 Java 2 Platform Enterprise Edition

以下の節では、Java 2 Platform Enterprise Edition (J2EE) について説明します。

Java 2 Platform Enterprise Edition (J2EE) の概要

今日のグローバルなビジネス環境において、組織は活動範囲を広げ、コストを削減し、応答時間を短くするため、カスタマー、従業員、提供業者、および他のビジネス・パートナーが容易にアクセスできるサービスを提供しなければなりません。これらのサービスは、以下の特性を持っている必要があります。

- グローバルなビジネス環境の要件を満たすため、高い可用性を持つ
- ユーザーのプライバシーと企業の統合を保護するため、セキュリティーが保証されている
- 商取引が正確かつ迅速に処理されるように、信頼性とスケーラビリティーを持つ

ほとんどの場合、これらのサービスは、各層が特定の目的を満たす複数層アプリケーションによって提供されます。 Java™ 2 Platform Enterprise Edition を使用すれば、これらの複数層サービスの開発に伴うコストと複雑さが削減され、企業の要件に応じて素早く展開して容易に拡張できるサービスを実現できます。

Java 2 Enterprise Edition では、以下のようなエレメントとして提供されている標準アーキテクチャーを定義することにより、これらの利点が実現しています。

- Java 2 Enterprise Edition Application Model。複数層のシン・クライアント・サービスを開発するための標準的なアプリケーション・モデルです。
- Java 2 Enterprise Edition Platform。Java 2 Enterprise Edition アプリケーションをホストする標準プラットフォームです。
- Java 2 Enterprise Edition Compatibility Test Suite。ある Java 2 Enterprise Edition Platform 製品が Java 2 Enterprise Edition Platform 標準に準拠しているかどうかを検査するためのものです。
- Java 2 Enterprise Edition Reference Implementation。 Java 2 Enterprise Edition の機能デモを行い、Java 2 Enterprise Edition Platform の作動可能な定義を提供するためのものです。

関連概念:

- 521 ページの『Java 2 Platform Enterprise Edition』

Java 2 Platform Enterprise Edition

Java™ 2 Platform Enterprise Edition は、Java 2 Enterprise Edition アプリケーションをホストするランタイム環境を提供します。このランタイム環境では、Java 2 Enterprise Edition 製品がサポートしなければならない 4 つのコンポーネント・タイプが定義されています。

- アプリケーション・クライアントは、通常はデスクトップ・コンピューターで実行される GUI プログラムである、Java プログラム言語のプログラムです。アプリケーション・クライアントは、Java 2 Enterprise Edition 中間層のすべての機能にアクセスできます。
- アプレットは、通常は Web ブラウザーで実行される GUI コンポーネントですが、アプレット・プログラミング・モデルをサポートしている他のさまざまなアプリケーションや装置でも実行できます。
- サーブレット、JavaServer Page (JSP)、フィルター、および Web イベント・リスナーは、通常では Web ブラウザーで実行され、Web クライアントからの HTTP 要求に応答できます。サーブレット、JSP、およびフィルターを使用すれば、アプリケーションのユーザー・インターフェースとなる HTML ページを生成できます。また、他のアプリケーション・コンポーネントによって利用される、XML や他のフォーマットのデータも生成できます。サーブレット、JSP テクノロジーによって作成されたページ、Web フィルター、および Web イベント・リスナーは、この仕様では *Web コンポーネント* と総称されています。 Web アプリケーションは、Web コンポーネントと、HTML ページなどの他のデータで構成されています。
- Enterprise JavaBeans™ (EJB) コンポーネントは、トランザクションをサポートしている管理対象環境で実行されます。 Enterprise Bean には、通常では Java 2 Enterprise Edition アプリケーション用のビジネス・ロジックが含まれています。

上記のアプリケーション・コンポーネントは、展開方法や管理方法に基づいて、3 つのカテゴリーに分けることができます。

- Java 2 Enterprise Edition サーバーで展開、管理、および実行されるコンポーネント。
- Java 2 Enterprise Edition サーバーで展開および管理されるが、クライアント・マシンにロードされて実行されるコンポーネント。
- 展開方法と管理方法がこの仕様では完全に定義されていないコンポーネント。アプリケーション・クライアントはこのカテゴリーに含めることができます。

これらのコンポーネントの実行時サポートは、コンテナー によって提供されます。

関連概念:

- 522 ページの『Java 2 Platform Enterprise Edition コンテナー』
- 530 ページの『Enterprise Java Beans』

Java 2 Platform Enterprise Edition コンテナー

コンテナーは、基礎となる Java™ 2 Platform Enterprise Edition API のフェデレーテッド・ビューをアプリケーション・コンポーネントに提供します。典型的な Java 2 Platform Enterprise Edition 製品では、各アプリケーション・コンポーネント・タイプごとに 1 つのコンテナー (アプリケーション・クライアント・コンテナー、アプレット・コンテナー、Web コンテナー、および Enterprise Bean コンテナー) が備わっています。コンテナー・ツールは、展開用のアプリケーション・コンポーネントをパッケージするためのファイル・フォーマットを理解することもできます。

仕様によると、これらのコンテナーは、Java 2 Platform Enterprise Edition, Standard Edition V1.3.1 仕様の J2SE で定義されている、Java 互換のランタイム環境を提供する必要があります。この仕様は、それぞれの Java 2 Enterprise Edition 製品がサポートしなければならない標準サービスのセットを定義しています。それらの標準サービスは以下のとおりです。

- HTTP サービス
- HTTPS サービス
- Java トランザクション API
- リモート呼び出しメソッド
- Java IDL
- JDBC API
- Java メッセージ・サービス
- Java 命名およびディレクトリー・インターフェース
- JavaMail
- JavaBeans™ 活動化フレームワーク
- XML 構文解析用の Java API
- コネクター・アーキテクチャー
- Java 認証および許可サービス

関連概念:

- 524 ページの『Java Naming and Directory Interface (JNDI)』
- 530 ページの『Enterprise Java Beans』

Java 2 Platform Enterprise Edition サーバー

基礎となる Java™ 2 Platform Enterprise Edition コンテナーは、そのコンテナー自体が一部となっているサーバーです。通常、Java 2 Enterprise Edition Product Provider は、既存のトランザクション処理インフラストラクチャーを J2SE テクノロジーと組み合わせて使用して、Java 2 Platform Enterprise Edition サーバー・サイド機能をインプリメントします。 Java 2 Platform Enterprise Edition クライアント機能は、通常では J2SE テクノロジーに基づいて構築されています。

注: IBM® WebSphere® Application Server は、 Java 2 Platform Enterprise Edition に準拠したサーバーです。

Java 2 Enterprise Edition のデータベース要件

Java™ 2 Enterprise Edition Platform では、ビジネス・データの保管用に、JDBC API を介してアクセス可能なデータベースが必要です。このデータベースには、Web コンポーネント、Enterprise Bean、およびアプリケーション・クライアント・コンポーネントがアクセスできます。このデータベースは、アプレットからはアクセスできなくてもかまいません。

関連概念:

- 289 ページの『第 14 章 Java アプリケーション・サポートの概要』

Java Naming and Directory Interface (JNDI)

JNDI を使用すれば、Java™ プラットフォーム・ベースのアプリケーションは、複数の命名およびディレクトリー・サービスにアクセスできます。これは、Java Enterprise アプリケーション・プログラミング・インターフェース (API) セットの一部です。 JNDI により、開発者は、複数の異なる命名およびディレクトリー・サービスに対応したポータブル・アプリケーションを作成できます。対応できるものには、ファイル・システムや、Lightweight Directory Access Protocol (LDAP)、Novell Directory Services、および Network Information System (NIS) などのディレクトリー・サービス、また Common Object Request Broker Architecture (CORBA)、Java Remote Method Invocation (RMI)、および Enterprise JavaBeans™ (EJB) などの分散オブジェクト・システムがあります。

JNDI API には 2 つの部分があります。アプリケーション・コンポーネントが命名およびディレクトリー・サービスへのアクセスに使用するアプリケーション・レベル・インターフェースと、命名およびディレクトリー・サービスのプロバイダーにアタッチするためのサービス・プロバイダー・インターフェースです。

Java トランザクション管理

Java™ 2 Enterprise Edition では、分散トランザクション管理のアプリケーション・プログラミングが容易になっています。 Java 2 Enterprise Edition には、Java Transaction API および Java Transaction Service (JTS) という 2 つの仕様により、分散トランザクションのサポートが組み込まれています。 JTA は、インプリメンテーションやプロトコルに依存しない高水準 API であり、アプリケーションやアプリケーション・サーバーがトランザクションにアクセスすることを可能にします。さらに、JTA は常に使用可能になっています。

DB2 Universal JDBC ドライバーと、DB2® JDBC Type 2 Driver for Linux, UNIX® and Windows® は、 JTA および JTS 仕様をインプリメントしています。

JTA は、トランザクション・マネージャーと、分散トランザクション・システムに関与する部分 (リソース管理プログラム、アプリケーション・サーバー、およびトランザクション・アプリケーション) との間にある、標準の Java インターフェースを指定します。

JTS は、JTA をサポートし、API の下のレベルで OMG Object Transaction Service (OTS) 1.1 仕様の Java マッピングをインプリメントする、トランザクション・マネージャーのインプリメンテーションを指定します。 JTS は、IIOP を使用してトランザクションを伝搬します。

JTA と JTS により、Java 2 Enterprise Edition サーバーでは、トランザクション管理の重荷がコンポーネント開発者から取り除かれます。開発者は、設計時または展開時に、展開記述子で宣言ステートメントを使用して、EJB テクノロジー・ベースのコンポーネントのトランザクション・プロパティーを定義できます。トランザクション管理の責任は、アプリケーション・サーバーが引き受けます。

DB2 および WebSphere® Application Server 環境において、 WebSphere Application Server にはトランザクション・マネージャーの役割が想定され、 DB2 はリソース

管理プログラムとして機能します。 WebSphere Application Server および DB2 が整合分散トランザクションを提供できるように、 WebSphere Application Server は、JTS と JTA の一部をインプリメントし、 JDBC ドライバーも JTA の一部をインプリメントします。

WebSphere Application Server 環境では、JDBC ドライバーで自動的にその環境が検出されるので、 DB2 を JTA に対応するように構成する必要はありません。

DB2 JDBC Type 2 Driver には、次の 2 つの `DataSource` クラスが用意されています。

- `COM.ibm.db2.jdbc.DB2ConnectionPoolDataSource`
- `COM.ibm.db2.jdbc.DB2XADataSource`

DB2 Universal JDBC ドライバー には、次の 2 つの `DataSource` クラスが用意されています。

- `com.ibm.db2.jcc.DB2ConnectionPoolDataSource`
- `com.ibm.db2.jcc.DB2XADataSource`

WebSphere Application Server では、データベースへの DB2 接続がプールされます。アプリケーションが分散トランザクションに関与する場合は、WebSphere Application Server で DB2 データ・ソースを定義するときに `COM.ibm.db2.jdbc.DB2XADataSource` クラスを使用する必要があります。

WebSphere Application Server で DB2 を構成する方法について詳しくは、以下の URL にある WebSphere Application Server InfoCenter を参照してください。

`http://www-4.ibm.com/software/webservers/appserv/library.html`

JTA メソッドを使用する分散トランザクションの例

分散トランザクションについて説明する際の最も良い方法は、ローカル・トランザクションと対比させることです。ローカル・トランザクションの場合、JDBC アプリケーションはデータベースへの変更を永続的なものにし、以下の方法のいずれかで作業単位の終わりを示します。

- 1 つ以上の SQL ステートメントを実行した後で、 `Connection.commit` または `Connection.rollback` メソッドを呼び出します。
- アプリケーションの初めに `Connection.setAutoCommit(true)` メソッドを呼び出し、各 SQL ステートメントの後の変更をコミットします。

526 ページの図 61 では、ローカル・トランザクションを実行するコードを概説しています。

```
con1.setAutoCommit(false);  // Set autocommit off
// execute some SQL
...
con1.commit();              // Commit the transaction
// execute some more SQL
...
con1.rollback();            // Roll back the transaction
con1.setAutoCommit(true);   // Enable commit after every SQL statement
...
// Execute some more SQL, which is automatically committed after
// every SQL statement.
```

図 61. ローカル・トランザクションの例

一方、分散トランザクションに加わるアプリケーションは、分散トランザクション内で Connection.commit、Connection.rollback、または Connection.setAutoCommit(true) メソッドを呼び出すことはできません。分散トランザクションの場合、 Connection.commit または Connection.rollback メソッドはトランザクション境界を指示しません。その代わり、アプリケーションはアプリケーション・サーバーにトランザクション境界を管理させます。多くの場合、分散トランザクションには、同じデータ・ソースまたは異なるデータ・ソース (別の製造メーカーのデータ・ソースも含む) への複数の接続が関係します。

図 62 では、分散トランザクションを使用するアプリケーションが示されています。例の中のコードが実行されている間、アプリケーション・サーバーは、その同じ分散トランザクションの一部である他の EJB も実行しています。すべての EJB が utx.commit() を呼び出すと、分散トランザクション全体が、アプリケーション・サーバーによってコミットされます。何らかの EJB が正常に行われなかった場合、アプリケーション・サーバーは、その分散トランザクションに関連するすべての EJB によって行われたすべての作業をロールバックします。

```
javax.transaction.UserTransaction utx;
// Use the begin method on a UserTransaction object to indicate
// the beginning of a distributed transaction.
utx.begin();
...
// Execute some SQL with one Connection object.
// Do not call Connection methods commit or rollback.
...
// Use the commit method on the UserTransaction object to
// drive all transaction branches to commit and indicate
// the end of the distributed transaction.

utx.commit();
...
```

図 62. アプリケーション・サーバーでの分散トランザクションの例

527 ページの図 63 は、分散トランザクションを実行するために JTA メソッドを使用するプログラムを示しています。このプログラムは、トランザクション・マネージャーおよびトランザクション・アプリケーションの役割を果たします。 2 つの別々のデータ・ソースへの 2 つの接続が、単一の分散トランザクションの下で SQL 作業を行います。

```
class XASample
{
  javax.sql.XADataSource xaDS1;
  javax.sql.XADataSource xaDS2;
  javax.sql.XAConnection xaconn1;
  javax.sql.XAConnection xaconn2;
  javax.transaction.xa.XAResource xares1;
  javax.transaction.xa.XAResource xares2;
  java.sql.Connection conn1;
  java.sql.Connection conn2;

  public static void main (String args []) throws java.sql.SQLException
  {
    XASample xat = new XASample();
    xat.runThis(args);
  }
  // As the transaction manager, this program supplies the global
  // transaction ID and the branch qualifier. The global
  // transaction ID and the branch qualifier must not be
  // equal to each other, and the combination must be unique for
  // this transaction manager.
  public void runThis(String[] args)
  {
    byte[] gtrid = new byte[] { 0x44, 0x11, 0x55, 0x66 };
    byte[] bqual = new byte[] { 0x00, 0x22, 0x00 };
    int rc1 = 0;
    int rc2 = 0;

    try
    {

      javax.naming.InitialContext context = new javax.naming.InitialContext();
      /*
        * Note that javax.sql.XADataSource is used instead of a specific
        * driver implementation such as com.ibm.db2.jcc.DB2XADataSource,
        * which can be used only if this is a DB2 connection.
        */
      xaDS1 = (javax.sql.XADataSource)context.lookup("checkingAccounts");
      xaDS2 = (javax.sql.XADataSource)context.lookup("savingsAccounts");

      // The XADatasource contains the user ID and password.
      // Get the XAConnection object from each XADataSource
      xaconn1 = xaDS1.getXAConnection();
      xaconn2 = xaDS2.getXAConnection();

      // Get the java.sql.Connection object from each XAConnection
      conn1 = xaconn1.getConnection();
      conn2 = xaconn2.getConnection();

      // Get the XAResource object from each XAConnection
      xares1 = xaconn1.getXAResource();
      xares2 = xaconn2.getXAResource();
```

図 63. JTA を使用する分散トランザクションの例 (1/4)

```
      // Create the Xid object for this distributed transaction.
      // This example uses the com.ibm.db2.jcc.DB2Xid implementation
      // of the Xid interface. This Xid can be used with any JDBC driver
      // that supports JTA.
      javax.transaction.xa.Xid xid1 =
                              new com.ibm.db2.jcc.DB2Xid(100, gtrid, bqual);

      // Start the distributed transaction on the two connections.
      // The two connections do NOT need to be started and ended together.
      // They might be done in different threads, along with their SQL operations.
      xares1.start(xid1, javax.transaction.xa.XAResource.TMNOFLAGS);
      xares2.start(xid1, javax.transaction.xa.XAResource.TMNOFLAGS);
...
      // Do the SQL operations on connection 1.
      // Do the SQL operations on connection 2.
...
      // Now end the distributed transaction on the two connections.
      xares1.end(xid1, javax.transaction.xa.XAResource.TMSUCCESS);
      xares2.end(xid1, javax.transaction.xa.XAResource.TMSUCCESS);

      // If connection 2 work had been done in another thread,
      // a thread.join() call would be needed here to wait until the
      // connection 2 work is done.

      try
      { // Now prepare both branches of the distributed transaction.
        // Both branches must prepare successfully before changes
        // can be committed.
        // If the distributed transaction fails, an XAException is thrown.
        rc1 = xares1.prepare(xid1);
        if(rc1 == javax.transaction.xa.XAResource.XA_OK)
        { // Prepare was successful. Prepare the second connection.
          rc2 = xares2.prepare(xid1);
          if(rc2 == javax.transaction.xa.XAResource.XA_OK)
          { // Both connections prepared successfully and neither was read-only.
            xares1.commit(xid1, false);
            xares2.commit(xid1, false);
          }
          else if(rc2 == javax.transaction.xa.XAException.XA_RDONLY)
          { // The second connection is read-only, so just commit the
            // first connection.
            xares1.commit(xid1, false);
          }
        }
        else if(rc1 == javax.transaction.xa.XAException.XA_RDONLY)
        { // SQL for the first connection is read-only (such as a SELECT).
          // The prepare committed it. Prepare the second connection.
          rc2 = xares2.prepare(xid1);
          if(rc2 == javax.transaction.xa.XAResource.XA_OK)
          { // The first connection is read-only but the second is not.
            // Commit the second connection.
            xares2.commit(xid1, false);
          }
          else if(rc2 == javax.transaction.xa.XAException.XA_RDONLY)
          { // Both connections are read-only, and both already committed,
            // so there is nothing more to do.
          }
        }
      }
```

図 63. JTA を使用する分散トランザクションの例 (2/4)

```
catch (javax.transaction.xa.XAException xae)
  { // Distributed transaction failed, so roll it back.
    // Report XAException on prepare/commit.
    System.out.println("Distributed transaction prepare/commit failed. " +
                       "Rolling it back.");
    System.out.println("XAException error code = " + xae.errorCode);
    System.out.println("XAException message = " + xae.getMessage());
    xae.printStackTrace();
    try
    {
      xares1.rollback(xid1);
    }
    catch (javax.transaction.xa.XAException xae1)
    { // Report failure of rollback.
      System.out.println("distributed Transaction rollback xares1 failed");
      System.out.println("XAException error code = " + xae1.errorCode);
      System.out.println("XAException message = " + xae1.getMessage());
    }
    try
    {
      xares2.rollback(xid1);
    }
    catch (javax.transaction.xa.XAException xae2)
    { // Report failure of rollback.
      System.out.println("distributed Transaction rollback xares2 failed");
      System.out.println("XAException error code = " + xae2.errorCode);
      System.out.println("XAException message = " + xae2.getMessage());
    }
  }

  try
  {
    conn1.close();
    xaconn1.close();
  }
  catch (Exception e)
  {
    System.out.println("Failed to close connection 1: " + e.toString());
    e.printStackTrace();
  }
  try
  {
    conn2.close();
    xaconn2.close();
  }
  catch (Exception e)
  {
    System.out.println("Failed to close connection 2: " + e.toString());
    e.printStackTrace();
  }
}
```

図 63. JTA を使用する分散トランザクションの例 (3/4)

```
    catch (java.sql.SQLException sqe)
    {
      System.out.println("SQLException caught: " + sqe.getMessage());
      sqe.printStackTrace();
    }
    catch (javax.transaction.xa.XAException xae)
    {
      System.out.println("XA error is " + xae.getMessage());
      xae.printStackTrace();
    }
    catch (javax.naming.NamingException nme)
    {
      System.out.println(" Naming Exception: " + nme.getMessage());
    }
  }
}
```

図 63. JTA を使用する分散トランザクションの例 (4/4)

推奨: パフォーマンスを向上させるために、 1 つの分散トランザクションが完了してから、別の分散トランザクションやローカル・トランザクションを開始してください。

関連概念:

- 524 ページの『Java トランザクション管理』

Enterprise Java Beans

Enterprise Java™ Beans アーキテクチャーは、コンポーネント・ベースの分散ビジネス・アプリケーションを開発および展開するためのコンポーネント・アーキテクチャーです。 Enterprise Java Beans アーキテクチャーを使用して作成されたアプリケーションは、一度作成すれば、Enterprise Java Beans 仕様をサポートする任意のサーバー・プラットフォームで展開できます。 Java 2 Enterprise Edition アプリケーションは、Enterprise Java Beans (EJBs) を使用して、Session Bean と Entity Bean が組み込まれた、サーバー・サイドのビジネス・コンポーネントをインプリメントします。

Session Bean は、ビジネス・サービスを表し、ユーザー間で共用されません。 Entity Bean は、マルチユーザーの分散トランザクション・オブジェクトであり、永続データを表します。 EJB アプリケーションのトランザクション境界は、コンテナー管理トランザクションか Bean 管理トランザクションを指定することによって設定できます。

サンプル・プログラム `AccessEmployee.ear` は、Enterprise Java Beans を使用して、 DB2® データベースにアクセスする Java 2 Enterprise Edition アプリケーションをインプリメントします。このサンプルは `SQLLIB/samples/websphere` ディレクトリーにあります。

以下の EJB サンプル・アプリケーションは、2 つのビジネス・サービスを提供します。 1 つのサービスは、従業員の従業員番号により、ユーザーがその従業員に関する情報 (**sample** データベースの EMPLOYEE 表に保管されている) にアクセスで

きるようにします。もう 1 つのサービスは、ユーザーが従業員番号のリストを検索できるようにします。これにより、ユーザーは従業員データの照会に使用する従業員番号を取得できます。

以下のサンプルは、EJB を使用して、DB2 データベースにアクセスする Java 2 Enterprise Edition アプリケーションをインプリメントします。このサンプルは、Model-View-Controller (MVC) アーキテクチャーを使用しています。これは共通で使用される GUI アーキテクチャーです。 JSP は、View (表示コンポーネント) をインプリメントするために使用されています。サーブレットは、このサンプルで Controller (コントローラー) として機能します。これは、ワークフローを制御し、ユーザーの要求を、EJB によってインプリメントされた Model (モデル) に委任します。このサンプルの Model コンポーネントは、2 つの EJB (1 つの Session Bean と 1 つの Entity Bean) によって構成されます。 container-managed persistence (CMP) Bean である Employee は、サンプル・データベースの EMPLOYEE 表にある永続データを表す分散トランザクション・オブジェクトを表します。「コンテナー管理パーシスタンス」という用語は、Entity Bean で必要なすべてのデータベース・アクセスを EJB コンテナーが処理することを意味します。この Bean のコードには、データベース・アクセス (SQL) 呼び出しが含まれていません。その結果は、この Bean のコードは特定の永続ストレージ機構 (データベース) に束縛されません。 Session Bean である AccessEmployee は、Entity Bean のファサードとして機能し、一様なクライアント・アクセス戦略を提供します。このファサード設計により、EJB クライアントと Entity Bean の間のネットワーク・トラフィックは削減され、 EJB クライアントが Entity Bean に直接アクセスする場合よりも分散トランザクションが効率的になります。 DB2 データベースへのアクセスは、Session Bean か Entity Bean によって提供できます。このサンプル・アプリケーションの 2 つのサービスは、DB2 データベースにアクセスする両方のアプローチを示しています。以下の最初のサービスでは、Entity Bean が使用されています。

```
//=====================================================
// This method returns an employee's information by
// interacting with the entity bean located by the
// provided employee number
public EmployeeInfo getEmployeeInfo(String empNo)
throws java.rmi.RemoteException
}
Employee employee = null;
try
}
employee = employeeHome.findByPrimaryKey(new EmployeeKey(empNo));
EmployeeInfo empInfo = new EmployeeInfo(empNo);
//set the employee's information to the dependent value object
empInfo.setEmpno(employee.getEmpno());
empInfo.setFirstName (employee.getFirstName());
empInfo.setMidInit(employee.getMidInit());
empInfo.setLastName(employee.getLastName());
empInfo.setWorkDept(employee.getWorkDept());
empInfo.setPhoneNo(employee.getPhoneNo());
empInfo.setHireDate(employee.getHireDate());
empInfo.setJob(employee.getJob());
empInfo.setEdLevel(employee.getEdLevel());
empInfo.setSex(employee.getSex());
empInfo.setBirthDate(employee.getBirthDate());
empInfo.setSalary(employee.getSalary());
empInfo.setBonus(employee.getBonus());
empInfo.setComm(employee.getComm());
return empInfo;
}
```

```
catch (java.rmi.RemoteException rex)
{
......
```

従業員番号を表示する 2 番目のサービスでは、Session Bean である AccessEmployee が、 DB2 サンプル・データベースに直接アクセスします。

```
/============================================
* Get the employee number list.
* @return Collection
*/
public Collection getEmpNoList()
{
ResultSet rs = null;
PreparedStatement ps = null;
Vector list = new Vector();
DataSource ds = null;
Connection con = null;
try
{
ds = getDataSource();
con = ds.getConnection();
String schema = getEnvProps(DBschema);
String query = "Select EMPNO from " + schema + ".EMPLOYEE";
ps = con.prepareStatement(query);
ps.executeQuery();
rs = ps.getResultSet();
EmployeeKey pk;
while (rs.next())
{
pk = new EmployeeKey();
pk.employeeId = rs.getString(1);
list.addElement(pk.employeeId);
}
rs.close();
return list;
```

関連資料:

- 「*アプリケーション開発ガイド アプリケーションの構築および実行*」の『Java WebSphere のサンプル』

第 5 部 他のプログラミング・インターフェース

第 22 章 Perl でのプログラミング

Perl でのプログラミングに関する考慮事項 . . . 535
Perl の制約事項 535
Perl でのマルチスレッド・データベース・アクセス 535
Perl でのデータベース接続 536
Perl での取り出し結果 536
Perl のパラメーター・マーカー 537
Perl の SQLSTATE および SQLCODE 変数 . . . 537
Perl プログラムの例 538

Perl でのプログラミングに関する考慮事項

Perl は、数多くのオペレーティング・システムで自由に使用できるポピュラーなプログラム言語です。 Perl データベース・インターフェース (DBI) モジュール (http://www.ibm.com/software/data/db2/perl から利用可能) で DBD::DB2 ドライバー (http://www.perl.com から利用可能) を使用すると、 Perl を使った DB2® アプリケーションを作成することができます。

Perl はインタープリター言語であり、 Perl DBI モジュールは動的 SQL を使用するため、 DB2 アプリケーションのプロトタイプを素早く作成および修正する上で、 Perl は理想的な言語です。 Perl DBI モジュールは、 CLI および JDBC と大変よく似たインターフェースを使用するため、 Perl プロトタイプを簡単に CLI および JDBC に移植することができます。

大部分のデータベース・ベンダーは、 Perl DBI モジュールのデータベース・ドライバーを提供しています。これは、さまざまなデータベース・サーバーからデータにアクセスするアプリケーションを作成するために、 Perl を使用できることも意味します。たとえば、DBD::Oracle データベース・ドライバーを使用して Oracle データベースに接続する Perl DB2 アプリケーションを作成し、 Oracle データベースからデータを取り出し、 DBD::DB2 データベース・ドライバーを使用して DB2 データベースにデータを挿入することができます。

Perl の制約事項

Perl DBI モジュールがサポートするのは、動的 SQL だけです。複数回ステートメントを実行する必要がある場合には、ステートメントを準備する **prepare** 呼び出しを発行して、 Perl DB2® アプリケーションのパフォーマンスを改善することができます。

ワークステーションにインストールする DBD::DB2 ドライバー・バージョンの制限に関する最新情報については、 DBD::DB2 パッケージにある CAVEATS ファイルを参照してください。

Perl でのマルチスレッド・データベース・アクセス

Perl ではマルチスレッド・データベース・アクセスはサポートされていません。

Perl でのデータベース接続

Perl が DBI モジュールをロードできるようにするには、 DB2® アプリケーションに次の行を含める必要があります。

```
use DBI;
```

DBI モジュールは、**DBI->connect** ステートメントを使用してデータベース・ハンドル を作成すると、 DBD::DB2 ドライバーを自動的にロードします。以下の構文を使用します。

```
my $dbhandle = DBI->connect('dbi:DB2:dbalias', $userID, $password);
```

ここで、各パラメーターは以下のとおりです。

$dbhandle
connect ステートメントが戻すデータベース・ハンドル。

dbalias
DB2 データベース・ディレクトリーにカタログされている DB2 別名

$userID
データベースへの接続で使用するユーザー ID

$password
データベースへの接続で使用するユーザー ID のパスワード

Perl での取り出し結果

Perl DBI モジュールは動的 SQL しかサポートしていないため、 Perl DB2 アプリケーションではホスト変数は使用しないでください。

手順:

SQL 照会から結果を戻すには、以下のステップを実行します。

1. DBI->connect ステートメントを使用してデータベースに接続することにより、データベース・ハンドルを作成します。
2. 作成したデータベース・ハンドルからステートメント・ハンドルを作成します。たとえば、ストリング引き数として SQL ステートメントと一緒に **prepare** を呼び出し、データベース・ハンドルからステートメント・ハンドル *$sth* を戻すことができます。以下に示す Perl ステートメントは、その例です。

   ```
   my $sth = $dbhandle->prepare(
      'SELECT firstnme, lastname
         FROM employee '
      );
   ```

3. ステートメント・ハンドルで **execute** を呼び出して、 SQL ステートメントを実行します。 **execute** が正常に呼び出されると、 結果セットがステートメント・ハンドルに関連付けられます。 たとえば、以下の Perl ステートメントを使用すると、前の例で準備したステートメントを実行できます。

   ```
   #Note: $rc represents the return code for the execute call
   my $rc = $sth->execute();
   ```

4. **fetchrow()** への呼び出しを使用して、ステートメント・ハンドルに関連付けられた結果セットから行を取り出します。 Perl DBI は、列ごとに値を 1 つ指定

した配列として行を戻します。たとえば、以下の Perl ステートメントを使用すると、前の例にあるステートメント・ハンドルからすべての行を戻すことができます。

```
while (($firstnme, $lastname) = $sth->fetchrow()) {
   print "$firstnme $lastname¥n";
}
```

関連概念:

- 536 ページの『Perl でのデータベース接続』

Perl のパラメーター・マーカー

指定したフィールドごとに別々の入力値を使用して、準備したステートメントを実行できるようにするため、 Perl DBI モジュールはパラメーター・マーカーを使ってステートメントを準備し、実行します。 SQL ステートメントにパラメーター・マーカーを入れるには、疑問符 (?) 文字を使用します。

以下の Perl コードは、 SELECT ステートメントの WHERE 節のパラメーター・マーカーを受け入れるステートメント・ハンドルを作成します。その後、このコードは、入力値 25000 および 35000 を使用して 2 度ステートメントを実行し、パラメーター・マーカーを置換します。

```
my $sth = $dbhandle->prepare(
   'SELECT firstnme, lastname
      FROM employee
      WHERE salary > ?'
   );

my $rc = $sth->execute(25000);

⋮
my $rc = $sth->execute(35000);
```

Perl の SQLSTATE および SQLCODE 変数

Perl DBI のデータベース・ハンドルまたはステートメント・ハンドルに関連付けられた SQLSTATE を戻すには、 **state** メソッドを呼び出します。たとえば、データベース・ハンドル $dbhandle に関連付けられた SQLSTATE を戻すには、次の Perl ステートメントをアプリケーションに組み込みます。

```
my $sqlstate = $dbhandle->state;
```

Perl DBI のデータベース・ハンドルまたはステートメント・ハンドルに関連付けられた SQLCODE を戻すには、 **err** メソッドを呼び出します。 Perl DBI のデータベース・ハンドルまたはステートメント・ハンドルに関連付けられた SQLCODE のメッセージを戻すには、 **errstr** メソッドを呼び出します。たとえば、データベース・ハンドル $dbhandle に関連付けられた SQLCODE を戻すには、次の Perl ステートメントをアプリケーションに組み込みます。

```
my $sqlcode = $dbhandle->err;
```

Perl プログラムの例

以下に、Perl で作成されたアプリケーションの例を示します。

```
#!/usr/bin/perl
use DBI;

my $database='dbi:DB2:sample';
my $user='';
my $password='';

my $dbh = DBI->connect($database, $user, $password)
   or die "Can't connect to $database: $DBI::errstr";

my $sth = $dbh->prepare(
   q{ SELECT firstnme, lastname
      FROM employee }
   )
   or die "Can't prepare statement: $DBI::errstr";

my $rc = $sth->execute
   or die "Can't execute statement: $DBI::errstr";

print "Query will return $sth->{NUM_OF_FIELDS} fields.¥n¥n";
print "$sth->{NAME}->[0]: $sth->{NAME}->[1]¥n";

while (($firstnme, $lastname) = $sth->fetchrow()) {
   print "$firstnme: $lastname¥n";
}

# check for problems which may have terminated the fetch early
warn $DBI::errstr if $DBI::err;

$sth->finish;
$dbh->disconnect;
```

第 23 章 REXX でのプログラミング

REXX でのプログラミングに関する考慮事項 . . . 539
REXX の言語制限 540
REXX の言語制限 540
REXX における SQLEXEC、SQLDBS、および SQLDB2 の登録 540
REXX でのマルチスレッド・データベース・アクセス 541
REXX の日本語または中国語 (繁体字) EUC に関する考慮事項 541
REXX アプリケーションでの組み込み SQL . . . 541
REXX のホスト変数 543
REXX のホスト変数 543
REXX でのホスト変数名 544
REXX でのホスト変数の参照 544
REXX の標識変数 544
事前定義された REXX 変数 545
REXX の LOB ホスト変数 547
REXX における LOB ロケーター宣言の構文 547
REXX における LOB ファイル参照宣言の構文 548
REXX での LOB ホスト変数のクリア 549
REXX でのカーソル 549
REXX でサポートされている SQL データ・タイプ 549
REXX の実行要件 551
REXX アプリケーションの構築と実行 551
REXX のバインド・ファイル 552
REXX の API 構文 553
REXX からのストアード・プロシージャーの呼び出し 554
REXX でのストアード・プロシージャー . . . 554
REXX におけるストアード・プロシージャーの呼び出し 555
REXX におけるクライアントによるストアード・プロシージャーの呼び出しに関する考慮事項 556
REXX におけるサーバーによるストアード・プロシージャーの呼び出しに関する考慮事項 . . . 556
SQLDA 10 進フィールドの精度と SCALE 値の検索 556

REXX でのプログラミングに関する考慮事項

特定のホスト言語によるプログラミングの考慮事項について、以降のセクションで説明します。組み込み SQL ステートメント、言語制限、およびサポートされるホスト変数のデータ・タイプについての情報が含まれます。

注: REXX のサポートは DB2 バージョン 5 において確立され、今後 REXX のサポートを拡張する予定はありません。たとえば、REXX は、表名などの SQL オブジェクト ID を処理できません。これは 18 バイトより長いからです。 19 ～ 128 バイト長の表名などの、バージョン 5 より後の DB2 に追加された機能を使用したい場合には、 REXX 以外の言語でアプリケーションを作成する必要があります。

REXX はインタープリター言語なので、プリコンパイラー、コンパイラー、またはリンカーを使用しません。その代わり、3 つの DB2 API を使用して、REXX の DB2 アプリケーションを作成します。 DB2 のさまざまなエレメントをアクセスするには、以下の API を使用してください。

SQLEXEC
SQL 言語をサポートします。

SQLDBS
DB2 API のコマンド形式のバージョンをサポートします。

SQLDB2
コマンド行プロセッサーへの REXX 特定のインターフェースをサポートします。このインターフェースを使用する際の制限および詳細については、REXX の API 構文の説明を参照してください。

関連概念:

- 553 ページの『REXX の API 構文』

REXX の言語制限

以下の節では、REXX の言語制限について説明します。

REXX の言語制限

ステートメントまたはコマンドにトークンを含めることができます。それは、REXX 変数に対応できる SQLEXEC、SQLDBS、および SQLDB2 ルーチンに渡されます。この場合、REXX インタープリターは SQLEXEC または SQLDB2 を呼び出す前に、変数の値を置換します。

この状態を避けるには、ステートメントのストリングを、引用符 (' ' または " ") で囲んでください。引用符で囲まない場合、変数名と同じトークンがあると、そのトークンは REXX インタープリターによって解決され、SQLEXEC、SQLDBS、または SQLDB2 ルーチンには渡されません。

REXX/SQL では、コンパウンド SQL はサポートされません。

REXX/SQL ストアード・プロシージャーは、 Windows® オペレーティング・システムでサポートされていますが、 AIX® ではサポートされていません。

関連タスク:

- 540 ページの『REXX における SQLEXEC、SQLDBS、および SQLDB2 の登録』

REXX における SQLEXEC、SQLDBS、および SQLDB2 の登録

アプリケーションで、DB2 API のいずれかを使用する前、または SQL ステートメントを発行する前に、SQLDBS、SQLDB2、および SQLEXEC ルーチンを登録しなければなりません。この登録は、REXX インタープリターに REXX/SQL の入り口点を知らせます。 Windows ベースのプラットフォームと AIX プラットフォームでは、登録のための方法が少し異なります。

手順:

正しい構文でそれぞれのルーチンを登録するには、以下の例を使用します。

Windows ベースのプラットフォームでの登録のサンプル

```
/* ------------ Register SQLDBS with REXX  -------------------------*/
If Rxfuncquery('SQLDBS')  <> 0 then
    rcy = Rxfuncadd('SQLDBS','DB2AR','SQLDBS')
If rcy ¥= 0 then
    do
      say 'SQLDBS was not successfully added to the REXX environment'
      signal rxx_exit
    end

/* ------------ Register SQLDB2 with REXX  -------------------------*/
If Rxfuncquery('SQLDB2')  <> 0 then
    rcy = Rxfuncadd('SQLDB2','DB2AR','SQLDB2')
```

```
If rcy ¥= 0 then
    do
      say 'SQLDB2 was not successfully added to the REXX environment'
      signal rxx_exit
    end

/* ----------------- Register SQLEXEC with REXX  --------------------*/
If Rxfuncquery('SQLEXEC') <> 0 then
    rcy = Rxfuncadd('SQLEXEC','DB2AR','SQLEXEC')
If rcy ¥= 0 then
    do
      say 'SQLEXEC was not successfully added to the REXX environment'
      signal rxx_exit
    end
```

AIX での登録のサンプル

```
/* ------------ Register SQLDBS, SQLDB2 and SQLEXEC with REXX --------*/
rcy = SysAddFuncPkg("db2rexx")
If rcy ¥= 0 then
  do
    say 'db2rexx was not successfully added to the REXX environment'
    signal rxx_exit
  end
```

Windows ベースのプラットフォームでは、RxFuncAdd コマンドはすべてのセッションにつき 1 回だけ実行する必要があります。

AIX では、SysAddFuncPkg をすべての REXX/SQL アプリケーションで実行しなければなりません。

RXfuncadd API および SysAddFuncPkg API の詳細については、それぞれ Windows ベースのプラットフォームおよび AIX 版の REXX の資料に記載されています。

REXX でのマルチスレッド・データベース・アクセス

REXX ではマルチスレッド・データベース・アクセスはサポートされていません。

REXX の日本語または中国語 (繁体字) EUC に関する考慮事項

REXX アプリケーションは、日本語や中国語 (繁体字) の EUC 環境ではサポートされません。

REXX アプリケーションでの組み込み SQL

REXX アプリケーションは API を使用することにより、データベース・マネージャーAPI および SQL により提供される機能の大部分を使用できるようになります。 コンパイル言語で作成されたアプリケーションとは異なり、 REXX アプリケーションはプリコンパイルされません。その代わりに、動的 SQL ハンドラーがすべての SQL ステートメントを処理します。 REXX と呼び出し可能な API を組み合わせることにより、データベース・マネージャー機能の大部分にアクセスできます。組み込み SQL を使用する API の中には REXX が直接サポートしていないものもありますが、 DB2® コマンド行プロセッサーを使用すれば、 REXX アプリケーションからこれらの API にアクセスできます。

REXX はインタープリター言語なので、コンパイル済みホスト言語に比べてアプリケーションのプロトタイプの開発やデバッグが容易です。 REXX でコーディングされた DB2 アプリケーションは、コンパイル言語を使用した DB2 アプリケーションほどの性能は持ちませんが、プリコンパイル、コンパイル、リンクを行わず、またはさらに別のソフトウェアを使用せずに DB2 アプリケーションを作成する機能を提供できます。

SQL ステートメントの処理には、SQLEXEC ルーチンを使用してください。
SQLEXEC ルーチンの文字ストリング引き数は以下のエレメントから成ります。
- SQL キーワード
- 事前定義 ID
- ステートメント・ホスト変数

有効な SQL ステートメントを SQLEXEC ルーチンに渡すことにより、それぞれの要素を要求します。次の構文を使用してください。

```
CALL SQLEXEC 'statement'
```

SQL ステートメントは複数行に渡って継続が可能です。ステートメントのそれぞれの部分は単一引用符で囲み、以下に示すように、追加ステートメントのテキストとの区切りとしてコンマを使用してください。

```
CALL SQLEXEC 'SQL text',
             'additional text',
                  .
                  .
                  .
             'final text'
```

以下に、REXX での組み込み SQL の例を示します。

```
statement = "UPDATE STAFF SET JOB = 'Clerk' WHERE JOB = 'Mgr'"
CALL SQLEXEC 'EXECUTE IMMEDIATE :statement'
IF ( SQLCA.SQLCODE < 0) THEN
   SAY 'Update Error:  SQLCODE = ' SQLCA.SQLCODE
```

この例では、更新が正常に行われたかどうかを判断するために、 SQLCA 構造体の SQLCODE フィールドがチェックされています。

組み込み SQL ステートメントには、以下の規則が適用されます。
- 次の組み込み SQL ステートメントは、SQLEXEC ルーチンに直接渡すことができる。
  - CALL
  - CLOSE
  - COMMIT
  - CONNECT
  - CONNECT TO
  - CONNECT RESET
  - DECLARE
  - DESCRIBE
  - DISCONNECT
  - EXECUTE
  - EXECUTE IMMEDIATE
  - FETCH

- FREE LOCATOR
- OPEN
- PREPARE
- RELEASE
- ROLLBACK
- SET CONNECTION

他の SQL ステートメントは、SQLEXEC ルーチンとともに EXECUTE IMMEDIATE ステートメント、または PREPARE と EXECUTE ステートメントを使用して動的に処理される。

- REXX では、CONNECT および SET CONNECTION ステートメントでホスト変数を使用することはできない。
- カーソル名およびステートメント名は、次のように事前定義される。

  c1 ～ c100
  WITH HOLD オプションを使用せずに宣言した、範囲が *c1* ～ *c50* のカーソルのカーソル名、および WITH HOLD オプションを使用して宣言した、範囲が *c51* ～ *c100* のカーソルのカーソル名です。

  カーソル名の ID は、DECLARE、OPEN、FETCH、および CLOSE ステートメントに使用されます。これは、SQL 要求で使用されるカーソルを識別します。

  s1 ～ s100
  範囲が *s1* ～ *s100* のステートメント名です。

  ステートメント名の ID は、DECLARE、DESCRIBE、PREPARE、および EXECUTE ステートメントに使用されます。

  カーソル名およびステートメント名には、事前定義 ID を使用しなければならない。その他の名前は認められません。
- カーソルを宣言する際には、DECLARE ステートメント内のカーソル名とステートメント名が対応していなければならない。たとえば、*c1* をカーソル名として使用する場合、ステートメント名には *s1* を使用しなければなりません。
- SQL ステートメント内ではコメントを使用しない。

REXX のホスト変数

以下の節では、REXX プログラムでホスト変数を宣言して使用する方法について説明します。

REXX のホスト変数

ホスト変数は、SQL ステートメント内で参照される REXX の言語変数です。これにより、アプリケーションは入力データを DB2 に渡し、また DB2 から出力データを受け取ることができます。 REXX アプリケーションは LOB ロケーターおよび LOB ファイル参照変数を除き、ホスト変数を宣言する必要はありません。ホスト変数のデータ・タイプおよびサイズは、変数の参照のランタイムに決定されます。以下の節では、ホスト変数の命名と使用の際に従う規則について説明します。

関連概念:

- 544 ページの『REXX でのホスト変数名』
- 544 ページの『REXX でのホスト変数の参照』
- 544 ページの『REXX の標識変数』
- 547 ページの『REXX の LOB ホスト変数』
- 549 ページの『REXX での LOB ホスト変数のクリア』

関連資料:

- 545 ページの『事前定義された REXX 変数』
- 547 ページの『REXX における LOB ロケーター宣言の構文』
- 548 ページの『REXX における LOB ファイル参照宣言の構文』
- 549 ページの『REXX でサポートされている SQL データ・タイプ』

REXX でのホスト変数名

正しく命名された REXX 変数は、すべてホスト変数として使用できます。変数名の長さは 64 文字までです。ピリオドを変数名の最後の文字として使用しないでください。ホスト変数名には、英字、数字、および @、_、!、.、?、$ といった文字を使用できます。

REXX でのホスト変数の参照

REXX インタープリターは、プロシージャー内のストリングで引用符で囲まれていないものをすべて検査します。ストリングが現行の REXX 変数プール内の変数を表している場合には、 REXX がそのストリングを現行値と置き換えます。以下に、REXX におけるホスト変数の参照方法を示します。

```
CALL SQLEXEC 'FETCH C1 INTO :cm'
SAY 'Commission = ' cm
```

文字ストリングが数値データ・タイプに変換されないようにするため、ストリングを以下の例のように単一引用符で囲んでください。

```
VAR = '100'
```

REXX は、3 バイトの文字ストリング 100 に変数 *VAR* をセットします。単一引用符がストリングの一部になっている場合は、次の例に従ってください。

```
VAR = "'100'"
```

CHARACTER フィールドに数値データを挿入する場合、REXX インタープリターは、数値データを整数データと見なします。したがって、数値ストリングを明示的に連結して、単一引用符で囲む必要があります。

REXX の標識変数

REXX における標識変数のデータ・タイプは、小数点を持たない数です。以下に、INDICATOR キーワードを使用した REXX における標識変数の例を示します。

```
CALL SQLEXEC 'FETCH C1 INTO :cm INDICATOR :cmind'
IF ( cmind < 0 )
   SAY 'Commission is NULL'
```

上記の例では、cmind が負の値かどうか検査されます。負の値ではない場合、アプリケーションは cm の戻り値を使用することができます。 負の値の場合、取り出される値は NULL で、cm は使用されません。この場合、データベース・マネージャーはホスト変数の値を変更しません。

事前定義された REXX 変数

SQLEXEC、SQLDBS および SQLDB2 は一定の操作の結果として、事前定義 REXX 変数をセットします。 それらの変数は以下のとおりです。

RESULT

各操作により、戻りコードがセットされます。可能な値は以下のとおりです。

n *n* は、フォーマットされたメッセージのバイト数を示す正の値です。この値を戻すのは GET ERROR MESSAGE API のみです。

0 API が実行されています。REXX 変数 SQLCA には、API の完了状況が含まれます。 SQLCA.SQLCODE がゼロでない場合は、その値に関連したテキスト・メッセージが SQLMSG に含まれます。

-1 API を完了するために十分なメモリーがありません。要求されたメッセージは戻されません。

-2 SQLCA.SQLCODE が 0 にセットされます。メッセージは戻されません。

-3 SQLCA.SQLCODE に無効な SQLCODE が含まれています。メッセージは戻されません。

-6 SQLCA REXX 変数が作成できません。これは、十分なメモリーがないか、または何らかの理由で REXX 変数プールが使用できないということを示します。

-7 SQLMSG REXX 変数が作成できません。これは、十分なメモリーがないか、または何らかの理由で REXX 変数プールが使用できないということを示します。

-8 REXX 変数プールから SQLCA.SQLCODE REXX 変数を取り出すことができません。

-9 取り出しの際に、SQLCA.SQLCODE REXX 変数の切り捨てが行われました。この変数の長さは最大 5 バイトまでです。

-10 SQLCA.SQLCODE REXX 変数を、ASCII から有効な長整数に変換できません。

-11 REXX 変数プールから SQLCA.SQLERRML REXX 変数を取り出すことができます。

-12 取り出しの際に、SQLCA.SQLERRML REXX 変数の切り捨てが行われました。この変数の長さは最大 2 バイトまでです。

-13 SQLCA.SQLERRML REXX 変数を、ASCII から有効な短整数に変換できません。

-14 REXX 変数プールから SQLCA.SQLERRMC REXX 変数を取り出すことができません。

-15 取り出しの際に、SQLCA.SQLERRMC REXX 変数の切り捨てが行われました。この変数の長さは最大 70 バイトまでです。

-16 エラー・テキストに指定された REXX 変数をセットできません。

-17 REXX 変数プールから SQLCA.SQLSTATE REXX 変数を取り出すことができません。

-18 取り出しの際に、SQLCA.SQLSTATE REXX 変数の切り捨てが行われました。この変数の長さは最大 2 バイトまでです。

注: -8 ～ -18 の値を戻すのは、GET ERROR MESSAGE API のみです。

SQLMSG

SQLCA.SQLCODE が 0 でない場合、この値にはエラー・コードに関連したテキスト・メッセージが含まれます。

SQLISL

分離レベル。可能な値は以下のとおりです。

RR 反復可能読み取り。

RS 読み取り固定。

CS カーソル固定。これがデフォルトです。

UR 非コミット読み取り。

NC コミットなし。(NC は、一部のホスト、AS/400、または iSeries サーバーでしかサポートされていません。)

SQLCA

SQL ステートメントの処理の後に更新された SQLCA 構造体と、DB2 API が呼び出されます。

SQLRODA

CALL ステートメントを使用して呼び出される、ストアード・プロシージャーの入出力 SQLDA 構造体です。データベース・アプリケーション・リモート・インターフェース (DARI) API を使用して呼び出される、出力 SQLDA ストアード・プロシージャーでもあります。

SQLRIDA

データベース・アプリケーション・リモート・インターフェース (DARI) API を使用して呼び出される、ストアード・プロシージャーの入力 SQLDA 構造体です。

SQLRDAT

データベース・アプリケーション・リモート・インターフェース (DARI) API を使用して呼び出される、サーバー・プロシージャーの SQLCHAR 構造体です。

関連資料:

- 「*管理 API リファレンス*」の『SQLCA』
- 「*管理 API リファレンス*」の『SQLCHAR』
- 「*管理 API リファレンス*」の『SQLDA』

REXX の LOB ホスト変数

REXX ホスト変数に LOB 列を取り出してくる場合、この列は単純 (つまり、カウントはされない) ストリングとして保管されます。これは、文字ベースの SQL タイプ (たとえば、CHAR、VARCHAR、GRAPHIC、LONG など) すべてと同じ方法で処理されます。入力では、ホスト変数の内容のサイズが 32K を超える場合、または以下に説明する基準を満たしている場合には、適切な LOB タイプが割り当てられます。

REXX SQL では、以下に示すホスト変数のストリングの内容により、LOB タイプが決定されます。

ホスト変数のストリングの内容	使用される LOB タイプ
:hv1='通常の引用符付きストリングが 32K を超えている'	CLOB
:hv2="'組み込み区切り引用符 "," 付きのストリングが 32K を超えている'"	CLOB
:hv3="G'G で開始する組み込み区切り単一引用符 "," 付きの DBCS が 32K を超えている'"	DBCLOB
:hv4="BIN'BIN で開始する組み込み区切り単一引用符 "," 付きのストリングが任意の長さである'"	BLOB

REXX における LOB ロケーター宣言の構文

以下の図は、REXX における LOB ロケーター・ホスト変数の宣言の構文を示しています。

REXX における LOB ロケーター・ホスト変数の構文

アプリケーション内で LOB ロケーター・ホスト変数を宣言しなければなりません。これらの宣言が出てくると、REXX/SQL はプログラムのそれ以降の部分で、宣言されたホスト変数をロケーターとして取り扱います。ロケーターの値は、内部形式で REXX 変数に保管されます。

以下に例を示します。

```
CALL SQLEXEC 'DECLARE :hv1, :hv2 LANGUAGE TYPE CLOB LOCATOR'
```

エンジンから戻された LOB により表現されるデータは、以下に示す形式の FREE LOCATOR ステートメントを使用して、 REXX/SQL 内で解放することができます。

FREE LOCATOR ステートメントの構文

以下に例を示します。

```
CALL SQLEXEC 'FREE LOCATOR :hv1, :hv2'
```

REXX における LOB ファイル参照宣言の構文

アプリケーション内で LOB ファイル参照ホスト変数を宣言しなければなりません。これらの宣言が出てくると、REXX/SQL はプログラムのそれ以降の部分で、宣言されたホスト変数を LOB ファイル参照として取り扱います。

以下の図は、REXX における LOB ファイル参照ホスト変数の宣言の構文を示しています。

REXX ファイル参照宣言

以下に例を示します。

```
CALL SQLEXEC 'DECLARE :hv3, :hv4 LANGUAGE TYPE CLOB FILE'
```

REXX におけるファイル参照変数には、3 つのフィールドが含まれます。上記の例では、以下のものがその 3 つのフィールドに相当します。

hv3.FILE_OPTIONS.
アプリケーションによりセットされ、ファイルの使用法を指示します。

hv3.DATA_LENGTH.
DB2 によりセットされ、ファイルのサイズを指示します。

hv3.NAME.
アプリケーションにより、LOB ファイルの名前に設定されます。

FILE_OPTIONS の場合は、アプリケーションが以下のキーワードを設定します。

キーワード (整数値)
意味

READ (2) ファイルが入力に使用されます。オープン、読み取り、クローズできるのは、正規のファイルです。ファイル内のデータの長さは、(アプリケーションの要求側のコードにより) ファイルのオープン時に計算されます。

CREATE (8) 出力において、新しいファイルを作成します。ファイルがすでに存在している場合はエラーとなります。ファイルの長さ (バイト単位) は、ファイル参照変数構造の DATA_LENGTH フィールドに戻されます。

OVERWRITE (16)
出力において、ファイルがすでに存在する場合はそれを上書きし、そうでない場合は新しいファイルを作成します。ファイルの長さ (バイト単位) は、ファイル参照変数構造の DATA_LENGTH フィールドに戻されます。

APPEND (32) ファイルがすでに存在する場合はそこに出力が追加され、そうでな

い場合は新しいファイルが作成されます。ファイルに追加されるデータ (ファイル全体の長さではない) の長さ (バイト単位) は、ファイル参照変数構造の DATA_LENGTH フィールドに戻されます。

注: REXX では、ファイル参照ホスト変数はコンパウンド変数です。したがって、NAME、NAME_LENGTH、および FILE_OPTIONS フィールドは、宣言するだけでなく、値も設定しなければなりません。

REXX での LOB ホスト変数のクリア

Windows® ベースのプラットフォームでは、プログラムの終了後も効力を持つ REXX SQL LOB ロケーターおよびファイル参照ホスト変数宣言を、明示的にクリアしなければならない場合があります。これは、実行中のセッションがクローズされるまでアプリケーション・プロセスが終了しないためです。 REXX SQL LOB 宣言がクリアされないと、 LOB アプリケーションの実行後に同一セッション内で実行されている他のアプリケーションの妨げになります。

宣言をクリアする構文を示します。

```
CALL SQLEXEC "CLEAR SQL VARIABLE DECLARATIONS"
```

このステートメントは、LOB アプリケーションの終端にコーディングしなければなりません。直前のアプリケーションで宣言がクリアされていない場合があるので、宣言をクリアするための回避的な手段として、このステートメントを任意の場所にコーディングできます (たとえば、REXX SQL アプリケーションの最初)。

REXX でのカーソル

REXX においてカーソルを宣言する際、カーソルは照会と関連付けられます。照会は、PREPARE ステートメントで割り当てられたステートメント名と関連付けられます。参照したホスト変数はパラメーター・マーカーで表されます。以下の例は、動的 SELECT ステートメントに関連する DECLARE ステートメントを示しています。

```
prep_string = "SELECT TABNAME FROM SYSCAT.TABLES WHERE TABSCHEMA = ?"
CALL SQLEXEC 'PREPARE S1 FROM :prep_string';
CALL SQLEXEC 'DECLARE C1 CURSOR FOR S1';
CALL SQLEXEC 'OPEN C1 USING :schema_name';
```

関連資料:

- 549 ページの『REXX でサポートされている SQL データ・タイプ』

REXX でサポートされている SQL データ・タイプ

特定の定義済み REXX のデータ・タイプは、 DB2 の列のタイプに対応しています。以下の表は、 SQLEXEC および SQLDBS が REXX 変数をどのように解釈して、その内容を DB2 のデータ・タイプに変換するかを示しています。

注: DB2 ホスト言語で、DATALINK データ・タイプをサポートするホスト変数はありません。

表 80. REXX 宣言にマップされる SQL 列タイプ

SQL 列タイプ[1]	REXX データ・タイプ	SQL 列タイプ記述
SMALLINT (500 または 501)	小数点を持たない -32 768 ～ 32 767 の数	16 ビットの符号付き整数
INTEGER (496 または 497)	小数点を持たない -2 147 483 648 ～ 2 147 483 647 の数	32 ビットの符号付き整数
REAL[2] (480 または 481)	-3.40282346 x 10^{38} ～ 3.40282346 x 10^{38} の浮動小数の数	単精度浮動小数点
DOUBLE[3] (480 または 481)	-1.79769313 x 10^{308} ～ 1.79769313 x 10^{308} の浮動小数の数	倍精度浮動小数点
DECIMAL(*p*,*s*) (484 または 485)	小数点を持つ数	パック 10 進数
CHAR(*n*) (452 または 453)	前後に引用符 (') を持つストリング。2 つの引用符を除くと、長さが *n* になる。 先行および後続ブランク、または浮動小数の E 以外の非数値文字を持つ、長さ *n* のストリング	長さが *n* の固定長文字ストリング (*n* の範囲は 1 ～ 254 まで)
VARCHAR(*n*) (448 または 449)	CHAR(*n*) と等しい	長さが *n* の可変長文字ストリング。 *n* の範囲は 1 ～ 4000 まで
LONG VARCHAR (456 または 457)	CHAR(*n*) と等しい	長さが *n* の可変長文字ストリング。 *n* の範囲は 1 ～ 32 700 まで
CLOB(*n*) (408 または 409)	CHAR(*n*) と等しい	長さが *n* のラージ・オブジェクト可変長文字ストリング。 *n* の範囲は 1 ～ 2 147 483 647 まで
CLOB ロケーター変数[4] (964 または 965)	DECLARE *:var_name* LANGUAGE TYPE CLOB LOCATOR	サーバー上の CLOB エンティティーを識別する
CLOB ファイル参照変数[4] (920 または 921)	DECLARE *:var_name* LANGUAGE TYPE CLOB FILE	CLOB データを含むファイルの記述子
BLOB(*n*) (404 または 405)	前後にアポストロフィーを持つストリングで、BIN が先行する。先行の BIN と 2 つのアポストロフィーを除くと、*n* 文字となる	長さが *n* のラージ・オブジェクト可変長バイナリー・ストリング。 *n* の範囲は 1 ～ 2 147 483 647 まで
BLOB ロケーター変数[4] (960 または 961)	DECLARE *:var_name* LANGUAGE TYPE BLOB LOCATOR	サーバー上の BLOB エンティティーを識別する
BLOB ファイル参照変数[4] (916 または 917)	DECLARE *:var_name* LANGUAGE TYPE BLOB FILE	BLOB データを含むファイルの記述子
DATE (384 または 385)	CHAR(*10*) と等しい	10 バイトの文字ストリング
TIME (388 または 389)	CHAR(*8*) と等しい	8 バイトの文字ストリング
TIMESTAMP (392 または 393)	CHAR(*26*) と等しい	26 バイトの文字ストリング
注: 以下のデータ・タイプは、DBCS 環境でのみ使用可能です。		
GRAPHIC(*n*) (468 または 469)	前後にアポストロフィーを持つストリングで、G または N が先行する。先行の文字と 2 つのアポストロフィーを除くと、 *n* 個の DBCS 文字となる	長さが *n* の固定長 GRAPHIC ストリング。 *n* の範囲は 1 ～ 127 まで
VARGRAPHIC(*n*) (464 または 465)	GRAPHIC(*n*) と等しい	長さが *n* の可変長 GRAPHIC ストリング。 *n* の範囲は 1 ～ 2 000 まで
LONG VARGRAPHIC (472 または 473)	GRAPHIC(*n*) と等しい	長さが *n* の長可変長 GRAPHIC ストリング。 *n* の範囲は、1 ～ 16 350 まで

表 80. REXX 宣言にマップされる SQL 列タイプ (続き)

SQL 列タイプ[1]	REXX データ・タイプ	SQL 列タイプ記述
DBCLOB(*n*) (412 または 413)	GRAPHIC(*n*) と等しい	長さが *n* のラージ・オブジェクト可変長 GRAPHIC ストリング。 *n* の範囲は 1 ～ 1 073 741 823 まで
DBCLOB ロケーター変数[4] (968 または 969)	DECLARE *:var_name* LANGUAGE TYPE DBCLOB LOCATOR	サーバー上の DBCLOB エンティティーを識別する
DBCLOB ファイル参照変数[4] (924 または 925)	DECLARE *:var_name* LANGUAGE TYPE DBCLOB FILE	DBCLOB データを含むファイルの記述子

注:

1. **列タイプ**の最初の番号は、標識変数が提供されないことを示し、2 番目の番号は標識変数が提供されることを示す。標識変数は、NULL 値を示したり、切り捨てられたストリングの長さを保持するのに必要です。
2. FLOAT(*n*)。ここで 0 < *n* < 25 の場合、REAL と同義。 SQLDA での REAL と DOUBLE の違いは長さの値です (4 または 8)。
3. 以下の SQL タイプは、DOUBLE と同義です。
   - FLOAT
   - FLOAT(*n*)。ここで、*n* の取る範囲は 24 < *n* < 54。
   - DOUBLE PRECISION
4. これは列タイプではなく、ホスト変数タイプである。

関連概念:

- 549 ページの『REXX でのカーソル』

REXX の実行要件

以下の節では、REXX の実行要件について説明します。

REXX アプリケーションの構築と実行

REXX アプリケーションではプリコンパイル、およびリンクは行われません。以下の指示では、Windows オペレーティング・システムおよび AIX オペレーティング・システムで REXX アプリケーションを構築して実行する方法について説明しています。

制約事項:

Windows ベースのプラットフォームの場合、アプリケーション・ファイルの拡張子は .CMD になっている必要があります。この拡張子を付けると、アプリケーションをオペレーティング・システムのコマンド・プロンプトから直接実行できます。AIX 上で、アプリケーション・ファイルには、任意の拡張子を付けることができます。

手順:

REXX アプリケーションを構築して実行するには、以下のようにします。

- Windows オペレーティング・システムの場合、アプリケーション・ファイルには、任意の名前を付けることができます。 ファイルを作成後、次のように REXX インタープリターを起動して、アプリケーションをオペレーティング・システムのコマンド・プロンプトから実行できます。

```
REXX file_name
```

- AIX では、アプリケーションは、次の 2 つの方法で実行することができます。
  - シェル・コマンド・プロンプトで、rexx name と入力する。 name は REXX プログラムの名前です。
  - REXX プログラムの最初の行に「マジック・ナンバー」(#!) が含まれており、それが REXX/6000 インタープリターの常駐するディレクトリーを識別する場合は、シェル・コマンド・プロンプトでその名前を入力すれば、 REXX プログラムを実行することができます。たとえば、 REXX/6000 インタープリター・ファイルが /usr/bin ディレクトリーにある場合は、次の行を、REXX プログラムの最初の行として組み込みます。

    ```
    #! /usr/bin/rexx
    ```

    そうすれば、シェル・コマンド・プロンプトで次のコマンドを入力することによって、プログラムを実行可能にできます。

    ```
    chmod +x name
    ```

    シェル・コマンド・プロンプトでファイル名を入力することによって、 REXX プログラムを実行します。

注: AIX では、LIBPATH 環境変数をセットして、 REXX SQL ライブラリー db2rexx があるディレクトリーを含めます。以下に例を示します。

```
export LIBPATH=/lib:/usr/lib:/usr/lpp/db2_08_01/lib
```

REXX のバインド・ファイル

REXX アプリケーションをサポートするために、5 つのバインド・ファイルが提供されています。それらのファイルの名前は、DB2UBIND.LST ファイルの中に組み込まれています。各バインド・ファイルは、それぞれ別々の分離レベルを使用してプリコンパイルされます。したがって、5 つの異なるパッケージがデータベース内に保管されます。

以下に、5 つのバインド・ファイルを示します。

DB2ARXCS.BND
カーソル固定の分離レベルをサポートします。

DB2ARXRR.BND
反復可能読み取りの分離レベルをサポートします。

DB2ARXUR.BND
非コミット読み取りの分離レベルをサポートします。

DB2ARXRS.BND
読み取り固定の分離レベルをサポートします。

DB2ARXNC.BND
コミットなしの分離レベルをサポートします。この分離レベルは、一部のホスト、AS/400、または iSeries データベース・サーバーを作動させる際に使用されます。他のデータベースでは、非コミット読み取りの分離レベルと同様に動作します。

注: これらのファイルを、データベースに明示的にバインドしなければならない場合もあります。

SQLEXEC ルーチンを使用する場合は、カーソル固定により作成されたパッケージがデフォルトのパッケージとして使用されます。他の分離レベルを使用する必要がある場合は、データベースに接続する前に SQLDBS CHANGE SQL ISOLATION LEVEL API を使用して分離レベルを変更できます。分離レベルを変更すると、その後に続く SQLEXEC ルーチンに対する呼び出しが、新たに指定した分離レベルと関連付けられます。

Windows ベースの REXX アプリケーションは、セッション内の他の REXX プログラムの設定が変更されていないことを確認しない限り、デフォルトの分離レベルが有効であると見なしません。 REXX アプリケーションは、データベースに接続する前に分離レベルを明示的に設定しなければなりません。

REXX の API 構文

DB2 API を呼び出すには、次の構文により SQLDBS ルーチンを使用します。

```
CALL SQLDBS 'command string'
```

SQLDBS ルーチンを使用しても、使用する DB2® API を呼び出せない場合、REXX アプリケーション内から DB2 コマンド行プロセッサー (CLP) を呼び出して、その API を呼び出せます。ただし、DB2 CLP は、標準出力装置または特定のファイルのどちらかに出力するように命令します。その理由で、REXX アプリケーションは、呼び出された DB2 API からその出力に直接アクセスできません。また、呼び出された API が正常に処理されたかどうかを容易には判断することもできません。 SQLDB2 API は DB2 CLP へのインターフェースを提供しています。そのインターフェースは、各呼び出し後にコンパウンドの REXX 変数である SQLCA を設定して、呼び出された API ごとに処理が正常だったか失敗だったかを、直接 REXX アプリケーションにフィードバックできます。

SQLDB2 ルーチンを使用して、次の構文により DB2 API を呼び出すことができます。

```
CALL SQLDB2 'command string'
```

`'command string'` は、コマンド行プロセッサー (CLP) で処理可能なストリングです。

SQLDB2 を使用して DB2 API を呼び出すことは、次の場合の他は CLP を直接呼び出すのと同じです。

- 実行可能 CLP の呼び出しが SQLDB2 の呼び出しで置換された場合 (他のすべての CLP オプションとパラメーターは、同じように指定されている)。
- SQLDB2 の呼び出し後に REXX のコンパウンド変数 SQLCA が設定されたが、実行可能 CLP の呼び出し後には設定されなかった場合。
- SQLDB2 を呼び出した時、CLP のディスプレイ出力のデフォルトはオフに設定されていますが、実行可能 CLP を呼び出す場合、ディスプレイ出力はオンに設定されています。 CLP のディスプレイ出力をオンに切り替えるには、 SQLDB2 に `+o` または `-o-` オプションを渡します。

SQLDB2 の呼び出し後、REXX 変数だけが SQLCA に設定されるので、 DB2 API を呼び出すには、このルーチンだけを使用してください。 DB2 API は SQLDBS

インターフェースですが、SQLCA 以外のデータを戻しませんし、現在のところ組み込まれてはいません。 したがって SQLDB2 では、以下の DB2 API だけがサポートされています。

- データベースの活動化 (Activate Database)
- ノードの追加 (Add Node)
- DB2 バージョン 1 用バインド (Bind for DB2 Version 1)(1) (2)
- DB2 バージョン 2 または 5 用バインド (Bind for DB2 Version 2 or 5)(1)
- ノードでのデータベース作成 (Create Database at Node)
- ノードでのデータベースのドロップ (Drop Database at Node)
- ノードのドロップの検査 (Drop Node Verify)
- データベースの非活動化 (Deactivate Database)
- 登録取り消し (Deregister)
- ロード (Load)(3)
- 照会のロード (Load Query)
- プログラムのプリコンパイル (Precompile Program)(1)
- パッケージの再バインド (Rebind Package)(1)
- データベース・パーティション・グループの再分散 (Redistribute Database Partition Group)
- 登録 (Register)
- データベース・マネージャーの開始 (Start Database Manager)
- データベース・マネージャーの停止 (Stop Database Manager)

SQLDB2 がサポートする DB2 API に関する注:

1. これらのコマンドは、SQLDB2 インターフェースを介して CONNECT ステートメントを必要とします。 SQLDB2 インターフェースを使用した接続では、SQLEXEC インターフェースに接続できません。また、SQLEXEC インターフェースを使用した接続では、SQLDB2 インターフェースに接続できません。
2. SQLDB2 インターフェースを介して Windows® ベースのプラットフォームでサポートされます。
3. Load API 用の任意指定の出力パラメーター `pLoadInfoOut` は、 REXX のアプリケーションに戻りません。

注: SQLDB2 ルーチンは、上にリストされた DB2 API のためだけに使用されるように意図されていますが、 SQLDBS ルーチンを通じてサポートされていない、他の DB2 API のためにも使用することができます。あるいは、REXX アプリケーション内から CLP を介してアクセスすることが可能です。

REXX からのストアード・プロシージャーの呼び出し

以下の節では、REXX アプリケーションからストアード・プロシージャーを呼び出す方法について説明します。

REXX でのストアード・プロシージャー

REXX SQL アプリケーションは、SQL CALL ステートメントを使用して、データベース・サーバーでストアード・プロシージャーを呼び出せます。ストアード・プロシージャーは、AIX® システム上の REXX を除き、そのサーバー上でサポートされる任意の言語で作成することができます。 (クライアント・アプリケーションは

AIX 上の REXX で作成できますが、他の言語と同様、AIX 上の REXX で作成されたストアード・プロシージャーを呼び出すことはできません。)

関連概念:

• 555 ページの『REXX におけるストアード・プロシージャーの呼び出し』

REXX におけるストアード・プロシージャーの呼び出し

CALL ステートメントを使用すると、クライアント・アプリケーションがサーバー・ストアード・プロシージャーとの間でデータをやりとりできるようになります。入出力データ用のインターフェースは、ホスト変数のリストになっています。REXX は一般に、ホスト変数のタイプとサイズをその内容に基づいて判別するため、 CALL に渡される出力専用変数は予期出力と同じタイプとサイズを持つダミーの データを用いて初期化されます。

データは、CALL ステートメントの USING DESCRIPTOR 構文を使用し、 SQLDA REXX 変数を介して、ストアード・プロシージャーに渡されます。以下の表に SQLDA が設定される方法を示します。表の中の ':value' は、アプリケーションに必要な値を含む REXX ホスト変数のステムです。 DESCRIPTOR の場合、'n' は SQLDA の特定の *sqlvar* エレメントを示す数値です。右側の数字は、表の後の注番号を示します。

表 81. CALL ステートメントを使用したストアード・プロシージャーの、クライアント側の REXX SQLDA

USING DESCRIPTOR	:value.SQLD	1	
	:value.n.SQLTYPE	1	
	:value.n.SQLLEN	1	
	:value.n.SQLDATA	1	2
	:value.n.SQLDIND	1	2

注:

1. ストアード・プロシージャーを呼び出す前に、クライアント・アプリケーションは適切なデータを使用して REXX 変数を初期化しなければならない。

   SQL CALL ステートメントが実行されると、データベース・マネージャーはストレージを割り振り、 REXX 変数プールから REXX 変数の値を取り出します。 CALL ステートメントで使用される SQLDA では、データベース・マネージャーは、SQLTYPE および SQLLEN 値に基づいて、ストレージを SQLDATA および SQLIND フィールドに割り振ります。

   REXX ストアード・プロシージャーの場合 (つまり、呼び出されるプロシージャーが Windows® ベースの REXX で作成されている)、クライアントにより CALL ステートメントまたは DARI API のいずれかのタイプから渡されるデータは、以下の事前定義名を使用してデータベース・サーバーの REXX 変数プールに入れられます。

   SQLRIDA
   REXX 入力 SQLDA 変数の事前定義名

   SQLRODA
   REXX 出力 SQLDA 変数の事前定義名

2. ストアード・プロシージャーが終了すると、データベース・マネージャーはストアード・プロシージャーからも変数の値を取り出す。それらの値はクライアント・アプリケーションに戻され、クライアントの REXX 変数プールの中に置かれます。

関連概念:

- 556 ページの『REXX におけるクライアントによるストアード・プロシージャーの呼び出しに関する考慮事項』
- 556 ページの『REXX におけるサーバーによるストアード・プロシージャーの呼び出しに関する考慮事項』
- 556 ページの『SQLDA 10 進フィールドの精度と SCALE 値の検索』

関連資料:

- 「*SQL リファレンス 第 2 巻*」の『CALL ステートメント』

REXX におけるクライアントによるストアード・プロシージャーの呼び出しに関する考慮事項

CALL ステートメントでホスト変数を使用するときは、サーバー・プロシージャーからホスト変数に戻されるデータすべてと互換性のあるタイプの値に、各ホスト変数を初期化してください。この初期化は、対応する標識が負であっても実行しなければなりません。

記述子を使用するときは、SQLDATA を初期化して、サーバー・プロシージャーから戻されるデータすべてと互換性のあるタイプのデータを含める必要があります。この初期化は、SQLIND フィールドに負の値が入っていても実行しなければなりません。

関連資料:

- 549 ページの『REXX でサポートされている SQL データ・タイプ』

REXX におけるサーバーによるストアード・プロシージャーの呼び出しに関する考慮事項

事前定義された出力 sqlda SQLRODA の SQLDATA フィールドおよび SQLIND (NULL 可能タイプの場合) のすべてが初期化されていることを確認してください。たとえば、SQLRODA.SQLD が 2 の場合、(たとえ、対応する標識が負で、データがクライアントに戻されなくても) 次のフィールドには同じデータが入っているはずです。

- SQLRODA.1.SQLDATA
- SQLRODA.2.SQLDATA

SQLDA 10 進フィールドの精度と SCALE 値の検索

REXX プログラムで SQLDA 出力を初期化した場合、データベース・マネージャーから戻された SQLDA 構造体から 10 進フィールドの精度とスケール値を取り出すには、 `sqllen.scale` 値と `sqllen.precision` 値を使用します。 以下に例を示します。

```
.
.
.
/* INITIALIZE ONE ELEMENT OF OUTPUT SQLDA */
io_sqlda.sqld = 1
io_sqlda.1.sqltype = 485           /* DECIMAL DATA TYPE */
io_sqlda.1.sqllen.scale  = 2       /* DIGITS RIGHT OF DECIMAL POINT */
io_sqlda.1.sqllen.precision  = 7   /* WIDTH OF DECIMAL  */
io_sqlda.1.sqldata = 00000.00      /* HELPS DEFINE DATA FORMAT */
io_sqlda.1.sqlind = -1             /* NO INPUT DATA */
.
.
.
```

第 24 章 DB2 WebSphere MQ 関数を使用したアプリケーションの作成

WebSphere MQ 機能の概要 559
WebSphere MQ メッセージング 561
WebSphere MQ 関数を使用したメッセージ送信 . . 564
WebSphere MQ 関数を使用したメッセージ取り出し 566
WebSphere MQ アプリケーション間の接続 . . . 568
WebSphere MQ 関数を使用した要求/応答通信 . . 569
WebSphere MQ 関数を使用したパブリッシュ/サブスクライブ 571

WebSphere MQ 機能の概要

WebSphere® MQ は、メッセージ操作やデータベース操作を結合する機能を提供します。場合によっては、単一作業単位のアトミック・トランザクションとして結合することもできます。この機能は UNIX® および Windows® での WebSphere MQ でサポートされ、スキーマ DB2MQ および DB2MQ1C を使った非トランザクションおよびトランザクション MQ ユーザー定義関数も使用できます。

DB2® にはさまざまな WebSphere MQ 機能が付属しており、 SQL ステートメントにメッセージング操作を含めることができます。つまり、サポートされるすべての言語 (C、Java™、SQL など) で書かれたアプリケーションは、任意のデータベース・インターフェースを使ってこの機能を利用できます。以下のすべての例は SQL です。この SQL は、さまざまな標準的な方法を使って他のプログラム言語から使用できます。上記のような WebSphere MQ メッセージング・スタイルがすべてサポートされます。

基本的な構成では、 WebSphere MQ サーバーは DB2 と一緒にデータベース・サーバー・マシン上に配置されます。 WebSphere MQ 機能は DB2 にインストールされ、 WebSphere MQ サーバーへのアクセスを提供します。 DB2 クライアントは、DB2 サーバーにアクセスできる任意のマシンに配置することができます。複数のクライアントがデータベースを介して同時に WebSphere MQ 機能にアクセスできます。提供される機能を利用して、DB2 クライアントは SQL ステートメント内でメッセージング操作を実行できます。このようなメッセージング操作により、DB2 アプリケーションは互いの間、または他の WebSphere MQ アプリケーションとの間で通信することができます。

DB2 データベースで WebSphere MQ 機能を使用可能にするには、**enable_MQFunctions** コマンドを使用します。このコマンドは簡単なデフォルト構成を自動的に設定し、クライアント・アプリケーションは追加の管理アクションを必要とせずにこれを利用することができます。詳しい説明は、「enable_MQFunctions」および「disable_MQFunctions」のトピックを参照してください。デフォルト構成を使用して、アプリケーション・プログラマーは簡単なインターフェースをすばやく開発することができます。さらに、必要に応じて追加機能を構成していくことができます。

デフォルト構成を使って簡単なメッセージを送信するには、以下の SQL ステートメントを使用します。

例 1:

```
VALUES DB2MQ.MQSEND('simple message')
```

これによって、 『simple message』 というメッセージが WebSphere MQ キュー・マネージャーおよびデフォルト構成で指定されたキューに送られます。

WebSphere MQ の Application Messaging Interface (AMI) は、メッセージング・アクションと、これらのアクションの実行方法を指示する定義とを明確に分離します。これらの定義は外部リポジトリー・ファイルに保管され、 AMI 管理ツールを使って管理されます。これにより、AMI アプリケーションの開発と保守が簡単になります。 DB2 Universal Database™ に付属の WebSphere MQ 機能は、 AMI WebSphere MQ インターフェースに基づいています。 AMI では、構成情報を保管するための AMI リポジトリーという外部構成ファイルを使用できます。デフォルト構成には、 DB2 Universal Database で使用するよう構成された WebSphere MQ AMI リポジトリーも含まれます。

WebSphere MQ AMI の 2 つの主要な概念であるサービス・ポイントとポリシーは、 DB2 の WebSphere MQ 機能にも取り込まれています。サービス・ポイントとは、メッセージが送受信される論理上のエンドポイントです。 AMI リポジトリーでは、WebSphere MQ キュー名とキュー・マネージャーを使って各サービス・ポイントが定義されます。ポリシーは、特定のメッセージング操作に使われるサービス・オプションの品質を定義します。サービスの主な品質には、メッセージの優先順位と持続性があります。デフォルトのサービス・ポイントとポリシー定義が提供されており、開発者はこれを使用して、アプリケーションをさらに単純化することができます。 559 ページの 『例 1』 に含まれる例を以下のように書き換えて、デフォルトのサービス・ポイントおよびポリシー名を明示的に指定できます。

例 2:
```
VALUES DB2MQ.MQSEND('DB2.DEFAULT.SERVICE',
                    'DB2.DEFAULT.POLICY',
                    'simple message')
```

キューは、キューやアプリケーションが格納されているサーバー上で 1 つまたは複数のアプリケーションによって処理できます。多くの構成では、さまざまなアプリケーションや目的をサポートするために、複数のキューが定義されます。このため、 WebSphere MQ 要求を出すときにはさまざまなサービス・ポイントを定義することがしばしば重要になります。以下の例では、この点が示されています。

例 3:
```
VALUES DB2MQ.MQSEND('ODS_Input', 'simple message')
```

この例では、ポリシーが指定されていないので、デフォルト・ポリシーが使用されます。

制限

送信または受信を行う関数を使用するとき、メッセージ・タイプ VARCHAR の最大長は、スキーマ DB2MQ の場合は 4000 文字、スキーマ DB2MQ1C の場合は 32,000 文字です。タイプ CLOB のメッセージを送信または受信するときの最大長は、 DB2MQ スキーマの場合は 1 MB です。これらの値は、 MQPublish を使ってメッセージをパブリッシュするときの最大メッセージ・サイズでもあります。

CLOB メッセージと VARCHAR メッセージの両方を扱う場合には、さまざまな関数が必要になることがあります。 MQ 関数では、一般的に CLOB と VARCHAR は同じ構文を使用します。唯一の違いは、名前の終わりに CLOB が付くことです。たとえば、MQREAD に対応する CLOB 用の関数は MQREADCLOB です。

エラー・コード

WebSphere MQ 関数の戻りコードは、「MQSeries® アプリケーション・メッセージング・インターフェース」マニュアルの『付録 B』にあります。コンポーネント ID AMI を使用する他のメッセージは、「DB2 Universal Database メッセージおよびコード」 にリストされています。

関連概念:

- 19 ページの『MQSeries 使用可能性』
- 561 ページの『WebSphere MQ メッセージング』
- 564 ページの『WebSphere MQ 関数を使用したメッセージ送信』
- 566 ページの『WebSphere MQ 関数を使用したメッセージ取り出し』
- 568 ページの『WebSphere MQ アプリケーション間の接続』
- 569 ページの『WebSphere MQ 関数を使用した要求/応答通信』
- 571 ページの『WebSphere MQ 関数を使用したパブリッシュ/サブスクライブ』
- 「*IBM DB2 Information Integrator アプリケーション開発者向けガイド*」の『DB2 内で WebSphere MQ 機能を使用する方法』

関連タスク:

- 「*アプリケーション開発ガイド アプリケーションの構築および実行*」の『DB2 WebSphere MQ 関数のセットアップ』

関連資料:

- 「*SQL 管理ルーチン*」の『MQSEND スカラー関数』
- 「*SQL 管理ルーチン*」の『MQRECEIVE スカラー関数』
- 「*SQL 管理ルーチン*」の『MQREAD スカラー関数』
- 「*SQL 管理ルーチン*」の『MQPUBLISH スカラー関数』
- 「*SQL 管理ルーチン*」の『MQSUBSCRIBE スカラー関数』
- 「*SQL 管理ルーチン*」の『MQUNSUBSCRIBE スカラー関数』
- 「*SQL 管理ルーチン*」の『MQREADALL 表関数』
- 「*SQL 管理ルーチン*」の『MQRECEIVEALL 表関数』
- 「*SQL 管理ルーチン*」の『MQREADCLOB スカラー関数』
- 「*SQL 管理ルーチン*」の『MQRECEIVECLOB スカラー関数』
- 「*SQL 管理ルーチン*」の『MQREADALLCLOB 表関数』
- 「*SQL 管理ルーチン*」の『MQRECEIVEALLCLOB 表関数』
- 「コマンド・リファレンス」の『db2mqlsn - MQ Listener コマンド』
- 「コマンド・リファレンス」の『enable_MQFunctions』
- 「コマンド・リファレンス」の『disable_MQFunctions』

WebSphere MQ メッセージング

DB2® WebSphere® MQ 機能は、データグラム、パブリッシュ/サブスクライブ (p/s)、および要求/応答 (r/r) という 3 つのメッセージング・モデルをサポートします。データグラムとして送られるメッセージは 1 つの宛先に送信され、応答は期待されません。 p/s モデルでは、1 つまたは複数のパブリッシャーがメッセージをパ

ブリッシュ・サービスに送信します。このサービスが、そのメッセージを 1 つまたは複数のサブスクライバーに配布します。要求/応答はデータグラムと同様ですが、送信側は応答を受け取ることを期待します。

WebSphere MQ 自体は、移送するメッセージの特定の構造を必要としたり、サポートしたりするわけではありません。 C、Cobol、または XML ストリングとして形式化されるメッセージをサポートするのは、他の製品 (たとえば MQSeries® Integrator (MQSI)) です。 MQSI で構造化されたメッセージは、メッセージ・リポジトリーによって定義されます。一般的に自己記述型の構造を持つ XML メッセージもまた、リポジトリーを介して管理できます。メッセージは非構造化にすることもできます。その場合、メッセージ内容を構文解析または構成するためのユーザー・コードが必要です。そのようなメッセージは、しばしば半構造化されます。つまり、バイト位置または固定された区切り文字を使ってメッセージ内の各フィールドを分離します。そのような半構造化されたメッセージは、 WebSphere MQ Assist Wizard を介してサポートされます。 XML メッセージは、DB2 XML エクステンダーを介してサポートされます。

すべてのデータベース・アプリケーションが同じ DB2 サーバーに接続するとき、WebSphere MQ DB2 機能を使った最も基本的なメッセージングが発生します。クライアントは、データベース・サーバーから見てローカルであるか、またはネットワーク環境に分散されている場合があります。

簡単なシナリオを考えます。ユーザー定義ストリングをデフォルト・サービス・ロケーションに送るために、クライアント A が MQSEND 関数を呼び出します。その後、データベース・サーバー上の DB2 の中で WebSphere MQ 関数が実行されます。その後のある時点で、デフォルト・サービスによって定義されたキューの先頭にあるメッセージを削除してそれをクライアントに戻すために、クライアント B が MQRECEIVE 関数を呼び出します。ここでもまた、この操作のために使われる WebSphere MQ 関数は DB2 によって実行されます。

データベース・クライアントは、簡単なメッセージングをさまざまな方法で使用できます。メッセージングの一般的な使用方法として、たとえば以下のものがあります。

データ収集
1 つの情報源またはさまざまな複数の情報源から、メッセージの形で情報が受け取られます。情報源は、たとえば SAP のような市販のアプリケーション、または組織内で開発されたアプリケーションです。そのようなデータはキューから受信され、後で処理または分析するためにデータベース表に保管されます。

ワークロードの分散
同一アプリケーションの複数インスタンスによって共有されるキューに対して、処理要求が通知されます。 1 つのインスタンスが処理の一部を実行できる状態になると、実行すべき処理要求を含んでいるメッセージをキューの先頭から受け取ります。この技法を使用して、複数のインスタンスは、プールされた要求の単一キューによって表されるワークロードを共有できます。

アプリケーション信号方式
複数のプロセスがコラボレーションする状況では、プロセスの処理を調整するためにしばしばメッセージが使用されます。このようなメッセージには、

実行すべき処理に関するコマンドや要求を含めることができます。通常、この種の信号方式は片方向です。つまり、メッセージの発信側は応答を期待しません。詳しくは、 『WebSphere MQ 関数を使用した要求/応答通信』のトピックを参照してください。

アプリケーション通知

データを送る発信側が応答を期待しないという点で、通知は信号方式と似ています。ただし、通知には通常、すでに発生したビジネス・イベントに関するデータが含まれます。パブリッシュ/サブスクライブは、より拡張された通知形式です。詳しくは、 『WebSphere MQ 関数を使用したパブリッシュ/サブスクライブ』のトピックを参照してください。

以下のシナリオは、先程の簡単なシナリオを拡張して、リモート・メッセージングを取り入れたものです。つまり、マシン A とマシン B の間で、以下のようなステップでメッセージが送信されます。

1. DB2 クライアントが MQSEND 呼び出しを実行します。その際、マシン B 上のリモート・キューを表すよう定義されたターゲット・サービスを指定します。
2. WebSphere MQ DB2 機能が、メッセージを送る実際の WebSphere MQ 処理を実行します。マシン A 上の WebSphere MQ サーバーはメッセージを受け取って、サービス・ポイント定義および現在のマシン A の WebSphere MQ 構成によって定義された宛先に送信します。サーバーは、これがマシン B 上のキューであることを判別した後、マシン B 上の WebSphere MQ サーバーにメッセージを送ります (必要に応じて透過的に再試行します)。
3. マシン B 上の WebSphere MQ サーバーはマシン A 上のサーバーからメッセージを受け取って、マシン B 上の宛先キューにそれを格納します。
4. マシン B 上の WebSphere MQ クライアントは、キューの先頭にあるメッセージを要求します。

関連概念:

- 19 ページの『MQSeries 使用可能性』
- 559 ページの『WebSphere MQ 機能の概要』
- 564 ページの『WebSphere MQ 関数を使用したメッセージ送信』
- 566 ページの『WebSphere MQ 関数を使用したメッセージ取り出し』
- 568 ページの『WebSphere MQ アプリケーション間の接続』
- 569 ページの『WebSphere MQ 関数を使用した要求/応答通信』
- 571 ページの『WebSphere MQ 関数を使用したパブリッシュ/サブスクライブ』
- 「*IBM DB2 Information Integrator アプリケーション開発者向けガイド*」の『DB2 内で WebSphere MQ 機能を使用する方法』

関連タスク:

- 「*アプリケーション開発ガイド アプリケーションの構築および実行*」の『DB2 WebSphere MQ 関数のセットアップ』

関連資料:

- 「*SQL 管理ルーチン*」の『MQSEND スカラー関数』
- 「*SQL 管理ルーチン*」の『MQRECEIVE スカラー関数』
- 「*SQL 管理ルーチン*」の『MQREAD スカラー関数』

- 「*SQL 管理ルーチン*」の『MQPUBLISH スカラー関数』
- 「*SQL 管理ルーチン*」の『MQSUBSCRIBE スカラー関数』
- 「*SQL 管理ルーチン*」の『MQUNSUBSCRIBE スカラー関数』
- 「*SQL 管理ルーチン*」の『MQREADALL 表関数』
- 「*SQL 管理ルーチン*」の『MQRECEIVEALL 表関数』
- 「*SQL 管理ルーチン*」の『MQREADCLOB スカラー関数』
- 「*SQL 管理ルーチン*」の『MQRECEIVECLOB スカラー関数』
- 「*SQL 管理ルーチン*」の『MQREADALLCLOB 表関数』
- 「*SQL 管理ルーチン*」の『MQRECEIVEALLCLOB 表関数』
- 「コマンド・リファレンス」の『db2mqlsn - MQ Listener コマンド』
- 「コマンド・リファレンス」の『enable_MQFunctions』
- 「コマンド・リファレンス」の『disable_MQFunctions』

WebSphere MQ 関数を使用したメッセージ送信

DB2® ユーザーは MQSEND を使用して、どのデータを送信するか、どこに送信するか、いつ送信するかを指定します。業界では、これを「応答不要送信」(『Send and Forget』) といいます。送信側は単にメッセージを送信するだけで、メッセージが宛先に確実に届くようにする、WebSphere® MQ の保証された送達プロトコルを信頼します。以下に、その例を示します。

『例 4』 では、ポリシー highPriority を使用してユーザー定義ストリングをサービス・ポイント myPlace に送ります。

例 4:
```
VALUES DB2MQ.MQSEND('myplace','highPriority','test')
```

『例 4』 のポリシー highPriority は AMI リポジトリーで定義されたポリシーであり、 WebSphere MQ 優先順位を最高レベルに設定し、場合によっては他のサービス品質 (たとえばパーシスタンス) も調整します。

メッセージの内容は、 SQL データとユーザー指定データの有効な任意の組み合わせにすることができます。これには、ネストされた関数、演算子、キャストが含まれます。たとえば、VARCHAR 列 LASTNAME、FIRSTNAME、および DEPARTMENT (部署) を持つ表 EMPLOYEE について考えます。 DEPARTMENT 5LGA に属するそれぞれの従業員についてこの情報を含んだメッセージを送信する場合、以下のようにします。

例 5:
```
SELECT DB2MQ.MQSEND(LASTNAME || ' ' || FIRSTNAME || ' ' || DEPARTMENT)
  FROM EMPLOYEE WHERE DEPARTMENT = '5LGA'
```

整数として定義された AGE (年齢) という列がこの表に含まれる場合、ステートメントに以下のようなコードを追加してこの情報を含めます。

例 6:
```
SELECT DB2MQ.MQSEND (LASTNAME || ' ' || FIRSTNAME || ' ' || DEPARTMENT ||
  ' ' || char(AGE)) FROM EMPLOYEE WHERE DEPARTMENT = '5LGA'
```

AGE 列ではなく、タイプ CLOB の RESUME (履歴) 列が表 EMPLOYEE に 含まれる場合、 DEPARTMENT 5LGA に属するそれぞれの従業員についての情報を含んだメッセージは、以下のステートメントを使って発行できます。

例 7:
```
SELECT DB2MQ.MQSEND(clob(LASTNAME) || ' ' || clob(FIRSTNAME) ||
  ' ' || clob(DEPARTMENT) || ' ' || RESUME))
  FROM EMPLOYEE WHERE DEPARTMENT = '5LGA'
```

以下の例は、有効な任意の SQL 式を使ってメッセージ内容を作成する方法を示しています。 VARCHAR 列 DEPT_NO (部署番号) および DEPT_NAME (部署名) を含む別の表 DEPT があるとします。従業員の LASTNAME と DEPT_NAME を含むメッセージは、以下のようにして送信できます。

例 8:
```
SELECT DB2MQ.MQSEND(e.LASTNAME || ' ' || d.DEPTNAME)
  FROM EMPLOYEE e, DEPT d
  WHERE e.DEPARTMENT = d.DEPTNAME
```

関連概念:

- 19 ページの『MQSeries 使用可能性』
- 559 ページの『WebSphere MQ 機能の概要』
- 561 ページの『WebSphere MQ メッセージング』
- 566 ページの『WebSphere MQ 関数を使用したメッセージ取り出し』
- 568 ページの『WebSphere MQ アプリケーション間の接続』
- 569 ページの『WebSphere MQ 関数を使用した要求/応答通信』
- 571 ページの『WebSphere MQ 関数を使用したパブリッシュ/サブスクライブ』
- 「*IBM DB2 Information Integrator アプリケーション開発者向けガイド*」の『DB2 内で WebSphere MQ 機能を使用する方法』

関連タスク:

- 「*アプリケーション開発ガイド アプリケーションの構築および実行*」の『DB2 WebSphere MQ 関数のセットアップ』

関連資料:

- 「*SQL 管理ルーチン*」の『MQSEND スカラー関数』
- 「*SQL 管理ルーチン*」の『MQRECEIVE スカラー関数』
- 「*SQL 管理ルーチン*」の『MQREAD スカラー関数』
- 「*SQL 管理ルーチン*」の『MQPUBLISH スカラー関数』
- 「*SQL 管理ルーチン*」の『MQSUBSCRIBE スカラー関数』
- 「*SQL 管理ルーチン*」の『MQUNSUBSCRIBE スカラー関数』
- 「*SQL 管理ルーチン*」の『MQREADALL 表関数』
- 「*SQL 管理ルーチン*」の『MQRECEIVEALL 表関数』
- 「*SQL 管理ルーチン*」の『MQREADCLOB スカラー関数』
- 「*SQL 管理ルーチン*」の『MQRECEIVECLOB スカラー関数』
- 「*SQL 管理ルーチン*」の『MQREADALLCLOB 表関数』
- 「*SQL 管理ルーチン*」の『MQRECEIVEALLCLOB 表関数』
- 「コマンド・リファレンス」の『db2mqlsn - MQ Listener コマンド』

- 「コマンド・リファレンス」の『enable_MQFunctions』
- 「コマンド・リファレンス」の『disable_MQFunctions』

WebSphere MQ 関数を使用したメッセージ取り出し

WebSphere® MQ DB2® 関数を使用して、メッセージを受信あるいは読み取ることができます。受信と読み取りの違いは、読み取りの場合、メッセージはキューから除去されずにキューの先頭に戻されます。受信操作の場合、メッセージはキューから除去されます。受信操作を使用してメッセージを取り出す場合、そのメッセージは一度だけ取り出せます。読み取り操作を使用してメッセージを取り出す場合、そのメッセージは何度でも取り出すことができます。以下の例は、取り出し操作を示しています。

例 9:
```
VALUES DB2MQ.MQREAD()
```

566 ページの 『例 9』 で戻される VARCHAR ストリングには、デフォルトのサービス品質を使用するデフォルト・サービスによって定義されているキューの先頭にあるメッセージが含まれます。読み取るメッセージが存在しない場合、NULL 値が戻されます。この操作によって、キューは変更されません。

例 10:
```
VALUES DB2MQ.MQRECEIVE('Employee_Changes')
```

566 ページの 『例 10』 は、デフォルト・ポリシーを使って Employee_Changes サービスによって定義されているキューの先頭から、メッセージが除去されます。

DB2 機能の 1 つとして、ユーザー定義の (または DB2 によって提供された) 関数から表を生成することができます。この表関数機能を利用して、キューの内容を DB2 Universal Database™ 表としてマテリアライズすることができます。以下は、この機能の最も簡単な例を示しています。

例 11:
```
SELECT t.* FROM table ( DB2MQ.MQREADALL()) t
```

『例 11』 の照会によって戻される表には、デフォルト・サービスによって定義されたキュー内のすべてのメッセージと、これらのメッセージに関するメタデータが含まれます。

メッセージのみを戻すには、この例を以下のように書き換えることができます。

例 12:
```
SELECT t.MSG FROM table (DB2MQ.MQREADALL()) t
```

表関数によって戻される表は、データベースから直接取り出される表とまったく同じです。つまり、この表をさまざまな方法を利用できます。たとえば、この表の内容を別の表と結合したり、キュー内のメッセージ数をカウントすることができます。

例 13:
```
SELECT t.MSG, e.LASTNAME FROM table (DB2MQ.MQREADALL() ) t, EMPLOYEE e
  WHERE t.MSG = e.LASTNAME
```

例 14:
```
SELECT COUNT(*) FROM table (DB2MQ.MQREADALL()) t
```

さらに、表関数の上にビューを作成することによって、表がキューから作成されたことを隠すこともできます。たとえば、以下の例では、 NEW_EMPLOYEES というサービスによって参照されるキューの上に NEW_EMP というビューが作成されます。

例 15:
```
CREATE VIEW NEW_EMP (msg) AS SELECT t.msg
   FROM table(DB2MQ.MQREADALL(NEW_EMPLOYEES)) t
```

『例 15』 の場合、メッセージ全体を含むただ 1 つの列を使ってビューが定義されています。メッセージの構造が単純であれば (たとえば固定長の 2 つのフィールドからなる場合)、単純に DB2 組み込み関数を使ってメッセージを 2 つの列に分けて構文解析することができます。たとえば、特定のキューに送られるすべてのメッセージに、常に 18 文字の姓と 18 文字の名が含まれることがわかっている場合、以下のようにして、各フィールドが別々の列となるビューを定義できます。

例 16:
```
CREATE VIEW NEW_EMP2 AS SELECT left(t.msg,18) AS LNAME, right(t.msg,18) AS FNAME
   FROM table(DB2MQ.MQREADALL()) t
```

メッセージの区切り構造を各列にマップする DB2 表関数およびビューを作成するために、 DB2 デベロップメント・センター機能の 1 つである WebSphere MQ Assist Wizard を使用できます。

1 つまたは複数のメッセージの内容をデータベースに保管したい場合がしばしばあります。メッセージ内容を操作および保管する SQL の機能を最大限に活用して、これを行うことができます。以下は、これを示す簡単な例です。

例 17:
```
INSERT INTO MESSAGES SELECT t.msg FROM table (DB2MQ.MQRECEIVEALL()) t
```

VARCHAR(2000) の 1 つの列を持つ表 MESSAGES があるとします。上記のステートメントによって、メッセージがデフォルト・サービス・キューから表に挿入されます。この技法は、さまざまな状況で利用することができます。

関連概念:

- 19 ページの『MQSeries 使用可能性』
- 559 ページの『WebSphere MQ 機能の概要』
- 561 ページの『WebSphere MQ メッセージング』
- 564 ページの『WebSphere MQ 関数を使用したメッセージ送信』
- 568 ページの『WebSphere MQ アプリケーション間の接続』
- 569 ページの『WebSphere MQ 関数を使用した要求/応答通信』
- 571 ページの『WebSphere MQ 関数を使用したパブリッシュ/サブスクライブ』
- 「*IBM DB2 Information Integrator アプリケーション開発者向けガイド*」の『DB2 内で WebSphere MQ 機能を使用する方法』

関連タスク:

- 「*アプリケーション開発ガイド アプリケーションの構築および実行*」の『DB2 WebSphere MQ 関数のセットアップ』

関連資料:

- 「*SQL 管理ルーチン*」の『MQSEND スカラー関数』

- 「*SQL 管理ルーチン*」の『MQRECEIVE スカラー関数』
- 「*SQL 管理ルーチン*」の『MQREAD スカラー関数』
- 「*SQL 管理ルーチン*」の『MQPUBLISH スカラー関数』
- 「*SQL 管理ルーチン*」の『MQSUBSCRIBE スカラー関数』
- 「*SQL 管理ルーチン*」の『MQUNSUBSCRIBE スカラー関数』
- 「*SQL 管理ルーチン*」の『MQREADALL 表関数』
- 「*SQL 管理ルーチン*」の『MQRECEIVEALL 表関数』
- 「*SQL 管理ルーチン*」の『MQREADCLOB スカラー関数』
- 「*SQL 管理ルーチン*」の『MQRECEIVECLOB スカラー関数』
- 「*SQL 管理ルーチン*」の『MQREADALLCLOB 表関数』
- 「*SQL 管理ルーチン*」の『MQRECEIVEALLCLOB 表関数』
- 「コマンド・リファレンス」の『db2mqlsn - MQ Listener コマンド』
- 「コマンド・リファレンス」の『enable_MQFunctions』
- 「コマンド・リファレンス」の『disable_MQFunctions』

WebSphere MQ アプリケーション間の接続

ほとんどのソリューションでは、アプリケーション統合が利用されています。購入したアプリケーションを既存のインフラストラクチャーに統合する場合も、新しく開発したアプリケーションを既存の環境に統合する場合も、異種のさまざまなサブシステムを集めて 1 つの機能するシステムに統合しなければなりません。

WebSphere® MQ は、アプリケーション統合に欠かせないツールとして広く認められています。ほとんどのハードウェア、ソフトウェア、および言語処理環境で利用できる WebSphere MQ は、さまざまな異種のアプリケーションを互いに統合する方法を提供します。

アプリケーション統合のシナリオ、および DB2® でそれを行う方法については、『WebSphere MQ 関数を使用した要求/応答通信』、および『WebSphere MQ 関数を使用したパブリッシュ/サブスクライブ』のトピックを参照してください。

関連概念:

- 19 ページの『MQSeries 使用可能性』
- 559 ページの『WebSphere MQ 機能の概要』
- 561 ページの『WebSphere MQ メッセージング』
- 564 ページの『WebSphere MQ 関数を使用したメッセージ送信』
- 566 ページの『WebSphere MQ 関数を使用したメッセージ取り出し』
- 569 ページの『WebSphere MQ 関数を使用した要求/応答通信』
- 571 ページの『WebSphere MQ 関数を使用したパブリッシュ/サブスクライブ』
- 「*IBM DB2 Information Integrator アプリケーション開発者向けガイド*」の『DB2 内で WebSphere MQ 機能を使用する方法』

関連タスク:

- 「*アプリケーション開発ガイド アプリケーションの構築および実行*」の『DB2 WebSphere MQ 関数のセットアップ』

関連資料:

- 「*SQL 管理ルーチン*」の『MQSEND スカラー関数』
- 「*SQL 管理ルーチン*」の『MQRECEIVE スカラー関数』
- 「*SQL 管理ルーチン*」の『MQREAD スカラー関数』
- 「*SQL 管理ルーチン*」の『MQPUBLISH スカラー関数』
- 「*SQL 管理ルーチン*」の『MQSUBSCRIBE スカラー関数』
- 「*SQL 管理ルーチン*」の『MQUNSUBSCRIBE スカラー関数』
- 「*SQL 管理ルーチン*」の『MQREADALL 表関数』
- 「*SQL 管理ルーチン*」の『MQRECEIVEALL 表関数』
- 「*SQL 管理ルーチン*」の『MQREADCLOB スカラー関数』
- 「*SQL 管理ルーチン*」の『MQRECEIVECLOB スカラー関数』
- 「*SQL 管理ルーチン*」の『MQREADALLCLOB 表関数』
- 「*SQL 管理ルーチン*」の『MQRECEIVEALLCLOB 表関数』
- 「コマンド・リファレンス」の『db2mqlsn - MQ Listener コマンド』
- 「コマンド・リファレンス」の『enable_MQFunctions』
- 「コマンド・リファレンス」の『disable_MQFunctions』

WebSphere MQ 関数を使用した要求/応答通信

要求/応答 (R/R) 通信方式は、 1 つのアプリケーションが別のアプリケーションのサービスを要求する際に広く使われる技法です。その 1 つの方法として、ある処理の実行を要求するメッセージを要求側がサービス提供側に送る方式があります。処理が完了すると、提供側は要求側に結果を戻すか、または単に完了したことを通知します。ただし、このような基本的なメッセージング技法には、送信側の要求とサービス提供者の応答を結び付ける方法がありません。要求側が処理を続行するために応答を待たないような場合には、それぞれの要求とその応答を関連付ける何らかのメカニズムを使用する必要があります。開発者がこのようなメカニズムを自分で作成する必要はありません。 WebSphere® MQ は、通信におけるメッセージの関連付けを可能にする相関 ID を提供します。

このメカニズムの使用方法は多数ありますが、最も簡単な方法は、たとえば以下のように、要求側が既知の相関 ID を使ってメッセージにマークを付けることです。

例 18:

```
VALUES DB2MQ.MQSEND ('myRequester','myPolicy','SendStatus:cust1','Req1')
```

このステートメントは、要求の相関 ID を示すために、最終パラメーター Req1 を上記の MQSEND ステートメントに追加します。この特定の要求に対する応答を受け取るには、対応する MQRECEIVE ステートメントを使用します。その際、以下のようにして、この相関 ID と一致する指定されたサービスによって定義された最初のメッセージを選択して取り出します。

例 19:

```
VALUES DB2MQ.MQRECEIVE('myReceiver','myPolicy','Req1')
```

要求を発行するアプリケーションがビジー状態の場合、応答が送られる前に要求側が MQRECEIVE を発行しても、この相関 ID に一致するメッセージは見つかりません。

サービス要求と相関 ID の両方を受け取るには、以下のようなステートメントを使用します。

例 20:

```
SELECT msg, correlid
  FROM table (VALUES DB2MQ.MQRECEIVEALL('aServiceProvider','myPolicy',1)) t
```

これによって、最初の要求のメッセージと相関 ID がサービス aServiceProvider から戻されます。

サービスの実行が完了すると、 aRequester によって示されたキューに応答メッセージが送られます。その間、要求側は他の処理を行っていたかもしれません。実際、最初のサービス要求に対する応答が設定された時間内に受信されるという保証はありません。このようなアプリケーション・レベルのタイムアウトは、開発者が管理しなければなりません。要求側はポーリングによって、応答の存在を検出する必要があります。このような時間に依存しない非同期処理の利点は、要求側とサービス提供者が互いに完全に独立して処理できることです。この方法は、アプリケーションが断続的に接続される環境や、複数の要求または応答が集約された後で処理されるバッチ指向の環境において利用することができます。この種の集約は、データウェアハウス定期的に更新するデータウェアハウス環境や、オペレーショナル・データ・ストアでしばしば使用されます。

関連概念:

- 19 ページの『MQSeries 使用可能性』
- 559 ページの『WebSphere MQ 機能の概要』
- 561 ページの『WebSphere MQ メッセージング』
- 564 ページの『WebSphere MQ 関数を使用したメッセージ送信』
- 566 ページの『WebSphere MQ 関数を使用したメッセージ取り出し』
- 568 ページの『WebSphere MQ アプリケーション間の接続』
- 571 ページの『WebSphere MQ 関数を使用したパブリッシュ/サブスクライブ』
- 「*IBM DB2 Information Integrator アプリケーション開発者向けガイド*」の『DB2 内で WebSphere MQ 機能を使用する方法』

関連タスク:

- 「*アプリケーション開発ガイド アプリケーションの構築および実行*」の『DB2 WebSphere MQ 関数のセットアップ』

関連資料:

- 「*SQL 管理ルーチン*」の『MQSEND スカラー関数』
- 「*SQL 管理ルーチン*」の『MQRECEIVE スカラー関数』
- 「*SQL 管理ルーチン*」の『MQREAD スカラー関数』
- 「*SQL 管理ルーチン*」の『MQPUBLISH スカラー関数』
- 「*SQL 管理ルーチン*」の『MQSUBSCRIBE スカラー関数』
- 「*SQL 管理ルーチン*」の『MQUNSUBSCRIBE スカラー関数』
- 「*SQL 管理ルーチン*」の『MQREADALL 表関数』
- 「*SQL 管理ルーチン*」の『MQRECEIVEALL 表関数』
- 「*SQL 管理ルーチン*」の『MQREADCLOB スカラー関数』
- 「*SQL 管理ルーチン*」の『MQRECEIVECLOB スカラー関数』

- 「*SQL 管理ルーチン*」の『MQREADALLCLOB 表関数』
- 「*SQL 管理ルーチン*」の『MQRECEIVEALLCLOB 表関数』
- 「コマンド・リファレンス」の『db2mqlsn - MQ Listener コマンド』
- 「コマンド・リファレンス」の『enable_MQFunctions』
- 「コマンド・リファレンス」の『disable_MQFunctions』

WebSphere MQ 関数を使用したパブリッシュ/サブスクライブ

データベース・メッセージングの技法には、単純なデータ・パブリッシュと自動パブリッシュの 2 つがあります。

単純なデータ・パブリッシュ

アプリケーション統合でよく見られるシナリオは、 1 つのアプリケーションが関連イベントについて他のアプリケーションに通知するという方法です。これは、別のアプリケーションによってモニターされているキューにメッセージを送信することにより、簡単に行えます。メッセージの内容は、ユーザー定義ストリングにするか、またはデータベース列から構成することができます。単純なメッセージは、MQSEND 関数を使って送るだけでよい場合がほとんどです。そのようなメッセージを複数の受信側に同時に送る必要がある場合、 WebSphere® MQ AMI の配布リスト機能を使用できます。

配布リストは AMI 管理ツールを使って定義されます。配布リストは、個々のサービスからなるリストです。配布リストに送られるメッセージは、そのリスト内で定義されたすべてのサービスに転送されます。この方法は、一部のサービスが常にあらゆるメッセージを受け取る必要があることがわかっている場合に特に便利です。以下の例は、配布リスト interestedParties へのメッセージ送信を示しています。

例 21:
```
VALUES DB2MQ.MQSEND('interestedParties', 'information of general interest');
```

特定のサービスが受け取るメッセージをより詳細に制御したい場合には、パブリッシュ/サブスクライブ機能が必要です。パブリッシュ/サブスクライブ・システムは、拡張が容易な、保護された環境を提供します。この環境では、複数のパブリッシャーからメッセージを受け取るよう、多数のサブスクライバーを登録することができます。この機能をサポートするために、 MQSeries® Integrator または the WebSphere MQ パブリッシュ/サブスクライブ機能で MQPublish インターフェースを使用することができます。

MQPublish を使用すれば、ユーザーは、メッセージを関連付けるトピックを任意に指定できます。トピックによって、サブスクライバーは受け入れるメッセージをより明確に指定できます。これを行うためのステップは、以下のとおりです。

1. WebSphere MQ 管理者が MQSeries Integrator パブリッシュ/サブスクライブ機能を構成します。
2. 関連するアプリケーションは、MQSI 構成によって定義されたサブスクリプション・ポイントにサブスクライブし、オプションで、とくに関心のあるトピックを指定します。それぞれのサブスクライバーは関係するトピックを選択し、さらに MQSeries Integrator バージョン 2 のコンテンツ・ベースのサブスクリプション

技法を利用できます。その際、サービス名によって表されるキューがサブスクライバーを定義することに注意する必要があります。

3. DB2® アプリケーションがサービス・ポイント Weather (天気) にメッセージをパブリッシュします。メッセージは、天気が Sleet (みぞれ) であることを示し、Austin (オースティン) 市というトピックを含んでいます。こうして、オースティン市の天候がみぞれであることを、関心のあるサブスクライバーに通知します。
4. 実際にメッセージをパブリッシュする操作は、 DB2 に付属の WebSphere MQ 機能によって処理されます。メッセージは、サービス名 Weather を使って MQSeries Integrator に送られます。
5. MQSI は Weather サービスからのメッセージを受け入れ、 MQSI 構成によって定義された処理をすべて実行して、どのサブスクリプションに該当するかを判別します。その後、MQSI は基準を満たすキューのサブスクライバーにメッセージを転送します。
6. Weather サービスにサブスクライブし、 Austin に関心があると登録したアプリケーションは、受信サービスにおいてメッセージ Sleet を受け取ります。

すべてのデフォルトおよび NULL トピックを使ってこのデータをパブリッシュするには、以下のステートメントを使用します。

例 22:
```
SELECT DB2MQ.MQPUBLISH (LASTNAME || ' ' || FIRSTNAME || ' ' || DEPARTMENT
  ||' ' || char(AGE)) FROM EMPLOYEE WHERE DEPARTMENT = '5LGA'
```

すべてのパラメーターを完全に指定し、メッセージを単純化して LASTNAME だけを含めるようにすると、ステートメントは以下のようになります。

例 23:
```
SELECT DB2MQ.MQPUBLISH('HR_INFO_PUB', 'SPECIAL_POLICY', LASTNAME,
  'ALL_EMP:5LGA', 'MANAGER') FROM EMPLOYEE WHERE DEPARTMENT = '5LGA'
```

このステートメントは、 SPECIAL_POLICY サービスを使用して HR_INFO_PUB パブリッシュ・サービスにメッセージをパブリッシュします。メッセージは、送信側の相関 ID が MANAGER であることを示します。トピック・ストリング (『ALL_EMP:5LGA』) は、コロン (『:』) によって連結された複数のトピックを指定できることを示しています。この例では 2 つのトピックが使用され、サブスクライバーは ALL_EMP または 5LGA のみのいずれかに登録してこれらのメッセージを受け取ることができます。パブリッシュされたメッセージを受け取るには、まず特定のトピックが含まれるメッセージに関心があることを登録し、メッセージ送信先のサブスクライバー・サービス名を示す必要があります。その際、AMI サブスクライバー・サービスがブローカー・サービスと受信側サービスを定義することに注意する必要があります。ブローカー・サービスはサブスクライバーがパブリッシュ/サブスクライブ・ブローカーと通信する方法を管理し、受信側サービスはサブスクリプション要求に一致するメッセージの送り先です。以下のステートメントは、トピック ALL_EMP への関心を登録します。

例 24:
```
VALUES DB2MQ.MQSUBSCRIBE('aSubscriber', 'ALL_EMP')
```

アプリケーションがサブスクライブを完了すると、トピック ALL_EMP を含んでパブリッシュされるメッセージは、サブスクライバー・サービスによって定義された受信側サービスに転送されます。 1 つのアプリケーションが複数のサブスクリプションを同時に使用できます。サブスクリプションを満たすメッセージを得るには、

任意の標準的なメッセージ検索関数を使用できます。たとえば、サブスクライバー・サービス aSubscriber が受信側サービスを aSubscriberReceiver と定義している場合、以下のステートメントは、最初のメッセージを削除せずに読み取ります。

例 25:
```
VALUES DB2MQ.MQREAD('aSubscriberReceiver')
```

パブリッシュされたメッセージとトピックを判別するには、いずれかの表関数を使用します。以下のステートメントは、aSubscriberReceiver から最初の 5 つのメッセージを受信して、メッセージとトピックの両方を表示します。

例 26:
```
SELECT t.msg, t.topic FROM table (DB2MQ.MQRECEIVEALL('aSubscriberReceiver',5)) t
```

トピック ALL_EMP を含むすべてのメッセージを読み取るには、以下のステートメントを発行して、SQL の機能を最大限に活用することができます。

例 27:
```
SELECT t.msg FROM table (DB2MQ.MQREADALL('aSubscriberReceiver')) t
  WHERE t.topic = 'ALL_EMP'
```

注: 制約がある MQRECEIVEALL を使用する場合、 ALL_EMP トピックを含むパブリッシュされたメッセージだけでなく、キュー全体が消費されることに注意してください。これは、制約が適用される前に表関数が実行されるためです。

特定のトピックへのサブスクライブにもはや関心がなくなった場合、以下のようなステートメントを使用して、明示的にアンサブスクライブする必要があります。

例 28:
```
VALUES DB2MQ.MQUNSUBSCRIBE('aSubscriber', 'ALL_EMP')
```

上記のステートメントが発行された後、パブリッシュ/サブスクライブ・ブローカーは、このサブスクリプションを満たすメッセージをもはや送りません。

自動パブリッシュ

データベース・メッセージングのもう 1 つの重要な技法は、自動パブリッシュです。 DB2 のトリガー機能を使用して、メッセージをトリガー起動の一部として自動的にパブリッシュすることができます。データの自動パブリッシュを行う技法は他にもありますが、トリガー・ベースの方法を使用すると、管理者や開発者はメッセージ内容をかなり自由に組み立てて、トリガー・アクションを柔軟に定義することができます。他のトリガーの場合と同様に、実行の頻度とコストを考慮する必要があります。以下の例は、WebSphere MQ DB2 関数でトリガーを使用する方法を示しています。

以下の例は、従業員が新しく採用されるたびに、メッセージを簡単にパブリッシュする方法を示しています。 NEW_EMP への関心を登録して HR_INFO_PUB サービスにサブスクライブしているすべてのユーザーまたはアプリケーションは、それぞれの新規採用者の名前と部署、および日付を含むメッセージを受け取ります。

例 29:
```
CREATE TRIGGER new_employee AFTER INSERT ON employee
  REFERENCING NEW AS n FOR EACH ROW MODE DB2SQL
  VALUES DB2MQ.MQPUBLISH('HR_INFO_PUB',current date||''||
    'LASTNAME'||''DEPARTMENT,'NEW_EMP')
```

関連概念:

- 19 ページの『MQSeries 使用可能性』
- 559 ページの『WebSphere MQ 機能の概要』
- 561 ページの『WebSphere MQ メッセージング』
- 564 ページの『WebSphere MQ 関数を使用したメッセージ送信』
- 566 ページの『WebSphere MQ 関数を使用したメッセージ取り出し』
- 568 ページの『WebSphere MQ アプリケーション間の接続』
- 569 ページの『WebSphere MQ 関数を使用した要求/応答通信』
- 「*IBM DB2 Information Integrator アプリケーション開発者向けガイド*」の『DB2 内で WebSphere MQ 機能を使用する方法』

関連タスク:

- 「*アプリケーション開発ガイド アプリケーションの構築および実行*」の『DB2 WebSphere MQ 関数のセットアップ』

関連資料:

- 「*SQL 管理ルーチン*」の『MQSEND スカラー関数』
- 「*SQL 管理ルーチン*」の『MQRECEIVE スカラー関数』
- 「*SQL 管理ルーチン*」の『MQREAD スカラー関数』
- 「*SQL 管理ルーチン*」の『MQPUBLISH スカラー関数』
- 「*SQL 管理ルーチン*」の『MQSUBSCRIBE スカラー関数』
- 「*SQL 管理ルーチン*」の『MQUNSUBSCRIBE スカラー関数』
- 「*SQL 管理ルーチン*」の『MQREADALL 表関数』
- 「*SQL 管理ルーチン*」の『MQRECEIVEALL 表関数』
- 「*SQL 管理ルーチン*」の『MQREADCLOB スカラー関数』
- 「*SQL 管理ルーチン*」の『MQRECEIVECLOB スカラー関数』
- 「*SQL 管理ルーチン*」の『MQREADALLCLOB 表関数』
- 「*SQL 管理ルーチン*」の『MQRECEIVEALLCLOB 表関数』
- 「コマンド・リファレンス」の『db2mqlsn - MQ Listener コマンド』
- 「コマンド・リファレンス」の『enable_MQFunctions』
- 「コマンド・リファレンス」の『disable_MQFunctions』

第 25 章 WebSphere

以下の節では、WebSphere の接続プールとステートメント・キャッシュについて説明します。

企業データへの接続

多くの企業は、データを zSeries® サーバーのような大規模なシステムに保管します。 Web アプリケーションには、そのデータにアクセスするための手段が必要です。

DB2® Connect は、Windows® ベースのアプリケーションまたは UNIX® ベースのアプリケーションに、それらの大規模システムに接続してそこに保管されているデータを使用する機能を提供します。また、DB2 は、接続プールや接続コンセントレーターなどの独自の機能のセットも備えています。

WebSphere の接続プールとデータ・ソース

リソースは、データベースにアクセスしようとするたびに、そのデータベースに接続しなければなりません。データベース接続はオーバーヘッドを引き起こします。リソースは、接続を作成し、保守し、必要なくなったときに解放しなければなりません。

注: ここに載せられている情報は、 WebSphere® Application Server for UNIX®/LINUX/Windows® のバージョン 4 に適用されます。

アプリケーションの全体的なデータベース・オーバーヘッドは、特に Web ベースのアプリケーションで高くなります。なぜなら、Web ユーザーの方が接続および切断を行う頻度が高いからです。さらに、インターネットの性質上、通常ではユーザー対話の割合が小さくなっています。対話自体の間も、より多くの労力が接続と切断に費やされます。さらに、インターネット要求は事実上どこからでもやって来るので、使用量は大規模になる可能性があり、予測が困難です。

IBM® WebSphere Application Server を使用すれば、管理者はアプリケーション・サーバー上の複数のアプリケーションで共用できるデータベース接続のプールを確立することにより、これらのオーバーヘッドに関する問題を扱えます。

接続プールでは、接続オーバーヘッドを複数のユーザー要求に分散することにより、将来の要求のためにリソースが保存されます。

接続プールを確立するには、WebSphere 接続プールを使用するか、JDBC 2.1 Optional Package API によって提供される DB2® 接続プール・サポートを使用できます。

WebSphere 接続プールは、JDBC 2.1 Optional Package API 仕様のインプリメンテーションです。したがって、この接続プールのプログラミング・モデルは、JDBC 2.1 Core 仕様および JDBC 2.1 Optional Package API 仕様で指定されているものと

同じです。つまり、WebSphere Application Server で作成されたデータ・ソースを介して接続を取得するアプリケーションは、接続プールや JTA に対応した接続などの、JDBC 2.1 の機能から益を得ることができます。

加えて、WebSphere 接続プールには、最高のパフォーマンスを得られるように管理者がプールを調整するための追加機能が備わっています。また、一般的なベンダー固有の SQLException の知識がないプログラマーがアプリケーションを作成するための WebSphere 例外も用意されています。ベンダー固有の SQLException は、すべてが WebSphere 例外にマップされるわけではありません。依然として、アプリケーションはベンダー固有の SQLException を扱うようにコード化する必要があります。ただし、最も一般的なリカバリー可能例外は、WebSphere 例外にマップされます。

WebSphere を介して取得されるデータ・ソースは、JDBC 2.1 Optional Package API をインプリメントするデータ・ソースです。接続プールを備えており、選択されたベンダー固有のデータ・ソースに応じて、2 フェーズ・コミット・プロトコル・トランザクション (JTA 対応) に参加できる接続を提供できます。

`AccessEmployee.ear` ファイルの `AccessEmployee` プログラムは、WebSphere `DataSource` を使用して DB2 データベースにアクセスします。

関連資料:

- 「*アプリケーション開発ガイド アプリケーションの構築および実行*」の『Java WebSphere のサンプル』

WebSphere 接続プールの利点

接続プールにより、接続を必要とするあらゆるアプリケーション (特に Web ベース・アプリケーション) の応答時間が改善されます。

ユーザーが Web を介してリソースへの要求を行うと、そのリソースはデータ・ソースにアクセスします。ほとんどのユーザー要求では、データ・ソースが接続プールから既存の接続を見付けて使用するので、新しい接続を作成するオーバーヘッドは発生しません。要求が満たされ、応答がユーザーに戻されると、リソースは接続を再利用のために接続プールに戻します。この場合も、切断のオーバーヘッドは回避されます。

各ユーザー要求のコストは、接続または切断のコストのほんの何分の一かです。初期リソースを使用してプール内に接続を作成した後は、既存の接続が再利用されるので、付加的なオーバーヘッドは問題になりません。

準備済みステートメントのキャッシングは、WebSphere® 接続プールが Web ベース・アプリケーションの応答時間を改善するもう 1 つの機構です。

接続では、以前の準備済みステートメントのキャッシュが使用できます。準備済みステートメントが接続で新しく要求されたとき、キャッシュにそのステートメントが入っていればそれが戻されます。このキャッシングにより、準備済みステートメントのコストのかかる作成回数が削減され、応答時間が改善されます。このキャッシュは、同じステートメントを何度も準備する傾向のあるアプリケーションで役立ちます。

応答時間の改善に加え、WebSphere 接続プールは、データベースの抽象層を提供します。ここでは、クライアント・アプリケーションをバッファーに入れ、アプリケーションに、さまざまなデータベースが同じように動作しているように見せることができます。

このバッファリングにより、アプリケーション・コードでは一般的なベンダー固有の SQLException を扱うのではなく、 WebSphere 接続プールの例外を扱うことになるので、アプリケーション・データベースの切り替えが容易になっています。

WebSphere でのステートメント・キャッシング

WebSphere® には、以前に準備したステートメントをキャッシュに入れるための機構が備わっています。準備済みステートメントをキャッシュに入れると、`PreparedStatement` が接続のキャッシュに存在する場合にアプリケーションがそれをその接続で再利用し、新しい `PreparedStatement` を作成する必要を回避できるので、応答時間が改善されます。

アプリケーションが接続で `PreparedStatement` を作成すると、最初にその接続のキャッシュが検索され、同じ SQL ストリングを持つ `PreparedStatement` がすでに存在しているかどうかが判別されます。この検索は、`prepareStatement()` メソッドで SQL ステートメントのストリング全体を使用することによって行われます。一致するものが見付かったら、キャッシュに入っている PreparedStatement が戻されて使用されます。見付からない場合は、新しい `PreparedStatement` が作成され、アプリケーションに戻されます。

準備済みステートメントは、アプリケーションによってクローズされると、接続のステートメント・キャッシュに戻されます。デフォルトでは、接続プール全体について、100 個だけの準備済みステートメントをキャッシュに保持できます。たとえば、プールに 10 個の接続が存在している場合、それら 10 個の接続についてキャッシュに入れられる準備済みステートメントは 100 個を超えることができません。これにより、データベースに対して並行してオープンされる準備済みステートメントの数が限定されるため、データベースとの間でリソース問題が生じるのを避けるのに役立ちます。

エレメントが接続の準備済みステートメント・キャッシュから除去されるのは、その時点でキャッシュに入っている準備済みステートメントの数が `statementCacheSize` (デフォルトでは 100) を超えたときだけです。キャッシュから除去されなければならなくなった準備済みステートメントは、除去されて廃棄ステートメントのベクトルに追加されます。準備済みステートメントが除去されたメソッドが終了すると、廃棄ステートメント・ベクトルにある準備済みステートメントがただちにデータベースに対してクローズされます。したがって、任意の時点において、最近廃棄されたステートメントの数に 100 を加えた数のステートメントをデータベースに対してオープンすることができます。余分の準備済みステートメントはメソッドの終了後にクローズされます。

キャッシュに入れられる準備済みステートメントの数はデータ・ソースで構成できます。各キャッシュは、準備済みステートメントに関するアプリケーションの要件に従って調整する必要があります。

第 6 部 セキュリティー・プラグイン

第 26 章 セキュリティー・プラグイン

セキュリティー・プラグイン 581
セキュリティー・プラグイン・ライブラリーの位置 585
セキュリティー・プラグインの命名規則 586
セキュリティー・プラグインの 2 パーツのユーザー ID のサポート 587
セキュリティー・プラグインの 32 ビットと 64 ビットに関する考慮事項 590
セキュリティー・プラグインの問題判別 590
グループ検索プラグインの展開 592
ユーザー ID/パスワード・プラグインのデプロイ 593
GSS-API プラグインのデプロイ 595
Kerberos プラグインのデプロイ 597

セキュリティー・プラグイン

DB2® での認証は、セキュリティー・プラグイン を使用して行われます。セキュリティー・プラグインは動的にロードできるライブラリーであり、 DB2 が以下の機能を提供するためにロードします。

- グループ検索プラグイン: 特定のユーザーのグループ・メンバーシップ情報を検索します。
- クライアント認証プラグイン: DB2 クライアント上で認証を管理します。
- サーバー認証プラグイン: DB2 サーバー上で認証を管理します。

DB2 は、次の 2 つのプラグイン認証メカニズムをサポートしています。

- ユーザー ID/パスワード認証と呼ばれる、ユーザー ID とパスワードを使用した認証。認証タイプ CLIENT、SERVER、SERVER_ENCRYPT、DATA_ENCRYPT、および DATA_ENCRYPT_CMP が、ユーザーの認証を行う方法と場所を決定します。使用される認証タイプは、 *authentication* データベース構成パラメーターを使用して指定される認証タイプに依存します。これらの認証タイプはすべて、ユーザー ID/パスワード認証プラグインを使用してインプリメントされます。
- GSS-API を使用した認証。これは、以前は *Generic Security Service Application Program Interface Version 2* (IETF RFC2743) および *Generic Security Service API Version 2: C-Bindings* (IETF RFC2744) として知られていました。 Kerberos 認証も、GSS-API を使用してインプリメントされます。認証タイプ KERBEROS、GSSPLUGIN、 KRB_SERVER_ENCRYPT、および GSS_SERVER_ENCRYPT は、GSS-API 認証プラグインを使用します。 KRB_SERVER_ENCRYPT および GSS_SERVER_ENCRYPT は、 GSS-API 認証とユーザー ID/パスワード認証の両方をサポートしますが、 GSS-API 認証のほうが望ましい認証タイプです。

各プラグインは独立して使用するか、1 つ以上の他のプラグインと併せて使用することができます。たとえば、サーバー認証プラグインだけを使用して、クライアントおよびグループの認証については DB2 のデフォルトを受け入れることができます。また、グループまたはクライアントの認証プラグインのみを使用することもできます。ただし、クライアントとサーバーの両方のプラグインが必要となるのは、GSS-API 認証プラグインの場合のみです。

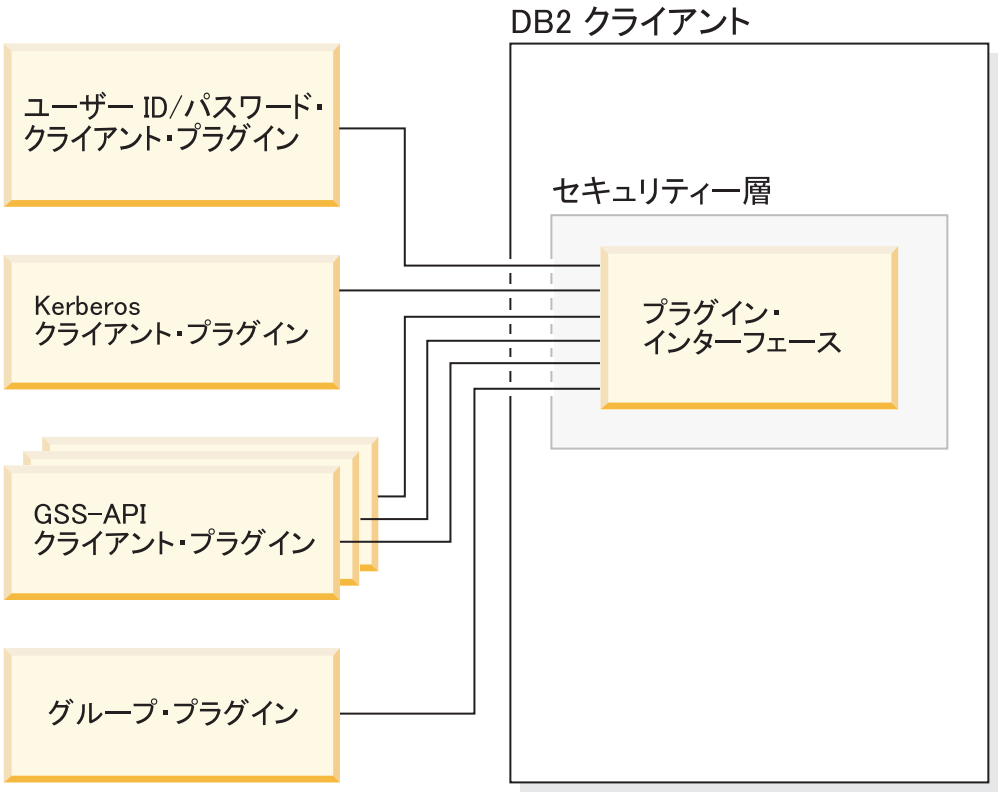
DB2 バージョン 8.2 では、デフォルトの動作として、オペレーティング・システム・レベルの認証メカニズムをインプリメントするユーザー ID/パスワード・プラグインが使用されます。 DB2 のこれより前のすべてのリリースでは、デフォルト動作として、プラグインのインプリメンテーションなしで、オペレーティング・シ

ステム・レベルの認証が直接使用されます。 DB2 バージョン 8.2 のクライアント・サイドの Kerberos サポートは、 Solaris、AIX®、Windows®、および IA32 Linux オペレーティング・システムで使用可能ですが、デフォルトでは Windows でのみ有効になっています。

DB2 には、グループ検索用、ユーザー ID/パスワード認証用、および Kerberos 認証用のプラグインのセットが組み込まれています。セキュリティー・プラグイン・アーキテクチャーを用いる場合は、独自のプラグインを作成するか、またはサード・パーティーからプラグインを購入することによって DB2 の認証動作をカスタマイズできます。

DB2 クライアント上のセキュリティー・プラグインのデプロイメント:

DB2 クライアントは 1 つのグループ・プラグイン、 1 つのユーザー ID/パスワード認証プラグインをサポートでき、特定の GSS-API プラグインについて DB2 サーバーと折衝します。この折衝では、クライアントが DB2 サーバー側のインプリメントされている GSS-API プラグインのリストをスキャンして、クライアントにインプリメントされている認証プラグインと一致する認証プラグイン名を探します。サーバー側のプラグインのリストは、サーバー上にインプリメントされているすべてのプラグインの名前が含まれた、ユーザー指定のデータベース・マネージャー構成パラメーター値です。以下の図は、 DB2 クライアント上のセキュリティー・プラグイン・インフラストラクチャーを描写しています。

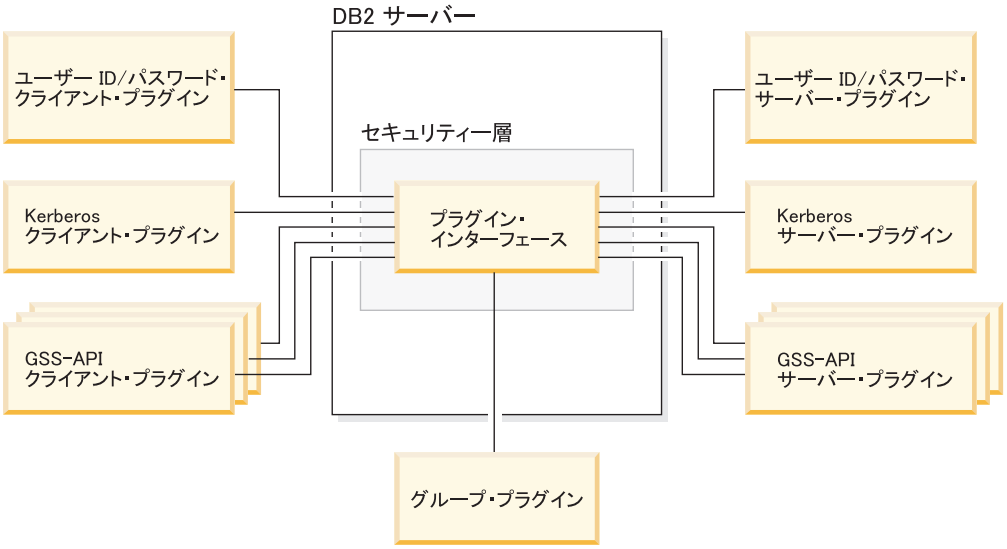


DB2 サーバー上のセキュリティー・プラグインのデプロイメント:

DB2 サーバーは 1 つのグループ・プラグイン、1 つのユーザー ID/パスワード認証プラグイン、および複数の GSS-API プラグインをサポートできます。複数の

GSS-API プラグインは、データベース・マネージャー構成パラメーター値でリストとして指定されます。このリストの中の 1 つの GSS-API プラグインのみ、Kerberos プラグインにすることができます。

サーバー・サイドのセキュリティー・プラグインのデプロイに加えて、データベース・サーバー上にもクライアント許可プラグインをデプロイする必要があるかもしれません。 db2start および db2trc などのインスタンス・レベルの操作を実行する際、 DB2 はクライアント認証プラグインを使用してその操作に対して許可検査を実行します。したがって、データベース・マネージャー構成パラメーター *authentication* で指定されたタイプの、認証プラグインのクライアント版をインストールする必要があります。データベース・サーバー上にクライアント認証プラグインをデプロイしていない場合、 db2start などのインスタンス・レベルの操作は失敗します。たとえば、認証タイプが SERVER で、ユーザー指定のクライアント・プラグインが使用されていない場合、 DB2 はフォールバックして IBM 提供のデフォルト・クライアント・オペレーティング・システム・プラグインを使用することになります。以下の図は、DB2 サーバー上のセキュリティー・プラグイン・インフラストラクチャーを描写しています。



セキュリティー・プラグインの有効化:

システム管理者は、プラグインに関係した特定のデータベース・マネージャー構成パラメーターを更新することにより、各認証メカニズムに使用するプラグインの名前を指定できます。これらのパラメーターがヌルの場合、それらはデフォルトとして、DB2 提供のグループ検索、ユーザー ID/パスワード管理、または Kerberos (認証に Kerberos が設定されている場合。サーバー・サイドのみ) 用のプラグインになります。ただし、DB2 はデフォルト GSS-API プラグインは提供していません。したがって、システム管理者は、認証で認証タイプ GSSPLUGIN を指定するなら、*srvcon_gssplugin_list* で認証プラグインを指定する必要もあります。

DB2 によるセキュリティー・プラグインのロード方法:

DB2 サーバーには、データベース・マネージャーの始動時に、データベース・マネージャー構成パラメーターで識別されたサポートされるすべてのプラグインがロードされます。

データベース接続およびインスタンス接続では、DB2 クライアントが、接続操作中にサーバーとの間で折衝されたセキュリティー・メカニズムに基づいて適切なプラグインをロードします。クライアント・アプリケーションにより、複数のセキュリティー・プラグインが並行してロードされ、使用される可能性もあります。これは、たとえば、異なるインスタンスから異なるデータベースへの同時接続があるスレッド化されたプログラムで発生する可能性があります。

許可を必要とする、データベース接続やインスタンス接続以外のアクション (データベース構成の更新、データベース・マネージャーの開始と停止、DB2 トレースのオン/オフなど) では、 DB2 クライアント・プログラムが、別のデータベース・マネージャー構成パラメーターで指定されたプラグインをロードします。 *authentication* に GSSPLUGIN が設定されている場合、 DB2 は *local_gssplugin* で指定されたプラグインを使用します。 *authentication* に KERBEROS が設定されている場合、 DB2 は *clnt_krb_plugin* で指定されたプラグインを使用します。その他の場合は、DB2 は *clnt_pw_plugin* で指定されたプラグインを使用します。

セキュリティー・プラグインの作成:

セキュリティー・プラグインを作成する場合は、 DB2 が呼び出す標準認証機能をインプリメントする必要があります。使用可能なプラグイン・タイプに関してインプリメントする必要がある機能は、以下のとおりです。

グループ検索
ユーザーが所属するグループのリストを取得します。

ユーザー ID/パスワード認証
デフォルトのセキュリティー・コンテキストを識別し (クライアントのみ)、パスワードを検証して任意で変更し、指定されたストリングが有効なユーザーを表しているかどうか判別し (サーバーのみ)、クライアントで提供されたユーザー ID またはパスワードをサーバーに送信される前に変更し (クライアントのみ)、指定されたユーザーに関連付けられた DB2 許可 ID を戻します。

GSS-API 認証
必要な GSS-API 関数をインプリメントし、デフォルトのセキュリティー・コンテキストを識別し (クライアントのみ)、ユーザー ID およびパスワードを基に初期証明書を生成して任意でパスワードを変更し (クライアントのみ)、セキュリティー・チケットを作成して受け入れ、指定された GSS-API セキュリティー・コンテキストに関連付けられた DB2 許可 ID を戻します。

関連概念:

- 「*管理ガイド: インプリメンテーション*」の『サーバーでの認証メソッド』
- 585 ページの『セキュリティー・プラグイン・ライブラリーの位置』
- 599 ページの『DB2 によるセキュリティー・プラグインのロード方法』
- 611 ページの『セキュリティー・プラグイン API』

関連資料:

- 「*管理ガイド: パフォーマンス*」の『authentication - 「認証タイプ」構成パラメーター』

- 「*管理ガイド: パフォーマンス*」の『srvcon_auth - サーバーでの着信接続の認証タイプ構成パラメーター』
- 「*アプリケーション開発ガイド アプリケーションの構築および実行*」の『Security プラグインのサンプル』

セキュリティー・プラグイン・ライブラリーの位置

セキュリティー・プラグインを (自分で作成するか、第三者から購入することによって) 取得したら、それらをデータベース・サーバー上の特定の場所にコピーする必要があります。

DB2® は、クライアント・サイドのユーザー認証プラグインを次のディレクトリーで探します。

- UNIX® 32 ビット: `$DB2PATH/security32/plugin/client`
- UNIX 64 ビット: `$DB2PATH/security64/plugin/client`
- WINDOWS 32 ビットおよび 64 ビット: `$DB2PATH¥security¥plugin¥<instance name>¥client`

注: Windows® の場合、サブディレクトリーの <instance name> および client は自動的には作成されません。これらは、インスタンスの所有者が手動で作成しなければなりません。

DB2 は、サーバー・サイドのユーザー認証プラグインを次のディレクトリーで探します。

- UNIX 32 ビット: `$DB2PATH/security32/plugin/server`
- UNIX 64 ビット: `$DB2PATH/security64/plugin/server`
- WINDOWS 32 ビットおよび 64 ビット: `$DB2PATH¥security¥plugin¥<instance name>¥server`

注: Windows の場合、サブディレクトリーの <instance name> および server は自動的には作成されません。これらは、インスタンスの所有者が手動で作成しなければなりません。

DB2 は、グループ・プラグインを次のディレクトリーで探します。

- UNIX 32 ビット: `$DB2PATH/security32/plugin/group`
- UNIX 64 ビット: `$DB2PATH/security64/plugin/group`
- WINDOWS 32 ビットおよび 64 ビット: `$DB2PATH¥security¥plugin¥<instance name>¥group`

注: Windows の場合、サブディレクトリーの <instance name> および group は自動的には作成されません。これらは、インスタンスの所有者が手動で作成しなければなりません。

関連概念:

- 581 ページの『セキュリティー・プラグイン』
- 599 ページの『DB2 によるセキュリティー・プラグインのロード方法』

関連タスク:

- 592 ページの『グループ検索プラグインの展開』
- 593 ページの『ユーザー ID/パスワード・プラグインのデプロイ』
- 595 ページの『GSS-API プラグインのデプロイ』
- 597 ページの『Kerberos プラグインのデプロイ』

関連資料:

- 601 ページの『セキュリティー・プラグイン・ライブラリーに関する制約事項』

セキュリティー・プラグインの命名規則

セキュリティー・プラグイン・ライブラリーは、それぞれのプラットフォーム用の適切なファイル名拡張子を持っていなければなりません。これらの拡張子は、オペレーティング・システム別に次のようになっています。

- Windows®: .DLL
- AIX®: .a
- Linux、HP IPF、および Solaris: .so
- PA-RISC 上の HPUX: .sl

たとえば、`MyPlugin` というセキュリティー・プラグイン・ライブラリーがあるとします。サポートされている各オペレーティング・システムに対する適切なライブラリー・ファイル名は、次のとおりです。

- Windows 32 ビット: MyPlugin.dll
- Windows 64 ビット: MyPlugin64.dll
- AIX 32 または 64 ビット: MyPlugin.a
- SUN 32 または 64 ビット、Linux 32 または 64 ビット、IPF 上の HP 32 または 64 ビット: MyPlugin.so
- PA-RISC 上の HP-UX 32 または 64 ビット: MyPlugin.sl

注: 接尾部 64 は、64 ビット Windows のセキュリティー・プラグインのライブラリー名にのみ必要です。

データベース・マネージャー構成でセキュリティー・プラグインの名前を指定する際は、接尾部 ″64″ のないライブラリーの絶対パス名を使用し、ファイル拡張子と名前の修飾パス部分は省略してください。オペレーティング・システムのいかんを問わず、 `MyPlugin` というセキュリティー・プラグイン・ライブラリーは、次のように登録されます。

```
UPDATE DBM CFG USING CLNT_PW_PLUGIN MyPlugin
```

セキュリティー・プラグイン名は、大文字小文字の区別があり、ライブラリー名と完全に一致している必要があります。 DB2® は、データベース・マネージャー構成パラメーターの値を使用してライブラリー・パスを組み立て、そのライブラリー・パスを使用してセキュリティー・プラグイン・ライブラリーをロードします。

セキュリティー・プラグイン名の競合を防ぐために、プラグインには、使用する認証方式、およびそのプラグインを作成した会社の識別シンボルに基づく名前を付けることをお勧めします。たとえば、 `Foo, Inc.` という会社が `somemethod` という認

証方式をインプリメントするプラグインを作成した場合、そのプラグインには FOOsomemethod.DLL などの名前を付けることができます。

プラグイン名の最大長 (ファイル拡張子および接尾部の 64 を含まない) は、 32 バイトまでに制限されています。データベース・サーバーによってサポートされるプラグインの最大数はありませんが、データベース・マネージャー構成内のコンマで区切られたプラグインのリストの最大長は 255 バイトです。組み込みファイル sqlenv.h には、以下の 2 つの制限を設定する 2 つの定義があります。

```
#define SQL_PLUGIN_NAME_SZ     32     /* plug-in name */
#define SQL_SRVCON_GSSPLUGIN_LIST_SZ 255 /* GSS API plug-in list */
```

セキュリティー・プラグイン・ライブラリー・ファイルには、以下のファイル許可がなければなりません。

- インスタンス所有者によって所有される。
- システム上のすべてのユーザーが読み取れる。
- システム上のすべてのユーザーが実行できる。

関連概念:

- 「*管理ガイド: パフォーマンス*」の『構成パラメーター』
- 581 ページの『セキュリティー・プラグイン』
- 585 ページの『セキュリティー・プラグイン・ライブラリーの位置』

関連タスク:

- 「*管理ガイド: パフォーマンス*」の『構成パラメーターによる DB2 の構成』
- 592 ページの『グループ検索プラグインの展開』
- 593 ページの『ユーザー ID/パスワード・プラグインのデプロイ』
- 595 ページの『GSS-API プラグインのデプロイ』
- 597 ページの『Kerberos プラグインのデプロイ』

関連資料:

- 「*コマンド・リファレンス*」の『UPDATE DATABASE MANAGER CONFIGURATION コマンド』
- 「*管理ガイド: パフォーマンス*」の『clnt_krb_plugin - クライアント Kerberos プラグイン構成パラメーター』
- 「*管理ガイド: パフォーマンス*」の『clnt_pw_plugin - クライアント・ユーザー ID パスワード・プラグイン構成パラメーター』

セキュリティー・プラグインの 2 パーツのユーザー ID のサポート

DB2® UDB for Linux、UNIX®、および Windows® 製品は、2 パーツのユーザー ID と、 2 パーツのユーザー ID から 2 パーツの許可 ID へのマッピングをサポートしています。

たとえば、ドメインとユーザー ID からなる Windows オペレーティング・システムの 2 パーツのユーザー ID (例、 MEDWAY¥peter) について考えます。この 2 パーツのユーザー ID の例で、MEDWAY はドメインであり、 peter はユーザー名です。DB2 では、この 2 パーツのユーザー ID を、 1 パーツの authid と 2 パーツの authid のどちらにマップするかを指定することができます。

バージョン 8.2 より前の DB2 では、 1 パーツの authid にマップする 1 パーツのユーザー ID しかありませんでした。 DB2 バージョン 8.2 では、デフォルトで、1 パーツのユーザー ID が 1 パーツの authid にマップし、 2 パーツのユーザー ID が 1 パーツの authid にマップします。 2 パーツのユーザー ID から 2 パーツの authid へのマッピングもサポートされていますが、デフォルト動作ではありません。

これにより、ユーザーは以下を使用してデータベースに接続できます。

```
db2 connect to db user MEDWAY¥peter using pw
```

この例では、デフォルト動作が使用される場合、ユーザー ID MEDWAY¥peter は許可 ID PETER にマップされます。 2 パーツのユーザー ID から 2 パーツの許可 ID へのマッピングのサポートが使用される場合、この例では許可 ID は MEDWAY¥PETER となります。

DB2 の 2 パーツの authid にマップする 2 パーツのユーザー ID の認証サポートを使用可能にするには、データベース・マネージャー構成パラメーターを設定することによって、 2 パーツのユーザー ID から 2 パーツの authid へのマッピングを実行するセキュリティー・プラグインを使用可能にする必要があります。これらの構成パラメーターについては、以下で説明します。

DB2 は、2 セットの認証プラグインを提供しています。 1 つは、1 パーツの authid へのユーザー ID のマッピング (すなわち、 1 パーツのユーザー ID から 1 パーツの authid へのマッピングと、 2 パーツのユーザー ID から 1 パーツの authid へのマッピング) 専用です。 2 番目のセットは、1 パーツのユーザー ID を 1 パーツの authid にマップし、 2 パーツのユーザー ID を 2 パーツの authid にマップするための柔軟性を追加するものです。

作業環境におけるユーザー名が、異なる場所で定義された複数のアカウント (ローカル・アカウント、ドメイン・アカウント、およびトラステッド・ドメイン・アカウントなど) にマップされることがある場合は、 2 パーツの authid のマッピングを使用可能にするプラグインを指定できます。

PETER などの 1 パーツの authid と、ドメインとユーザー ID が結合した MEDWAY¥PETER のような 2 パーツの authid は、機能的に異なる authid であることに留意することは重要です。これらの一方の authid に関連付けられている特権セットは、他方の authid に関連付けられている特権セットとは完全に異なります。 1 パーツの authid と 2 パーツの authid を扱う際には注意が必要です。

次の表では、DB2 で提供されているプラグインの種類と、特定の認証インプリメンテーション用のプラグイン名を示しています。

表 82. DB2 セキュリティー・プラグイン

認証タイプ	1 パーツのユーザー ID のプラグインの名前	2 パーツのユーザー ID のプラグインの名前
ユーザー ID/パスワード (クライアント)	IBMOSauthclient	IBMOSauthclientTwoPart
ユーザー ID/パスワード (サーバー)	IBMOSauthserver	IBMOSauthserverTwoPart
Kerberos	IBMkrb5	IBMkrb5TwoPart

注: Windows 64 ビット・プラットフォームでは、ここにリストされているプラグイン名に ″64″ が付加されます。

2 パーツのユーザー ID を 2 パーツの authid にマップするには、 2 パーツのプラグイン (これはデフォルトのプラグインではありません) を使用することを指定する必要があります。セキュリティー・プラグインは、セキュリティー関連のデータベース・マネージャー構成パラメーターを設定することによって、インスタンス・レベルで指定します。それは次のようにします。

2 パーツのユーザー ID を 2 パーツの authid にマップするサーバー認証の場合は、以下のように設定する必要があります。

- SRVCON_PW_PLUGIN を IBMOSauthserverTwoPart に設定
- CLNT_PW_PLUGIN を IBMOSauthclientTwoPart に設定

2 パーツのユーザー ID を 2 パーツの authid にマップするクライアント認証の場合は、以下のように設定する必要があります。

- SRVCON_PW_PLUGIN を IBMOSauthserverTwoPart に設定
- CLNT_PW_PLUGIN を IBMOSauthclientTwoPart に設定

2 パーツのユーザー ID を 2 パーツの authid にマップする Kerberos 認証の場合は、以下のように設定する必要があります。

- SRVCON_GSSPLUGIN_LIST を IBMOSkrb5TwoPart に設定
- CLNT_KRB_PLUGIN を IBMkrb5TwoPart に設定

セキュリティー・プラグイン・ライブラリーは、 Microsoft® Windows Security Account Manager 互換形式で指定された 2 パーツのユーザー ID を受け入れます。これは、たとえば domain¥user ID の形式です。接続時の DB2 の認証プロセスおよび許可プロセスでは、ドメインとユーザー ID の両方の情報が使用されます。

ユーザー ID/パスワード・プラグインまたは Kerberos プラグインを必要とする認証タイプを指定している場合は、「1 パーツのユーザー ID のプラグインの名前」列にリストされているプラグインがデフォルトで使用されます。

新規データベースを作成する際は、既存データベース内の 1 パーツ authid との競合を防止するため、 2 パーツのプラグインをインプリメントすることを考慮するようお勧めします。 2 パーツの authid の認証を使用する新規データベースは、単一パーツの authid を使用するデータベースとは別のインスタンス内で作成する必要があります。

関連概念:

- 「*管理ガイド: インプリメンテーション*」の『DB2 for Windows NT および Windows NT セキュリティーの紹介』

関連タスク:

- 「*管理ガイド: インプリメンテーション*」の『DB2 for Windows NT 認証でのグループおよびドメイン・セキュリティーの使用』

関連資料:

- 「*管理ガイド: パフォーマンス*」の『clnt_pw_plugin - クライアント・ユーザー ID パスワード・プラグイン構成パラメーター』
- 「*管理ガイド: パフォーマンス*」の『srvcon_pw_plugin - サーバーでの着信接続用のユーザー ID-パスワード・プラグイン構成パラメーター』

セキュリティー・プラグインの 32 ビットと 64 ビットに関する考慮事項

通常、32 ビット DB2® インスタンスは、32 ビット・セキュリティー・プラグインを使用し、 64 ビット DB2 インスタンスは 64 ビット・セキュリティー・プラグインを使用します。しかし、64 ビット・インスタンス上では、 DB2 は、32 ビット・プラグイン・ライブラリーを必要とする 32 ビット・アプリケーションをサポートします。

32 ビットと 64 ビットの両方のアプリケーションが実行できるデータベース・インスタンスは、ハイブリッド・インスタンスといいます。ハイブリッド・インスタンスがあり、32 ビット・アプリケーションを実行しようとしている場合は、必要な 32 ビット・セキュリティー・プラグインが、 32 ビット・プラグイン・ディレクトリー内に用意されていることを確認してください。 UNIX® オペレーティング・システム上のハイブリッド DB2 インスタンスの場合は、 `security32` および `security64` というディレクトリーが表示されます。 Windows® 64 ビットのハイブリッド・インスタンスの場合、 32 ビットと 64 ビットの両方のセキュリティー・プラグインが同一のディレクトリー内にありますが、 64 ビット・プラグイン名には、接尾部の 64 が付いています。

32 ビット・インスタンスを 64 ビット・インスタンスに移行する場合は、 64 ビット用に再コンパイルされたセキュリティー・プラグイン用のバージョンを取得する必要があります。

64 ビット・プラグイン・ライブラリーを提供しないベンダーからセキュリティー・プラグインを取得した場合は、 32 ビット・アプリケーションを実行する 64 ビット・スタブをインプリメントできます。この場合、セキュリティー・プラグインは、ライブラリーではなく外部プログラムになります。

関連概念:

- 581 ページの『セキュリティー・プラグイン』
- 585 ページの『セキュリティー・プラグイン・ライブラリーの位置』

関連タスク:

- 「*アプリケーション開発ガイド アプリケーションの構築および実行*」の『32 ビット環境から 64 ビット環境へのアプリケーションの移行』

セキュリティー・プラグインの問題判別

セキュリティー・プラグインの問題は、SQL エラー経由と管理ログ経由の 2 つの経路で報告されます。

以下は、セキュリティー・プラグインに関係した SQLCODE 値です。

- SQLCODE -1365 は、db2start または db2stop の間にプラグイン・エラーが発生すると戻されます。
- SQLCODE -1366 は、ローカル許可の問題がある場合に戻されます。
- SQLCODE -30082 は、すべての接続に関係したプラグイン・エラーで戻されます。

管理ログは、セキュリティー・プラグインのデバッグおよび管理のための良い情報源です。 UNIX® 上で管理ログを参照するには、sqllib/db2dump/<instance name>.nfy を調べます。 Windows オペレーティング・システム上で管理ログを参照するには、「イベント ビューア」ツールを使用します。「イベント ビューア」ツールは、Windows オペレーティング・システムの「スタート」ボタンから 「設定」->「コントロール パネル」->「管理ツール」->「イベント ビューア」の順にナビゲートすると見付かります。以下は、セキュリティー・プラグインに関係した管理ログ値です。

- 13000 は、GSS-API セキュリティー・プラグイン API の呼び出しがエラーによって失敗し、オプションのエラー・メッセージが戻されたことを示します。

```
SQLT_ADMIN_GSS_API_ERROR (13000)
Plug-in "<plug-in name>" received error code "<error code>" from
GSS API  "<gss api name>" with the error message "<error message>"
```

- 13001 は、DB2® セキュリティー・プラグイン API の呼び出しがエラーによって失敗し、オプションのエラー・メッセージが戻されたことを示します。

```
SQLT_ADMIN_PLUGIN_API_ERROR(13001)
Plug-in "<plug-in name>" received error code "<error code>" from DB2
security plug-in API "<gss api name>" with the error message
"<error message>"
```

- 13002 は、DB2 がプラグインをアンロードできなかったことを示します。

```
SQLT_ADMIN_PLUGIN_UNLOAD_ERROR (13002)
Unable to unload plug-in "<plug-in name>". No further action required.
```

- 13003 は、無効なプリンシパル名を示します。

```
SQLT_ADMIN_INVALID_PRIN_NAME (13003)
The principal name "<principal name>" used for "<plug-in name>"
is invalid. Fix the principal name.
```

- 13004 は、プラグイン名が無効なことを示します。パス区切り文字 (UNIX の場合は "/"、Windows® の場合は "¥") をプラグイン名の一部として使用することはできません。

```
SQLT_ADMIN_INVALID_PLGN_NAME (13004)
The plug-in name "<plug-in name>" is invalid. Fix the plug-in name.
```

- 13005 は、セキュリティー・プラグインがロードできなかったことを示します。プラグインを正しいディレクトリーに入れ、該当するデータベース・マネージャー構成パラメーターを更新してください。

```
SQLT_ADMIN_PLUGIN_LOAD_ERROR (13005)
Unable to load plug-in "<plug-in name>". Verify the plug-in existence and
directory where it is located is correct.
```

- 13006 は、セキュリティー・プラグインによって予期しないエラーが検出されたことを示します。すべての db2support 情報を収集し、可能であれば db2trc を取り込んで、 IBM® サポートに支援を依頼してください。

```
SQLT_ADMIN_PLUGIN_UNEXP_ERROR (13006)
Plug-in encountered unexpected error. Contact IBM Support for further assistance.
```

注: Windows 64 ビットのデータベース・サーバーで上でセキュリティー・プラグインを使用しているときに、セキュリティー・プラグインのロード・エラーが表示された場合は、セキュリティー・プラグインの 32 ビットと 64 ビットに関する考慮事項およびセキュリティー・プラグインの命名規則のトピックを参照してください。 64 ビット・プラグイン・ライブラリーには、ライブラリー名に 64 という接尾部が付いていなければなりませんが、セキュリティー・プラグインのデータベース・マネージャー構成パラメーターへの入力にはこの接尾部は含めません。

関連概念:

- 「システム・モニター *ガイドおよびリファレンス*」の『イベント・モニター』
- 114 ページの『SQLCODE、SQLSTATE、および SQLWARN フィールドのエラー情報』
- 192 ページの『C および C++ における SQLSTATE および SQLCODE 変数』
- 765 ページの『IBM のリレーショナル・データベース・システム間での SQLCODE と SQLSTATE の相違点』
- *Troubleshooting Guide* の『DB2 トレース (db2trc)』
- 581 ページの『セキュリティー・プラグイン』
- 590 ページの『セキュリティー・プラグインの 32 ビットと 64 ビットに関する考慮事項』
- 611 ページの『セキュリティー・プラグイン API』

関連資料:

- 「*SQL リファレンス 第 2 巻*」の『CREATE EVENT MONITOR ステートメント』
- 「コマンド・リファレンス」の『db2trc - トレース・コマンド』
- 605 ページの『セキュリティー・プラグインのエラー・メッセージ』
- 647 ページの『GSS-API 認証プラグインに必要な API および定義』
- 612 ページの『グループ検索プラグイン用の API』
- 622 ページの『ユーザー ID/パスワード認証プラグインの API』

グループ検索プラグインの展開

DB2 セキュリティー・システムのグループ検索動作をカスタマイズする場合は、独自のグループ検索プラグインを開発するか、または第三者からこれを購入できます。

データベース管理システムに適したグループ検索プラグインを入手したら、それを展開することができます。

手順:

グループ検索プラグインをデータベース・サーバー上に展開するには、以下のステップを実行します。

1. グループ検索プラグイン・ライブラリーをサーバーのグループ・プラグイン・ディレクトリーに置きます。

2. データベース・マネージャー構成パラメーター *group_plugin* をプラグインの名前で更新します。

グループ検索プラグインをデータベース・クライアント上に展開するには、以下のステップを実行します。

1. グループ検索プラグイン・ライブラリーをクライアントのグループ・プラグイン・ディレクトリーに置きます。
2. データベース・マネージャー構成パラメーター *group_plugin* をプラグインの名前で更新します。

関連概念:

- 581 ページの『セキュリティー・プラグイン』
- 585 ページの『セキュリティー・プラグイン・ライブラリーの位置』
- 586 ページの『セキュリティー・プラグインの命名規則』

関連タスク:

- 593 ページの『ユーザー ID/パスワード・プラグインのデプロイ』
- 595 ページの『GSS-API プラグインのデプロイ』
- 597 ページの『Kerberos プラグインのデプロイ』

関連資料:

- 「*管理ガイド: パフォーマンス*」の『group_plugin - グループ・プラグイン構成パラメーター』

ユーザー ID/パスワード・プラグインのデプロイ

DB2 セキュリティー・システムのユーザー ID/パスワード認証動作をカスタマイズする場合は、独自のユーザー ID/パスワード認証プラグインを開発するか、または第三者からこれを購入できます。

データベース管理システムに適したユーザー ID/パスワード認証プラグインを入手したら、それらをデプロイすることができます。

すべてのユーザー ID/パスワード・ベース認証プラグインは、意図しているそれらプラグインの使用法に応じて、クライアントのプラグイン・ディレクトリーか、サーバーのプラグイン・ディレクトリーのいずれかに置く必要があります。プラグインがクライアントのプラグイン・ディレクトリーに置かれる場合、それはローカル許可検査のためと、クライアントがサーバーと接続しようとするときに使用されます。プラグインがサーバーのプラグイン・ディレクトリーに置かれる場合、それはサーバーへの着信接続の処理のためと、 USER または GROUP キーワードが指定されずに GRANT ステートメントが発行された場合の authid の存在とその有効性の検査のために使用されます。

ほとんどの場合、ユーザー ID/パスワード認証で必要となるのは、サーバー・サイドのプラグインのみです。一般にそれほど役に立ちませんが、クライアントのユーザー ID/パスワード・プラグインのみを使用することも可能です。非常に稀なことですが、クライアントとサーバーの両方に、一致するユーザー ID/パスワード・プラグインが必要になることもあります。

手順:

ユーザー ID/パスワード認証プラグインをデータベース・サーバー上にデプロイするには、以下のステップを実行します。

1. ユーザー ID/パスワード認証プラグイン・ライブラリーをサーバーのプラグイン・ディレクトリーに置きます。
2. データベース・マネージャー構成パラメーター *srvcon_pw_plugin* をサーバー・プラグインの名前で更新します。

   このプラグインは、サーバーが接続 (CONNECT) 要求およびアタッチ (ATTACH) 要求を処理する際に使用します。
3. 次のいずれかを行ってください。
   - データベース・マネージャー構成パラメーター *srvcon_auth* に、CLIENT、SERVER、SERVER_ENCRYPT、DATA_ENCRYPT、または DATA_ENCRYPT_CMP の認証タイプを設定します。あるいは、
   - データベース・マネージャー構成パラメーター *srvcon_auth* に NOT_SPECIFIED を設定し、 *authentication* に CLIENT、 SERVER、 SERVER_ENCRYPT、 DATA_ENCRYPT、または DATA_ENCRYPT_CMP の認証タイプを設定します。

ユーザー ID/パスワード認証プラグインをデータベース・クライアント上にデプロイするには、以下のステップを実行します。

1. ユーザー ID/パスワード認証プラグイン・ライブラリーをクライアント上のクライアント・プラグイン・ディレクトリーに置きます。
2. データベース・マネージャー構成パラメーター *clnt_pw_plugin* をクライアント・プラグインの名前で更新します。

   このプラグインは、認証がどこで行われているかに関係なく、つまり、クライアント認証が使用可能なとき以外にもロードされ、呼び出されます。

ユーザー ID/パスワード認証プラグインを使用したクライアント、サーバー、またはゲートウェイでのローカル許可に関しては、以下のステップを実行します。

1. ユーザー ID/パスワード認証プラグイン・ライブラリーをクライアント、サーバー、またはゲートウェイ上のクライアント・プラグイン・ディレクトリーに置きます。
2. データベース・マネージャー構成パラメーター *clnt_pw_plugin* をプラグインの名前で更新します。
3. *authentication* データベース・マネージャー構成パラメーターに、 CLIENT、 SERVER、 SERVER_ENCRYPT、 DATA_ENCRYPT、または DATA_ENCRYPT_CMP を設定します。

関連概念:

- 581 ページの『セキュリティー・プラグイン』
- 585 ページの『セキュリティー・プラグイン・ライブラリーの位置』

関連タスク:

- 592 ページの『グループ検索プラグインの展開』
- 595 ページの『GSS-API プラグインのデプロイ』

関連資料:

- 「*管理ガイド: パフォーマンス*」の『authentication - 「認証タイプ」構成パラメーター』
- 「*SQL リファレンス 第 2 巻*」の『GRANT (データベース権限) ステートメント』
- 「*管理ガイド: パフォーマンス*」の『clnt_pw_plugin - クライアント・ユーザー ID パスワード・プラグイン構成パラメーター』
- 「*管理ガイド: パフォーマンス*」の『srvcon_auth - サーバーでの着信接続の認証タイプ構成パラメーター』
- 「*管理ガイド: パフォーマンス*」の『srvcon_pw_plugin - サーバーでの着信接続用のユーザー ID-パスワード・プラグイン構成パラメーター』

GSS-API プラグインのデプロイ

DB2 セキュリティー・システムの認証動作をカスタマイズする場合は、 GSS-API を使用して独自の認証プラグインを開発するか、または第三者からこれを購入できます。

データベース管理システムに適した GSS-API 認証プラグインを入手したら、それらをデプロイすることができます。

GSS-API または Kerberos プラグインの場合は、クライアントとサーバー上に一致するプラグイン・タイプがなければなりません。クライアントとサーバー上のプラグインは、同一のベンダーのものである必要はありませんが、これらは互換性のある GSS-API トークンを生成して使用する必要があります。たとえば、Kerberos プラグインは標準化されているので、クライアントとサーバー上にはどのような組み合わせの Kerberos プラグインをデプロイしても構いません。しかし、それほど標準化されていない *x.509* 証明書などの種々の GSS-API メカニズムのインプリメンテーション間には、完全な互換性はない可能性があります。

すべての GSS-API 認証プラグインは、意図しているそれらプラグインの使用法に応じて、クライアントのプラグイン・ディレクトリーか、サーバーのプラグイン・ディレクトリーのいずれかに置く必要があります。プラグインがクライアントのプラグイン・ディレクトリーに置かれる場合、それはローカル許可検査のためと、クライアントがサーバーと接続しようとするときに使用されます。プラグインがサーバーのプラグイン・ディレクトリーに置かれる場合、それはサーバーへの着信接続の処理のためと、 USER または GROUP キーワードが指定されずに GRANT ステートメントが発行された場合の authid の存在とその有効性の検査のために使用されます。

手順:

GSS-API 認証プラグインをデータベース・サーバー上にデプロイするには、以下のステップを実行します。

1. GSS-API 認証プラグイン・ライブラリーをサーバー上のサーバー・プラグイン・ディレクトリーに置きます。このディレクトリーには、多数の GSS-API プラグインをコピーすることができます。

2. データベース・マネージャー構成パラメーター *srvcon_gssplugin_list* を、GSS-API プラグイン・ディレクトリーにインストールされたプラグインの名前の順番のコンマ区切りのリストで更新します。
3. 次のいずれかを行ってください。
   - データベース・マネージャー構成パラメーター *srvcon_auth* に、GSSPLUGIN を設定します。あるいは、
   - データベース・マネージャー構成パラメーター *srvcon_auth* に NOT_SPECIFIED を設定し、 *authentication* に GSSPLUGIN を設定します。

GSS-API 認証プラグインをデータベース・クライアント上にデプロイするには、以下のステップを実行します。

1. GSS-API 認証プラグイン・ライブラリーをクライアント上のクライアント・プラグイン・ディレクトリーに置きます。このディレクトリーには、多数の GSS-API プラグインをコピーすることができます。クライアントは、クライアント上にあるサーバーのプラグインのリストに含まれている最初の GSS-API を選出することによって、 CONNECT/ATTACH の際の認証に適した GSS-API プラグインを選択します。
2. オプション: クライアントがアクセスするデータベースをカタログし、クライアントが GSS-API 認証プラグインのみを認証メカニズムとして受け入れることを示します。以下に例を示します。

   ```
   CATALOG DB testdb AT NODE testnode AUTHENTICATION GSSPLUGIN
   ```

GSS-API 認証プラグインを使用したクライアント、サーバー、またはゲートウェイでのローカル許可に関しては、以下のステップを実行します。

1. GSS-API 認証プラグイン・ライブラリーをクライアント、サーバー、またはゲートウェイ上のクライアント・プラグイン・ディレクトリーに置きます。
2. データベース・マネージャー構成パラメーター *local_gssplugin* をプラグインの名前で更新します。
3. *authentication* データベース・マネージャー構成パラメーターに、 GSSPLUGIN、または GSS_SERVER_ENCRYPT を設定します。

関連概念:

- 581 ページの『セキュリティー・プラグイン』
- 585 ページの『セキュリティー・プラグイン・ライブラリーの位置』

関連資料:

- 「*管理ガイド: パフォーマンス*」の『authentication - 「認証タイプ」構成パラメーター』
- 「*コマンド・リファレンス*」の『CATALOG DATABASE コマンド』
- 「*管理ガイド: パフォーマンス*」の『local_gssplugin - ローカル・インスタンス・レベル許可に使用する GSS API プラグイン構成パラメーター』
- 「*管理ガイド: パフォーマンス*」の『srvcon_auth - サーバーでの着信接続の認証タイプ構成パラメーター』
- 「*管理ガイド: パフォーマンス*」の『srvcon_gssplugin_list - サーバーでの着信接続用の GSS API プラグインのリスト構成パラメーター』

- 「*アプリケーション開発ガイド アプリケーションの構築および実行*」の『Security プラグインのサンプル』

Kerberos プラグインのデプロイ

DB2 セキュリティー・システムの Kerberos 認証動作をカスタマイズする場合は、独自の Kerberos 認証プラグインを開発するか、または第三者からこれを購入できます。

データベース管理システムに適した Kerberos 認証プラグインを入手したら、それらをデプロイすることができます。

手順:

Kerberos 認証プラグインをデータベース・サーバー上にデプロイするには、以下のステップを実行します。

1. Kerberos 認証プラグイン・ライブラリーをサーバー上のサーバー・プラグイン・ディレクトリーに置きます。
2. 順番のコンマ区切りのリストとして指定されるデータベース・マネージャー構成パラメーター *srvcon_gssplugin_list* を更新して、 Kerberos サーバー・プラグイン名を含めます。このリストの中の 1 つのプラグインのみを Kerberos プラグインにすることができます。

   このリストがブランクで、 *authentication* に KERBEROS または KRB_SVR_ENCRYPT が設定されている場合は、デフォルトの DB2 Kerberos プラグイン IBMkrb5 が使用されます。
3. 次のいずれかを行ってください。
   - データベース・マネージャー構成パラメーター *srvcon_auth* に、 KERBEROS または KRB_SERVER_ENCRYPT の認証タイプを設定します。あるいは、
   - データベース・マネージャー構成パラメーター *srvcon_auth* に NOT_SPECIFIED を設定し、 *authentication* に KERBEROS または KRB_SERVER_ENCRYPT の認証タイプを設定します。

Kerberos 認証プラグインをデータベース・クライアント上にデプロイするには、以下のステップを実行します。

1. Kerberos 認証プラグイン・ライブラリーをクライアント上のクライアント・プラグイン・ディレクトリーに置きます。
2. データベース・マネージャー構成パラメーター *clnt_krb_plugin* をクライアント Kerberos プラグインの名前で更新します。

   clnt_krb_plugin がブランクの場合、 DB2 はクライアントは Kerberos 認証を使用できないとみなします。サーバーがプラグインをサポートできない場合にのみ、これは適切です。詳しくは、セキュリティー・プラグインの使用上の制限を参照してください。サーバーとクライアントの両方がセキュリティー・プラグインをサポートしている場合は、サーバーがリストされた *IBMkrb5* という名前の GSS-API プラグインを持っているので、クライアントは *clnt_krb_plugin* の値を使用しません。

Kerberos 認証プラグインを使用したクライアント、サーバー、またはゲートウェイでのローカル許可に関しては、以下のステップを実行します。

a. Kerberos 認証プラグイン・ライブラリーをクライアント、サーバー、またはゲートウェイ上のクライアント・プラグイン・ディレクトリーに置きます。
b. データベース・マネージャー構成パラメーター *clnt_krb_plugin* をプラグインの名前で更新します。
c. *authentication* データベース・マネージャー構成パラメーターに、KERBEROS または KRB_SERVER_ENCRYPT を設定します。

DB2 が提供する Kerberos プラグインの名前は、`IBMkrb5` です。

3. オプション: クライアントがアクセスするデータベースをカタログし、クライアントが Kerberos 認証プラグインのみを使用することを示します。以下に例を示します。

```
CATALOG DB testdb AT NODE testnode AUTHENTICATION KERBEROS
        TARGET PRINCIPAL service/host@REALM
```

注: Kerberos をサポートするプラットフォームの場合、 `IBMkrb5` ライブラリーはクライアント・プラグイン・ディレクトリーに置かれます。 Kerberos プラグインは GSS-API プラグインを使用してインプリメントされているので、 DB2 はこのライブラリーを有効な GSS-API プラグインとして認識します。

関連概念:

- 581 ページの『セキュリティー・プラグイン』
- 585 ページの『セキュリティー・プラグイン・ライブラリーの位置』

関連タスク:

- 592 ページの『グループ検索プラグインの展開』
- 593 ページの『ユーザー ID/パスワード・プラグインのデプロイ』
- 595 ページの『GSS-API プラグインのデプロイ』

関連資料:

- 「*管理ガイド: パフォーマンス*」の『authentication - 「認証タイプ」構成パラメーター』
- 「コマンド・リファレンス」の『CATALOG DATABASE コマンド』
- 「*管理ガイド: パフォーマンス*」の『clnt_krb_plugin - クライアント Kerberos プラグイン構成パラメーター』
- 「*管理ガイド: パフォーマンス*」の『srvcon_auth - サーバーでの着信接続の認証タイプ構成パラメーター』
- 「*管理ガイド: パフォーマンス*」の『srvcon_gssplugin_list - サーバーでの着信接続用の GSS API プラグインのリスト構成パラメーター』

第 27 章 セキュリティー・プラグインの作成

DB2 によるセキュリティー・プラグインのロード方法 599
セキュリティー・プラグイン・ライブラリーに関する制約事項 601
セキュリティー・プラグインの戻りコード. 603
セキュリティー・プラグインのエラー・メッセージ 605
セキュリティー・プラグイン API の呼び出し順序 606

DB2 によるセキュリティー・プラグインのロード方法

各プラグイン・ライブラリーには、以下のような、プラグイン・タイプに応じた特定の名前を持つ初期化関数が含まれていなければなりません。

- サーバー・サイド認証プラグイン: db2secServerAuthPluginInit()
- クライアント・サイド認証プラグイン: db2secClientAuthPluginInit()
- グループ・プラグイン: db2secGroupPluginInit()

この関数は、プラグイン初期化関数と呼ばれます。プラグイン初期化関数は、指定されたプラグインを初期化し、プラグインの関数を呼び出すために必要な情報を DB2® に提供します。プラグイン初期化関数は、以下のパラメーターを受け入れます。

- DB2 がサポートできる関数ポインター構造の最高バージョン番号
- インプリメンテーションを必要とするすべての API を指すポインターを含む構造を指すポインター
- db2diag.log ファイルにログ・メッセージを追加する関数を指すポインター
- エラー・メッセージ・ストリングを指すポインター
- エラー・メッセージの長さ

以下は、グループ検索プラグインの初期化関数の関数シグニチャーです。

```
SQL_API_RC SQL_API_FN db2secGroupPluginInit(
 db2int32 version,
 void *group_fns,
  db2secLogMessage *logMessage_fn,
  char **errormsg,
 db2int32  *errormsglen);
```

注: プラグイン・ライブラリーは、C または C++ でのみインプリメントできます。プラグイン・ライブラリーを C++ でコンパイルする場合は、 `extern "C"` を使用してすべての関数を宣言する必要があります。 DB2 は、基礎オペレーティング・システムの動的ローダーを利用して、 C++ のユーザー作成プラグイン・ライブラリーの内部で使用されている C++ コンストラクターおよびデストラクターを処理します。

これは、プラグイン・ライブラリー内の、規定の関数名を持たなければならない唯一の関数です。その他のプラグイン関数は、初期化関数から戻された関数ポインターを通して参照されます。サーバー・プラグインは、db2start の際にサーバー上にロードされます。クライアント・プラグインは、クライアント上で必要とされるとき

にロードされます。 DB2 は、プラグイン・ライブラリーをロードするとすぐに、この関数の位置を解決して呼び出します。この関数の具体的なタスクは、以下のとおりです。

- 関数ポインターを、適切な関数構造を指すポインターにキャストする
- ライブラリー内の他の関数を指すポインターに入力する
- 戻される関数ポインター構造のバージョン番号に入力する

DB2 はプラグイン初期化関数を複数回呼び出すことがあります。このことが起こるのは、アプリケーションが動的に DB2 クライアント・ライブラリーをロードしてからこれをアンロードして再ロードし、再ロードの前と後の両方にプラグインから認証関数を実行する場合です。このような場合は、プラグイン・ライブラリーがアンロードされず、したがって再ロードもされないことがあります。ただし、この動作はオペレーティング・システムによって異なります。

DB2 がプラグイン初期化関数を複数回呼び出す別の例は、データベース・サーバー自体がクライアントとして振る舞うことがある、ストアード・プロシージャーやフェデレーテッド・システム呼び出しの実行時です。データベース・サーバー上のクライアント・プラグインとサーバー・プラグインが同じファイル内にある場合、DB2 はプラグイン初期化関数を 2 回呼び出す可能性があります。

`db2secGroupPluginInit` が複数回呼び出されたことを検出した場合、プラグインは、プラグイン・ライブラリーを終了して再初期化するよう指示されたときと同じ処理をする必要があります。したがって、プラグイン初期化関数は、`db2secPluginTerm` を呼び出すと実行されるクリーンアップ全体を実行してから、再び関数ポインターのセットを戻す必要があります。

UNIX® オペレーティング・システムを稼働している DB2 サーバーでは、 DB2 は db2start の後にプラグイン・ライブラリーを異なるプロセスで複数回ロードして再初期化することがあります。

関連概念:

- 581 ページの『セキュリティー・プラグイン』
- 585 ページの『セキュリティー・プラグイン・ライブラリーの位置』

関連資料:

- 601 ページの『セキュリティー・プラグイン・ライブラリーに関する制約事項』
- 603 ページの『セキュリティー・プラグインの戻りコード』
- 606 ページの『セキュリティー・プラグイン API の呼び出し順序』
- 615 ページの『db2secGroupPluginInit - グループ・プラグインの初期化』
- 616 ページの『db2secPluginTerm - グループ・プラグイン・リソースのクリーンアップ』
- 629 ページの『db2secClientAuthPluginInit - クライアント認証プラグインの初期化』
- 641 ページの『db2secServerAuthPluginInit - サーバー認証プラグインの初期化』

セキュリティー・プラグイン・ライブラリーに関する制約事項

以下は、プラグイン・ライブラリーの作成に関連した制約事項です。

C-linkage

プラグイン・ライブラリーは、C-linkage とリンクされていなければなりません。プロトタイプ、プラグインのインプリメントに必要なデータ構造、およびエラー・コード定義を規定するヘッダー・ファイルは、C/C++ の場合にのみ準備されます。プラグイン・ライブラリーが C++ としてコンパイルされている場合は、 DB2 がロード時に解決する関数を extern ″C″ を用いて宣言する必要があります。

.NET 共通言語ランタイムはサポートされていません。

プラグイン・ライブラリーのソース・コードのコンパイルおよびリンクにおいて、 .NET 共通言語ランタイム (CLR) はサポートされません。

シグナル・ハンドラー

プラグイン・ライブラリーは、シグナル・ハンドラーをインストールしたり、シグナル・マスクを変更したりしてはなりません。なぜならば、これをすると、DB2 のシグナル・ハンドラーが妨げられるからです。 DB2 のシグナル・ハンドラーが妨げられると、プラグイン・コード自体にあるトラップを含めたエラーを報告してリカバリーする DB2 の機能が著しく妨げられます。さらに、プラグイン・ライブラリーは、C++ 例外を出してはなりません。なぜなら、これも DB2 のエラー処理を妨げるからです。

スレッド・セーフ

プラグイン・ライブラリーは、スレッド・セーフおよび再入可能でなければなりません。プラグイン初期化関数は、再入可能でなくてもよい唯一の API です。なぜなら、プラグイン初期化関数は異なるプロセスから複数回呼び出される可能性がありますが、その場合は、プラグインがすべての使用済みリソースをクリーンアップして、プラグイン自体を再初期化するからです。

終了ハンドラー、および標準 C ライブラリーとオペレーティング・システム呼び出しのオーバーライド

プラグイン・ライブラリーは、標準 C ライブラリーやオペレーティング・システム呼び出しをオーバーライドしてはなりません。さらに、プラグイン・ライブラリーは、終了ハンドラーや `pthread_atfork` ハンドラーにインストールしてはなりません。終了ハンドラーはプログラムが終了する前にアンロードされる可能性があるので、終了ハンドラーを使用することはお勧めしません。

ライブラリーの従属関係

Linux または Unix では、プラグイン・ライブラリーをロードするプロセスは、 setuid か setgid になります。このことは、プロセスが `$LD_LIBRARY_PATH`、`$SHLIB_PATH`、または `$LIBPATH` 環境変数を利用して従属ライブラリーを検索できないことを意味します。したがって、従属ライブラリーが次のような他の方法でアクセス可能にされていない限り、プラグイン・ライブラリーが他のライブラリーに従属していてはなりません。

- `/lib` または `/usr/lib` の中に入れる。
- それらが常駐するディレクトリーを OS ワイド (Linux 上の `ld.so.conf` ファイル内など) で指定する。

- プラグイン・ライブラリー自体の RPATH で指定する。

この制限は、Windows オペレーティング・システムには当てはまりません。

シンボルの重複

可能であれば、プラグイン・ライブラリーは、シンボルの重複の可能性を減らす使用可能オプション (アンバインドされた外部シンボル参照を削減するオプションなど) を用いてコンパイルおよびリンクする必要があります。たとえば、HP、Sun Solaris、および Linux 上で ″-Bsymbolic″ リンカー・オプションを使用するなら、シンボルの重複に関係した問題を防ぐことができます。ただし、AIX プラットフォームで作成されたプラグインの場合は、`"-brtl"` リンカー・オプションを使用すると問題が生じるので、これは明示的にも暗黙的にも使用しないでください。

32 ビット・アプリケーションと 64 ビット・アプリケーション

32 ビット・アプリケーションは、32 ビット・プラグインを使用する必要があります。 64 ビット・アプリケーションは、64 ビット・プラグインを使用する必要があります。詳細は、『セキュリティー・プラグインの 32 ビットと 64 ビットに関する考慮事項』のトピックを参照してください。

テキスト・ストリング

入力テキスト・ストリングがヌル終了になっているという保証はなく、出力ストリングがヌル終了である必要はありません。その代わり、すべての入力ストリングに対して整数の長さが指定され、戻される長さとして整数を指すポインターが指定されます。

authid パラメーターの引き渡し

DB2 がプラグインに渡す authid パラメーター (入力 authid パラメーター) には、埋め込みブランクが除かれた大文字の authid が含まれます。プラグインが DB2 に戻す authid パラメーター (出力 authid パラメーター) には特別な処理は必要ありませんが、 DB2 は authid を取り込むと、内部 DB2 規格に準じてこれを大文字にしてブランクを埋め込みます。

パラメーターのサイズ制限

プラグイン API は、パラメーターの長さ制限として以下を使用します。

```
#define DB2SEC_MAX_AUTHID_LENGTH 255
#define DB2SEC_MAX_USERID_LENGTH 255
#define DB2SEC_MAX_USERNAMESPACE_LENGTH 255
#define DB2SEC_MAX_PASSWORD_LENGTH 255
#define DB2SEC_MAX_DBNAME_LENGTH 128
```

特定のプラグイン・インプリメンテーションでは、許可 ID、ユーザー ID、およびパスワードの最大長は、小さくする必要があるか、あるいは強制的に小さくされる可能性があります。特に、DB2 UDB に付属しているオペレーティング・システム認証プラグインは、オペレーティング・システムの限界が上記の限界より低い場合、オペレーティング・システムが施行する最大ユーザー長、最大グループ長、および最大ネームスペース長の限界の制約を受けます。

関連概念:

- 581 ページの『セキュリティー・プラグイン』
- 585 ページの『セキュリティー・プラグイン・ライブラリーの位置』

セキュリティー・プラグインの戻りコード

すべてのセキュリティー・プラグイン API は、 API の実行の成功や失敗を示すために整数の値を戻す必要があります。戻りコード値 0 は、API が正常に実行したことを示します。 -3、-4、および -5 以外のすべての負の戻りコードは、API がエラーを検出したことを示します。

-3、-4、または -5 が付く戻りコードを除き、セキュリティー・プラグイン APIS から戻されるすべての負の戻りコードは、 SQLCODE -1365、SQLCODE -1366、または SQLCODE -30082 にマップされます。 -3、-4、および -5 の値は、 AUTHID が有効なユーザーまたはグループを表しているかどうかを示すために用いられます。

すべてのセキュリティー・プラグイン API の戻りコードは、 DB2 の組み込みディレクトリー `SQLLIB/include` にある `db2secPlugin.h` で定義されます。

すべてのセキュリティー・プラグインの戻りコードに関する詳細については、以下の表で説明しています。

表 83. セキュリティー・プラグインの戻りコード

戻りコード	定義値	意味	関連 API
0	DB2SEC_PLUGIN_OK	プラグイン API が正常に実行されました。	すべて
-1	DB2SEC_PLUGIN _UNKNOWNERROR	プラグイン API で想定外のエラーが発生しました。	すべて
-2	DB2SEC_PLUGIN_BADUSER	入力として渡されたユーザー ID が定義されていません。	db2secGenerateInitialCred db2secValidatePassword db2secRemapUserid db2secGetGroupsForUser
-3	DB2SEC_PLUGIN _INVALIDUSERORGROUP	このユーザーまたはグループがありません。 /entry>	db2secDoesAuthIDExist db2secDoesGroupExist
-4	DB2SEC_PLUGIN _USERSTATUSNOTKNOWN	ユーザー状況が不明です。これは DB2 ではエラーとして扱われません。これは、GRANT ステートメントが、 authid がユーザーまたはオペレーティング・システム・グループのどちらを表しているか判別するために使用します。	db2secDoesAuthIDExist
-5	DB2SEC_PLUGIN _GROUPSTATUSNOTKNOWN	グループ状況が不明です。これは DB2 ではエラーとして扱われません。これは、GRANT ステートメントが、 authid がユーザーまたはオペレーティング・システム・グループのどちらを表しているか判別するために使用します。	db2secDoesGroupExist
-6	DB2SEC_PLUGIN_UID_EXPIRED	ユーザー ID が期限切れです。	db2secValidatePassword db2GetGroupsForUser db2secGenerateInitialCred
-7	DB2SEC_PLUGIN_PWD_EXPIRED	パスワードが期限切れです。	db2secValidatePassword db2GetGroupsForUser db2secGenerateInitialCred
-8	DB2SEC_PLUGIN_USER_REVOKED	ユーザーが失効しています。	db2secValidatePassword db2GetGroupsForUser

表 83. セキュリティー・プラグインの戻りコード (続き)

戻りコード	定義値	意味	関連 API
-9	DB2SEC_PLUGIN _USER_SUSPENDED	ユーザーが一時失効しています。	db2secValidatePassword db2GetGroupsForUser
-10	DB2SEC_PLUGIN_BADPWD	パスワードが無効です。	db2secValidatePassword db2secRemapUserid db2secGenerateInitialCred
-11	DB2SEC_PLUGIN _BAD_NEWPASSWORD	新規パスワードが無効です。	db2secValidatePassword db2secRemapUserid
-12	DB2SEC_PLUGIN _CHANGEPASSWORD _NOTSUPPORTED	パスワード変更はサポートされていません。	db2secValidatePassword db2secRemapUserid db2secGenerateInitialCred
-13	DB2SEC_PLUGIN_NOMEM	メモリー不足のため、プラグインがメモリーを割り振れませんでした。	すべて
-14	DB2SEC_PLUGIN_DISKERROR	プラグインがディスク・エラーを検出しました。	すべて
-15	DB2SEC_PLUGIN_NOPERM	ファイルの許可が不適切なため、プラグインがファイルにアクセスできませんでした。	すべて
-16	DB2SEC_PLUGIN_NETWORKERROR	プラグインがネットワーク・エラーを検出しました。	すべて
-17	DB2SEC_PLUGIN _CANTLOADLIBRARY	プラグインが必要なライブラリーをロードできません。	db2secGroupPluginInit db2secClientAuthPluginInit db2secServerAuthPluginInit
-18	DB2SEC_PLUGIN_CANT _OPEN_FILE	欠落ファイルや不適切なファイル許可以外の理由のために、プラグインがファイルをオープンして読み取ることができません。	すべて
-19	DB2SEC_PLUGIN_FILENOTFOUND	ファイル・システムにファイルがないために、プラグインがファイルをオープンして読み取ることができません。	すべて
-20	DB2SEC_PLUGIN _CONNECTION_DISALLOWED	接続できるデータベース、または特定のデータベースに接続できる TCP/IP アドレスについての制約事項のために、プラグインが接続を拒否しています。	すべてのサーバー・サイドのプラグイン API。
-21	DB2SEC_PLUGIN_NO_CRED	GSS API プラグインのみ: 初期クライアント証明書がありません。	db2secGetDefaultLoginContext db2secServerAuthPluginInit
-22	DB2SEC_PLUGIN_CRED_EXPIRED	GSS API プラグインのみ: クライアント証明書が期限切れです。	db2secGetDefaultLoginContext db2secServerAuthPluginInit
-23	DB2SEC_PLUGIN _BAD_PRINCIPAL_NAME	GSS API プラグインのみ: プリンシパル名が無効です。	db2secProcessServerPrincipalName
-24	DB2SEC_PLUGIN _NO_CON_DETAILS	この戻りコードは、`db2secGetConDetails` コールバック (たとえば、DB2 からプラグインへの) によって戻され、 DB2 がクライアントの TCP/IP アドレスを判別できないことを示します。	db2secGetConDetails
-25	DB2SEC_PLUGIN _BAD_INPUT_PARAMETERS	プラグイン API を呼び出すとき、いくつかのパラメーターが無効か、または欠落しています。	すべて

表 83. セキュリティー・プラグインの戻りコード (続き)

戻りコード	定義値	意味	関連 API
-26	DB2SEC_PLUGIN _INCOMPATIBLE_VER	プラグインによって報告された API のバージョンに、DB2 との互換性がありません。	db2secGroupPluginInit db2secClientAuthPluginInit db2secServerAuthPluginInit
-27	DB2SEC_PLUGIN_PROCESS_LIMIT	プラグインが、新規プロセスを作成しようとしているときにリソースを使い尽くしました。	すべて
-28	DB2SEC_PLUGIN_NO_LICENSES	プラグインがユーザー・ライセンスの問題を検出しました。基礎メカニズムのライセンスが限界に達している可能性があります。	すべて

関連概念:

- 581 ページの『セキュリティー・プラグイン』
- 590 ページの『セキュリティー・プラグインの問題判別』

セキュリティー・プラグインのエラー・メッセージ

セキュリティー・プラグイン API でエラーが発生すると、 API は errormsg フィールドに ASCII テキスト・ストリングを戻して、戻りコードよりも具体的な問題の説明を提示することがあります。たとえば、errormsg ストリングに、 "File /home/db2inst1/mypasswd.txt does not exist." などのメッセージが含められます。 DB2 はこのストリングをまるごと DB2 管理通知ログに書き込み、さらに、短縮版をいくつかの SQL メッセージにトークンとして組み込みます。 SQL メッセージ内のトークンは限られた長さにしかできないため、これらのメッセージは短くし、これらのメッセージの重要な変数の部分がストリングの先頭にくるようにすることをお勧めします。デバッグに役立てるため、エラー・メッセージにはセキュリティー・プラグインの名前を追加することを考慮してください。

パスワード期限切れエラーなどの緊急でないエラーに関しては、errormsg ストリングは、 DIAGLEVEL データベース・マネージャー構成パラメーターに 4 が設定されている場合にのみダンプされます。

これらのエラー・メッセージ用のメモリーは、セキュリティー・プラグインによって割り振られる必要があります。したがって、プラグインは、このメモリーを解放するための API である db2secFreeErrormsg を備えていなければなりません。

errormsg フィールドは、 API がゼロ以外の値を戻した場合にのみ DB2 によってチェックされます。そのため、プラグインは、エラーがない場合は、この戻りエラー・メッセージ用のメモリーを割り振るべきではありません。

初期化時には、メッセージ・ロギング関数ポインター logMessage_fn が、グループ、クライアント、およびサーバーのプラグインに渡されます。プラグインはこの関数を使用してデバッグ情報を db2diag.log に記録できます。たとえば、次のようにします。

```
// Log an message indicate init successful
(*(logMessage_fn))(DB2SEC_LOG_CRITICAL,
                   "db2secGroupPluginInit successful",
                    strlen("db2secGroupPluginInit successful"));
```

db2secLogMessage 関数の各パラメーターについて詳しくは、各プラグイン・タイプの初期化 API を参照してください。

関連概念:

- 581 ページの『セキュリティー・プラグイン』
- 590 ページの『セキュリティー・プラグインの問題判別』
- 611 ページの『セキュリティー・プラグイン API』
- 587 ページの『セキュリティー・プラグインの 2 パーツのユーザー ID のサポート』

関連資料:

- 603 ページの『セキュリティー・プラグインの戻りコード』

セキュリティー・プラグイン API の呼び出し順序

DB2 がセキュリティー・プラグイン API を呼び出す主なシナリオは、次の 5 つです。

- クライアントでのデータベース接続。
- クライアント、サーバー、またはゲートウェイでのローカル許可。
- サーバーでのデータベース接続。
- サーバーでの GRANT ステートメント。
- サーバーで authid が所属するグループのリストを取得する。

注: DB2 サーバーは、ローカル許可が必要な db2start、db2stop、および db2trc などのデータベース・アクションを、クライアント・アプリケーションと同様に取り扱います。

DB2 セキュリティー・プラグイン API を呼び出す順序は、これらの各操作ごとに適宜異なります。これらの各シナリオにおいて DB2 が呼び出す API の順序を以下に示します。

クライアントでのデータベース接続

ユーザー構成認証タイプが CLIENT の場合、 DB2 クライアント・アプリケーションは以下のセキュリティー・プラグイン API を呼び出します。

- `db2secGetDefaultLoginContext();`
- `db2secValidatePassword();`
- `db2secFreetoken();`

暗黙的な認証の場合、すなわち、特定のユーザー ID やパスワードを指定せずに接続する場合は、ユーザー ID/パスワード・プラグインを使用していると、 `db2secValidatePassword` API が呼び出されます。必要に応じ、プラグイン作成者はこの API によって暗黙的な認証を禁止することができます。

暗黙的な認証において、データベース・マネージャー構成パラメーター *authentication* に CLIENT 以外が設定されている (これはサーバーでの認証を暗示する) 場合は、アプリケーションがユーザー ID/パスワード認証メカニズム用に以下のセキュリティー・プラグイン API を呼び出します。

- db2secGetDefaultLoginContext();
- db2secFreeToken();

暗黙的な認証において、 *authentication* に CLIENT 以外が設定されている (これはサーバーでの認証を暗示する) 場合は、アプリケーションが GSS-API プラグイン用に以下のセキュリティー・プラグイン API を呼び出します。 (gss_init_sec_context() を呼び出すときは、GSS_C_NO_CREDENTIAL が入力証明書として使用されます。)

- db2secGetDefaultLoginContext();
- db2secProcessServerPrincipalName();
- gss_init_sec_context();
- gss_release_buffer();
- gss_release_name();
- gss_delete_sec_context();
- db2secFreeToken();

サーバーから相互認証トークンが戻される場合には、 API *gss_init_sec_context()* が 2 回呼び出されることがあります。

明示的な認証において、*authentication* に CLIENT が設定されている場合は、 DB2 クライアント・アプリケーションが以下のセキュリティー・プラグイン API を呼び出します。

- db2secRemapUserid();
- db2secValidatePassword();
- db2secFreeToken();

明示的な認証において、*authentication* に CLIENT が設定されている場合は、アプリケーションがユーザー ID/パスワード認証メカニズム用に以下のセキュリティー・プラグイン API を呼び出します。

- db2secRemapUserid();

折衝された認証タイプが GSS-API または Kerberos の場合は、クライアント・アプリケーションが GSS-API プラグイン用に以下のセキュリティー・プラグイン API をこの順序で呼び出します。これらの API は、暗黙的な認証または明示的な認証 (ユーザー ID とパスワードの両方が指定されるデータベースへの接続) に使用されます。

- db2secProcessServerPrincipalName();
- db2secGenerateInitialCred(); (明示的な認証の場合のみ)
- gss_init_sec_context();
- gss_release_buffer ();
- gss_release_name();
- gss_release_cred();
- db2secFreeInitInfo();
- gss_delete_sec_context();
- db2secFreeToken();

サーバーから相互認証トークンが戻される場合には、 API gss_init_sec_context() が 2 回呼び出されることがあります。

クライアント、サーバー、またはゲートウェイでのローカル許可

ローカル許可の場合は、使用される DB2 コマンドが、以下のセキュリティー・プラグイン API を呼び出します。

- `db2secGetDefaultLoginContext();`
- `db2secGetGroupsForUser();`
- `db2secFreeToken();`
- `db2secFreeGroupList();`

これらの API が、ユーザー ID/パスワードと GSS-API の両方の認証メカニズム用に呼び出されます。

サーバーでのデータベース接続

データベース・サーバー上でのデータベース接続の場合は、 DB2 エージェント・プロセスまたはスレッドが、ユーザー ID/パスワード認証メカニズム用に以下のセキュリティー・プラグイン API を呼び出します。

- `db2secValidatePassword();` (*authentication* が CLIENT でない場合のみ)
- `db2secGetAuthIDs();`
- `db2secGetGroupsForUser();`
- `db2secFreeToken();`
- `db2secFreeGroupList();`

データベース・サーバー上でのデータベース接続の場合は、 DB2 エージェント・プロセスまたはスレッドが、 GSS-API 認証メカニズム用に以下のセキュリティー・プラグイン API を呼び出します。

- `gss_accept_sec_context();`
- `gss_release_buffer();`
- `db2secGetAuthIDs();`
- `db2secGetGroupsForUser();`
- `gss_delete_sec_context();`
- `db2secFreeToken();`
- `db2secFreeGroupList();`

サーバーでの GRANT ステートメント

USER または GROUP キーワードを指定しない GRANT ステートメント (たとえば、 `"GRANT CONNECT ON DATABASE TO user1"`) の場合、 DB2 は `user1` がユーザー、グループ、またはその両方のいずれであるかを判別できなければなりません。そのため、DB2 は以下のセキュリティー・プラグイン API を呼び出します。

- `db2secDoesGroupExist();`
- `db2secDoesAuthIDExist();`

サーバーで authid が所属するグループのリストを取得する

データベース・サーバーで、authid が所属するグループのリストを取得する必要がある場合、 DB2 は以下のセキュリティー・プラグイン API を、authid のみを入力として呼び出します。

- `db2secGetGroupsForUser();`

他のセキュリティー・プラグインからのトークンはありません。

関連概念:

- 581 ページの『セキュリティー・プラグイン』
- 611 ページの『セキュリティー・プラグイン API』

第 28 章 セキュリティー・プラグイン API

セキュリティー・プラグイン API 611
グループ・プラグイン API 612
グループ検索プラグイン用の API 612
db2secGroupPluginInit - グループ・プラグインの初期化. 615
db2secPluginTerm - グループ・プラグイン・リソースのクリーンアップ 616
db2secGetGroupsForUser - ユーザーのグループのリストの取得 616
db2secDoesGroupExist - グループの存在のチェック 620
db2secFreeGroupListMemory - グループ・リストのメモリーの解放 621
db2secFreeErrormsg - エラー・メッセージのメモリーの解放 622
ユーザー認証プラグイン API 622
ユーザー ID/パスワード認証プラグインの API 622
db2secClientAuthPluginInit - クライアント認証プラグインの初期化 629
db2secClientAuthPluginTerm - クライアント認証プラグイン・リソースのクリーンアップ . . . 630
db2secRemapUserid - ユーザー ID およびパスワードの再マップ. 631
db2secGetDefaultLoginContext - デフォルト・ログイン・コンテキストの取得 633
db2secGenerateInitialCred - 初期証明書の生成 635
db2secValidatePassword - パスワードの検証 . . 637
db2secProcessServerPrincipalName - サーバーから戻されたサービス・プリンシパル名の処理. . . 639
db2secFreeToken - トークンが保持しているメモリーの解放 640
db2secFreeInitInfo - db2secGenerateInitialCred() が保持しているリソースのクリーンアップ . . . 641
db2secServerAuthPluginInit - サーバー認証プラグインの初期化 641
db2secServerAuthPluginTerm - サーバー認証プラグイン・リソースのクリーンアップ 643
db2secGetAuthIDs - 認証 ID の取得 644
db2secDoesAuthIDExist - 認証 ID の存在の検査 646
GSS-API プラグイン API 647
GSS-API 認証プラグインに必要な API および定義 647
GSS-API 認証プラグインに関する制約事項 . . 648
セキュリティー・プラグイン API のバージョン管理 649

セキュリティー・プラグイン API

ユーザーが DB2® の許可動作をカスタマイズできるようにするため、 DB2 は既存プラグインを変更したり、新規セキュリティー・プラグインを作成したりするために使用できる API を公開しています。

セキュリティー・プラグインを作成するときは、 DB2 が呼び出す標準認証関数をインプリメントする必要があります。使用可能な 3 つのプラグイン・タイプに関してインプリメントする必要がある機能は、以下のとおりです。

グループ検索
特定のユーザーのグループ・メンバーシップ情報を検索し、指定されたストリングが有効なグループ名を表しているかどうか判別します。

ユーザー ID/パスワード認証
この認証は、デフォルトのセキュリティー・コンテキストを識別し (クライアントのみ)、パスワードを検証して任意で変更し、指定されたストリングが有効なユーザーを表しているかどうか判別し (サーバーのみ)、クライアントで規定されているユーザー ID またはパスワードをサーバーに送信される前に変更し (クライアントのみ)、指定されたユーザーに関連付けられた DB2 許可 ID を戻します。

GSS-API 認証
この認証は、必要な GSS-API 関数をインプリメントし、デフォルトのセキュリティー・コンテキストを識別し (クライアント・サイドのみ)、ユーザー

ID およびパスワードを基に初期証明書を生成して任意でパスワードを変更し (クライアント・サイドのみ)、セキュリティー・チケットを作成して受け入れ、指定された GSS-API セキュリティー・コンテキストに関連付けられた DB2 許可 ID を戻します。

以下は、プラグイン API の説明に使用される用語の定義です。

プラグイン
DB2 がユーザー作成認証関数にアクセスするためにロードする、動的にロード可能なライブラリー。

暗黙的な認証
ユーザー ID またはパスワードが指定されないデータベースへの接続。

明示的な認証
ユーザー ID とパスワードの両方が指定されるデータベースへの接続。

authid データベース内での権限および特権が付与された個人またはグループを表す内部 ID。内部では、DB2 authid は大文字で、8 文字以上です (8 文字まではブランクが埋め込まれます)。現在のところ、DB2 は、7 ビット ASCII で表記できる authid、ユーザー ID、パスワード、グループ名、ネームスペース、およびドメイン名を必要とします。 authid の最大長は 30 文字です。

ローカル許可
許可をインプリメントしているサーバーまたはクライアントでのローカルな許可で、データベース接続以外の許可の必要なアクション (データベース・マネージャーの開始と停止、 DB2 トレースのオン/オフ、データベース・マネージャー構成の更新など) を実行する権限がユーザーにあるかどうかを検査します。

ネームスペース
ユーザーの集合またはグループ。この中で個々のユーザー ID はユニークでなければなりません。一般的な例としては、Windows® ドメインと Kerberos レルムがあります。たとえば、Windows ドメイン ″usa.company.com″ では、すべてのユーザー名がユニークでなければなりません。たとえば、″user1@usa.company.com″ などとなります。他のドメインにある同一のユーザー ID (たとえば、″user1@canada.company.com″) は、別のユーザーを表します。完全修飾ユーザー ID には、ユーザー ID とネームスペースのペア (たとえば ″user@domain.name″ または ″domain¥user″) が含まれます。

関連概念:

• 581 ページの『セキュリティー・プラグイン』

グループ・プラグイン API

グループ検索プラグイン用の API

グループ検索プラグイン・ライブラリー用には、以下の API をインプリメントする必要があります。

```
SQL_API_RC SQL_API_FN db2secGroupPluginInit(
 db2int32 version,
 void *group_fns,
 db2secLogMessage *logMessage_fn,
 char **errormsg,
 db2int32  *errormsglen);
```

注: 上記の関数は、関数 *logMessage_fn を指すポインターを入力として取ります。以下はそのプロトタイプです。

```
SQL_API_RC (SQL_API_FN db2secLogMessage) (
  db2int32 level,
 void *data,
 db2int32 length);
```

```
SQL_API_RC SQL_API_FN db2secPluginTerm(char **errormsg,
      db2int32  *errormsglen);
```

```
SQL_API_RC SQL_API_FN db2secGetGroupsForUser(
 const char *authid,
 db2int32  authidlen,
 const char *userid,
 db2int32  useridlen,
 const char *usernamespace,
 db2int32  usernamespacelen,
 db2int32  usernamespacetype,
 const char *dbname,
 db2int32  dbnamelen,
 const void *token,
 db2int32 tokentype,
 db2int32 location,
 const char *authpluginname,
 db2int32  authpluginname,
 char **grouplist,
 db2int32  *numgroups,
 char **errormsg,
 db2int32  *errormsglen);
```

```
SQL_API_RC SQL_API_FN db2secDoesGroupExist(
  const char *groupname,
  db2int32  groupnamelen
  char **errormsg,
  db2int32 *errormsglen);
```

```
SQL_API_RC SQL_API_FN db2secFreeGroupListMemory(
  char *ptr,
  char **errormsg,
  db2int32 *errormsglen);
```

```
SQL_API_RC SQL_API_FN db2secFreeErrormsg(char *msgtobefree);
```

外部で解決できなければならない唯一の API は、db2secGroupPluginInit() です。この関数は、void * パラメーターをとり、それは以下のタイプにキャストする必要があります。

```
typedef struct db2secGroupFunctions_1
{
  db2int32 version;
  db2int32 plugintype;
  SQL_API_RC (SQL_API_FN * db2secGetGroupsForUser) (
    const char *authid,
    db2int32  authidlen,
    const char *userid,
    db2int32  useridlen,
    const char *usernamespace,
    db2int32  usernamespacelen,
    db2int32  usernamespacetype,
    const char *dbname,
```

```
    db2int32  dbnamelen,
    const void *token,
    db2int32 tokentype,
    db2int32 location,
    const char *authpluginname,
    db2int32  authpluginnamelen,
    void **grouplist,
    db2int32  *numgroups,
    char **errormsg,
    db2int32  *errormsglen);

 SQL_API_RC (SQL_API_FN * db2secDoesGroupExist)(
    const char *groupname,
    db2int32  groupnamelen,
    char **errormsg,
    db2int32  *errormsglen);

  SQL_API_RC (SQL_API_FN * db2secFreeGroupListMemory)(
    void *ptr,
    char **errormsg,
    db2int32  *errormsglen);

  SQL_API_RC (SQL_API_FN * db2secFreeErrormsg)(
    char *msgtobefree);

 SQL_API_RC (SQL_API_FN * db2secPluginTerm)(
    char **errormsg,
    db2int32  *errormsglen);

} db2secGroupFunctions_1;
```

db2secGroupPluginInit() は、外部で使用可能な残りの関数のアドレスを割り当てます。

注: _1 はこれが API のバージョン 1 に対応する構造であることを示します。後続のインターフェース・バージョンの拡張子は _2、_3 というようになります。

関連概念:

- 581 ページの『セキュリティー・プラグイン』
- 611 ページの『セキュリティー・プラグイン API』

関連タスク:

- 592 ページの『グループ検索プラグインの展開』

関連資料:

- 615 ページの『db2secGroupPluginInit - グループ・プラグインの初期化』
- 616 ページの『db2secPluginTerm - グループ・プラグイン・リソースのクリーンアップ』
- 616 ページの『db2secGetGroupsForUser - ユーザーのグループのリストの取得』
- 620 ページの『db2secDoesGroupExist - グループの存在のチェック』
- 621 ページの『db2secFreeGroupListMemory - グループ・リストのメモリーの解放』
- 622 ページの『db2secFreeErrormsg - エラー・メッセージのメモリーの解放』

db2secGroupPluginInit - グループ・プラグインの初期化

これは、DB2 がプラグイン・ライブラリーをロードした直後に呼び出すライブラリーの初期化関数です。 functions ポインターは、インターフェース・バージョン用の適切な group_functions 構造にキャストされる必要があります。

C API 構文:

```
SQL_API_RC SQL_API_FN db2secGroupPluginInit(
  db2int32 version,
  void *group_fns,
  db2secLogMessage *logMessage_fn,
  char **errormsg,
  db2int32 *errormsglen);
```

入力:

db2int32 version

DB2 が現在サポートする API の最高バージョン番号。

*db2secLogMessage *logMessage_fn*

DB2 が提供する関数を指すポインター。プラグインはこの関数を呼び出して、デバッグや通知のための追加のエラー・ストリングを db2diag.log に記録できます。最初のパラメーターは db2secPlugin.h 内の定義を使用する必要があり、最後の 2 つのパラメーターはメッセージ・ストリングとその長さです。最初のパラメーターで使用される定義は、以下のとおりです。

```
#define DB2SEC_LOG_NONE       0   - ロギングなし
#define DB2SEC_LOG_CRITICAL   1   - 重大エラーを検出した
#define DB2SEC_LOG_ERROR      2   - エラーを検出した
#define DB2SEC_LOG_WARNING    3   - 警告
#define DB2SEC_LOG_INFO       4   - 通知
```

DB2SEC_LOG_INFO 定義を使用する場合、メッセージ・テキストは、*diaglevel* データベース・マネージャー構成パラメーターに 4 が設定されている場合にのみ db2diag.log に記録されます。

出力:

*void *group_fns*

DB2 が提供する db2secGroupFunction_1 構造用のメモリーを指すポインター。 DB2 の将来のバージョンでは、異なるバージョンの API が存在している可能性があるので、これは、プラグインがインプリメントしている API のバージョンに対応する db2secGroupFunction_1 構造を指すポインターにキャストする必要があります。

*char **errormsg*

API が正常に実行されない場合にこのパラメーターに戻されることがある、プラグインによって割り振られた ASCII ストリングのアドレスを指すポインター。

*db2int32 *errormsglen*

*char **errormsg* のエラー・メッセージ・ストリングの長さを示す整数を指すポインター。

関連概念:

- 581 ページの『セキュリティー・プラグイン』

- 611 ページの『セキュリティー・プラグイン API』

関連資料:

- 612 ページの『グループ検索プラグイン用の API』

db2secPluginTerm - グループ・プラグイン・リソースのクリーンアップ

この関数は、DB2 がプラグインをアンロードする直前に呼び出します。これは、プラグイン・ライブラリーが保持しているリソースの適切なクリーンアップを実行します。たとえば、プラグインによって割り振られたメモリーを解放し、まだオープンしているファイルをクローズし、ネットワーク接続をクローズします。これらのリソースを解放するためにその記録を保持することは、プラグインが行います。

C API 構文:

```
SQL_API_RC SQL_API_FN db2secPluginTerm(char **errormsg,
          db2int32  *errormsglen);
```

出力:

*char **errormsg*
API が正常に実行されない場合にこのパラメーターに戻されることがある、プラグインによって割り振られた ASCII ストリングのアドレスを指すポインター。

*db2int32 *errormsglen*
*char **errormsg* のエラー・メッセージ・ストリングの長さを示す整数を指すポインター。

関連概念:

- 581 ページの『セキュリティー・プラグイン』
- 611 ページの『セキュリティー・プラグイン API』

関連資料:

- 「*管理ガイド: パフォーマンス*」の『diaglevel - 「診断エラー・キャプチャー・レベル」構成パラメーター』
- 612 ページの『グループ検索プラグイン用の API』

db2secGetGroupsForUser - ユーザーのグループのリストの取得

この関数は、ユーザーが所属するグループのリストを取得するために DB2 が呼び出します。

C API 構文:

```
SQL_API_RC SQL_API_FN db2secGetGroupsForUser(
  const char *authid,
  db2int32 authidlen,
  const char *userid,
  db2int32 useridlen,
  const char *usernamespace,
  db2int32 usernamespacelen,
  db2int32 usernamespacetype,
  const char *dbname,
```

```
  db2inst32 dbnamelen,
  const void *token,
  db2int32 tokentype,
  db2int32 location,
  const char *authpluginname,
  db2int32  authpluginnamelen,
  void **grouplist,
  db2int32  *numgroups,
    char **errormsg,
  db2int32 *errormsglen);
```

入力:

*const char *authid*
: DB2 が提供する唯一の入力フィールドです。このフィールドの値は SQL authid なので、これは、末尾ブランクのないすべて大文字のストリングの形式になります。プラグインは、他の入力パラメーターに関係なく、 authid が所属するグループのリストを戻せなければなりません。これが判別できない場合は、短縮されたリストまたは空のリストを戻しても差し支えありません。

  ユーザーが存在しない場合、この関数は DB2SEC_PLUGIN_BADUSER を戻す必要があります。 authid には関連するグループがなくても差し支えないため、 DB2 は存在しないユーザーのケースをエラーとして扱いません。これには、たとえば、 db2secGetAuthids 関数がオペレーティング・システムに存在しない authid を戻した場合があります。この authid にはグループが関連付けられていませんが、それでもこれには直接特権を割り当てることができます。

  プラグインがその authid だけからでは完全なグループのリストを戻せない場合、グループ・サポートに関連した特定の SQL 関数になんらかの制限が生じる可能性があります。考えられる問題のシナリオについては、このトピックにある『不完全なグループのリストが戻された場合に生じる可能性のある問題』と題する注記を参照してください。

db2int32 authidlen
: authid の長さ。

*const char *userid*
: これは authid に対応するユーザー ID です。非接続のシナリオで、サーバー上でこの API が呼び出されたときは、これには値が入れられません。

db2int32 useridlen
: ユーザー ID の長さ。

*const void *token*
: 認証プラグインによって提供されるデータを指すポインター。これは DB2 には参照されません。必要に応じて、プラグイン作成者はこれを使用してユーザーおよびグループの情報を調整することができます。この値はすべての場合に与えられるわけではなく、その場合は NULL になります。使用されている認証プラグインが GSS-API ベースの場合、このトークンには GSS-API コンテキスト・ハンドル (gss_ctx_id_t) が設定されます。

db2int32 tokentype
: 認証プラグインによって提供されるデータのタイプを示します。使用されている認証プラグインが GSS-API ベースの場合、このトークンには GSS-API

コンテキスト・ハンドル (gss_ctx_id_t) が設定されます。使用されている認証プラグインがユーザー ID/パスワード・ベースの場合、これは汎用タイプになります。 db2secPlugin.h 内の以下の定義を参照してください。

- #define DB2SEC_GENERIC 0 -- これは、トークンがユーザー ID/パスワード・ベースのプラグインからのものであることを示します。
- #define DB2SEC_GSSAPI_CTX_HANDLE 1 -- これは、トークンが GSS-API (Kerberos を含む) ベースのプラグインからのものであることを示します。

db2int32 location
DB2 がクライアント・サイドとサーバー・サイドのどちらでプラグインを呼び出すかを示します。 db2secPlugin.h 内の以下の定義を参照してください。

- #define DB2SEC_SERVER_SIDE 0 -- グループ・プラグインはデータベース・サーバー上で呼び出されます。
- #define DB2SEC_CLIENT_SIDE 1 -- グループ・プラグインはクライアント上で呼び出されます。

*const char *usernamespace*
取得されたユーザー ID が属するネームスペース。ユーザー ID が使用不可の場合、これに値は入れられません。

db2int32 usernamespacelen
ネームスペース・フィールドの長さ。

db2int32 usernamespacetype
ネームスペースのタイプ。可能な値は、DB2SEC_NAMESPACE_SAM_COMPATIBLE (torolab¥myname のようなユーザー名のスタイルに対応)、または DB2SEC_NAMESPACE_USER_PRINCIPAL (myname@torolab.ibm.com のようなユーザー名のスタイルに対応) です。現在のところ、DB2 は DB2SEC_NAMESPACE_SAM_COMPATIBLE しかサポートしていません。ユーザー ID がない場合、これには DB2SEC_USER_NAMESPACE_UNDEFINED が入れられます。すべての定義は db2secPlugin.h にあります。

*const char *dbname*
これは接続先のデータベースの名前です。

db2int32 dbnamelen
dbname で指定されるデータベース名の長さです。

*const char *authpluginname*
これは、トークンのデータを提供した認証プラグインの名前です。プラグインは、正しいグループ・メンバーシップを判別するためにこの情報を使用することがあります。 authid が認証されない場合 (たとえば、authid が現行接続ユーザーと一致しない場合) には、これは与えられないことがあります。

db2int32 authpluginnamelen
authpluginname の長さ。

出力:

*void **grouplist*
グループのリストは、連結された varchars (varchar とは、最初のバイトが後続のバイトの数を示す文字配列です) が含まれている、プラグインによって割り振られたメモリーのセクションを指すポインターとして戻されなければなりません。長さは unsigned char であり、このためグループ名の最大長は 255 文字までに制限されます。すなわち、unsigned char (1 バイト) を使用してグループ名の長さを示しているので、最大長は 255 です。以下に例を示します。

```
"¥006GROUP1¥007MYGROUP¥008MYGROUP3"
```

各グループ名は、有効な DB2 authid でなければなりません。この配列のメモリーは、プラグインによって割り振られる必要があります。したがって、プラグインは、 DB2 がメモリーを解放するために呼び出す db2secFreeGroupListMemory() プラグイン関数などの関数を備えている必要があります。

*db2int32 *numgroups*
grouplist に含まれるグループの数。

*char **errormsg*
API が正常に実行されない場合にこのパラメーターに戻されることがある、プラグインによって割り振られた ASCII ストリングのアドレスを指すポインター。

*db2int32 *errormsglen*
*char **errormsg* のエラー・メッセージ・ストリングの長さを示す整数を指すポインター。

API から不完全なグループのリストが戻された場合に生じる可能性のある問題:

この API から DB2 UDB に不完全なグループのリストが戻された場合は、以下の問題が生じる可能性があります。

- DYNAMICRULES BIND (あるいは、パッケージがスタンドアロン・アプリケーションとして実行している場合は、 DEFINEDBIND または INVOKEDBIND) が指定された組み込み SQL アプリケーション。 DB2 は SYSADM メンバーシップをチェックします。そして、アプリケーションが、 SYSADM のメンバーであることによって付与される暗黙的な DBADM 権限に依存している場合、このアプリケーションは失敗します。
- CREATE SCHEMA ステートメントで代替許可が提供される。 CREATE SCHEMA ステートメント内にネストされた CREATE ステートメントがある場合、 AUTHORIZATION NAME パラメーターに対してグループ検索が実行されます。
- DYNAMICRULES DEFINERUN/DEFINEBIND が指定された組み込み SQL アプリケーションがあり、そのパッケージがルーチン・コンテキストで実行している。 DB2 はルーチン定義者の SYSADM メンバーシップをチェックします。そして、アプリケーションが、 SYSADM のメンバーであることによって付与される暗黙的な DBADM 権限に依存している場合、このアプリケーションは失敗します。
- MPP 環境での jar ファイルの処理。 MPP 環境では、jar 処理要求が、セッション authid とともにコーディネーター・ノードから送信されます。カタログ・ノー

ドは要求を受信すると、セッション authid (jar 処理要求を実行するユーザー) の特権に基づいて jar ファイルを処理します。

- jar ファイルのインストール。セッション authid は、SYSADM、DBADM、または CREATEIN のいずれかの (jar スキーマに対する暗黙的または明示的な) 権限を有している必要があります。セッション authid の含まれるグループに対しては上記の権限が付与されているが、セッション authid に明示的には付与されていない場合や、データベース構成パラメーターによって定義されたグループのメンバーシップによって SYSADM メンバーシップが判別されたために、 SYSADM のみが保持されている場合は、操作は失敗します。
- jar ファイルの除去。セッション authid は、SYSADM、DBADM、または DROPIN のいずれかの (jar スキーマに対する暗黙的または明示的な) 権限を有しているか、 jar ファイルの定義者である必要があります。セッション authid の含まれるグループに対しては上記の権限が付与されているが、セッション authid に明示的には付与されておらず、セッション authid が Jar ファイルの定義者でもない場合や、データベース構成パラメーターによって定義されたグループのメンバーシップによって SYSADM メンバーシップが判別されたために、 SYSADM のみが保持されている場合は、操作は失敗します。
- jar ファイルの置き換え。これは、jar ファイルを除去した後に、jar ファイルをインストールするのと同じことです。上記の両方が当てはまります。

- ビューの再生成。これは ALTER TABLE、ALTER COLUMN、SET DATA TYPE VARCHAR/VARGRAPHIC ステートメントによって、またはマイグレーションの際に起動されます。 DB2 はビュー定義者の SYSADM メンバーシップをチェックします。アプリケーションが、 SYSADM グループのメンバーであることによって付与される暗黙的な DBADM 権限に依存している場合、このアプリケーションは失敗します。
- SET SESSION_USER ステートメントが発行される場合。その後の DB2 操作は、このステートメントで指定された authid のコンテキストの下で実行されます。必要な特権が SESSION_USER のグループのいずれかによって所有されているものの、 SESSION_USER authid に明示的に付与されてはいない場合、それらの操作は失敗します。

関連概念:

- 581 ページの『セキュリティー・プラグイン』
- 611 ページの『セキュリティー・プラグイン API』

関連資料:

- 612 ページの『グループ検索プラグイン用の API』

db2secDoesGroupExist - グループの存在のチェック

この関数は、authid がグループを表しているかどうか判別するために使用されます。 groupname が存在する場合、この関数は DB2SEC_PLUGIN_OK を戻して正常であることを示します。グループ名が有効でない場合は、DB2SEC_PLUGIN_INVALIDUSERORGROUP が戻されなければなりません。この API は、入力が有効なグループかどうか判別できない場合は、DB2SEC_PLUGIN_GROUPSTATUSNOTKNOWN を戻すこともできます。無効なグループや不明なグループが戻される場合、 DB2 は USER キーワードおよび GROUP キーワードのない GRANT ステートメントを発行するときに、 authid が

グループかユーザーかを判別できない可能性があり、その結果 SQLCODE -569、SQLSTATE 56092 のエラーがユーザーに戻されます。

C API 構文:

```
SQL_API_RC SQL_API_FN db2secDoesGroupExist(
 const char *groupname,
 db2int32  groupnamelen,
 char **errormsg,
 db2int32  *errormsglen
);
```

入力:

*const char *groupname*
末尾ブランクなしの大文字の authid。

db2int32 groupnamelen
groupname の長さ。

出力:

*char **errormsg*
API が正常に実行されない場合にこのパラメーターに戻されることがある、プラグインによって割り振られた ASCII ストリングのアドレスを指すポインター。

*db2int32 *errormsglen*
*char **errormsg* のエラー・メッセージ・ストリングの長さを示す整数を指すポインター。

関連概念:

- 581 ページの『セキュリティー・プラグイン』
- 611 ページの『セキュリティー・プラグイン API』

関連資料:

- 612 ページの『グループ検索プラグイン用の API』

db2secFreeGroupListMemory - グループ・リストのメモリーの解放

この関数は、ptr が指すメモリーを DB2 が必要としなくなったことをプラグイン・ライブラリーに知らせます。プラグインはこのメモリーを解放する必要があります。

C API 構文:

```
SQL_API_RC SQL_API_FN db2secFreeGroupListMemory(
 void *ptr
 char **errormsg,
 db2int32  *errormsglen);
```

入力:

*void *ptr*
解放されるメモリーを指すポインター。

出力:

*char **errormsg*
API が正常に実行されない場合にこのパラメーターに戻されることがある、プラグインによって割り振られた ASCII ストリングのアドレスを指すポインター。

*db2int32 *errormsglen*
*char **errormsg* のエラー・メッセージ・ストリングの長さを示す整数を指すポインター。

関連概念:

- 581 ページの『セキュリティー・プラグイン』
- 611 ページの『セキュリティー・プラグイン API』

関連資料:

- 612 ページの『グループ検索プラグイン用の API』

db2secFreeErrormsg - エラー・メッセージのメモリーの解放

この関数は、以前のプラグイン API 呼び出しのエラー・メッセージを保持するために使用されているメモリーを解放するために DB2 が呼び出します。これは、エラー・メッセージを一緒に戻さない唯一の API です。この関数がエラーを戻す場合、DB2 はそれをログに記録して続行します。

C API 構文:

```
SQL_API_RC SQL_API_FN db2secFreeErrormsg(char *msgtobefree);
```

入力:

*char *msgtobefree*
以前の API 呼び出しで割り振られたエラー・メッセージを指すポインター。

関連概念:

- 581 ページの『セキュリティー・プラグイン』
- 611 ページの『セキュリティー・プラグイン API』

関連資料:

- 612 ページの『グループ検索プラグイン用の API』

ユーザー認証プラグイン API

ユーザー ID/パスワード認証プラグインの API

ユーザー ID/パスワード・プラグイン・ライブラリー用には、以下のクライアント・サイド API をインプリメントする必要があります。

```
SQL_API_RC SQL_API_FN db2secClientAuthPluginInit(
  db2int32 version,
  void *client_fns,
  db2secLogMessage *logMessage_fn,
  char **errormsg,
  db2int32 *errormsglen);
```

注: 上記の関数は、関数 *logMessage_fn を指すポインターを入力として取ります。以下はそのプロトタイプです。

```
    SQL_API_RC (SQL_API_FN db2secLogMessage) (
      db2int32 level,
      void *data,
      db2int32 length);

SQL_API_RC SQL_API_FN db2secClientAuthPluginTerm(
  char **errormsg,
  db2int32 *errormsglen);

/* Only used for gssapi: */
db2secGenerateInitialCred(
  const char *userid,
  db2int32 useridlen,
  const char *usernamespace,
  db2int32 usernamespacelen,
  db2int32  usernamespacetype,
  const char *password,
  db2int32 passwordlen,
  const char *newpassword,
  db2int32 newpasswordlen,
  const char *dbname,
  db2int32  dbnamelen,
  gss_cred_id_t *pGSSCredHandle,
  void **initInfo,
  char **errormsg,
  db2int32 *errormsglen);

/* Optional */
SQL_API_RC SQL_API_FN db2secRemapUserid(
  char userid[DB2SEC_MAX_USERID_LENGTH],
  db2int32  *useridlen,
  db2int32 useridtype,
  char usernamespace[DB2SEC_MAX_USERNAMESPACE_LENGTH],
  db2int32  *usernamespacelen,
  db2int32  *usernamespacetype,
  char password[DB2SEC_MAX_PASSWORD_LENGTH],
  db2int32  *passwordlen,
  char newpassword[DB2SEC_MAX_PASSWORD_LENGTH],
  db2int32  *newpasswordlen,
  const char *dbname,
  db2int32  dbnamelen,
  char **errormsg,
  db2int32 *errormsglen);

SQL_API_RC SQL_API_FN db2secGetDefaultLoginContext(
  char authid[DB2SEC_MAX_AUTHID_LENGTH],
  db2int32  *authidlen,
  char userid[DB2SEC_MAX_USERID_LENGTH],
  db2int32  *useridlen,
  db2int32 useridtype,
  char usernamespace[DB2SEC_MAX_USERNAMESPACE_LENGTH],
  db2int32  *usernamespacelen,
  db2int32  *usernamespacetype,
  const char *dbname,
```

```
  db2int32  dbnamelen,
  void **token,
  char **errormsg,
  db2int32 *errormsglen);

SQL_API_RC SQL_API_FN db2secValidatePassword(
  const char *userid,
  db2int32 useridlen,
  const char *usernamespace,
  db2int32 usernamespacelen,
  db2int32  usernamespacetype,
  const char *password,
  db2int32 passwordlen,
  const char *newpassword,
  db2int32 newpasswordlen,
  const char *dbname,
  db2int32  dbnamelen,
  db2Uint32  connection_details,
  void **token,
  char **errormsg,
  db2int32 *errormsglen);

/* This is only for GSS-API */
SQL_API_RC SQL_API_FN db2secProcessServerPrincipalName(
  const void *name,
  db2int32 nameLen,
  gss_name_t *gssName,
  char **errormsg,
  db2int32 *errormsglen);

/* Functions to free memory held by the DLL */
SQL_API_RC SQL_API_FN db2secFreeToken(
  void *token,
  char **errormsg,
  db2int32 *errormsglen);

SQL_API_RC SQL_API_FN db2secFreeErrormsg(char *errormsg);

SQL_API_RC SQL_API_FN db2secFreeInitInfo(
  void *initInfo,
  char **errormsg,
  db2int32 *errormsglen);
```

外部で解決できなければならない唯一の API は、db2secClientAuthPluginInit() です。この関数は、void * パラメーターをとり、それは以下のいずれかにキャストする必要があります。

```
typedef struct db2secUseridPasswordClientAuthFunctions_1
{
   db2int32 version;
   db2int32 plugintype;

SQL_API_RC (SQL_API_FN * db2secGetDefaultLoginContext)(
  char authid[DB2SEC_MAX_AUTHID_LENGTH],
  db2int32  *authidlen,
  char userid[DB2SEC_MAX_USERID_LENGTH],
  db2int32  *useridlen,
  db2int32 useridtype,
  char usernamespace[DB2SEC_MAX_USERNAMESPACE_LENGTH],
  db2int32  *usernamespacelen,
  db2int32  *usernamespacetype,
  const char *dbname,
  db2int32  dbnamelen,
  void **token,
  char **errormsg,
  db2int32 *errormsglen);

SQL_API_RC (SQL_API_FN * db2secRemapUserid)(  // Optional
```

```
  char userid[DB2SEC_MAX_USERID_LENGTH],
  db2int32  *useridlen,
  char usernamespace[DB2SEC_MAX_USERNAMESPACE_LENGTH],
  db2int32  *usernamespacelen,
  db2int32  *usernamespacetype,
  char password[DB2SEC_MAX_PASSWORD_LENGTH],
  db2int32  *passwordlen,
  char newpassword[DB2SEC_MAX_PASSWORD_LENGTH],
  db2int32  *newpasswordlen,
  const char *dbname,
  db2int32  dbnamelen,
  char **errormsg,
  db2int32 *errormsglen);

SQL_API_RC (SQL_API_FN * db2secValidatePassword)(
  const char *userid,
  db2int32 useridlen,
  const char *usernamespace,
  db2int32 usernamespacelen,
  db2int32  usernamespacetype,
  const char *password,
  db2int32 passwordlen,
  const char *newpassword,
  db2int32 newpasswordlen,
  const char *dbname,
  db2int32  dbnamelen,
  db2Uint32  connection_details,
  void **token,
  char **errormsg,
  db2int32 *errormsglen);

SQL_API_RC (SQL_API_FN * db2secFreeToken)(
  void **token,
  char **errormsg,
  db2int32 *errormsglen);

SQL_API_RC (SQL_API_FN * db2secFreeErrormsg)(char *errormsg);


SQL_API_RC (SQL_API_FN * db2secClientAuthPluginTerm)(
  char **errormsg,
  db2int32 *errormsglen);
}
```

または

```
typedef struct db2secGssapiClientAuthFunctions_1
{
   db2int32 version;
   db2int32 plugintype;

SQL_API_RC (SQL_API_FN * db2secGetDefaultLoginContext) (
  char authid[DB2SEC_MAX_AUTHID_LENGTH],
  db2int32  *authidlen,
  char userid[DB2SEC_MAX_USERID_LENGTH],
  db2int32  *useridlen,
  db2int32 useridtype,
  char usernamespace[DB2SEC_MAX_USERNAMESPACE_LENGTH],
  db2int32  *usernamespacelen,
  db2int32  *usernamespacetype,
  const char *dbname,
  db2int32  dbnamelen,
  void **token,
  char **errormsg,
  db2int32 *errormsglen);

SQL_API_RC (SQL_API_FN * db2secProcessServerPrincipalName) (
```

```
  const void *data,
  gss_name_t *gssName,
  char **errormsg,
  db2int32 *errormsglen);

SQL_API_RC (SQL_API_FN * db2secGenerateInitialCred) (
  const char *userid,
  db2int32 useridlen,
  const char *usernamespace,
  db2int32 usernamespacelen,
  db2int32 usernamespacetype,
  const char *password,
  db2int32 passwordlen,
  const char *newpassword,
  db2int32 newpasswordlen,
  const char *dbname,
  db2int32  dbnamelen,
  gss_cred_id_t *pGSSCredHandle,
  void **initInfo,
  char **errormsg,
  db2int32 *errormsglen);

SQL_API_RC (SQL_API_FN * db2secFreeToken)(
  void *token,
  char **errormsg,
  db2int32 *errormsglen);

SQL_API_RC (SQL_API_FN * db2secFreeErrormsg)(char *errormsg);


SQL_API_RC (SQL_API_FN * db2secFreeInitInfo) (
  void *initInfo,
  char **errormsg,
  db2int32 *errormsglen);

SQL_API_RC (SQL_API_FN * db2secClientAuthPluginTerm) (
  char **errormsg,
  db2int32 *errormsglen);

/* GSS-API specific functions -- refer to db2secPlugin.h
   for parameter list*/

  OM_uint32 (SQL_API_FN * gss_init_sec_context )(<parameter list>);
  OM_uint32 (SQL_API_FN * gss_delete_sec_context )(<parameter list>);
  OM_uint32 (SQL_API_FN * gss_display_status )(<parameter list>);
  OM_uint32 (SQL_API_FN * gss_release_buffer )(<parameter list>);
  OM_uint32 (SQL_API_FN * gss_release_cred )(<parameter list>);
  OM_uint32 (SQL_API_FN * gss_release_name )(<parameter list>);
  }
```

ユーザー ID/パスワード・プラグインを作成する場合は、
db2secUseridPasswordClientAuthFunctions_1 を使用する必要があります。
GSS-API (Kerberos を含む) プラグインを作成する場合は、
db2secGssapiClientAuthFunctions_1 を使用する必要があります。

ユーザー ID/パスワード・プラグイン・ライブラリー用には、以下のサーバー・サイド API をインプリメントする必要があります。

```
db2secServerAuthPluginInit(
  db2int32 version,
  void *server_fns,
  db2secGetConDetails *getConDetails_fn,
  db2secLogMessage *logMessage_fn,
  char **errormsg,
  db2int32 *errormsglen);
```

上記の関数は、関数 *logMessage_fn および関数 *getConDetails_fn を指すポインターを入力としてとります。以下はそのプロトタイプです。

```
SQL_API_RC (SQL_API_FN db2secLogMessage) (
  db2int32 level,
  void *data,
  db2int32 length);

SQL_API_RC (SQL_API_FN db2secGetConDetails)(
  db2int32 conDetailsVersion,
  const void *pConDetails);
```

この関数は、その 2 番目のパラメーター pConDetails として、次のように定義された構造を指すポインターをとります。

```
typedef struct db2sec_con_details_1
{
  db2int32 clientProtocol;
  db2Uint32 clientIPAddress;
  db2Uint32 connect_info_bitmap;
  db2int32 dbnameLen;
  char dbname[DB2SEC_MAX_DBNAME_LENGTH + 1];
} db2sec_con_details_1;
```

この関数と構造の説明については、詳細記述のセクションを参照してください。

```
db2secServerAuthPluginTerm(
  char **errormsg,
  db2int32 *errormsglen);
```

```
SQL_API_RC SQL_API_FN db2secValidatePassword(
  const char *userid,
  db2int32 useridlen,
  const char *usernamespace,
  db2int32 usernamespacelen,
  db2int32  usernamespacetype,
  const char *password,
  db2int32 passwordlen,
  const char *newpasswd,
  db2int32  newpasswdlen,
  const char *dbname,
  db2int32  dbnamelen,
  db2Uint32  connection_details,
  void **token,
  char **errormsg,
  db2int32 *errormsglen);
```

```
SQL_API_RC SQL_API_FN db2secGetAuthIDs(
  const char *userid,
  db2int32 useridlen,
  const char *usernamespace,
  db2int32 usernamespacelen,
  db2int32  usernamespacetype,
  const char *dbname,
  db2int32  dbnamelen,
  void **token,
  char SystemAuthid[DB2SEC_MAX_AUTHID_LENGTH],
  db2int32  SystemAuthidlen,
  char InitialSessionAuthID[DB2SEC_MAX_AUTHID_LENGTH],
  db2int32  *InitialSessionAuthIdlen,
  char username[DB2SEC_MAX_USERID_LENGTH],
  db2int32  *usernamelen,
  db2int32 *initsessionidtype,
  char **errormsg,
  db2int32 *errormsglen);
```

```
SQL_API_RC SQL_API_FN db2secDoesAuthIDExist(
  const char *authid,
  db2int32 authidlen,
  const char *errormsg,
  db2int32 *errormsglen);

SQL_API_RC SQL_API_FN db2secFreeToken(
  void *token,
  char **errormsg,
  db2int32 *errormsglen);

SQL_API_RC SQL_API_FN db2secFreeErrormsg(char *errormsg);
```

外部で解決できなければならない唯一の API は、db2secServerAuthPluginInit() です。この関数は、void * パラメーターをとり、それは以下のいずれかにキャストする必要があります。

```
typedef struct db2secUseridPasswordServerAuthFunctions_1
{
   db2int32 version;
   db2int32 plugintype;

  /* parameter lists left blank for readability
    see above for parameters */
 SQL_API_RC (SQL_API_FN * db2secValidatePassword)(<parameter list>);
  SQL_API_RC (SQL_API_FN * db2secGetAuthIDs)(<parameter list);
  SQL_API_RC (SQL_API_FN * db2secDoesAuthIDExist)(<parameter list>);
  SQL_API_RC (SQL_API_FN * db2secFreeToken)(<parameter list>);
  SQL_API_RC (SQL_API_FN * db2secFreeErrormsg)(<parameter list>);
 SQL_API_RC (SQL_API_FN * db2secServerAuthPluginTerm)();
} userid_password_server_auth_functions;
```

または

```
typedef struct db2secGssapiServerAuthFunctions_1
{
 db2int32 version;
  db2int32 plugintype;
  gss_buffer_desc serverPrincipalName;
  gss_cred_id_t ServerCredHandle;
  SQL_API_RC (SQL_API_FN * db2secGetAuthIDs)(<parameter list);
  SQL_API_RC (SQL_API_FN * db2secDoesAuthIDExist)(<parameter list>);
  SQL_API_RC (SQL_API_FN * db2secFreeToken)(<parameter list>);
  SQL_API_RC (SQL_API_FN * db2secFreeErrormsg)(<parameter list>);
  SQL_API_RC (SQL_API_FN * db2secServerAuthPluginTerm)();

 /* GSS-API specific functions
 refer to db2secPlugin.h for parameter list*/
 OM_uint32 (SQL_API_FN * gss_accept_sec_context )(<parameter list>);
 OM_uint32 (SQL_API_FN * gss_display_name )(<parameter list>);
 OM_uint32 (SQL_API_FN * gss_delete_sec_context )(<parameter list>);
 OM_uint32 (SQL_API_FN * gss_display_status )(<parameter list>);
 OM_uint32 (SQL_API_FN * gss_release_buffer )(<parameter list>);
 OM_uint32 (SQL_API_FN * gss_release_cred )(<parameter list>);
 OM_uint32 (SQL_API_FN * gss_release_name )(<parameter list>);

} gssapi_server_auth_functions;
```

ユーザー ID/パスワード・プラグインを作成する場合は、db2secUseridPasswordServerAuthFunctions_1 を使用する必要があります。GSS-API (Kerberos を含む) プラグインを作成する場合は、db2secGssapiServerAuthFunctions_1 を使用する必要があります。

関連概念:

- 581 ページの『セキュリティー・プラグイン』
- 611 ページの『セキュリティー・プラグイン API』

関連タスク:

- 593 ページの『ユーザー ID/パスワード・プラグインのデプロイ』

関連資料:

- 616 ページの『db2secGetGroupsForUser - ユーザーのグループのリストの取得』
- 629 ページの『db2secClientAuthPluginInit - クライアント認証プラグインの初期化』
- 630 ページの『db2secClientAuthPluginTerm - クライアント認証プラグイン・リソースのクリーンアップ』
- 631 ページの『db2secRemapUserid - ユーザー ID およびパスワードの再マップ』
- 633 ページの『db2secGetDefaultLoginContext - デフォルト・ログイン・コンテキストの取得』
- 635 ページの『db2secGenerateInitialCred - 初期証明書の生成』
- 637 ページの『db2secValidatePassword - パスワードの検証』
- 639 ページの『db2secProcessServerPrincipalName - サーバーから戻されたサービス・プリンシパル名の処理』
- 640 ページの『db2secFreeToken - トークンが保持しているメモリーの解放』
- 641 ページの『db2secFreeInitInfo - db2secGenerateInitialCred() が保持しているリソースのクリーンアップ』
- 641 ページの『db2secServerAuthPluginInit - サーバー認証プラグインの初期化』
- 643 ページの『db2secServerAuthPluginTerm - サーバー認証プラグイン・リソースのクリーンアップ』
- 644 ページの『db2secGetAuthIDs - 認証 ID の取得』
- 646 ページの『db2secDoesAuthIDExist - 認証 ID の存在の検査』

db2secClientAuthPluginInit - クライアント認証プラグインの初期化

これは、DB2 がプラグイン・ライブラリーをロードした直後に呼び出すプラグイン・ライブラリーの初期化関数です。 `functions` ポインターは、インターフェース・バージョン用の適切な `client_auth_functions` 構造にキャストされる必要があります。

C API 構文:

```
SQL_API_RC SQL_API_FN db2secClientAuthPluginInit(
 db2int32 version,
 void *client_fns,
 db2secLogMessage *logMessage_fn,
 char **errormsg,
 db2int32  *errormsglen);
```

入力:

db2int32 version
DB2 が現在サポートする API の最高バージョン番号。

*db2secLogMessage *logMessage_fn*

DB2 が提供する関数を指すポインター。プラグインはこの関数を呼び出して、デバッグや通知のための追加のエラー・ストリングを db2diag.log に記録できます。最初のパラメーターは db2secPlugin.h 内の定義を使用する必要があり、最後の 2 つのパラメーターはメッセージ・ストリングとその長さです。最初のパラメーターで使用される定義は、以下のとおりです。

```
#define DB2SEC_LOG_NONE       0   - ロギングなし
#define DB2SEC_LOG_CRITICAL   1   - 重大エラーを検出した
#define DB2SEC_LOG_ERROR      2   - エラーを検出した
#define DB2SEC_LOG_WARNING    3   - 警告
#define DB2SEC_LOG_INFO       4   - 通知
```

DB2SEC_LOG_INFO 定義を使用する場合、メッセージ・テキストは、*diaglevel* データベース・マネージャー構成パラメーターに 4 が設定されている場合にのみ db2diag.log に記録されます。

出力:

*void *client_fns*

DB2が client_auth_functions 構造用に提供しているメモリーを指すポインター。 DB2 の将来のバージョンでは、異なるバージョンの API が存在している可能性があるので、これは、プラグインがインプリメントしている API のバージョンに対応する client_auth_functions 構造を指すポインターにキャストする必要があります。

*char **errormsg*

API が正常に実行されない場合にこのパラメーターに戻されることがある、プラグインによって割り振られた ASCII ストリングのアドレスを指すポインター。

*db2int32 *errormsglen*

*char **errormsg* のエラー・メッセージ・ストリングの長さを示す整数を指すポインター。

関連概念:

- 581 ページの『セキュリティー・プラグイン』
- 611 ページの『セキュリティー・プラグイン API』

関連資料:

- 「*管理ガイド: パフォーマンス*」の『diaglevel - 「診断エラー・キャプチャー・レベル」構成パラメーター』
- 622 ページの『ユーザー ID/パスワード認証プラグインの API』

db2secClientAuthPluginTerm - クライアント認証プラグイン・リソースのクリーンアップ

この関数は、DB2 がプラグインをアンロードする直前に呼び出します。これは、プラグイン・ライブラリーが保持しているリソースの適切なクリーンアップを実行します。たとえば、プラグインによって割り振られたメモリーを解放し、まだオープンしているファイルをクローズし、ネットワーク接続をクローズします。これらのリソースを解放するためにその記録を保持することは、プラグインが行います。

C API 構文:

```
SQL_API_RC SQL_API_FN db2secClientAuthPluginTerm(
 char **errormsg
 db2int32  *errormsglen);
```

出力:

*char **errormsg*
API が正常に実行されない場合にこのパラメーターに戻されることがある、プラグインによって割り振られた ASCII ストリングのアドレスを指すポインター。

*db2int32 *errormsglen*
*char **errormsg* のエラー・メッセージ・ストリングの長さを示す整数を指すポインター。

関連概念:

- 581 ページの『セキュリティー・プラグイン』
- 611 ページの『セキュリティー・プラグイン API』

関連資料:

- 622 ページの『ユーザー ID/パスワード認証プラグインの API』

db2secRemapUserid - ユーザー ID およびパスワードの再マップ

この関数はクライアント・サイドで呼び出され、特定のユーザー ID およびパスワード (そしておそらく新規パスワードおよび usernamespace) を、接続時に指定された値とは異なる値に再マップする機能を提供します。 DB2 は、接続時に少なくともユーザー ID およびパスワードが指定されている場合にのみ、この関数を呼び出します。これは、プラグインがユーザー ID を自らユーザー ID/パスワードのペアに再マップすることを防止します。この関数はオプションであり、提供されなければ呼び出されません。

C API 構文:

```
SQL_API_RC SQL_API_FN db2secRemapUserid(
  char userid[DB2SEC_MAX_USERID_LENGTH],
  db2int32  *useridlen,
  char usernamespace[DB2SEC_MAX_USERNAMESPACE_LENGTH],
  db2int32  *usernamespacelen,
  db2int32  *usernamespacetype,
  char password[DB2SEC_MAX_PASSWORD_LENGTH],
  db2int32  *passwordlen,
  char newpassword[DB2SEC_MAX_PASSWORD_LENGTH],
  db2int32  *newpasswordlen,
  const char *dbname,
  db2int32 dbnamelen,
  char **errormsg,
  db2int32 *errormsglen);
```

入力:

*const char *dbname*
クライアントの接続先のデータベースの名前。

db2int32 dbnamelen
dbname の長さ。

入出力:

char userid[DB2SEC_MAX_USERID_LENGTH]
再マップされるユーザー ID。入力ユーザー ID 値がある場合は出力ユーザー ID 値がなければならず、それは入力ユーザー ID 値と同じか、あるいは異なる値になる可能性があります。入力ユーザー ID 値がない場合、プラグインは出力ユーザー ID 値を戻すべきではありません。

*db2int32 *useridlen*
userid パラメーターに戻されるユーザー ID の長さ。

char usernamespace[DB2SEC_MAX_USERNAMESPACE_LENGTH]
ユーザー ID が属するネームスペース。この再マップはオプションです。この関数に usernamespace が入力として提供されず、この関数が出力として値を提供している場合、 usernamespace は CLIENT 認証の場合にのみ DB2 によって使用され、他の認証タイプでは無視されます。

*db2int32 *usernamespacelen*
古い usernamespace の長さと新規の usernamespace の長さ。
usernamespacetype が DB2SEC_NAMESPACE_SAM_COMPATIBLE でなければならないという制限のもとでは、 DB2 の現在のバージョンでサポートされる最大長は 15 バイトになります。

*db2int32 *usernamespacetype*
古い namespacetype と新規の namespacetype。現行バージョンの DB2 においてサポートされる唯一のネームスペース・タイプは、
DB2SEC_NAMESPACE_SAM_COMPATIBLE です。

char password[DB2SEC_MAX_PASSWORD_LENGTH]
再マップされるパスワード。パスワードが入力として渡された場合、プラグインは新規パスワードを出力する必要があり、これは元のパスワードとは異なる可能性があります。入力として渡されたパスワードがない場合、プラグインは出力パスワードを戻すべきではありません。

*db2int32 *passwordlen*
パスワードの長さ。

char newpassword[DB2SEC_MAX_PASSWORD_LENGTH]
パスワードが変更される場合の新規パスワード。

注: これは、DB2 がクライアント上またはサーバー上のどちらか (AUTHENTICATION dbm 構成パラメーターの値に応じる) の db2secValidatePassword 関数の newpassword フィールドに渡す新規パスワードです。新規パスワードが入力として渡された場合は、出力の新規パスワードが必要であり、別の新規パスワードが存在する可能性もあります。入力として渡された新規パスワードがない場合、プラグインは出力の新規パスワードを戻すべきではありません。

*db2int32 *newpasswordlen*
新規パスワードの長さ。

出力:

*char **errormsg*
API が正常に実行されない場合にこのパラメーターに戻されることがある、プラグインによって割り振られた ASCII ストリングのアドレスを指すポインター。

*db2int32 *errormsglen*
*char **errormsg* のエラー・メッセージ・ストリングの長さを示す整数を指すポインター。

関連概念:

- 581 ページの『セキュリティー・プラグイン』
- 611 ページの『セキュリティー・プラグイン API』

関連資料:

- 622 ページの『ユーザー ID/パスワード認証プラグインの API』

db2secGetDefaultLoginContext - デフォルト・ログイン・コンテキストの取得

この関数は、デフォルト・ログイン・コンテキストに関連したユーザーを判別するため、すなわち、ユーザー ID を明示的に指定しない (データベースに対する暗黙的な認証か、ローカル許可) で DB2 コマンドを呼び出すユーザーの DB2 authid を判別するために DB2 が呼び出します。この関数は、authid とユーザー ID の両方を戻さなければなりません。

C API 構文:

```
db2secGetDefaultLoginContext(
 char authid[DB2SEC_MAX_AUTHID_LENGTH],
 db2int32  *authidlen,
 char userid[DB2SEC_MAX_USERID_LENGTH],
 db2int32  *useridlen,
 db2int32  useridtype,
 char usernamespace[DB2SEC_MAX_USERNAMESPACE_LENGTH],
 db2int32  *usernamespacelen,
 db2int32  *usernamespacetype,
 const char *dbname,
 db2int32  dbnamelen,
 void **token,
 char **errormsg,
 db2int32  *errormsglen);
```

入力:

*const char *dbname*
これには、データベース接続のコンテキストでこの呼び出しが使用される場合に、接続先のデータベースの名前が入ります。ローカル許可アクションやインスタンス・アタッチの場合、このパラメーターは NULL になります。

db2int32 dbnamelen
dbname の長さ。

db2int32 useridtype
DB2 がプロセスの実際のユーザー、または実質的なユーザーを求めているかどうかを指定します。

出力:

char authid[DB2SEC_MAX_AUTHID_LENGTH]
authid が戻されるフィールド。戻される値は DB2 authid の命名規則に準拠していなければなりません。そうでなければ、ユーザーは要求されたアクションの実行を許可されません。

*db2int32 *authidlen*
戻される authid の長さ。

char userid[DB2SEC_MAX_USERID_LENGTH]
ユーザー ID が戻されるフィールド。

*db2int32 *useridlen*
戻されるユーザー ID の長さ。

*void **token*
これはプラグインが、そのプラグインでの後の認証呼び出しや、またはグループ検索プラグインに渡す、プラグインによって割り振られたデータを指すポインターです。このデータの構造は、プラグイン作成者によって決定されます。

char usernamespace[DB2SEC_MAX_USERNAMESPACE_LENGTH]
戻されるネームスペースの長さ。 usernamespacetype が DB2SEC_NAMESPACE_SAM_COMPATIBLE でなければならないという制限のもとでは、 DB2 の現在のバージョンでサポートされる最大長は 15 バイトになります。

*db2int32 *usernamespacelen*
戻されるネームスペースの長さ。 usernamespacetype が DB2SEC_NAMESPACE_SAM_COMPATIBLE でなければならないという制限のもとでは、 DB2 の現在のバージョンでサポートされる最大長は 15 バイトになります。

*db2int32 *usernamespacetype*
上記のとおり。現行バージョンの DB2 においてサポートされる唯一のネームスペース・タイプは、 DB2SEC_NAMESPACE_SAM_COMPATIBLE です。

*char **errormsg*
API が正常に実行されない場合にこのパラメーターに戻されることがある、プラグインによって割り振られた ASCII ストリングのアドレスを指すポインター。

*db2int32 *errormsglen*
*char **errormsg* のエラー・メッセージ・ストリングの長さを示す整数を指すポインター。

関連概念:

- 581 ページの『セキュリティー・プラグイン』
- 611 ページの『セキュリティー・プラグイン API』

関連資料:

- 「*管理ガイド: パフォーマンス*」の『authentication - 「認証タイプ」構成パラメーター』

• 622 ページの『ユーザー ID/パスワード認証プラグインの API』

db2secGenerateInitialCred - 初期証明書の生成

この関数は、渡されるユーザー ID およびパスワードを基に初期 GSS-API 証明書を取得します。 Kerberos の場合、これは TGT です。 pGSSCredHandle に戻される証明書ハンドルは、 gss_init_sec_context() で使用されるハンドルであり、INITIATE 証明書か BOTH 証明書のいずれかでなければなりません。この関数は、ユーザー ID、そしておそらくパスワードが指定されている場合にのみ呼び出されます。それ以外の場合、 DB2 は gss_init_sec_context() を呼び出すときに GSS_C_NO_CREDENTIAL を指定して、現行ログイン・コンテキストから取得されるデフォルト証明書が使用されることを示します。

C API 構文:

```
db2secGenerateInitialCred(
 const char *userid,
 db2int32  useridlen,
 const char *usernamespace,
 db2int32  usernamspacelen,
 db2int32  usernamespacetype,
 const char *password,
 db2int32  passwordlen,
 const char *newpassword,
 db2int32  newpasswordlen,
 const char *dbname,
 db2int32  dbnamelen,
 gss_cred_id_t *pGSSCredHandle,
 void **initInfo,
 char **errormsg,
 db2int32  *errormsglen);
```

入力:

*const char *userid*
パスワードが検証されるユーザー ID。

db2int32 useridlen
ユーザー ID の長さ。

*const char *usernamespace*
取得されたユーザー ID が属するネームスペース。

db2int32 usernamespacelen
ネームスペース・フィールドの長さ。

db2int32 usernamespacetype
ネームスペースのタイプ。

*const char *password*
検証されるパスワード。これは、プラグインに渡される前に DB2 によって暗号化解除されます。

db2int32 passwordlen
newpassword の長さ。

*const char *newpassword*
パスワードが変更される場合の新規パスワード。変更が要求されない場合、これは NULL になります。これが非ヌルの場合、この関数は、新規パスワ

ードに変更する前に古いパスワードを検証する必要があります。プラグインは、パスワードの変更要求を受け入れなくても構いませんが、受け入れない場合は、古いパスワードを検証せずに即時に DB2SEC_PLUGIN_CHANGEPASSWORD_NOTSUPPORTED を戻す必要があります。

db2int32 newpasswordlen
newpassword の長さ。

*const char *dbname*
接続先のデータベースの名前。この関数はこれを無視しても差し支えありません。あるいは、特定のデータベースへのアクセスを、有効なパスワードを特別に持つユーザーのみに限定する場合は、この関数は DB2SEC_PLUGIN_CONNECTION_DISALLOWED を戻すことができます。

db2int32 dbnamelen
dbname の長さ。

出力:

*gss_cred_id_t *pGSSCredHandle*
GSS-API 証明書ハンドルを指すポインター。

*void **initInfo*
DB2 に対して隠されているデータを指すポインター。プラグインはこれを使用して、証明書ハンドルの生成プロセスで割り振られたリソースのリストを保守できます。 DB2 は認証の終わりに `db2secFreeInitInfo()` を呼び出し、その時点でプラグインがこれらのリソースを解放できます。プラグインは、このようなリストを保守する必要がない場合は、NULL を戻す必要があります。

*char **errormsg*
API が正常に実行されない場合にこのパラメーターに戻されることがある、プラグインによって割り振られた ASCII ストリングのアドレスを指すポインター。

注: この API では、戻り値が無効なユーザー ID またはパスワードを示している場合には、エラー・メッセージは作成されるべきではありません。エラー・メッセージは、API の中に、この API が正しく戻ることを妨げる内部エラーがある場合にのみ戻される必要があります。

*db2int32 *errormsglen*
*char **errormsg* のエラー・メッセージ・ストリングの長さを示す整数を指すポインター。

関連概念:

- 581 ページの『セキュリティー・プラグイン』
- 611 ページの『セキュリティー・プラグイン API』

関連資料:

- 622 ページの『ユーザー ID/パスワード認証プラグインの API』

db2secValidatePassword - パスワードの検証

この関数は、データベース接続操作時に、ユーザー ID とパスワードのスタイルの認証方式を提供します。

注: プラグイン・コードは、クライアント・アプリケーションの特権で実行されます。認証に特別な特権 (ルートなど) が必要な場合、プラグイン作成者は、このことを考慮する必要があります。

この API は、パスワードが正常な場合は DB2SEC_PLUGIN_OK (正常) を、また、パスワードが無効な場合は DB2SEC_PLUGIN_BADPWD などのエラー・コードを戻す必要があります。

この API は、*authentication* に CLIENT が設定されている場合にのみ、クライアント・サイドで呼び出されます。

C API 構文:

```
SQL_API_RC SQL_API_FN db2secValidatePassword(
 const char *userid,
 db2int32  useridlen,
 const char *usernamespace,
 db2int32  usernamespacelen,
 db2int32  usernamespacetype,
 const char *password,
 db2int32  passwordlen,
 const char *newpassword,
 db2int32  newpasswordlen,
 const char *dbname,
 db2int32  dbnamelen,
  db2Uint32  connection_details,
  void **token,
  char **errormsg,
  db2int32 *errormsglen);
```

入力:

*const char *userid*
: パスワードが検証されるユーザー ID。

db2int32 useridlen
: ユーザー ID の長さ。

*const char *password*
: 検証されるパスワード。これは、渡される前に DB2 によって暗号化解除されます。

db2int32 passwordlen
: 指定されるパスワードの長さ。

*const char *newpassword*
: パスワードが変更される場合の新規パスワード。変更が要求されない場合、このパラメーターは NULL になります。このパラメーターが非ヌルの場合、この関数は、古いパスワードを新規パスワードに変更する前に検証する必要があります。プラグインは、パスワードの変更要求を受け入れなくても構いませんが、受け入れない場合は、古いパスワードを検証せずに即時に DB2SEC_PLUGIN_CHANGEPASSWORD_NOTSUPPORTED を戻す必要があります。

db2int32 newpasswordlen
newpassword の長さ。

*const char *dbname*
接続先のデータベースの名前。この関数はこれを無視しても差し支えありません。あるいは、特定のデータベースへのアクセスを、有効なパスワードを特別に持つユーザーのみに限定する方針をとっている場合は、この関数は DB2SEC_PLUGIN_CONNECTIONREFUSED を戻すことができます。

db2int32 dbnamelen
dbname の長さ。

db2int32 usernamespace
取得されたユーザー ID が属するネームスペース。

db2int32 usernamespacelen
ネームスペース・フィールドの長さ。

db2int32 usernamespacetype
ネームスペースのタイプ。可能な値は、DB2SEC_NAMESPACE_SAM_COMPATIBLE (torolab¥myname のようなユーザー名のスタイルに対応)、または DB2SEC_NAMESPACE_USER_PRINCIPAL (myname@torolab.ibm.com のようなユーザー名のスタイルに対応) です。現在のところ、DB2 は DB2SEC_NAMESPACE_SAM_COMPATIBLE しかサポートしていません。ユーザー ID がない場合、これには DB2SEC_USER_NAMESPACE_UNDEFINED が入れられます。すべての定義は `db2secPlugin.h` にあります。

db2Uint32 connection_details
現在のところ以下の 2 つのフィールドから成るビット・フィールド。

- 1 ビットは、接続がローカル (ipc を使用しているか、あるいは EEE クラスター内の `db2nodes.cfg` 内にあるノードのいずれかからの接続である) か、リモート (ネットワークまたはループバックを経由) かを示します。これによって、プラグインは同一のマシン上のクライアントがパスワードなしで DB2 サーバーに接続できるかどうか判別できます。現在のところ、 DB2 は同一のマシン上のクライアントからのパスワードなしの接続を常時許可します (クライアントが接続特権を持つ場合)。
- 1 ビットは、ユーザー ID のソースが `db2secGetDefaultLoginContext` のデフォルトであるか、それとも接続時に明示的に指定されているかを示します。ビット値は、`db2secPlugin.h` 内で次のように定義されています。

  ```
  #define DB2SEC_USERID_FROM_OS 0x00000001
  ```

  `DB2SEC_USERID_FROM_OS` は、ユーザー ID は OS から取得され、接続ステートメントで明示的には指定されていないことを示します。

  ```
  #define DB2SEC_CONNECTION_ISLOCAL 0x00000002
  ```

  `DB2SEC_CONNECTION_ISLOCAL` はローカル接続を示します。

  ```
  #define DB2SEC_VALIDATING_ON_SERVER_SIDE 0x0000004
  ```

  `DB2SEC_VALIDATING_ON_SERVER_SIDE` は、 DB2 がサーバー・サイドからパスワードの検証を呼び出しているかどうかを示します。

暗黙的な認証での DB2 のデフォルト動作では、パスワード検証なしの接続が許可されます。しかし、プラグイン開発者には、DB2SEC_PLUGIN_BADPASSWORD エラーを戻すことによって暗黙的な認証を禁止するという選択肢もあります。

*void **token*
この接続中の後続のプラグイン API 呼び出し (db2secGetAuthIDs、db2secGetGroupsForUser) に渡されるデータを指すポインター。

出力:

*char **errormsg*
API が正常に実行されない場合にこのパラメーターに戻されることがある、プラグインによって割り振られた ASCII ストリングのアドレスを指すポインター。

*db2int32 *errormsglen*
*char **errormsg* のエラー・メッセージ・ストリングの長さを示す整数を指すポインター。

関連概念:

- 581 ページの『セキュリティー・プラグイン』
- 611 ページの『セキュリティー・プラグイン API』

関連資料:

- 622 ページの『ユーザー ID/パスワード認証プラグインの API』

db2secProcessServerPrincipalName - サーバーから戻されたサービス・プリンシパル名の処理

この関数は、サーバーから戻されたサービス・プリンシパル名を処理し、gss_init_sec_context() で使用される gss_name_t 内部形式のプリンシパル名を戻します。さらにこの関数は、 Kerberos 認証の場合に db ディレクトリーにカタログされているサービス・プリンシパル名を処理するためにも呼び出されます。通常、この変換では、gss_import_name() API が採用されます。コンテキストが確立されれば、 gss_name_t オブジェクトは gss_release_name() の呼び出しを通して解放されます。この関数は、gssName が有効な GSS 名を指していれば DB2SEC_PLUGIN_OK を戻します。プリンシパル名が無効な場合は DB2SEC_PLUGIN_BAD_PRINCIPAL_NAME エラー・コードが戻されます。

C API 構文:

```
SQL_API_RC SQL_API_FN db2secProcessServerPrincipalName(
 const void *name,
 db2int32 nameLen,
 gss_name_t *gssName,
 char **errormsg,
 db2int32  *errormsglen);
```

入力:

*const void *name*
GSS_C_NT_USER_NAME 形式のサービス・プリンシパルの名前 (例: service/host@REALM)。

db2int32 nameLen
テキストのサービス・プリンシパル名の長さ。

出力:

*gss_name_t *gssName*
GSS-API 内部形式の出力サービス・プリンシパル名を指すポインター。

*char **errormsg*
API が正常に実行されない場合にこのパラメーターに戻されることがある、プラグインによって割り振られた ASCII ストリングのアドレスを指すポインター。

*db2int32 *errormsglen*
*char **errormsg* のエラー・メッセージ・ストリングの長さを示す整数を指すポインター。

関連概念:

- 581 ページの『セキュリティー・プラグイン』
- 611 ページの『セキュリティー・プラグイン API』

関連資料:

- 622 ページの『ユーザー ID/パスワード認証プラグインの API』

db2secFreeToken - トークンが保持しているメモリーの解放

この関数は、DB2 が token によって保持されているメモリーを必要としなくなったときに呼び出します。プラグインはこのメモリーを解放する必要があります。

C API 構文:

```
SQL_API_RC SQL_API_FN db2secFreeToken(
 void *token
 char **errormsg,
 db2int32  *errormsglen);
```

入力:

*void *token*
解放されるメモリーを指すポインター。

出力:

*char **errormsg*
API が正常に実行されない場合にこのパラメーターに戻されることがある、プラグインによって割り振られた ASCII ストリングのアドレスを指すポインター。

*db2int32 *errormsglen*
*char **errormsg* のエラー・メッセージ・ストリングの長さを示す整数を指すポインター。

関連概念:

- 581 ページの『セキュリティー・プラグイン』
- 611 ページの『セキュリティー・プラグイン API』

関連資料:

- 622 ページの『ユーザー ID/パスワード認証プラグインの API』

db2secFreeInitInfo - db2secGenerateInitialCred() が保持しているリソースのクリーンアップ

この関数は、db2secGenerateInitialCred() によって割り振られたリソースを解放します。これには、たとえば、基礎メカニズム・コンテキストのハンドルや、GSS-API 証明書キャッシュ用に作成された証明書キャッシュが含まれます。

C API 構文:

```
SQL_API_RC SQL_API_FN db2secFreeInitInfo(
 void *initinfo,
 char **errormsg,
 db2int32  *errormsglen);
```

入力:

*void *initinfo*
: DB2 に対して隠されているデータを指すポインター。このポインターは、証明書ハンドルの生成プロセスで割り振られたリソースのリストを保守するために使用されます。これらのリソースは、この API を呼び出すことによって解放されます。

出力:

*char **errormsg*
: API が正常に実行されない場合にこのパラメーターに戻されることがある、プラグインによって割り振られた ASCII ストリングのアドレスを指すポインター。

*db2int32 *errormsglen*
: *char **errormsg* のエラー・メッセージ・ストリングの長さを示す整数を指すポインター。

関連概念:

- 581 ページの『セキュリティー・プラグイン』
- 611 ページの『セキュリティー・プラグイン API』

関連資料:

- 622 ページの『ユーザー ID/パスワード認証プラグインの API』

db2secServerAuthPluginInit - サーバー認証プラグインの初期化

これは、DB2 がライブラリーをロードした直後に呼び出すライブラリーの初期化関数です。 functions ポインターは、インターフェース・バージョン用の適切な server_auth_functions 構造にキャストされる必要があります。 GSS-API の場合は、プラグインが、初期化時に gssapi_server_auth_functions 構造内部の serverPrincipalName 変数にサーバーのプリンシパル名を入れ、 *serverCredHandle* 変数にサーバーの証明書ハンドルを提供します。プリンシパル名および証明書ハン

ドルを保持するために割り振られているメモリーを解放するのは、
db2secServerAuthPluginTerm() クリーンアップ関数の行うべき事柄です。

C API 構文:

```
SQL_API_RC SQL_API_FN db2secServerAuthPluginInit(
 db2int32 version,
  void *server_fns,
  db2secGetConDetails *getConDetails_fn,
  db2secLogMessage *logMessage_fn,
  char **errormsg,
  db2int32 *errormsglen);
```

入力:

db2int32 version

DB2 が現在サポートする API の最高バージョン番号。

*db2secGetConDetails *getConDetails_fn*

これは、DB2 が提供する関数を指すポインターです。プラグインは、いずれかの他の認証 API でこの関数を呼び出して、データベース接続に関する詳細を取得することができます。これらの詳細には、接続に関連した通信メカニズム (TCP/IP の場合は IP アドレスなど) についての情報が含まれ、これは、プラグイン作成者が認証についての決定をする際に参照しなければならない可能性があります。たとえば、プラグインは、特定のユーザーが特定の IP アドレスから接続しようとしているのでない場合、そのユーザーの接続を禁止できます。このコールバックの使用は任意です。

データベース接続に関係しない状況でこのコールバックが呼び出された場合、この関数は DB2SEC_PLUGIN_NO_CON_DETAILS を戻し、それ以外の場合は、この関数は正常なら 0 を戻します。

パラメーター getConDetails_fn は、 db2sec_con_details 構造を指すポインターと、使用される db2sec_con_details 構造を示すバージョン番号という 2 つの入力パラメーターをとります。現行のバージョン番号は 1 です。正常に戻されると、db2sec_con_details 構造に以下の詳細が取り込まれます。

- サーバーへの接続に使用されるプロトコル。プロトコル定義のリストは、ファイル sqlenv.h *lpar;SQL_PROTOCOL_* にあります。
- プロトコルが TCP/IP の場合、サーバーへのインバウンド接続の TCP/IP アドレス。
- クライアントが接続しようとしているデータベースの名前。これは、インスタンス・アタッチの場合は設定されません。
- db2secValidatePassword() API の connection_details パラメーター内に記述されるのと同じ詳細を含む、接続情報のビットマップ。

*db2secLogMessage *logMessage_fn*

DB2 が提供する関数を指すポインター。プラグインはこの関数を呼び出して、デバッグや通知のための追加のエラー・ストリングを db2diag.log に記録できます。最初のパラメーターは db2secPlugin.h 内の定義を使用する必要があり、最後の 2 つのパラメーターはメッセージ・ストリングとその長さです。最初のパラメーターで使用される定義は、以下のとおりです。

```
#define DB2SEC_LOG_NONE       0   - ロギングなし
#define DB2SEC_LOG_CRITICAL   1   - 重大エラーを検出した
#define DB2SEC_LOG_ERROR      2   - エラーを検出した
#define DB2SEC_LOG_WARNING    3   - 警告
#define DB2SEC_LOG_INFO       4   - 通知
```

DB2SEC_LOG_INFO 定義を使用する場合、メッセージ・テキストは、*diaglevel* データベース・マネージャー構成パラメーターに 4 が設定されている場合にのみ db2diag.log に記録されます。

出力:

*void *server_fns*

DB2が server_auth_functions 構造用に提供しているメモリーを指すポインター。 DB2 の将来のバージョンでは、異なるバージョンの API が存在している可能性があるので、これは、プラグインがインプリメントしている API のバージョンに対応する server_auth_functions 構造を指すポインターにキャストする必要があります。

server_auth_functions の内部の plugintype 変数には、DB2SEC_PLUGIN_TYPE_USERID_PASSWORD、DB2SEC_PLUGIN_TYPE_GSSAPI、または DB2SEC_PLUGIN_TYPE_KERBEROS のいずれかを設定する必要があります。将来のバージョンの API では、他の値も定義される可能性があります。

*char **errormsg*

API が正常に実行されない場合にこのパラメーターに戻されることがある、プラグインによって割り振られた ASCII ストリングのアドレスを指すポインター。

*db2int32 *errormsglen*

*char **errormsg* のエラー・メッセージ・ストリングの長さを示す整数を指すポインター。

関連概念:

- 581 ページの『セキュリティー・プラグイン』
- 611 ページの『セキュリティー・プラグイン API』

関連資料:

- 「*管理ガイド: パフォーマンス*」の『diaglevel - 「診断エラー・キャプチャー・レベル」構成パラメーター』
- 622 ページの『ユーザー ID/パスワード認証プラグインの API』

db2secServerAuthPluginTerm - サーバー認証プラグイン・リソースのクリーンアップ

この関数は、DB2 がプラグインをアンロードする直前に呼び出します。これは、プラグイン・ライブラリーが保持しているリソースの適切なクリーンアップを実行します。たとえば、プラグインによって割り振られたメモリーを解放し、まだオープンしているファイルをクローズし、ネットワーク接続をクローズします。これらのリソースを解放するためにその記録を保持することは、プラグインが行います。

C API 構文:

```
SQL_API_RC SQL_API_FN db2secServerAuthPluginTerm(char **errormsg,
 db2int32  *errormsglen);
```

出力:

*char **errormsg*
API が正常に実行されない場合にこのパラメーターに戻されることがある、プラグインによって割り振られた ASCII ストリングのアドレスを指すポインター。

*db2int32 *errormsglen*
*char **errormsg* のエラー・メッセージ・ストリングの長さを示す整数を指すポインター。

関連概念:

- 581 ページの『セキュリティー・プラグイン』
- 611 ページの『セキュリティー・プラグイン API』

関連資料:

- 622 ページの『ユーザー ID/パスワード認証プラグインの API』

db2secGetAuthIDs - 認証 ID の取得

この関数は、認証ユーザーに対する SQL authid を戻します。これは、ユーザー ID/パスワードと GSS-API の両方の認証方式において、データベース接続時に呼び出されます。

C API 構文:

```
SQL_API_RC SQL_API_FN db2secGetAuthIDs(
 const char *userid,
 db2int32  useridlen,
 const char *usernamespace,
 db2int32  usernamespacelen,
 db2int32  usernamespacetype,
 const char *dbname,
 db2int32  dbnamelen,
 void **token,
 char SystemAuthID[DB2SEC_MAX_AUTHID_LENGTH],
 db2int32  *SystemAuthIDlen,
 char InitialSessionAuthID[DB2SEC_MAX_AUTHID_LENGTH],
 db2int32  *InitialSessionAuthIDlen,
 char username[DB2SEC_MAX_USERID_LENGTH],
 db2int32  *usernamelen,
 db2int32 *initsessionidtype,
 char **errormsg,
 db2int32  *errormsglen);
```

入力:

*const char * userid*
認証ユーザー。 GSS-API の場合、これはブランクです。

db2int32 useridlen
ユーザー ID の長さ。

*void **token*
プラグインが db2secGetGroupsForUser 呼び出しに渡すデータ。 GSS-API の場合、これはコンテキスト・ハンドル (gss_ctx_id_t) です。通常、これは入力のみのパラメーターであり、この値は db2secValidatePassword から取り込まれます。認証がクライアントで行われ、そのために db2secValidatePassword が呼び出されない場合には、これは出力パラメーターにもなります。

*const char *dbname*
接続先のデータベースの名前。プラグインはこれを無視しても差し支えありません。あるいはプラグインは、同一のユーザーが異なるデータベースに接続する際に別々の authid を戻すこともできます。

db2int32 dbnamelen
dbname の長さ。

*const char *usernamespace*
取得されたユーザー ID が属するネームスペース。

db2int32 usernamespacelen
ネームスペース・フィールドの長さ。

db2int32 usernamespacetype
上記のとおり。現行バージョンの DB2 においてサポートされる唯一のネームスペース・タイプは、 DB2SEC_NAMESPACE_SAM_COMPATIBLE です。

出力:

char SystemAuthID[DB2SEC_MAX_AUTHID_LENGTH]
認証ユーザーの ID に対応するシステム authid。サイズは 255 ですが、DB2 は現在のところ 30 までしか使用できません。

*db2int32 *SystemAuthiIDlen*
戻される SystemAuthId の長さ。

char InitialSessionAuthid[DB2SEC_MAX_AUTHID_LENGTH]
これは、この接続セッションに使用される authid です。これは通常 SystemAuthID と同じですが、 set session authorization ステートメントを発行する場合などのある特定の場合には異なることがあります。サイズは 255 ですが、DB2 は現在のところ 30 までしか使用できません。

*db2int32 *InitialSessionAuthidlen*
戻される InitialSessionAuthID の長さ。

char username[DB2SEC_MAX_USERID_LENGTH]
認証ユーザーと authid に対応するユーザー名。これは監査のためにのみ使用され、「ユーザー ID」フィールドに記録されます。プラグインがこのフィールドを入力しない場合は、DB2 が userid からこれをコピーします。

*db2int32 *usernamelen*
戻されるユーザー ID の長さ。

*db2int32 *initsessionidtype*
InitialSessionAuthid が役割か authid かを示すセッション authid タイプ。プラグインは、 DB2SEC_ID_TYPE_AUTHID (0) または

DB2SEC_ID_TYPE_ROLE (1) (これらは db2secPlugin.h で定義されています) のいずれかを戻す必要があります。現在のところ、DB2 は authid (DB2SEC_ID_TYPE_AUTHID) のみをサポートしています。

*char **errormsg*
API が正常に実行されない場合にこのパラメーターに戻されることがある、プラグインによって割り振られた ASCII ストリングのアドレスを指すポインター。

*db2int32 *errormsglen*
*char **errormsg* のエラー・メッセージ・ストリングの長さを示す整数を指すポインター。

関連概念:

- 581 ページの『セキュリティー・プラグイン』
- 611 ページの『セキュリティー・プラグイン API』

関連資料:

- 622 ページの『ユーザー ID/パスワード認証プラグインの API』

db2secDoesAuthIDExist - 認証 ID の存在の検査

この関数は、authid が個々のユーザーを表しているかどうか (たとえば、この関数がこの authid を外部ユーザーにマップできるかどうか) を判別します。この関数は、これが正常 (authid が有効) な場合は DB2SEC_PLUGIN_OK を、無効な場合は DB2SEC_PLUGIN_INVALID_USERORGROUP を、存在を判別できない場合は DB2SEC_PLUGIN_USERSTATUSNOTKNOWN を戻す必要があります。

C API 構文:

```
SQL_API_RC SQL_API_FN db2secDoesAuthIDExist(
 const char *authid,
 db2int32  authidlen,
 const char *errormsg,
 db2int32  *errormsglen);
```

入力:

*const char *authid*
検証する authid。これは、末尾ブランクなしの大文字になります。

db2int32 authidlen
authid の長さ。

出力:

*char **errormsg*
API が正常に実行されない場合にこのパラメーターに戻されることがある、プラグインによって割り振られた ASCII ストリングのアドレスを指すポインター。

*db2int32 *errormsglen*
*char **errormsg* のエラー・メッセージ・ストリングの長さを示す整数を指すポインター。

関連概念:

- 581 ページの『セキュリティー・プラグイン』
- 611 ページの『セキュリティー・プラグイン API』

関連資料:

- 622 ページの『ユーザー ID/パスワード認証プラグインの API』

GSS-API プラグイン API

GSS-API 認証プラグインに必要な API および定義

このトピックでは、 DB2 セキュリティー・プラグイン・インターフェースに必要な全 GSS-API のリストを提示します。サポートされている API は、*Generic Security Service Application Program Interface, Version 2* (IETF RFC2743) および *Generic Security Service API Version 2: C-Bindings* (IETF RFC2744) の仕様に準拠しています。 GSS-API ベース・プラグインをインプリメントする前に、これらの仕様について完全に理解しておく必要があります。

表 84. GSS-API 認証プラグインに必要な API および定義

名前		説明
クライアント・サイド API		
	gss_init_sec_context	対等アプリケーションとのセキュリティー・コンテキストを開始します。
サーバー・サイド API		
	gss_accept_sec_context	対等アプリケーションによって開始されたセキュリティー・コンテキストを受け入れます。
	gss_display_name	内部フォーマットの名前をテキストに変換します。
共通 API		
	gss_delete_sec_context	確立されたセキュリティー・コンテキストを削除します。
	gss_display_status	GSS-API 状況コードに関連したテキスト・エラー・メッセージを取得します。
	gss_release_buffer	バッファーを削除します。
	gss_release_cred	GSS-API 証明書に関連したローカル・データ構造を解放します。
	gss_release_name	内部フォーマットの名前を削除します。
必要な定義		
	GSS_C_DELEG_FLAG	委任が要求されました。
	GSS_C_EMPTY_BUFFER	gss_buffer_desc にデータが含まれていないことを意味します。
	GSS_C_GSS_CODE	GSS メジャー状況コードを示します。
	GSS_C_INDEFINITE	メカニズムがコンテキストの有効期限をサポートしていないことを示します。
	GSS_C_MECH_CODE	GSS マイナー状況コードを示します。
	GSS_C_MUTUAL_FLAG	相互認証が要求されました。

表 84. GSS-API 認証プラグインに必要な API および定義 (続き)

名前		説明
	GSS_C_NO_BUFFER	gss_buffer_t 変数が有効な gss_buffer_desc 構造を指していないことを示します。
	GSS_C_NO_CHANNEL_BINDINGS	通信チャネル・バインディングがありません。
	GSS_C_NO_CONTEXT	gss_ctx_id_t 変数が有効なコンテキストを指していないことを示します。
	GSS_C_NO_CREDENTIAL	gss_cred_id_t 変数が有効な証明書ハンドルを指していないことを示します。
	GSS_C_NO_NAME	gss_name_t 変数が有効な内部名を指していないことを示します。
	GSS_C_NO_OID	デフォルト認証メカニズムを使用します。
	GSS_C_NULL_OID_SET	デフォルト・メカニズムを使用します。
	GSS_S_COMPLETE	API が正常に完了しました。
	GSS_S_CONTINUE_NEEDED	プロセスが完了しておらず、対等機能から受け取った応答トークンを指定して API をもう一度呼び出す必要があります。

関連概念:

- 581 ページの『セキュリティー・プラグイン』
- 611 ページの『セキュリティー・プラグイン API』

関連資料:

- 648 ページの『GSS-API 認証プラグインに関する制約事項』

GSS-API 認証プラグインに関する制約事項

以下は、GSS-API 認証プラグインに関する制約事項のリストです。

- デフォルト・セキュリティー・メカニズムが常時採用されるため、OID についての考慮事項はありません。
- gss_init_sec_context() で要求される GSS サービスは、相互認証および委任だけです。 DB2 は常に委任のためのチケットを要求しますが、現在のところそのチケットを使用して新規チケットを生成することはありません。
- デフォルト・コンテキスト時刻のみが要求されます。
- gss_delete_sec_context() からのコンテキスト・トークンは、クライアントからサーバー、またはその逆には送信されません。
- 匿名はサポートされていません。
- チャネル・バインディングはサポートされていません。
- 初期証明書の有効期限が切れた場合、DB2 はそれを自動的に更新しません。
- GSS-API 仕様は、gss_init_sec_context() または gss_accept_sec_context() が失敗しても、対等機能に送信するトークンがいずれかの関数によって戻されなければならないことを規定しています。しかし、DRDA の制限のため、DB2 は gss_init_sec_context() が失敗し、かつ最初の呼び出しでトークンを生成した場合にのみトークンを送信できます。

関連概念:

- 581 ページの『セキュリティー・プラグイン』
- 611 ページの『セキュリティー・プラグイン API』

関連資料:

- 647 ページの『GSS-API 認証プラグインに必要な API および定義』

セキュリティー・プラグイン API のバージョン管理

将来の DB2® のリリースでは、バージョンの異なるセキュリティー・プラグイン API が必要になる可能性があるので、 DB2 は API のバージョン番号付けをサポートしています。これらのバージョン番号は、DB2 UDB バージョン 8.2 用の 1 を最初とする整数になります。 DB2 がセキュリティー・プラグイン API に渡すバージョン番号は、 DB2 がサポートできる API の最高バージョン番号であり、構造のバージョン番号に対応します。プラグインは、それより高い API バージョンをサポートできる場合、 DB2 が要求したバージョン用の関数ポインターを戻さなければなりません。プラグインがそれより低いバージョンの API しかサポートできない場合は、その低いバージョン用の関数ポインターを入れる必要があります。いずれのケースでも、セキュリティー・プラグイン API は、サポートしている API のバージョン番号を、関数構造のバージョン・フィールドに入れて戻す必要があります。

DB2 の場合、セキュリティー・プラグインのバージョン番号は、必要な場合にのみ変化します。これには、たとえば、API のパラメーターが変更された場合があります。バージョン番号が DB2 のリリース番号とともに自動的に変わるわけではありません。

関連概念:

- 581 ページの『セキュリティー・プラグイン』
- 611 ページの『セキュリティー・プラグイン API』

第 7 部 一般的な DB2 アプリケーションの概念

第 29 章 各国語サポート

照合順序の概説. 653
照合順序 653
照合順序に基づく文字比較 655
TRANSLATE 関数による、大文字小文字に無関係な文字比較 656
EBCDIC および ASCII 照合順序におけるソート順序の相違 657
データベースの作成時に指定される照合順序 . . 658
照合順序のサンプル 660
コード・ページとロケール 661
コード・ページ値の導出. 661
アプリケーション・プログラム中のロケールの導出 661
DB2 によるロケールの導出方法 662
アプリケーションに関する考慮事項 662
各国語サポートとアプリケーション開発に関する考慮事項 662
各国語サポートと SQL ステートメント . . . 664
リモート・ルーチン 665
混合コード・ページ環境でのパッケージ名の考慮事項 665
プリコンパイルおよびバインド用のアクティブ・コード・ページ. 666
アプリケーション実行用のアクティブ・コード・ページ. 666
異なるコード・ページ間での文字変換 666
コード・ページ変換はいつ行われるか 667
コード・ページ変換時の文字置換. 668
サポートされるコード・ページ変換 668
コード・ページ変換の拡張係数 669
DBCS 文字セット 670
拡張 UNIX コード (EUC) 文字セット 671
DBCS 環境での CLI、ODBC、JDBC、および SQLJ プログラム 672
日本語および中国語 (繁体字) EUC および UCS-2 コード・セットに関する考慮事項. 673
日本語および中国語 (繁体字) EUC および UCS-2 コード・セットに関する考慮事項 . . . 673
EUC および 2 バイトが混在するクライアントおよびデータベースに関する考慮事項 674
中国語 (繁体字) のユーザーへの文字変換に関する考慮事項 675
日本語または中国語 (繁体字) EUC アプリケーションにおけるグラフィック・データ 675
コード・ページが異なる状況におけるアプリケーション開発 677
混合コード・セット環境におけるクライアント・ベースのパラメーターの検証 680
混合コード・セット環境における DESCRIBE ステートメント 681
混合コード・セット環境における固定長および可変長データ 683
混合コード・セット環境におけるコード・ページ変換によるストリング長のオーバーフロー. . . 683
Unicode データベースに接続されるアプリケーション 685

照合順序の概説

以下の節では、照合順序と、文字比較が実行される方法について説明しています。

照合順序

データベース・マネージャーは、*照合順序* によって文字データを比較します。照合順序とは、一組の文字のうちどちらを大きくまたは小さく、あるいは等しく判断するかの順序付けです。

注: FOR BIT DATA 属性および BLOB データで定義した文字ストリング・データは、バイナリー・ソート順序に従ってソートされます。

たとえば、照合順序は特定の文字の小文字と大文字を同等に判断するために使用されます。

データベース・マネージャーにより、カスタム照合順序を持つデータベースの作成が可能となります。以降の節では、データベースで使用する特定の照合順序の決定と実際の使用に役立つ情報を示します。

データベースの 1 バイト文字はそれぞれ、内部では 0〜255 (16 進数表記で X'00'〜X'FF') のユニークな番号で表されます。 この番号を、文字のコード・ポイント といいます。ひとかたまりの文字に割り当てられた数字をコード・ページ と呼びます。照合順序は、ソート後の順序列の中で各文字を配置したい位置とコード・ポイントとの間のマッピングです。位置の数値は、照合順序における文字の重み と呼ばれます。最も単純な照合順序では、コード・ポイントと重みとが同じです。これは基本順序 と呼ばれます。

たとえば、文字 B と b のコード・ポイントがそれぞれ X'42' と X'62' であるとします。照合順序表に従い、いずれもソートの重みが X'42' (B) である場合、これらは同様に照合されます。 B のソートの重みが X'9E' で、 b のソートの重みが X'9D' の場合は、 b が B の前にソートされます。照合順序表は各文字の重みを指定します。表はコード・ページとは異なります。コード・ページは各文字のコード・ポイントを指定します。

次の例を考えてください。 ASCII 文字 A〜Z は X'41'〜X'5A' で表されます。照合順序を記述する場合、これらが (無関係な文字が間に入らずに) ソートされて連続する順序になっていれば、 X'41'、X'42'、... X'59'、X'5A' と記述できます。

マルチバイト文字の 16 進数値もまた重みとして使用されます。たとえば、2 バイト文字 A と B のコード・ポイントがそれぞれ X'8260' と X'8261' であるとします。そして、X'82'、X'60'、および X'61' の照合の重みを使用して、これらの 2 つの文字をコード・ポイントに従ってソートします。

照合順序における重みは、ユニークである必要はありません。たとえば、ある文字の大文字の文字と小文字に同じ重みを与えることができます。

照合順序によりすべての 256 コード・ポイントの重みを決めると、照合順序の指定が容易になります。各文字の重みは、その文字のコード・ポイントを使用して判別することができます。

すべての場合に、 DB2 Universal Database™ (DB2 UDB) はデータベース作成時に指定された照合表を使用します。 コード・ポイント表に現れる順序でマルチバイト文字をソートする場合は、データベースの作成時に照合順序として IDENTITY を指定しなければなりません。

いったん照合順序が定義されると、そのデータベースでそれ以降行われるすべての文字比較はその照合順序によって実行されます。 FOR BIT DATA または BLOB データとして定義された文字データを除き、その照合順序はすべての SQL 比較および ORDER BY 文節で使用され、索引や統計のセットアップにも使用されます。

以下の場合には、問題が発生する可能性があります。

- アプリケーションが、データベースから取り出したソート済みデータを、別の照合順序を使用してソートされたアプリケーション・データとマージする。
- アプリケーションが、あるデータベースから取り出したソート済みデータを、別のデータベースから取り出したソート済みデータとマージするが、それらのデータベースで異なる照合順序が使用されている。
- アプリケーションが、ソート済みデータに関して、関係する照合順序には当てはまらない事項を想定している。たとえば、特定の照合順序では、数字の照合順序が英字より小さくならないことがあります。

最後に覚えておくべき点として、文字コード・ポイントの直接比較に基づくソートの結果は、IDENTITY 照合順序を使用して配列された照会結果にのみ一致します。

関連概念:

- 「*SQL リファレンス 第 1 巻*」の『文字変換』
- 「*管理ガイド: プランニング*」の『DB2 Universal Database での Unicode のインプリメンテーション』
- 655 ページの『照合順序に基づく文字比較』

照合順序に基づく文字比較

SYSTEM、NLSCHAR、COMPATIBILITY、またはユーザー定義の照合オプションを指定することで、データベースの照合順序が確定されると、2 つの文字の比較はコード・ポイント値を直接に比較する代わりに、その重みを比較することによって実行されます。

ユニークでない重みが使用されると、異なる文字が同じ文字として比較される場合があります。このため、ストリング比較は次の 2 段階のプロセスをとることがあります。

1. 各ストリングの文字を重みにより比較する。
2. ステップ 1 での比較の結果、同等である場合は、コード・ポイント値に基づき各ストリングの文字を比較する。

照合順序に 256 のユニークな重みが含まれる場合には、最初のステップのみが実行されます。照合順序が基本順序の場合は、2 番目のステップのみが実行されます。いずれの場合もパフォーマンスが向上します。 Unicode データベースの場合、照合オプションが SYSTEM または IDENTITY である場合、照合シーケンスは IDENTITY になり、2 番目のステップだけが実行されます。

SQL_CS_IDENTITY_16BIT 照合オプションを指定した Unicode データベースは、UTF-8 バイナリー順序ではなく、 CESU-8 バイナリー順序に基づき、データベース内の CHAR または VARCHAR データを照合します。 CESU-8 は *Compatibility Encoding Scheme for UTF-16: 8-Bit (UTF-16 互換の 8 ビット・エンコード・スキーム)* の略であり、 Unicode Consortium Web サイト (www.unicode.org) で入手可能な *Unicode Technical Report #26* で指定されています。 CESU-8 は、Unicode 補足文字を除き、バイナリーが UTF-8 と同じです。つまり、 Unicode 補足文字は 16 ビットの Basic Multilingual Plane (BMP または Plane 0) の外部で定義されています。 UTF-8 エンコードでは、補足文字は 1 つの 4 バイト・シーケンスによって表されますが、CESU-8 では同じ文字に 2 つの 3 バイト・シーケンスが必要です。Unicode データベースでは、文字データ・セットは UTF-8 で保管され、グラフィック・データは UCS-2 で保管されます。 SQL_CS_NONE 照合の場合、UTF-8 の非補足文字と UCS-2 は同一のバイナリー照合順序を持ちますが、UTF-8 の補足文字は UCS-2 において異なった順序で照合されます。

照合オプションが UCA (Unicode Collation Algorithm) タイプである Unicode データベースでは、バイナリー的には同一ではないものの意味論的に等しい文字は、同じように比較されます。このため、ストリング比較は次の 2 段階のプロセスをとることがあります。

1. *Unicode Technical Standard #10* で指定されたアルゴリズムごとに、各ストリングの文字を比較する。この Unicode Technical Standard #10 は、 Unicode Technical Consortium の Web サイト (www.unicode.org) で入手可能です。
2. ステップ 1 での比較の結果、同等である場合は、コード・ポイント値に基づき各ストリングの文字を比較する。

関連概念:

- 「*SQL リファレンス 第 1 巻*」の『文字変換』

TRANSLATE 関数による、大文字小文字に無関係な文字比較

大文字小文字に無関係な文字比較を実行するために、 TRANSLATE 関数を使用して、大文字小文字の混在した列データを大文字に変換した上で、データを選択および比較することができます。ただし、この変換は比較の目的でのみ行われます。次のデータを例にとって考えてみましょう。

```
Abel
abels
ABEL
abel
ab
Ab
```

次の SELECT ステートメントを作成したとします。

```
SELECT c1 FROM T1 WHERE TRANSLATE(c1) LIKE 'AB%'
```

以下を戻します。

```
ab
Ab
abel
Abel
ABEL
abels
```

″v1″ ビューの作成時に、以下の SELECT ステートメントも指定し、大文字でビューを比較して、大文字小文字が混在した形で INSERT 表を要求します。

```
CREATE VIEW v1 AS SELECT TRANSLATE(c1) FROM T1
```

データベース・レベルで、 **sqlecrea** (データベースの作成 API) の一部として照合順序を設定することができます。これにより、″a″ を ″A″ より前に処理するか、″A″ を ″a″ の後に処理するか、またはどちらも同じ重みで処理するかを決定することができます。同じ重みを持たせるならば、ORDER BY 文節を使用して照合またはソートするときに、等しく処理されます。 ″A″ と ″a″ はすべてのセンスで等しいため、 ″A″ は常に ″a″ の前に処理されます。ソートするときの唯一の基準は 16 進値です。

そのため、次のように入力すると

```
SELECT c1 FROM T1 WHERE c1 LIKE 'ab%'
```

以下を戻します。

```
ab
abel
abels
```

および

```
SELECT c1 FROM T1 WHERE c1 LIKE 'A%'
```

以下を戻します。

```
Abel
Ab
ABEL
```

次のステートメントは、

```
SELECT c1 FROM T1 ORDER BY c1
```

以下を戻します。

```
ab
Ab
abel
Abel
ABEL
abels
```

したがって、**sqlecrea** だけでなく、スカラー関数 TRANSLATE() を使用することもできます。ただし、照合順序を指定できるのは、**sqlecrea** だけであることに注意してください。コマンド行プロセッサー (CLP) から照合順序を指定することはできません。

次のように UCASE 関数を使用することもできます。ただし、この場合 DB2® は選択に索引を使用するのではなく、表スキャンを実行することに注意してください。

```
SELECT * FROM EMP WHERE UCASE(JOB) = 'NURSE'
```

関連資料:

- 「*SQL リファレンス 第 1 巻*」の『TRANSLATE スカラー関数』
- 「*SQL リファレンス 第 1 巻*」の『UCASE または UPPER スカラー関数』
- 「*管理 API リファレンス*」の『sqlecrea - データベースの作成』

EBCDIC および ASCII 照合順序におけるソート順序の相違

データベース内のデータをソートする順序は、そのデータベースに定義した照合順序によって決まります。たとえば、データベース A は EBCDIC コード・ページのデフォルトの照合順序を使用し、データベース B は ASCII コード・ページのデフォルトの照合順序を使用すると仮定します。これら 2 つのデータベースのソート順序には、以下の例で示すような違いがあります。

```
SELECT.....
  ORDER BY COL2

EBCDIC-Based Sort        ASCII-Based Sort

COL2                     COL2
----                     ----
V1G                      7AB
Y2W                      V1G
7AB                      Y2W
```

図 64. EBCDIC ベースの順序におけるソート順序と ASCII ベースの順序におけるソート順序の相違例

同様に、データベースでの文字比較は、そのデータベースに定義した照合順序によって決まります。それで、データベース A が EBCDIC コード・ページのデフォルトの照合順序を使用し、データベース B は ASCII コード・ページのデフォルトの照合順序を使用する場合、その 2 つのデータベースにおける文字比較の結果は異なります。以下に、その違いについて示します。

```
SELECT.....
  WHERE COL2 > 'TT3'

EBCDIC-Based Results     ASCII-Based Results

COL2                     COL2
----                     ----
TW4                      TW4
X72                      X72
39G
```

図 65. EBCDIC ベースの順序における文字比較と ASCII ベースの順序における文字比較の相違例

フェデレーテッド・データベースを作成する場合には、ユーザーの照合順序がデータ・ソースでの照合順序と一致するよう指定してください。こうすると、『プッシュダウン』 の機会が最大になるため、照会のパフォーマンスが向上する場合もあります。

関連概念:

- 「*管理ガイド: パフォーマンス*」の『フェデレーテッド照会を評価する場合の分析のガイドライン』

データベースの作成時に指定される照合順序

データベースの照合順序は、データベースの作成時に指定されます。 いったんデータベースを作成してしまうと、照合順序は変更できません。

CREATE DATABASE API は、データベース記述子ブロック (SQLEDBDESC) と呼ばれるデータ構造を受け入れます。この構造内では、ユーザー独自の照合順序を定義できます。

注: 独自の照合順序を定義できるのは、単一バイト・データベースの場合だけです。

データベースの照合順序の指定は次のように行ってください。

- 使用したい SQLEDBDESC 構造を渡すか、または

- NULL ポインターを渡す。オペレーティング・システム (現在の国別/地域別コードとコード・ページに基づく) の照合順序が使用されます。これは、SQL_CS_SYSTEM (0) と等しい SQLDBCSS を指定することと同じです。

SQLEDBDESC 構造には次のものが含まれています。

SQLDBCSS データベース照合順序のソースを示す 4 バイトの整数です。有効な値は以下のとおりです。

SQL_CS_SYSTEM
オペレーティング・システム (現在の国別/地域別コードとコード・ページに基づく) の照合順序が使用されます。

SQL_CS_SYSTEM_NLSCHAR
NLS バージョンの文字タイプ比較ルーチンを使用する、ユーザー指定の照合順序。

SQL_CS_IDENTITY_16BIT
SQL_CS_IDENTITY_16BIT 照合オプションを使用すれば、Unicode データベースを作成できます。SQL_CS_IDENTITY_16BIT がデフォルトの SQL_CS_NONE 照合オプションと異なっているところは、 Unicode データベース内の CHAR または VARCHAR データの照合に、UTF-8 バイナリー順序でなく CESU-8 バイナリー順序が使用されることです。 CESU-8 は *Compatibility Encoding Scheme for UTF-16: 8-Bit (UTF-16 互換の 8 ビット・エンコード・スキーム)* の略であり、 Unicode Consortium Web サイト (www.unicode.org) で入手可能な *Unicode Technical Report #26* で指定されています。 CESU-8 は、Unicode 補足文字を除き、バイナリーが UTF-8 と同じです。つまり、Unicode 補足文字は 16 ビットの Basic Multilingual Plane (BMP または Plane 0) の外部で定義されています。 UTF-8 エンコードでは、補足文字は 1 つの 4 バイト・シーケンスによって表されますが、CESU-8 では同じ文字に 2 つの 3 バイト・シーケンスが必要です。 Unicode データベースでは、文字データ・セットは UTF-8 で保管され、グラフィック・データは UCS-2 で保管されます。 SQL_CS_NONE 照合の場合、UTF-8 の非補足文字と UCS-2 は同一のバイナリー照合順序を持ちますが、UTF-8 の補足文字は UCS-2 において異なった順序で照合されます。 SQL_CS_IDENTITY_16BIT では、補足文字と非補足文字を含む DB2® Unicode データベース内のすべての文字が、同じバイナリー照合順序を持ちます。

SQL_CS_UCA_NO
SQL_CS_UCA400_NO 照合オプションを使用すれば、Unicode データベースを作成できます。

SQL_CS_UCA400_NO は、正規化を暗黙的にオンに設定した Unicode Standard バージョン 4.00 に基づき、 UCA (Unicode Collation Algorithm) 照合シーケンスを指定します。 UCA の詳細は、*Unicode Technical Standard #10* にあります。これは、Unicode Consortium Web サイト (www.unicode.org) で入手可能です。

SQL_CS_UCA_LTH
SQL_CS_UCA400_LTH 照合オプションを使用すれば、Unicode データベースを作成できます。SQL_CS_UCA400_LTH は、Unicode Standard バージョン 4.00 に基づいて UCA (Unicode Collation Algorithm) 照合シーケンスを指定しますが、すべてのタイ語文字を Royal Thai 辞書順序としてソートします。 UCA の詳細は、*Unicode Technical Standard #10* にあります。これは、 Unicode Consortium Web サイト (www.unicode.org) で入手可能です。

SQL_CS_USER
照合順序は、SQLDBUDC フィールドの値によって指定されます。

SQL_CS_NONE
照合順序は基本順序です。ストリングは最初のバイトから、単純なコード・ポイント比較により 1 バイトごとに比較されます。

注: これらの定数は、SQLENV 組み込みファイルに定義されています。

SQLDBUDC 256 バイトのフィールドです。 *n* 番目のバイトには、データベースのコード・ページの *n* 番目の文字のソートの重みが含まれています。 SQLDBCSS が SQL_CS_USER と異なる場合、このフィールドは無視されます。

関連資料:

- 「*管理 API リファレンス*」の『sqlecrea - データベースの作成』

照合順序のサンプル

デフォルトに使用されるワークステーションの照合順序ではなく、 EBCDIC 照合順序を使用したデータベースを作成しやすくするためのいくつかのサンプルが (組み込みファイルとして) 提供されています。

これらの組み込みファイルでの照合順序は、 SQLEDBDESC 構造の SQLDBUDC フィールドに指定することができます。これらの照合順序は、別の照合順序の構造のモデルとしても使うことができます。

照合順序を含む組み込みファイルは、以下のホスト言語で使用できます。

- C/C++
- COBOL
- FORTRAN

関連資料:

- 152 ページの『C および C++ の組み込みファイル』
- 200 ページの『COBOL の組み込みファイル』
- 222 ページの『FORTRAN の組み込みファイル』

コード・ページとロケール

以下の節では、コード・ページと、コード・ページおよびロケールが導出される方法について説明しています。

コード・ページ値の導出

アプリケーションのコード・ページ は、データベース接続時のアクティブ環境から導出されます。 DB2CODEPAGE レジストリー変数が設定されていれば、その値がアプリケーションのコード・ページとなります。ただし、DB2® はオペレーティング・システムから適当なコード・ページ値を決定するため、必ずしもレジストリー変数 DB2CODEPAGE を設定する必要はありません。 DB2CODEPAGE レジストリー変数を誤った値に設定すると、予測できない結果が生じる場合があります。

データベースのコード・ページ は、データベースの作成時に指定される (明示的にまたはデフォルトにより) 値から取得されます。以下は、さまざまなオペレーティング環境においてアクティブ環境 が定められる方法を定義したものです。

UNIX®
UNIX ベースの環境の場合、アクティブ環境は言語、区域およびコード・セットに関する情報を含むロケール設定値により決定されます。

Windows® オペレーティング・システム
Windows オペレーティング・システムの場合、DB2CODEPAGE 環境変数が設定されていない場合、そのコード・ページは、レジストリー内にある ANSI コード・ページの設定値から取得されます。

セクションのコード・ページ は、 SQL ステートメントで使用される表から導出されます。表が、暗黙的または明示的に CCSID ASCII で定義される場合、セクションのコード・ページは、データベースのコード・ページと同じです。表が、CCSID UNICODE で定義される場合、セクションのコード・ページは、Unicode コード・ページです。

関連資料:

- 「*管理ガイド: プランニング*」の『サポートされているテリトリー・コードおよびコード・ページ』

アプリケーション・プログラム中のロケールの導出

ロケールの設定は、Windows® と UNIX® ベースのシステムとでそれぞれ異なります。 UNIX ベースのシステムでは 2 つのロケールがあります。

- 環境ロケールは、使用したい言語、通貨記号などの指定を可能にする。
- プログラム・ロケールには、実行中のプログラムで使用されている言語、通貨記号などが含まれている。

Windows システムでは、「コントロール パネル」の「地域の設定」で国別設定ができます。ただし、UNIX ベースのシステムのような環境ロケールはありません。

プログラムは、開始されるとデフォルトの C ロケールを取得します。 環境ロケールのコピーは取得しません。 プログラム・ロケールを ″C″ 以外のいずれかに設定すると、 DB2 Universal Database は現行のプログラム・ロケールを使用して、アプリケーション環境のコード・ページおよびテリトリー設定を決定します。そうでない場合は、これらの値はオペレーティング・システム環境から得られます。 `setlocale()` はスレッド・セーフではないため、 `setlocale()` をアプリケーション内から出すと、プロセス全体に新しいロケールが設定されることに注意してください。

DB2 によるロケールの導出方法

UNIX® ベースのシステムでは、DB2® で使用されるアクティブ・ロケールはロケールの LC_CTYPE 部分によって決定されます。 詳細については、ご使用のオペレーティング・システムに対応する NLS の資料を参照してください。

- プログラム・ロケールの LC_CTYPE の値が ″C″ と異なる場合、 DB2 はその値を対応するコード・ページにマッピングすることによって、アプリケーションのコード・ページを決定します。
- LC_CTYPE が C と同じ値の場合 (C ロケール)、DB2 は、環境ロケールに従い、 `setlocale()` 関数を使用してプログラム・ロケールを設定します。
- LC_CTYPE が C の値のままである場合、 DB2 はデフォルトの設定として米国英語環境とコード・ページ 819 (ISO 8859-1) を想定します。
- LC_CTYPE がすでに C ではない場合、その新しい値が対応するコード・ページのマッピングに使用されます。

関連資料:

- 「*管理ガイド: プランニング*」の『サポートされているテリトリー・コードおよびコード・ページ』

アプリケーションに関する考慮事項

以下の節では、アプリケーションをコーディングするときに気を付ける必要のある考慮事項について説明しています。

各国語サポートとアプリケーション開発に関する考慮事項

静的 SQL ステートメント中の定数文字ストリングは、バインド時にアプリケーションのコード・ページからデータベースのコード・ページに変換され、このデータベースのコード・ページの表現形式で実行時に使用されます。 このような変換が適切でない場合にそれを避けるには、ストリング定数の代わりにホスト変数を使用できます。

プログラムに固定文字ストリングが含まれる場合、同じコード・ページを使用して、アプリケーションをプリコンパイル、バインド、コンパイル、および実行する必要があります。 Unicode データベースの場合には、ストリング定数の代わりにホスト変数を使用する必要があります。これは、バインド段階と実行段階のいずれにおいても、サーバーによるデータ変換が行われる可能性があるためです。プログラムの中で固定文字ストリングが使われている場合には、この点に注意する必要があります。これらの組み込みストリングは、バインド実行時に、そのバインド・フェーズで有効であるコード・ページに基づいて変換されます。 7 ビット ASCII 文字は DB2 Universal Database のサポートするすべてのコード・ページで共通なので、問題は発生しません。非 ASCII 文字については、バインドと実行において、同一の変換表が同一のコード・ページを用いて使用されているかどうか確認してください。

アプリケーションが取得する外部データは、アプリケーションのコード・ページに存在すると想定されます。これには、ファイルやユーザー入力から得られるデータも含まれます。アプリケーション以外のソースからのデータが、アプリケーションと同じコード・ページを使用していることを確認してください。

C または C++ のアプリケーションでグラフィック・データを使用するホスト変数を使用する場合は、特殊なプリコンパイラー、アプリケーション・パフォーマンス、アプリケーション設計を考慮する必要があります。アプリケーションで EUC コード・セットを処理する場合は、指針について該当するトピックを参照してください。

アプリケーションを開発するときは、次以降のトピックを検討する必要があります。それらのトピックで記述されている推奨事項に従わないと、予測不能な結果が生じる可能性があります。これらの条件はデータベース・マネージャーによっては検出できないため、エラー・メッセージや警告メッセージは出されません。たとえば、C アプリケーションに、 1 つの列が C1 CHAR(20) と定義されている表 T1 に対する処理を行う次の SQL ステートメントが含まれるとします。

```
     (0)  EXEC SQL CONNECT TO GLOBALDB;
     (1)  EXEC SQL INSERT INTO T1 VALUES ('a-constant');
          strcpy(sqlstmt, "SELECT C1 FROM T1 WHERE C1='a-constant');
     (2)  EXEC SQL PREPARE S1 FROM :sqlstmt;
```

ここで、

```
application code page at bind time = x
application code page at execution time = y
database code page = z
```

バインド実行時、ステートメント (1) の '*a-constant*' は、コード・ページ **x** からコード・ページ **z** に変換されます。この変換は、(x→z) と示すことができます。

実行時には、ステートメント (1) が実行されると、 '*a-constant*' (x→z) が表に挿入されます。 しかし、ステートメント (2) の WHERE 文節は '*a-constant*' (y→z) で実行されます。定数のコード・ポイントが 2 つの変換 (x→z と y→z) で値が異なるものだった場合、ステートメント (2) の SELECT は、ステートメント (1) によって挿入されたデータの取り出しに失敗することがあります。

関連概念:

- 164 ページの『C および C++ でのグラフィック・ホスト変数』
- 661 ページの『コード・ページ値の導出』

- 673 ページの『日本語および中国語 (繁体字) EUC および UCS-2 コード・セットに関する考慮事項』

各国語サポートと SQL ステートメント

SQL ステートメントのコーディングは言語に依存していません。 SQL キーワードは大文字、小文字、またはそれらが混在した形で入力できますが、いずれも示されているとおりに入力しなければなりません。 SQL ステートメントにおけるデータベース・オブジェクト名、ホスト変数、およびプログラム・ラベルは、ご使用のアプリケーション・コード・ページでサポートされる文字でなければなりません。

サーバーはファイル名を変換しません。 ファイル名をコーディングするには、ASCII 不変セットを使用するか、またはファイル・システムに物理的に保管される 16 進数値でパスを指定します。

マルチバイト環境では、不変文字セットに属さない特殊な文字が 4 つあります。それらは次の 4 つです。

- 2 バイトのパーセント文字と 2 バイトの下線記号文字。この 2 つは LIKE 処理で使用されます。
- 2 バイトの空白文字。GRAPHIC ストリングに空白を埋め込むために使用します。
- 2 バイトの置換文字。ソース・コード・ページとターゲット・コード・ページとの間に対応するマッピングがない場合のコード・ページ変換時に代替の置換文字として使用されます。

以下の表は、上記 4 つの文字の各コード・ポイントをコード・ページごとに示したものです。

表 85. 特殊 2 バイト文字のコード・ポイント

コード・ページ	2 バイト パーセント記号	2 バイト 下線記号	2 バイト 空白文字	2 バイト 置換文字
932	X'8193'	X'8151'	X'8140'	X'FCFC'
938	X'8193'	X'8151'	X'8140'	X'FCFC'
942	X'8193'	X'8151'	X'8140'	X'FCFC'
943	X'8193'	X'8151'	X'8140'	X'FCFC'
948	X'8193'	X'8151'	X'8140'	X'FCFC'
949	X'A3A5'	X'A3DF'	X'A1A1'	X'AFFE'
950	X'A248'	X'A1C4'	X'A140'	X'C8FE'
954	X'A1F3'	X'A1B2'	X'A1A1'	X'F4FE'
964	X'A2E8'	X'A2A5'	X'A1A1'	X'FDFE'
970	X'A3A5'	X'A3DF'	X'A1A1'	X'AFFE'
1381	X'A3A5'	X'A3DF'	X'A1A1'	X'FEFE'
1383	X'A3A5'	X'A3DF'	X'A1A1'	X'A1A1'
13488	X'FF05'	X'FF3F'	X'3000'	X'FFFD'
1363	X'A3A5'	X'A3DF'	X'A1A1'	X'A1E0'
1386	X'A3A5'	X'A3DF'	X'A1A1'	X'FEFE'

表 85. 特殊 2 バイト文字のコード・ポイント (続き)

コード・ページ	2 バイト パーセント記号	2 バイト 下線記号	2 バイト 空白文字	2 バイト 置換文字
5039	X'8193'	X'8151'	X'8140'	X'FCFC'

Unicode データベースでは、GRAPHIC スペースが X'0020' であり、 euc-Japan および euc-Taiwan データベースに使用される GRAPHIC スペースの X'3000' とは異なっています。 X'0020' と X'3000' のどちらも Unicode 規格のスペース文字です。GRAPHIC スペースのコード・ポイントの違いは、 EUC データベースのデータと Unicode データベースのデータを比較する際に考慮する必要があります。

関連資料:

- 「*SQL リファレンス 第 1 巻*」の『LIKE 述部』
- 671 ページの『拡張 UNIX コード (EUC) 文字セット』

リモート・ルーチン

リモートに実行されるルーチンをコーディングする場合は、次の考慮が必要です。

- ルーチンのデータは、ルーチンの作成時に暗黙的または明示的に指定された PARAMETER CCSID オプションで定義されるコード・ページでなければなりません。
- 特定の文字データ・タイプでルーチンとやり取りされるデータは、必要に応じて、セクションのコード・ページかルーチンのコード・ページに変換されたコード・ページでなければなりません。したがって、クライアント・アプリケーションのコード・ページがステートメントまたはルーチンのコード・ページと異なる場合は、文字タイプに数値データとデータ構造を渡してはなりません。このような変換を避けるには、データを BLOB のデータ・タイプを使用してバイナリー・ストリング形式で定義するか、または文字データを FOR BIT DATA として定義してデータを渡します。

デフォルト設定では、ルーチンを呼び出すと、それらはデフォルトの各国語環境で実行します。これは、データベースの各国語環境と一致しない場合があります。その結果、C `wchar_t` グラフィック・ホスト変数や関数など、国/地域またはコード・ページに特有の操作を使用すると、期待どおりに作動しないことがあります。 ルーチンの呼び出し時には、必ず適切な環境 (適用される場合) が初期化されるようにする必要があります。

混合コード・ページ環境でのパッケージ名の考慮事項

パッケージ名は、PRECOMPILE PROGRAM コマンドまたは API の呼び出し時に判別されます。デフォルト設定では、パッケージ名はアプリケーション・プログラムのソース・ファイルの最初の 8 バイト (ファイル拡張子は除く) に基づいて生成され、大文字で表示されます。任意に、名前を明示的に定義することができます。パッケージ名の由来とは無関係に、異なるコード・ページ環境で実行している場合、パッケージ名の文字が不変の文字セットを使用するようにしなければなりません。そうでないと、パッケージ名の修正に伴って、問題が発生することがあります。デ

ータベース・マネージャーがアプリケーションのパッケージを検出することができないか、または、クライアント側のツールがパッケージの正確な名前を表示しないということが生じます。

パッケージ名の文字のいずれかが、データベースのコード・ページの有効な文字に直接マッピングしない場合に、文字変換が理由となって、パッケージ名が修正されます。 そのような場合、置換文字は変換されない文字を置き換えます。この場合の修正後には、パッケージ名がアプリケーションのコード・ページに戻されても、元のパッケージ名と一致しなくなることがあります。この振る舞いが望ましくない例としては、コントロール・センターを使用してパッケージをリストしたり、その処理を行ったりする場合です。表示されるパッケージ名が、予期している名前と一致しないことがあります。

パッケージ名の変換問題を回避するには、アプリケーションとデータベースの双方のコード・ページで有効な文字だけを使用することです。

プリコンパイルおよびバインド用のアクティブ・コード・ページ

プリコンパイル/バインド時には、プリコンパイラーが実行中のアプリケーションとなります。プリコンパイルの要求に先立つデータベース接続の際のアクティブ・コード・ページは、プリコンパイル済みステートメント、および SQLCA 中に戻される文字データに適用されます。

関連概念:

- 666 ページの『アプリケーション実行用のアクティブ・コード・ページ』

アプリケーション実行用のアクティブ・コード・ページ

実行時は、データベース接続時のユーザー・アプリケーションのアクティブ・コード・ページが、その接続の続いている間有効となります。すべてのデータはこのコード・ページに基づいて解釈されます。これには、動的 SQL ステートメント、ユーザー入力データ、ユーザー出力データ、および SQLCA 中の文字フィールドが含まれます。

関連概念:

- 666 ページの『プリコンパイルおよびバインド用のアクティブ・コード・ページ』

異なるコード・ページ間での文字変換

最適なパフォーマンスを得るためには、アプリケーションが、常にアプリケーションから呼び出したステートメントと同じコード・ページを使用するのが理想的です。ただし、これが常に実用的または可能であるとは限りません。 DB2® 製品は、アプリケーションとデータベースが異なったコード・ページを使用できるようにするコード・ページ変換をサポートします。 1 つのコード・ページの文字は、データ保全性を保持するために別のコード・ページにマップされなければなりません。

コード・ページ変換はいつ行われるか

コード・ページ変換は、以下のような状況において行われます。

- データベースにアクセスするクライアントまたはアプリケーションが、呼び出されるステートメントのコード・ページとは別のコード・ページで実行されているとき。

  状況によっては、クライアント/サーバー文字変換を最小化したり除去することさえできる場合があります。たとえば、以下のような場合があります。

  – コード・ページ 850 を通常使用する Windows® クライアント・アプリケーションの環境に一致させるために、コード・ページ 850 を使用したデータベースを Windows NT 上で作成することができます。

    Windows ODBC アプリケーションを Windows データベース・クライアント内の IBM® DB2® ODBC ドライバーとともに使用する場合は、 `odbc.ini` または `db2cli.ini` ファイル内で TRANSLATEDLL および TRANSLATEOPTION キーワードを使用することによってこの問題が解決される場合があります。

  – コード・ページ 850 を通常使用するクライアント・アプリケーションの環境に一致させるために、コード・ページ 850 を使用したデータベースを AIX® 上で作成することができます。

  – ルーチンの作成時に表および PARAMETER CCSID オプションを作成する場合は、 CCSID オプションを指定しないようにしてください。

- PC/IXF ファイルをインポートするクライアントまたはアプリケーションが、インポートされているファイルと異なるコード・ページで作動している場合。

  このデータ変換は、クライアントがデータベース・サーバーにアクセスする前に、データベース・クライアント・マシンで行われます。そのアプリケーションがデータベースのコード・ページとは異なったコード・ページで作動している場合には、さらに別のデータ変換が行われます (前述のもの)。

  データ変換が行われる場合は、これはインポート・ユーティリティーの呼び出し方によっても異なります。

- ホスト、AS/400®、または iSeries サーバー上のデータへのアクセスに DB2 Connect が用いられる場合。この場合、データの受け取り側が文字データを変換します。たとえば、DB2 for MVS/ESA に送信されるデータは、DB2 for MVS/ESA によって、該当する MVS™ コード化文字セット ID (CCSID) に変換されます。 DB2 for MVS/ESA から DB2 Connect マシンに送り返されるデータは、 DB2 Connect によって変換されます。

以下の状況では、文字変換は行われ**ません**。

- ファイル名。ファイル名には ASCII 不変セットを使うか、またはファイル・システム中に物理的に保管される 16 進値でファイル名を指定してください。ファイル名を SQL ステートメントの一部として組み込むと、ステートメント変換の一部としてファイル名が変換されることに注意してください。
- FOR BIT DATA 属性が割り当てられている列のデータ、またはそれをターゲットとするデータ、処理結果が FOR BIT または BLOB となる SQL 操作で使用されるデータのうちいずれか。これらの場合、データはバイト・ストリームとして扱われ、変換は生じません。

注: FOR BIT DATA と定義された列に挿入されるリテラルは、変換される SQL ステートメントの一部であれば、変換されます。

- 必要なコード・ページの組み合わせに対するサポートがない、あるいはインストールされていない DB2 製品またはプラットフォーム。このような場合には、アプリケーションを実行しようとしても SQLCODE -332 (SQLSTATE 57017) が戻されます。

関連概念:

- 「*SQL リファレンス 第 1 巻*」の『文字変換』

コード・ページ変換時の文字置換

アプリケーションがあるコードから別のコードに変換されるとき、ターゲットのコード・ページで 1 つまたは複数の文字が表示されなくなる場合もあります。このことが生じた場合、DB2 はターゲット・ストリング内の表示されない文字の位置に、*置換* 文字を挿入します。この置換文字は、その後、ストリングの有効な部分と見なされます。置換が行われた場合には、SQLCA の SQLWARN10 標識が 'W' に設定されます。

注: WCHARTYPE CONVERT プリコンパイル・オプションを使用した場合、置換が行われても、その文字に警告のフラグが付けられることはありません。

関連概念:

- 180 ページの『C および C++ での WCHARTYPE プリコンパイラー・オプション』

関連資料:

- 「コマンド・リファレンス」の『PRECOMPILE コマンド』

サポートされるコード・ページ変換

データ変換が発生する場合、*ソース・コード・ページ* からターゲット・コード・ページ への変換が行われます。

ソース・コード・ページは、データのソースにより決定されます。アプリケーションのデータは、アプリケーションのコード・ページと同じソース・コード・ページを持ち、データベースからのデータはデータベースのコード・ページと同じソース・コード・ページを持ちます。

ターゲット・コード・ページの決定はより複雑で、最終的にデータが配置される場所だけでなく、中間操作の規則も考慮されます。

- データが中間の操作なしで直接アプリケーションからデータベースに移動された場合、ターゲット・コード・ページはデータベースのコード・ページとなる。
- PC/IXF ファイルからデータベースにデータがインポートされる場合、文字変換には次の 2 つの手順がある。
  1. PC/IXF ファイルのコード・ページ (ソース・コード・ページ) からアプリケーションのコード・ページ (ターゲット・コード・ページ) への変換
  2. アプリケーションのコード・ページ (ソース・コード・ページ) からデータベースのコード・ページ (ターゲット・コード・ページ) への変換

変換のステップが 2 つ発生する可能性がある場合は注意が必要です。文字データが失われる可能性をなくすためには、必ずサポートされる文字変換に従ってください。さらに、各グループ内で、ソースとターゲットの両方のコード・ページに存在する文字だけに、有効な変換が行われます。他の文字は、代入 として使用され、ターゲット・コード・ページからソース・コード・ページに戻されるときにのみ意味を持ちます (必ずしも上記の 2 つのステップの変換プロセスによって無意味な変換が行われるとは限りません)。 アプリケーションのコード・ページがデータベースのコード・ページと同じであれば、そのような問題は防げます。

- ソースがアプリケーションのコード・ページ、データベースのコード・ページ、FOR BIT DATA または BLOB データのいずれかである文字データに対する処理によりデータが導出される場合、データ変換は一連の規則に基づいたものとなる。最終的なターゲット・コード・ページが定められる前に、データ項目の一部またはすべてが、中間結果に変換されなければならない場合があります。

注: マルチバイト・コード・ページ同士でコード・ページ変換 (たとえば、DBCS と EUC) が行われる場合、ストリングの長さが変化する可能性があります。

関連概念:

- 「*SQL リファレンス 第 1 巻*」の『文字変換』
- 666 ページの『異なるコード・ページ間での文字変換』

関連資料:

- 「*管理ガイド: プランニング*」の『サポートされているテリトリー・コードおよびコード・ページ』

コード・ページ変換の拡張係数

アプリケーションが DB2 データベース・サーバーへの接続を正常に完了したならば、戻された SQLCA の次のフィールドを調べる必要があります。

- SQLERRMC フィールドの 2 番目のトークン (それぞれのトークンは X'FF' で区切られています) は、データベースのコード・ページを示します。 SQLERRMC フィールドの 9 番目のトークンは、アプリケーションのコード・ページを示します。アプリケーションのコード・ページを照会し、それをデータベースのコード・ページと比較すると、確立されている接続が文字変換の行われるタイプのものかどうかがアプリケーションに知らされます。
- SQLERRD 配列の 1 番目および 2 番目の項目。 SQLERRD(1) には、アプリケーション・コード・ページからデータベース・コード・ページに変換された場合に見込まれる、混合文字データ (CHAR データ・タイプ) の長さについての最大拡張係数または縮小係数に等しい整数値が入っています。 SQLERRD(2) には、データベース・コード・ページからアプリケーション・コード・ページへの変換が行われた場合に見込まれる、混合文字データ (CHAR データ・タイプ) の長さについての最大拡張係数または縮小係数に等しい整数値が入っています。 0 または 1 の値は、長さが変わらないことを示し、1 より大きい値は長さが長くなることを、負の値は切り捨てが生じることを示します。

GRAPHIC ストリング・データについての考慮事項は、コード・ページが異なる状況には当てはまりません。 このようなストリングのおのおのの長さは、データがア

プリケーションのコード・ページであるかデータベースのコード・ページであるかに関係なく、常に同じ文字数になります。

関連概念:

- 677 ページの『コード・ページが異なる状況におけるアプリケーション開発』

関連資料:

- 「*SQL リファレンス 第 2 巻*」の『CONNECT (タイプ 1) ステートメント』
- 「*SQL リファレンス 第 2 巻*」の『CONNECT (タイプ 2) ステートメント』

DBCS 文字セット

結合した 1 バイト文字セット (SBCS) または 2 バイト文字セット (DBCS) の各コード・ページでは、 1 バイト文字と 2 バイト文字の両方のコード・ポイントを使用できます。このことは、通常は 2 バイトのコード・ポイントの最初のバイトに対して未定義のコード・ポイントまたは割り当て済みのコード・ポイントのいずれかの剰余分と一緒に、単一バイト文字が混合しているコード表の 256 使用可能コード・ポイントのサブセットを予約することにより行います。以下の表で、これらのコード・ポイントについて示します。

表 86. 混合している文字セットのコード・ポイント

国/地域	サポートされている混合コード・ページ	単一バイト文字のコード・ポイント	2 バイト文字の最初のバイトのコード・ポイント
日本	932、943	X'00'-X'7F', X'A1'-X'DF'	X'81'-X'9F', X'E0'-X'FC'
日本	942	X'00'-X'80', X'A0'-X'DF', X'FD'-X'FF'	X'81'-X'9F', X'E0'-X'FC'
台湾	938 (*)	X'00'-X'7E'	X'81'-X'FC'
台湾	948 (*)	X'00'-X'80', X'FD', X'FE'	X'81'-X'FC'
韓国	949	X'00'-X'7F'	X'8F'-X'FE'
台湾	950	X'00'-X'7E'	X'81'-X'FE'
中国	1381	X'00'-X'7F'	X'8C'-X'FE'
韓国	1363	X'00'-X'7F'	X'81'-X'FE'
中国	1386	X'00'	X'81'-X'FE'
注: (*) これは古いコード・ページなので、お勧めしません。			

こうしたカテゴリーのどこにも割り当てられていないコード・ポイントは定義されていません。それは、単一バイトの未定義のコード・ポイントとして処理されます。

暗黙の DBCS コード表ごとに、有効な最初のバイトの 2 番目のバイトとしてそれぞれ利用できる 256 のコード・ポイントがあります。 2 番目のバイト値は、X'40'

から X'7E'、および X'80' から X'FE' までの任意の値にすることができます。DBCS 環境では、DB2 が個々の 2 バイト文字に対して妥当性検査を行うことはありませんのでご注意ください。

拡張 UNIX コード (EUC) 文字セット

EUC コード・ページはそれぞれ、1 バイト文字のコード・ポイントと、最大 3 つの異なるマルチバイト文字のコード・ポイント・セットを両方とも使用することができます。このサポートは、それぞれの暗黙の 1 バイト文字 SBCS コード・ページ ID で利用可能な 256 のコード・ポイントのサブセットを予約することにより行います。コード・ポイントの剰余分は未定義であったり、マルチバイト文字のエレメントとして割り当てられたり、マルチバイト文字の単一シフト接頭部として割り当てられたりします。以下の表で、これらのコード・ポイントについて示します。

表 87. 日本語 EUC コード・ポイント

グループ	1 番目のバイト	2 番目のバイト	3 番目のバイト	4 番目のバイト
G0	X'20'-X'7E'	n/a	n/a	n/a
G1	X'A1'-X'FE'	X'A1'-X'FE'	n/a	n/a
G2	X'8E'	X'A1'-X'FE'	n/a	n/a
G3	X'8E'	X'A1'-X'FE'	X'A1'-X'FE'	n/a

表 88. 韓国語 EUC コード・ポイント

グループ	1 番目のバイト	2 番目のバイト	3 番目のバイト	4 番目のバイト
G0	X'20'-X'7E'	n/a	n/a	n/a
G1	X'A1'-X'FE'	X'A1'-X'FE'	n/a	n/a
G2	n/a	n/a	n/a	n/a
G3	n/a	n/a	n/a	n/a

表 89. 中国語 (繁体字) EUC コード・ポイント

グループ	1 番目のバイト	2 番目のバイト	3 番目のバイト	4 番目のバイト
G0	X'20'-X'7E'	n/a	n/a	n/a
G1	X'A1'-X'FE'	X'A1'-X'FE'	n/a	n/a
G2	X'8E'	X'A1'-X'FE'	X'A1'-X'FE'	X'A1'-X'FE'
G3	n/a	n/a	n/a	n/a

表 90. 中国語 (簡体字) EUC コード・ポイント

グループ	1 番目のバイト	2 番目のバイト	3 番目のバイト	4 番目のバイト
G0	X'20'-X'7E'	n/a	n/a	n/a
G1	X'A1'-X'FE'	X'A1'-X'FE'	n/a	n/a
G2	n/a	n/a	n/a	n/a
G3	n/a	n/a	n/a	n/a

こうしたカテゴリーのどこにも割り当てられていないコード・ポイントは定義されていません。それは、単一バイトの未定義のコード・ポイントとして処理されます。

DBCS 環境での CLI、ODBC、JDBC、および SQLJ プログラム

JDBC および SQLJ プログラムは、DB2 CLI/ODBC ドライバーを使用して DB2® にアクセスするので、これらのプログラムは同一の構成ファイル (db2cli.ini) を使用します。 DBCS 環境で DB2 Universal Database にアクセスする Java™ プログラムを実行する場合には、この構成ファイルに以下のような項目を追加する必要があります。

PATCH1 = 65536
ドライバーは実際には GRAPHIC リテラルである文字リテラルの前に「G」を手動で挿入することを強制されます。 2 バイト環境で作業している際には、この PATCH1 値を必ず設定する必要があります。

PATCH1 = 64
ドライバーは、グラフィック出力ストリングを NULL で終了するように強制されます。この PATCH1 値は、2 バイト環境で Microsoft® Access を使用する際に必要です。 PATCH1 値も使用する必要があるならば、これら 2 つの値を加算して (64+65536)、PATCH1=65600 を設定します。複数の PATCH1 値を指定する方法については、下記の注 2 を参照してください。

PATCH2 = 7
ドライバーは、すべてのグラフィック列のデータ・タイプをストリング列のデータ・タイプにマップするように強制されます。この PATCH2 値は 2 バイト環境では必須です。

PATCH2 = 10
EUC (拡張 UNIX コード) 環境だけで使用されます。この PATCH2 値により、CLI ドライバーによって、文字変数 (CHAR、VARCHAR など) に、JDBC ドライバーに適した形式でデータが提供されます。この設定をしないと、これらの文字タイプのデータは JDBC では使用できません。

注:

1. これらのキーワードは、 db2cli.ini ファイルのデータベース固有のスタンザに設定されます。複数のデータベースに対してキーワードを設定したい場合には、db2cli.ini の各データベースのスタンザに対してそれを繰り返してください。
2. 複数の PATCH1 値を設定するには、個々の値を加算してから合計値を使用します。 PATCH1 を 64 と 65536 の両方に設定するには、PATCH1=65600 (64+65536) と設定します。すでに他の PATCH1 値を設定している場合には、既存の数値を、その既存の数値に新しい PATCH1 値を加えた合計によって置換してください。
3. 複数の PATCH2 値を設定するには、それらをコンマで区切ったストリング (PATCH1 オプションとは異なる) で指定します。 PATCH2 値を 1 および 7 に設定するには、PATCH2=″1,7″ とします。

日本語および中国語 (繁体字) EUC および UCS-2 コード・セットに関する考慮事項

以下の節では、日本語および中国語 (繁体字) EUC および UCS-2 コード・セットに関する考慮事項について説明しています。

日本語および中国語 (繁体字) EUC および UCS-2 コード・セットに関する考慮事項

拡張 UNIX® コード (EUC) とは、UNIX 系のオペレーティング環境において 1 つ以上 4 つまでの文字セットをサポートできる一連のエンコード規則のことを言います。このエンコード規則は、制御文字が文字セットの一部を分離させるために使用される、 7 ビットおよび 8 ビット・データをエンコードするための ISO 2022 定義に基づいています。 EUC に基づくコード・セットは、基本的には EUC エンコード規則に従いますが、固有のインスタンスに関連付けられた固有の文字セットをも識別します。たとえば、日本語用である IBM®-eucJP コード・セットは、 EUC エンコード規則に従って日本工業規格文字のエンコードも参照します。

グラフィック (つまり 2 バイト文字) データに対するデータベースおよびクライアント・アプリケーション・サポートは、長さが 2 バイト以上の文字エンコードを使用する EUC コード・ページで実行される場合、制限されてしまいます。 DB2 Universal Database 製品は、すべての文字が 2 バイト幅でなければならないという厳密な規則をグラフィック・データに適用します。このような規則のために、日本語および中国語 (繁体字) EUC コード・ページからはほとんどの文字が使用できません。このような状況の解決を目的として、日本語および中国語 (繁体字) EUC グラフィック・データを表示するためのサポートが、アプリケーション・レベルとデータベース・レベルの双方で備えられました。このサポートでは、まったく別個のエンコード・スキーマを使用します。

日本語もしくは中国語 (繁体字) EUC コード・ページで作成されたデータベースでは、グラフィック・データの保管および操作が Unicode UCS-2 コード・セットによって行われます。このコード・セットは、完全な Unicode 文字レパートリーの適格なサブセットの 1 つといえる 2 バイトのエンコード・スキーマです。 これと同様に、このコード・ページで実行されるアプリケーションは、グラフィック・データを UCS-2 エンコード・データとしてデータベース・サーバーに送信します。このサポートにより、EUC コード・ページで実行されるアプリケーションでも、 DBCS コード・ページで実行されるアプリケーションであるかのように同じタイプのデータにアクセスすることが可能になります。 UCS-2 に関連付けられた IBM 定義のコード・ページ ID は 1200 で、同じコード・ページの CCSID 番号は 13488 です。eucJP または eucTW データベースのグラフィック・データは、CCSID 番号 13488 を使用します。 Unicode データベースでは、GRAPHIC データに CCSID 1200 を使用します。

DB2 Universal Database は、UCS-2 を使用してエンコードできるすべての Unicode 文字をサポートします。ただし、文字の構成、分解、正規化は実行しません。Unicode 規格の詳細は、Unicode Consortium の Web サイトである www.unicode.org、および Addison Wesley Longman, Inc. 発行の Unicode Standard ブックの最新版から入手できます。

これらの文字セットを使用するアプリケーションまたはデータベースを処理する場合には、 UCS-2 エンコード・データの処理を考慮する必要があります。 UCS-2 グラフィック・データをアプリケーションの EUC コード・ページに変換すると、データ長が大きくなる可能性があります。また、大量のデータを表示しようとする場合、バッファーを割り振ったり、 1 連のフラグメント化にあるデータを変換および表示したりする必要の生じる場合があります。

以下の節では、そのような環境内にあるデータを処理する方法について説明します。以下の節の中では、EUC という用語が、日本語および中国語 (繁体字) EUC 文字セットだけを指して用いられています。以下の節の説明は、DB2 韓国語または中国語 (簡体字) EUC サポートにはあてはまりません。これら 2 つの文字セットのグラフィック・データは EUC エンコードによって表示されるからです。

関連概念:

- 669 ページの『コード・ページ変換の拡張係数』
- 683 ページの『混合コード・セット環境におけるコード・ページ変換によるストリング長のオーバーフロー』

関連資料:

- 「*管理ガイド: プランニング*」の『サポートされているテリトリー・コードおよびコード・ページ』
- 671 ページの『拡張 UNIX コード (EUC) 文字セット』

EUC および 2 バイトが混在するクライアントおよびデータベースに関する考慮事項

EUC および 2 バイトが混在しているコード・ページ環境にあるデータベース・ オブジェクトは、クライアントおよびデータベースのコード・ページ間で変換するために、オブジェクト名の長さが拡張または縮小することがあり、管理が複雑になります。特に、管理コマンドおよびユーティリティーの多くは、入力または出力パラメーターとして使用できる文字ストリングの長さに限界を設けて明示しています。これらの限界は、特に明示されていなければ、一般にクライアント側にも適用されます。たとえば、表名の限界は 128 バイトです。 2 バイト・コード・ページでは 128 バイトの文字ストリングが、 EUC コード・ページではもっと長く、たとえば 135 バイトとなることがあります。この 135 バイトの表名は、宛先の 2 バイト・データベースでは有効であっても、 REORGANIZE TABLE などのコマンドで入力パラメーターとして使用すると、無効と見なされます。同様に、データベースのコード・ページからアプリケーションのコード・ページへ変換した後、出力パラメーターに許可される最大長を超えてしまうことがあります。これは、変換エラーまたは出力データの切り捨てのいずれかの原因となります。

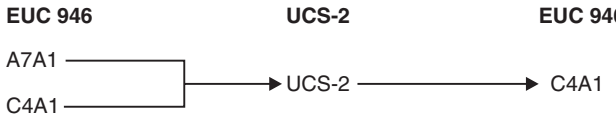
EUC と 2 バイト文字が混在している環境で管理コマンドおよびユーティリティーをよく使用する場合は、データベース・オブジェクトおよびその関連データを、サポートされる限界より長く定義する必要があります。 2 バイト・クライアントから EUC データベースを管理する場合は、 EUC クライアントから 2 バイト・データベースを管理するよりも、制限が少なくなります。通常、2 バイト文字ストリングの長さは、対応する EUC 文字ストリングの長さと同じかそれ以下です。この特性は一般に、文字ストリング長の限界を適用するよりも、問題が少なくてすみます。

注: SQL ステートメントの場合、入力パラメーターの妥当性検査は、ステートメント全体がデータベースのコード・ページに変換されるまで、実施されません。したがって、クライアントのコード・ページで表すときには、技術的に許可されるよりも長い文字ストリングを使用することができます。ただし、それはデータベースのコード・ページで表すときには、長さの要件に適合する必要があります。

中国語 (繁体字) のユーザーへの文字変換に関する考慮事項

中国語 (繁体字) の標準定義のため、2 バイトまたは EUC コード・ページと UCS-2 との間で、ある種の文字を変換する際に、副次作用が発生することがあります。変換した場合に、コード・セットの別の文字と同じ UCS-2 コード・ポイントを共用する文字が 189 文字 (187 の部首および 2 つの数字) あります。これらの文字を 2 バイトまたは EUC に戻すと、元のコード・ポイントではなく、同じ UCS-2 コード・ポイントを共用する同じ漢字のコード・ポイントに変換されます。表示された文字は同じように見えますが、実はコード・ポイントが異なっています。アプリケーションの設計によっては、この振る舞いを考慮に入れなければならないことがあります。

例として、EUC コード・ページ 964 のコード・ポイント A7A1 を UCS-2 に変換した後、元のコード・ページ EUC 946 に戻した場合にどうなるかを考えてみましょう。



上図のように、コード・ポイント A7A1 および C4A1 は、変換後、コード・ポイント C4A1 になります。

日本語または中国語 (繁体字) EUC アプリケーションにおけるグラフィック・データ

以降の情報では、EUC アプリケーション開発におけるグラフィック・データに関する考慮事項について説明します。これには、グラフィック定数、UDF 内のグラフィック・データ、ストアード・プロシージャー、 DBCLOB ファイル、および照合に関する情報が含まれます。

- グラフィック定数

  グラフィック定数 (すなわちリテラル) は、混合文字データとして分類され、SQL ステートメントの一部ともなります。日本語または中国語 (繁体字) EUC クライアントからの SQL ステートメント内のグラフィック定数は、データベース・サーバーによってエンコードされたグラフィックに、暗黙に変換されます。 EUC エンコード文字を構成する GRAPHIC リテラルであれば、 SQL アプリケーション内で使用することができます。そのようなリテラルは、EUC データベース・サーバーによってグラフィック・データベース・コード・セットに変換され、

UCS-2 になります。 EUC クライアントからのグラフィック定数には、CS0 7 ビット ASCII 文字や日本語 EUC CS2 (カタカナ) 文字などの、単一幅の文字を絶対に含めないでください。

- UDF

  UDF は、データベース・サーバーで呼び出される、データベースと同じコード・セットでエンコードされたデータを処理する手段です。日本語または中国語 (繁体字) コード・セットで実行されるデータベースの場合は、そのデータベースを作成する EUC コード・セットによって、混合文字データがエンコードされます。グラフィック・データは UCS-2 によってエンコードされます。 UDF は、UCS-2 によってエンコードされたグラフィック・データを認識して処理する必要があります。

  たとえば、VARCHAR という UDF を作成し、UDF によって GRAPHIC ストリングを混合文字ストリングに変換するとします。この場合 VARCHAR 関数は、データベースが EUC コード・セットで作成されているのであれば、 UCS-2 としてエンコードされた GRAPHIC ストリングを EUC 表記に変換しなければなりません。

- ストアード・プロシージャー

  日本語または中国語 (繁体字) EUC コード・セットで実行されるストアード・プロシージャーは、 UCS-2 によってエンコードされたグラフィック・データを認識および処理できなければなりません。そのようなコード・セットを使用する場合、ストアード・プロシージャーの入力/出力 SQLDA によって送受信したグラフィック・データは、UCS-2 を使用してエンコードします。

- DBCLOB ファイル

  DBCLOB ファイルに関して以下の重要な考慮事項があります。

  – DBCLOB ファイル・データは、アプリケーションの EUC コード・ページにあるものと見なされます。 EUC DBCLOB ファイルの場合、読み取り時にデータはクライアントで UCS-2 に変換され、書き込み時にはクライアントで UCS-2 から他形式に変換されます。
  – サーバー上で読み取られたり書き込まれるバイト数は、ファイル参照変数のデータ長フィールドに戻されます。このバイト数は、ファイルから読み取ったりファイルに書き込んだりした UCS-2 エンコード文字の数に基づいています。ファイルから読み取られたりファイルに書き込んだりする実際のバイト数は、サーバーがデータ長フィールドに書き込むバイト数よりも大きい場合があります。

- 照合

  グラフィック・データはバイナリー順序でソートされます。混合データは各バイトに適用されるデータベースの照合シーケンスでソートされます。同一の国/地域用であっても EUC コード・セットの場合と DBCS コード・セットの場合では文字を順序付ける仕方に違いがあるため、同じデータであっても EUC データベース内のデータをソートする場合と DBCS データベース内のデータをソートする場合とでは異なる結果が生じる可能性があります。

関連資料:

- 「*SQL リファレンス 第 1 巻*」の『GRAPHIC スカラー関数』

- 「*SQL リファレンス 第 2 巻*」の『SELECT ステートメント』
- 「*SQL リファレンス 第 1 巻*」の『GRAPHIC ストリング』

コード・ページが異なる状況におけるアプリケーション開発

アプリケーションのコード・ページやデータベースのコード・ページが使用する文字エンコード・スキーマによっては、ソース・コード・ページからターゲット・コード・ページに変換される際に、ストリングのデータ長が変わる可能性があります。このようなデータ長の変化は、多くの場合、異なるエンコード・スキーマ (たとえば DBCS と EUC) を使用するマルチバイト・コード・ページ間での変換に起因します。

ほとんどの場合、データ長が短くなるよりもデータ長が長くなる方が多くのあるいは深刻な問題が生じます。それは、メモリーの割り振りに余裕がある方が不足するよりも当然勝っているからです。 データ長拡張の可能性があるかどうかによっては、データの送信や取り出しに関するアプリケーションの考慮事項を別個に扱う必要が生じます。 さらに、データ長の拡張または縮小が見られそうな場合の*最善* 状況と*最悪* 状況の違いに注意することも重要です。 正の値 (拡張の可能性を示す) は、*最悪* 状況を乗算係数で示します。たとえば、SQLERRD(1) または SQLERRD(2) フィールドに 2 の値が示された場合、それは変換後のデータの処理にストレージのストリング長が変換前の最大 2 倍必要になることを意味します。これが*最悪* 標識です。 この例で*最善* の状況とは、変換後もデータ長が変わらないことです。

SQLERRD(1) または SQLERRD(2) に負の値 (縮小の可能性を示す) が示される場合も、同様の*最悪* 拡張係数を表します。たとえば、値 -1 は必要となるストレージの最大値が変換前のストリング長と同じであることを示します。負の値によって、ストレージのサイズが変換前よりも小さくてよいことが示される場合もありますが、受信側のアプリケーションがソース・データの構造をはじめから識別しているのでないかぎり、負の値がそのような状況を示すために使われることはほとんどありません。

変換後に拡張が行われても大丈夫なよう十分な量のストレージを割り振るためには、値 max_target_length と同量のストレージを割り振るようにしてください。この値は次のような計算から得られます。

1. データの拡張係数を判別する。

   アプリケーションからデータベースにデータを転送する場合

   ```
   expansion_factor = ABS[SQLERRD(1)]
   if expansion_factor = 0
      expansion_factor = 1
   ```

   データベースからアプリケーションにデータを転送する場合

   ```
   expansion_factor = ABS[SQLERRD(2)]
   if expansion_factor = 0
      expansion_factor = 1
   ```

   上記の計算において、ABS は絶対値を参照しています。

   expansion_factor = 0 のチェックは不可欠です。というのは、DB2 Universal Database 製品の中には SQLERRD(1) および SQLERRD(2) に 0 を戻すものがあ

るからです。そのようなサーバーでは、データの拡張または縮小が生じるようなコード・ページの変換をサポートしていません。このことは拡張係数 1 によって示されます。

2. 中間長を計算する。

```
temp_target_length = actual_source_length * expansion_factor
```

3. ターゲット・データ・タイプの最大長を判別する。

ターゲット・データ・タイプ	タイプの最大長 (type_maximum_length)
CHAR	254
VARCHAR	32 672
LONG VARCHAR	32 700
CLOB	2 147 483 647

4. ターゲットの最大長を判別する。

```
1  if temp_target_length < actual_source_length
      max_target_length = type_maximum_length
   else
2  if temp_target_length > type_maximum_length
      max_target_length = type_maximum_length
   else
3  max_target_length = temp_target_length
```

長さの計算中に生じるかもしれないオーバーフローを許容するには、上記のチェックがすべて必要になります。 それぞれのチェックを以下に詳しく説明します。

1 ステップ 2 の temp_target_length の計算中に数値™のオーバーフローが生じました。

2 つの正の値を乗算した結果がデータ・タイプの最大値を超えてしまう場合、*行が折り返され*、2 つの値のうちの大きい方に満たない分の値が返されます。

たとえば、2 バイト符号付き整数 (CLOB 以外のデータ・タイプの長さに使用される) の最大値は 32 767 です。仮に actual_source_length が 25 000 で拡張係数が 2 であるとすると、 temp_target_length は当然のことながら 50 000 になります。 2 バイト符号付き整数にこの値では大きすぎるので、行は折り返され、値 -15 536 が返されます。

CLOB データ・タイプの場合には、4 バイト符号付き整数が長さに使用されます。 4 バイト符号付き整数の最大値は 2 147 483 647 です。

2 temp_target_length がデータ・タイプの最大値を超過しています。

データ・タイプをステップ 3 にリストされた値よりも長くすることはできません。

変換によって、そのデータ・タイプで認められている以上のスペースが必要とされる場合には、より大きなデータ・タイプを使用すれば結果を保持できるようになるかもしれません。たとえば、CHAR(250) 値が変換後のストリングの保持に 500 バイトを必要とする場合、 CHAR 値の最大長は 254 バイトなのでそのストリングを保持することはできません。しかし、VARCHAR(500) を使用すれば、そのストリングを変換後も保持

できます。変換後のデータがデータ・タイプの限界を超えた場合に何が起こるかについては、混合コード・セット環境におけるコード・ページ変換によるストリング長のオーバーフローに関するトピックを参照してください。

3 `temp_target_length` は適切な長さになっています。

データベースへの接続時に返された SQLERRD(1) および SQLERRD(2) 値と上記の計算を基にして、文字変換後にはストリングの長さがどうなるかを判別できます。一般に、0 または 1 の値は、長さが変わらないことを示し、1 より大きい値は長さが長くなることを、負の値は切り捨てが生じることを示します。 ('0' の値が示されるのは下位の DB2 Universal Database 製品からだけであることに注意してください。) さらに、これらの値は他のデータベース・サーバー製品では定義されていないことにも注意してください。以下の表には、DB2 Universal Database を使用した場合のさまざまなアプリケーション・コード・ページとデータベース・コード・ページの組み合わせにおける理想的な値が示されています。

表 91. CONNECT 時の SQLCA.SQLERRD 設定値

アプリケーション・コード・ページ	データベース・コード・ページ	SQLERRD(1)	SQLERRD(2)
SBCS	SBCS	+1	+1
DBCS	DBCS	+1	+1
eucJP	eucJP	+1	+1
eucJP	DBCS	-1	+2
DBCS	eucJP	+2	-1
eucTW	eucTW	+1	+1
eucTW	DBCS	-1	+2
DBCS	eucTW	+2	-1
eucKR	eucKR	+1	+1
eucKR	DBCS	+1	+1
DBCS	eucKR	+1	+1
eucCN	eucCN	+1	+1
eucCN	DBCS	+1	+1
DBCS	eucCN	+1	+1

SQLERRD(1) または SQLERRD(2) 値にデータベース・サーバーまたはアプリケーション・クライアントでの拡張が示された場合は、以下の事項を考慮する必要があります。

- データベース・サーバーでの拡張

  SQLERRD(1) 項目にデータベース・サーバーでの拡張が示された場合、アプリケーションの側では、クライアントでは有効だった長さ依存の文字データがデータベース・サーバーにおいては変換の影響で無効になる可能性はないかについて考慮する必要が生じます。たとえば、DB2 製品では、列名の長さが 128 バイト以下でなければなりません。しかし、文字ストリングが DBCS コード・ページによって 128 バイトの長さでエンコードされていても、EUC コード・ページに変換すると、この 128 バイトの制限を超えてしまう場合があります。つまり、アプリ

ケーション・コード・ページとデータベース・コード・ページとが同じだったときには有効だった動作でも、これらのコード・ページが異なる場合には無効になり得る、ということです。コード・ページが異なる状況下で EUC および DBCS データベースを設計する際には、これらの点に注意してください。

- アプリケーションでの拡張

  SQLERRD(2) 項目にクライアント・アプリケーションでの拡張が示された場合、アプリケーションの側では、長さ依存の文字データが変換後にはどの程度拡張されるかを把握する必要が生じます。たとえば、CHAR(128) 桁の行を取り出すとします。この場合、データベース・コード・ページとアプリケーション・コード・ページが同じであれば、返されるデータの長さは当然 128 バイトで問題はありません。しかし、この両者のコード・ページが異なる場合には、 DBCS コード・ページでは 128 バイトでエンコードされたデータでも EUC コード・ページに変換すると 128 バイトを超えてしまう可能性があります。したがって、このような場合には、ストリング全体を取り出せるようにするため、追加のストレージを割り振る必要が生じてきます。

関連概念:

- 683 ページの『混合コード・セット環境におけるコード・ページ変換によるストリング長のオーバーフロー』

混合コード・セット環境におけるクライアント・ベースのパラメーターの検証

クライアントとサーバーとの間で文字データの拡張や縮小が行われることの重大な影響の 1 つとして、クライアント・アプリケーションとデータベース・サーバーとの間で受け渡しされるデータを検証する必要が生じることが挙げられます。クライアントとサーバー間でコード・ページが異なる場合、クライアントでは有効と判断されたデータが、コード・ページ変換の結果、データベース・サーバーでは無効になるという状況も十分に生じ得ます。これとは反対に、クライアントでは無効だったデータが、変換の結果、データベース・サーバーで有効になるという場合もあります。

クライアントとサーバー間でコード・ページが異なる場合には、特定のエンド・ユーザー・アプリケーションや API ライブラリーであらゆる処理が行えなくなる可能性があります。さらに、一部のパラメーター (ストリング長など) はクライアントでコマンドや API によって検証されるのに対し、 SQL ステートメント内のトークンはデータベースのコード・ページに変換されるまで検証されません。その検証の結果、コード・ページが異なる環境でも SQL ステートメントを使用するとデータベース・オブジェクト (表など) にアクセスできるのに、特定のコマンドもしくは API ではその同じオブジェクトにアクセスできないという状況が生じ得ます。

ここであるアプリケーションの例を考えてみましょう。そのアプリケーションはエンド・ユーザーが作成した表に含まれているデータを返そうとしています。その表の名前が 128 バイトよりも長くならないことに注目してください。ではさっそく、以下に示されているこのサンプル・アプリケーションのシナリオを考慮してみましょう。

1. DBCS データベースを作成します。 DBCS クライアントからは、表 (t1) が作成されました。この表の名前の長さはちょうど 128 バイトです。この表名にはス

トリングが EUC に変換されると 2 バイトよりも長くなる文字がいくつか含まれています。結果として、EUC 表記の場合には、この表名の長さは 131 バイトになります。 DBCS から DBCS への間の接続では拡張がないので、データベース環境では表名が 128 バイトになり、 CREATE TABLE は正常に実行されます。

2. EUC クライアントが DBCS データベースに接続します。このクライアントからは、表 (t2) が作成されました。この表の名前の長さは、EUC としてエンコードされた場合は 120 バイトで、 DBCS に変換されると 100 バイトになります。DBCS データベースの表名は 100 バイトです。 CREATE TABLE は正常に実行されました。
3. EUC クライアントから、表 (t3) が作成されました。この表の名前は 64 の EUC 文字 (131 バイト) で構成されています。この名前を DBCS に変換すると、この名前の長さは 128 バイトの制限に合わせて切り捨てられることになります。 CREATE TABLE は正常に実行されました。
4. EUC クライアントが DBCS データベース内の表のおのおの (t1、t2、および t3) に対してアプリケーションを呼び出したところ、結果は次のようになりました。

表	結果
t1	アプリケーションはこの表名を無効と判断します。長さが 131 バイトだからです。
t2	正しい結果が表示されます。
t3	アプリケーションはこの表名を無効と判断します。長さが 131 バイトだからです。

5. EUC クライアントによって CLP から DBCS データベースを照会します。この表名はクライアントでは 131 バイトですが、照会は成功します。この表名がサーバーでは 128 バイトになるからです。

混合コード・セット環境における DESCRIBE ステートメント

EUC データベースに対して DESCRIBE を実行すると、データベース内の GRAPHIC 列の定義に基づいた、混合文字と GRAPHIC 列についての情報が返されてきます。この情報は、クライアントのコード・ページに変換される前のサーバーのコード・ページに基づいています。

アプリケーション・コンテキスト (例: `VALUES SUBSTR(?,1,2)`) で解決された選択リスト項目に対して DESCRIBE を実行し、その結果文字データまたはグラフィック・データが関係していた場合、返されてくる SQLLEN 値に加えて、返されてくるコード・ページも評価してください。返されてきたコード・ページがアプリケーション・コード・ページと同じ場合、拡張は行われません。返されてきたコード・ページがデータベース・コード・ページと同じ場合は、拡張の可能性があります。FOR BIT DATA (コード・ページ 0) である選択リスト項目やアプリケーション・コード・ページ内の選択リスト項目は、アプリケーションに返されても変換されません。したがって、報告される長さに拡張や縮小はありません。

DBCS データベースにアクセスする EUC アプリケーションに関する考慮事項と EUC データベースにアクセスする DBCS アプリケーションに関する考慮事項は異なります。

- DBCS データベースにアクセスする EUC アプリケーション

  アプリケーションのコード・ページが EUC コード・ページの場合、 DBCS コード・ページのデータベースに対して DESCRIBE を発行すると、 CHAR および GRAPHIC 列について返された情報がデータベース・コンテキストに返されます。たとえば、 DESCRIBE の一環として返された CHAR(5) 列には、SQLLEN フィールドに 5 の値があります。 EUC 以外のデータの場合は、この列からデータを取り出すと、ストレージを 5 バイト割り振ることになります。 EUC データの場合には、このことはあてはまりません。 DBCS から EUC へのコード・ページ変換が行われると、CHAR 列の文字に使用されるエンコードに違いがあるため、データ長が大きくなる可能性があります。たとえば、中国語 (繁体字) 文字セットの場合、このようなデータ拡張が最大で 2 倍になります。つまり、DBCS エンコードでの最大文字長が 2 バイトのとき、EUC では 4 バイトになる可能性があります。日本語コード・セットの場合も、2 倍の拡張が生じ得ます。ただ中国語の場合とは異なり、日本語 DBCS で最大文字長が 2 バイトだったものが、日本語 EUC では 3 バイトになることもあることに注意してください。このような拡張は係数 1.5 によってのみ知ることができますが、日本語 DBCS では単一バイト・カタカナ文字がわずか 1 バイトなのに対し、日本語 EUC では 2 バイトと違いがあります。

  文字変換の結果としてのデータ長の変化は、混合文字データの場合にのみ起こるものです。グラフィック文字データのエンコードは、エンコード・スキーマがどのようなものであろうと、常に同じ長さ、つまり 2 バイトです。データを誤って失わないようにするため、コード・ページが異なるなどの状況が存在しているかどうか、およびそれが EUC アプリケーションと DBCS データベースとの間のものであるかどうかなどについて評価する必要があります。 データベース・コード・ページとアプリケーション・コード・ページがどのようなものかは、CONNECT ステートメントから返された SQLCA 内にあるトークンから判別できます。そのような状況が存在する場合、アプリケーションの側では、そのエンコード・スキーマの最大拡張係数に基づいて混合文字データ用の追加のストレージを割り振る必要が生じます。

- EUC データベースにアクセスする DBCS アプリケーション

  アプリケーション・コード・ページが DBCS コード・ページで、 DESCRIBE を EUC データベースに対して発行する場合にも、 EUC アプリケーションが DBCS データベースにアクセスする場合と同様の状況が生じます。ただしこちらの状況には、SQLLEN フィールドの値で示されるほどのストレージは必要とされません。この状況で最悪なケースとなるのは、すべてのデータが EUC で単一バイトまたは 2 バイトのどちらかであることです。この場合、SQLLEN に示されたのとまったく同じバイト数が DBCS エンコード・スキーマで必要になります。このような状況を除いては、SQLLEN で示されるよりも少ないバイト数で済みます。どの EUC 文字を保管するにも最大で 2 バイトあれば十分だからです。

関連概念:

- 661 ページの『コード・ページ値の導出』
- 669 ページの『コード・ページ変換の拡張係数』
- 683 ページの『混合コード・セット環境におけるコード・ページ変換によるストリング長のオーバーフロー』

関連資料:

- 「*SQL リファレンス 第 2 巻*」の『DESCRIBE ステートメント』

混合コード・セット環境における固定長および可変長データ

DBCS コード・ページと EUC コード・ページ間での変換時に生じ得るストリング長の変化を考慮に入れると、固定長データ・タイプを使わない方が良いかもしれません。ブランク埋め込みが必要になるかどうかによっては、DESCRIBE の実行後、SQLTYPE を固定長文字ストリングから可変長文字ストリングに変更する方が良いかもしれません。たとえば、EUC と DBCS 間の接続で CHAR(5) 列について最大拡張係数 2 が示されている場合、アプリケーションは 10 バイト割り振る必要があります。

SQLTYPE が固定長の場合、EUC アプリケーションは列を DBCS データ (それ自体は末尾部分の空白として 5 バイトを含めることができる) から変換された EUC データ・ストリームとして受け取ることになります。コード・ページ変換によってデータ・エレメントがその最大サイズまで至らない場合には、これにさらにブランクが埋め込まれます。 SQLTYPE が可変長であれば、CHAR(5) 列の中身の元の意味は保ったまま、ソースの 5 バイトに 5 ～ 10 バイトのターゲットを含めることができます。これと同様に、データが縮小する場合にも (DBCS アプリケーションと EUC データベース間で)、可変長データ・タイプの処理を考慮した方が良いかもしれません。

余分のスペースを割り振ったりデータ・タイプをプロモートする代わりに、データをフラグメント化させるという方法もあります。たとえば、同じ VARCHAR(3000) を選択して変換後も 6 000 バイト含められるようにするため、2 つの選択手段、すなわち `SUBSTR(VC3000, 1, LENGTH(VC3000)/2)` と `SUBSTR(VC3000, (LENGTH(VC3000)/2)+1)` を別々に実行して 2 つの VARCHAR(3000) アプリケーション域に分けることができます。このメソッドを使用できるのは、データ・タイプがもうこれ以上プロモートできないという場合です。たとえば、日本語 DBCS コード・ページでエンコードされ、最大長が 2 ギガバイトの CLOB では、日本語 EUC コード・ページでエンコードする際、そのサイズをおそらく最大 2 回プロモートできます。言い換えると、データをフラグメント化しなければならなくなるのは、 2 ギガバイトを超えデータ・タイプのサポートがなくなってからです。

混合コード・セット環境におけるコード・ページ変換によるストリング長のオーバーフロー

EUC と DBCS が混在するコード・ページ環境では、ストリング全体を 1 つの列に収めきれるだけのスペースを割り振っていないと、変換後に問題が生じる場合があります。この場合、ストリング長の最大拡張は 2 倍になります。拡張が列の容量を超える場合には、SQLCODE -334 (SQLSTATE 22524) が返されます。

このような結果、次のような、すぐには明らかにならなかったり前もっては考慮できない状況に陥ります。

- SQL ステートメントは 32 765 バイトよりも長くすることはできない。ステートメントがかなり複雑だったり、変換時に拡張しそうな定数やデータベース・オブジェクトを多数使用している場合には、意外に早くこの制限に到達してしまいます。
- 変換時拡張の結果として許可されている SQL ID の最大長は、短 ID で 8 バイト、長 ID で 128 バイト。
- 変換時に、ホスト言語 ID は最大長 255 バイトまで拡張できる。
- SQLCA 構造体内の文字フィールドが変換される場合も、最大長として定義されている以上の拡張は許可されていない。

混合コード・セット環境用のアプリケーションを設計する場合、以下の状況については、該当する資料を参照してください。
- 集合演算 (UNION、INTERSECT、および EXCEPT) の全選択における対応するストリング列
- 連結のオペランド
- 述部のオペランド (LIKE を除く)
- CASE ステートメントの結果式
- スカラー関数 COALESCE (および VALUE) の引き数
- IN 述部の IN リストの式値
- 複数行の VALUES 文節の対応する式

上記の状況では、データベース・コード・ページではなく、アプリケーション・コード・ページに対して起こる変換を考慮しています。

考慮しなければならない他の状況は、文字変換後のストリング長がそのデータ・タイプの限界を超えたり、ストアード・プロシージャーでコード・ページ変換が行われたりする場合です。
- 文字変換後の長さがデータ・タイプの限界を超える場合

  EUC と DBCS が混在するコード・ページ環境では、混合文字や GRAPHIC ストリングの長さがそのデータ・タイプで許可されている最大長を超えていると、変換後に問題が生じる場合があります。拡張後のストリング長がデータ・タイプの制限を超える場合、データ・タイプのプロモートは行われません。 代わりに、許可されている拡張の最大値を超えたことを示すエラー・メッセージが返されます。この状況は、挿入時よりも、述部の評価中によく起こります。挿入時には、アプリケーションが容易に列幅を識別でき、最大拡張係数も容易に把握できるからです。大半の場合、文字変換のこの副次作用は、最大長がもっと長い関連データ・タイプに値をキャストすることによって、回避できます。たとえば、CHAR 値の最大長は 254 バイトなのに対し、 VARCHAR の最大値は 32 672 バイトです。拡張によってデータ・タイプの最大長を超えてしまう場合には、SQLCODE -334 (SQLSTATE 22524) が返されます。
- ストアード・プロシージャーでコード・ページ変換が行われる場合

  ホスト変数に指定された混合文字データやグラフィック・データ、および sqleproc() または SQL CALL 呼び出しでの SQLDA は、アプリケーション・コード・ページとデータベース・コード・ページが異なる場合に変換が起きます。 変換の結果としてストリング長の拡張が行われる場合、その拡張に対応できるだけのスペースが割り振られていないと、SQLCODE -334 (SQLSTATE 22524)

が返されてしまいます。そのためストアード・プロシージャーを開発する際には、生じ得る拡張に備えて十分のスペースを割り振っておく必要があります。拡張に対応できるだけのスペースを割り振ることができるよう、可変長データ・タイプを使用するようにしてください。

関連資料:

- 「*SQL リファレンス 第 1 巻*」の『COALESCE スカラー関数』
- 「*SQL リファレンス 第 1 巻*」の『VALUE スカラー関数』
- 「*SQL リファレンス 第 1 巻*」の『全選択』
- 「*SQL リファレンス 第 2 巻*」の『VALUES ステートメント』
- 「*SQL リファレンス 第 2 巻*」の『CASE ステートメント』
- 「*SQL リファレンス 第 1 巻*」の『述部』

Unicode データベースに接続されるアプリケーション

どんなコード・ページ環境のアプリケーションでも、 Unicode データベースに接続できます。 Unicode データベースに接続するアプリケーションに対して、データベース・マネージャーは、文字ストリング・データを、アプリケーション・コード・ページとデータベース・コード・ページ (UTF-8) との間で変換します。 DB2 がコード・ページからの文字を UTF-8 に変換すると、文字のコード・ページおよびコード・ポイントに応じて、その文字を表す合計のバイトの数は増減します。 UTF-8 では 7 ビット ASCII は不変です。各 ASCII 文字は 1 バイトを要します。非 ASCII 文字は、それぞれ 1 バイトになります。 UTF-8 変換の詳細については、Unicode 標準についての文書を参照してください。

注: 混合コード・セットのアプリケーションに適用される情報は、Unicode データベースに接続するアプリケーションにも適用されます。

Unicode データベースでは、GRAPHIC データは UCS-2 ビッグ・エンディアン配列になります。コマンド行プロセッサーを使用してグラフィック・データを検索する場合には、グラフィック文字もクライアント・コード・ページに変換されます。この変換によって、コマンド行プロセッサーは、グラフィック文字を現行のフォントで表示できます。データベース・マネージャーによって UCS-2 文字がクライアント・コード・ページに変換される場合には、一部のデータが失われる可能性があります。データベース・マネージャーがクライアント・コード・ページ内の有効な文字に変換できない文字は、そのコード・ページのデフォルトの置換文字に変換されます。

DB2® バージョン 8 を始動すると、クライアントのコード・ページ設定がチェックされ、 UCS-2 GRAPHIC データの必要な変換すべてが実行されます。たとえば、非 Unicode アプリケーションが GRAPHIC データを送信する場合には、そのデータが UCS-2 データベースに保管される前に、その GRAPHIC データが UCS-2 に変換されます。逆に非 Unicode アプリケーションが GRAPHIC データを UCS-2 データベースから要求する場合には、アプリケーションがデータにアクセスする前に、その GRAPHIC データがアプリケーションのコード・ページに変換されます。

注: 以下の制約事項が適用されます。

- DB2 Load、Import、または Export ユーティリティーで DBCLOB ファイルを処理する場合、クライアントのコード・ページはチェックされません。
- GRAPHIC データが UCS-2 データベースから非 SBCS、非 EUC、または非 Unicode アプリケーションに取り出される場合には、それぞれのブランクに埋め込まれている ASCII ブランク文字 (U+0020) が UCS-2 GRAPHIC 列に置換されます。純粋な DBCS コード・ページには UCS-2 ブランクに相当するものがないので、このような置換が行われます。
- DATE、TIME、および TIMESTAMP データが、 GRAPHIC データ・タイプとして UCS-2 データベースから非 SBCS、非 EUC、または非 Unicode アプリケーションに取り出される場合には、これらのデータ・タイプが置換文字に変換されます。 UCS-2 データ・タイプには純粋な DBCS コード・ページ内に同等のものがない SBCS 文字が含まれているため、こうした置換が行われます。

バージョン 8 より前の DB2 では、 UCS-2 GRAPHIC データの自動変換は実行されませんでした。非 Unicode アプリケーションは Unicode へのおよび Unicode からの必要な変換を実行するか、 WCHARTYPE CONVERT オプションを設定して `wchar_t` を使用する必要がありました。バージョン 7 のクライアントが DB2 バージョン 8 サーバーに接続する場合、デフォルトでは、UCS-2 GRAPHIC データのデータ変換は実行されません。このデフォルトの動作をオーバーライドする場合には、 DB2GRAPHICUNICODESERVER レジストリー変数を `OFF` に設定できます。

DBCS データベースに接続するアプリケーションでは、グラフィック・データは、アプリケーションの DBCS コード・ページとデータベースの DBCS コード・ページとの間で変換されます。

関連概念:

- 「*管理ガイド: プランニング*」の『データ・タイプの Unicode 処理』
- 「*管理ガイド: プランニング*」の『Unicode データベースでのストリングの比較』
- 164 ページの『C および C++ でのグラフィック・ホスト変数』
- 665 ページの『混合コード・ページ環境でのパッケージ名の考慮事項』
- 674 ページの『EUC および 2 バイトが混在するクライアントおよびデータベースに関する考慮事項』
- 680 ページの『混合コード・セット環境におけるクライアント・ベースのパラメーターの検証』
- 681 ページの『混合コード・セット環境における DESCRIBE ステートメント』
- 683 ページの『混合コード・セット環境における固定長および可変長データ』
- 683 ページの『混合コード・セット環境におけるコード・ページ変換によるストリング長のオーバーフロー』

第 30 章 トランザクションの管理

リモート作業単位 687
マルチサイト更新に関する考慮事項 687
マルチサイト更新 688
マルチサイト更新をいつ使用するか 688
マルチサイト更新アプリケーションでの SQL ステートメント 689
マルチサイト更新アプリケーションのプリコンパイル 691
マルチサイト更新アプリケーションの構成パラメーターに関する考慮事項. 692
ホスト、AS/400、または iSeries サーバーへのアクセス 694
並行トランザクション 694
並行トランザクション 694
並行トランザクションを使用する際に気を付けるべき問題 695
並行トランザクションのデッドロックの回避 . . 696
セーブポイントとトランザクション 697
セーブポイントによるトランザクションの管理 697
アプリケーションのセーブポイントとコンパウンド SQL ブロックの比較. 699
セーブポイントの作成や制御に使用する SQL ステートメント 701
セーブポイントの使用に関する制約事項 . . . 702
セーブポイントおよびデータ定義言語 (DDL) 702
セーブポイントのネスト. 704
セーブポイントおよびバッファー化挿入 . . . 704
セーブポイントとカーソル・ブロック化 . . . 704
セーブポイントおよび XA に準拠したトランザクション・マネージャー. 705
X/Open XA インターフェース・プログラミングに関する考慮事項. 705
アプリケーション・リンケージと X/Open XA インターフェース 709
MTS および COM+ トランザクション管理 . . . 709
トランザクション・マネージャーとしての Microsoft Transaction Server (MTS) および Microsoft Component Services (COM+) 709
Microsoft Component Services (COM+) での疎結合サポート 711
Microsoft Transaction Server (MTS) および Microsoft Component Services (COM+) のトランザクション・タイムアウト 712
Microsoft Transaction Server (MTS) および Microsoft Component Services (COM+) を使用した ODBC および ADO 接続プール 713

リモート作業単位

作業単位とは、単一の論理トランザクションのことです。これは、すべての処理の実行が成功するか、または全体としては不成功と考えられる一連の SQL ステートメントから成り立っています。

リモート作業単位により、ユーザーまたはアプリケーション・プログラムは、作業単位あたり 1 つのロケーションでデータの読み取りまたは更新が行えます。これは、作業単位内での 1 つのデータベースへのアクセスをサポートします。アプリケーション・プログラムは複数のリモート・データベースにアクセスできますが、作業単位内では 1 つのデータベースにしかアクセスできません。

リモート作業単位には次の特徴があります。
- 作業単位あたり複数の要求がサポートされる。
- 作業単位あたり複数のカーソルがサポートされる。
- 各作業単位で 1 つのデータベースしかアクセスできない。
- アプリケーション・プログラムは、作業単位をコミットまたはロールバックします。特定のエラー条件では、サーバーは作業単位をロールバックします。

マルチサイト更新に関する考慮事項

以下の節では、マルチサイト更新と、マルチサイト更新を実行するアプリケーションを開発する方法について説明しています。

マルチサイト更新

マルチサイト更新 (*分散作業単位* (DUOW) および *2 フェーズ・コミット* ともいう) は、アプリケーションが、整合性を保証しながら、複数のリモート・データベース・サーバー上のデータを更新する機能です。マルチサイト更新のよい例として、銀行でのトランザクションを挙げることができます。このトランザクションでは、ある口座の預金が別のデータベース・サーバーの口座に移されます。このようなトランザクションでは、1 つの口座での借り方の更新のコミットが、別の口座への貸し方の更新の処理がコミットされる時点で実行されることが重要です。マルチサイト更新の考慮事項は、これらの口座によって表されるデータが 2 つの異なるデータベース・サーバーによって管理される状況に適用されます。

マルチサイト更新を使用すれば、 1 つの作業単位内で複数の DB2 Universal Database のデータベースの読み取りおよび更新を行えます。 DB2® Connect をインストールしているか、 DB2 Universal Database™ Enterprise Edition で提供されている DB2 Connect™ 機能を使用している場合、ホスト、AS/400®、または iSeries データベース・サーバー (DB2 Universal Database for z/OS and OS/390 および DB2 UDB for AS/400 など) とともに、マルチサイト更新を使用することもできます。他のデータベース・サーバーとのマルチサイト更新で DB2 Connect を使用する場合は、特定の制限があります。

トランザクション・マネージャーは、複数のデータベース間でのコミットを調整します。 TxSeries CICS® のようなトランザクション処理 (TP) モニター環境では、TP モニターは独自のトランザクション・マネージャーを使用します。そうでない場合は、DB2 とともに供給されるトランザクション・マネージャーが使用されます。DB2 Universal Database for UNIX®、および Windows® 32 ビット・オペレーティング・システムは、 XA (拡張アーキテクチャー) 準拠のリソース管理プログラムです。 DB2 Connect を使ってアクセスするホストおよび iSeries データベース・サーバーは、 XA 準拠のリソース管理プログラムです。また、DB2 Universal Database トランザクション・マネージャーは、 XA 準拠トランザクション・マネージャーでは*ない* こと、すなわちトランザクション・マネージャーが調整できるのは DB2 データベースだけであることに注意してください。

関連概念:

- 「*管理ガイド: プランニング*」の『X/Open 分散トランザクション処理のモデル』
- 「*DB2 Connect ユーザーズ・ガイド*」の『マルチサイト更新』

マルチサイト更新をいつ使用するか

マルチサイト更新は、複数のデータベースで処理を行い、データ保全を保持したい場合に最も有効です。 たとえば、銀行の各支店がそれぞれのデータベースを持っていた場合、現金転送アプリケーションは、次のことを行います。

1. 送金側のデータベースに接続する。
2. 送金側の口座残高を読み取り、金額が十分あることを確認する。
3. 送金側の口座残高から転送金額を差し引く。
4. 受取側のデータベースに接続する。
5. 受取側の口座残高に転送金額を加算する。
6. データベースをコミットする。

1 つの作業単位で資金の転送を行うことにより、両方のデータベースが更新されるか、またはいずれも更新されないようにできます。

マルチサイト更新アプリケーションでの SQL ステートメント

以下の表は、マルチサイト更新の SQL ステートメントのコーディング方法を説明しています。左の列は、マルチサイト更新を使用しない SQL ステートメントを示し、右の列は、マルチサイト更新と類似したステートメントを示しています。

表 92. RUOW およびマルチサイト更新 SQL ステートメント

RUOW ステートメント	マルチサイト更新ステートメント

CONNECT TO D1
  SELECT
  UPDATE
  COMMIT
 	
CONNECT TO D2
  INSERT
  COMMIT
 	
CONNECT TO D1
  SELECT
  COMMIT
CONNECT RESET
	CONNECT TO D1
	  SELECT
	  UPDATE
	
	CONNECT TO D2
	  INSERT
	  RELEASE CURRENT
	
	SET CONNECTION D1
	  SELECT
	  RELEASE D1
	  COMMIT

左列の SQL ステートメントは、各作業単位あたり 1 つのデータベースのみにアクセスします。これがリモート作業単位 (RUOW) アプリケーションです。

右列の SQL ステートメントは、1 つの作業単位内で複数のデータベースにアクセスします。これはマルチサイト更新アプリケーションです。

SQL ステートメントの中には、マルチサイト更新アプリケーションにおいて異なったコーディングおよび解釈をされるものもあります。

- 他のデータベースに接続する前に現行の作業単位をコミットまたはロールバックする必要はない。
- 他のデータベースへの接続時には、現行接続は切断されない。その代わりに、*休止* 状態となります。 CONNECT ステートメントが失敗しても、現行接続には影響しません。
- データベースへの現行または休止接続がすでに存在している場合、USER/USING 文節では接続することはできない。
- SET CONNECTION ステートメントを使用することにより、休止接続を現行接続に変更することができる。

  休止データベースに CONNECT ステートメントを出すことによっても、同じことが行える。この方式は、SQLRULES を STD に設定した場合は行えません。プリコンパイラー・オプション、SET CLIENT コマンド、API のいずれかを使用することにより、SQLRULES の値を設定できます。 SQLRULES (DB2) のデフォルトにより、CONNECT ステートメントを使用する接続の切り替えが可能になります。
- 選択時に、別のデータベースに切り換えてから元のデータベースに戻しても、カーソル位置は変わらない。

- CONNECT RESET は、現行接続の切断および現行作業単位の暗黙でのコミットを行わない。その代わり、このステートメントは、デフォルトのデータベース (すでに定義されている場合) に明示的に接続するのと同じことになります。暗黙の接続が定義されていない場合、SQLCODE -1024 (SQLSTATE 08003) が戻されます。
- RELEASE ステートメントを使用することにより、次の COMMIT で切断される接続をマークすることができる。 RELEASE CURRENT ステートメントは現行接続に適用され、 RELEASE *connection (接続名)* は指定した接続に適用され、RELEASE ALL ステートメントはすべての接続に適用されます。

  解放とマークされた接続は、次の COMMIT でドロップされるまで使用できる。ロールバックは接続のドロップを行わないため、すでに存在している接続への再試行が可能です。 DISCONNECT ステートメント (またはプリコンパイラー・オプション) を使用して、コミットまたはロールバック後に接続をドロップします。
- COMMIT ステートメントは、作業単位 (現行または休止) におけるすべてのデータベースをコミットする。
- ROLLBACK ステートメントは、作業単位におけるすべてのデータベースをロールバックし、すべてのデータベースで保持されるカーソルを、アクセスされたかどうかにかかわらずクローズする。
- すべての接続 (休止接続および解放とマークされた接続など) は、アプリケーションのプロセスが終了すると切断する。
- 接続が成功すると (オプション指定のない CONNECT ステートメントも含む。これは、現行接続の照会のみを実行する)、数値が SQLCA の SQLERRD(3) および SQLERRD(4) フィールドに戻される。

  SQLERRD(3) フィールドは、接続されたデータベースが作業単位において現在更新可能かどうかについての情報を戻します。次に、戻される可能性のある値を示します。

  1 更新可能

  2 読み取り専用

  SQLERRD(4) フィールドにより戻される、接続の現在の特性についての情報を次に示します。

  0 適用外。 この状態は、更新および 1 フェーズ・コミットの使用を行う下位レベルのクライアントから実行された場合にのみ発生する。

  1 1 フェーズ・コミット。

  2 1 フェーズ・コミット (読み取り専用)。この状態は、DB2® Connect の同期点マネージャーを開始しないで、 DB2 Connect™ を使ってアクセスするホスト、AS/400®、または iSeries データベース・マネージャーにのみ適用できます。

  3 2 フェーズ・コミット

  ツールまたはユーティリティーを作成中、接続が読み取り専用の場合にユーザーにメッセージを出したい場合があるかもしれません。

マルチサイト更新アプリケーションのプリコンパイル

マルチサイト更新アプリケーションをプリコンパイルするときには、 CLP 接続をタイプ 1 接続に設定する必要があります。そのように設定しないと、アプリケーションのプリコンパイルを試行したときに、 SQLCODE 30090 (SQLSTATE 25000) を受け取ります。以下のプリコンパイラー・オプションは、マルチサイト更新を使用するアプリケーションをプリコンパイルするときに使用します。

CONNECT (1 | 2)
このアプリケーションがマルチサイト更新アプリケーションで SQL 構文を使用することを示すには、 CONNECT 2 を指定してください。デフォルトの設定の CONNECT 1 は、SQL 構文の通常の (RUOW) 規則がアプリケーションに適用されることを意味します。

SYNCPOINT (ONEPHASE | TWOPHASE | NONE)
SYNCPOINT TWOPHASE および DB2® がトランザクションを調整するように指定する場合、 DB2 にはトランザクションの状態情報を管理するデータベースが必要です。アプリケーションを配置するときには、データベース・マネージャー構成パラメーター *tm_database* を構成して、このデータベースを定義する必要があります。

SQLRULES (DB2 | STD)
ISO/ANSI SQL92 を基にした DB2 規則または標準 (STD) 規則がマルチサイト更新アプリケーションで使用されるべきかどうかを指定します。 DB2 規則では、休止データベースに CONNECT ステートメントを出すことができます。 STD 規則ではできません。

DISCONNECT (EXPLICIT | CONDITIONAL | AUTOMATIC)
RELEASE ステートメントで解放とマークされたデータベースのみ (EXPLICIT)、オープンされている WITH HOLD カーソルがないすべてのデータベース (CONDITIONAL)、またはすべての接続 (AUTOMATIC) のいずれのデータベース接続が COMMIT 時に切断されるかを指定します。

マルチサイト更新プリコンパイラー・オプションは、最初のデータベース接続時に有効になります。 SET CLIENT API を使用することにより、接続がない場合に (接続が確立される前、またはすべての接続切断後)、接続の設定を置き換えることができます。 QUERY CLIENT API を使用することにより、アプリケーション・プロセスの現行接続の設定を照会することができます。

バインド・プログラムは、アプリケーション・プログラムで参照されるオブジェクトが存在しないと失敗します。マルチサイト更新アプリケーションに対処する方法は 3 つあります。

- アプリケーションを複数のファイルに分割し、それぞれが 1 つのデータベースにのみアクセスする。それぞれのファイルがアクセスする 1 つのデータベースに対して、各ファイルを用意し、バインドします。
- 各表がそれぞれのデータベースに存在することを確保できる。たとえば、銀行のそれぞれの支店のデータベースは、同じ表を持っています (データを除き)。
- 動的 SQL のみを使用できる。

関連概念:

- 689 ページの『マルチサイト更新アプリケーションでの SQL ステートメント』

関連資料:

- 「*SQL リファレンス 第 2 巻*」の『CONNECT (タイプ 1) ステートメント』
- 「*SQL リファレンス 第 2 巻*」の『CONNECT (タイプ 2) ステートメント』
- 「*管理 API リファレンス*」の『sqlesetc - クライアントの設定』
- 「*管理 API リファレンス*」の『sqleqryi - クライアント情報の照会』
- 「*コマンド・リファレンス*」の『PRECOMPILE コマンド』

マルチサイト更新アプリケーションの構成パラメーターに関する考慮事項

以下の構成パラメーターは、マルチサイト更新を実行するアプリケーションに影響を与えます。構成パラメーターは、*locktimeout* 以外は、データベース・マネージャー構成パラメーターです。 *locktimeout* は、データベース構成パラメーターです。

tm_database
: 2 フェーズ・コミット・トランザクションで、どのデータベースがトランザクション・マネージャーとして動作するか指定します。

resync_interval
: 未確定トランザクションを再び同期させるまでのシステムの待ち時間 (秒) を指定します。 (未確定トランザクションとは、2 フェーズ・コミットの第 1 段階まで成功し、第 2 段階で失敗するようなトランザクションのことです。)

locktimeout
: ロックの待ち時間がタイムアウトになり、指定したデータベースの現行のトランザクションをロールバックするまでの秒数を指定します。アプリケーションは、明示的に ROLLBACK を発行して、マルチサイト更新に関与しているすべてのデータベースをロールバックする必要があります。 *locktimeout* は、データベース構成パラメーターです。

tp_mon_name
: TP モニターがある場合その名前を指定します。

spm_resync_agent_limit
: SNA を使用してホスト、AS/400®、または iSeries サーバーと再同期操作を実行できる、同時エージェントの数を指定します。

spm_name
: - 同期点マネージャーが TCP/IP の 2 フェーズ・コミット接続で使用される場合、 *spm_name* はネットワーク内で一意的な ID でなければなりません。 DB2® インスタンスを作成する場合、 DB2 は *spm_name* のデフォルトとして TCP/IP ホスト名から派生したものを使用します。ユーザーの環境でこの値を受け入れることができない場合は、変更することができます。ホスト・データベース・サーバーと TCP/IP 接続できるようにするため、デフォルトを受け入れ可能にする必要があります。ホスト、AS/400、または iSeries データベース・サーバーとの SNA 接続の場合、この値はご使用の SNA 製品で定義した SNA LU プロファイルと一致していなければなりません。

- 同期点マネージャーが SNA の 2 フェーズ・コミット接続で使用される場合、同期点マネージャー名は 2 フェーズ・コミットに使用されている LU_NAME に設定されなければなりません。
- 同期点マネージャーが TCP/IP および SNA の両方に対して使用される場合、 2 フェーズ・コミットに使用されている LU_NAME を使用しなければなりません。

注: ホスト、AS/400、または iSeries データベース・サーバーを使った環境でのマルチサイト更新には、同期点マネージャーが必要です。

spm_log_size
現在の接続状態など接続に関する情報を記録するため、同期点マネージャーが使用する 1 次および 2 次ログ・ファイルの各ページ数 (4K バイト単位)。

ホスト、AS/400、または iSeries データベースとの間で、トランザクション・マネージャーによって調整されるマルチサイト更新をアプリケーションが実行する場合は、付加的な考慮事項があります。

関連概念:

- 「*DB2 Connect ユーザーズ・ガイド*」の『マルチサイト更新』
- 「*DB2 Connect ユーザーズ・ガイド*」の『マルチサイト更新と同期点管理プログラム』

関連タスク:

- 「*DB2 Connect ユーザーズ・ガイド*」の『コントロール・センターを使ったマルチサイト更新の使用可能化』

関連資料:

- 「*管理ガイド: パフォーマンス*」の『spm_log_path - 「同期点マネージャー・ログ・ファイル・パス」構成パラメーター』
- 「*管理ガイド: パフォーマンス*」の『resync_interval - 「トランザクション再同期インターバル」構成パラメーター』
- 「*管理ガイド: パフォーマンス*」の『tm_database - 「トランザクション・マネージャー・データベース名」構成パラメーター』
- 「*管理ガイド: パフォーマンス*」の『tp_mon_name - 「トランザクション・プロセッサー・モニター名」構成パラメーター』
- 「*管理ガイド: パフォーマンス*」の『locktimeout - 「ロック・タイムアウト」構成パラメーター』
- 「*管理ガイド: パフォーマンス*」の『spm_name - 「同期点マネージャー名」構成パラメーター』
- 「*管理ガイド: パフォーマンス*」の『spm_log_file_sz - 「同期点マネージャー・ログ・ファイル・サイズ」構成パラメーター』
- 「*管理ガイド: パフォーマンス*」の『spm_max_resync - 「同期点マネージャー再同期エージェント数の限度」構成パラメーター』

ホスト、AS/400、または iSeries サーバーへのアクセス

手順:

さまざまなデータベース・システムにアクセス (または更新) できるアプリケーションを開発する場合は、次の手順に従ってください。

1. ユーザーのアプリケーションがアクセスするすべてのデータベース・システムでサポートされているプリコンパイル/ BIND オプションおよび SQL ステートメントを使用する。たとえば、ストアード・プロシージャーは必ずしもすべてのプラットフォームでサポートされるわけではありません。

   IBM 製品の場合は、コーディングを始める*前* に、SQL の資料を参照してください。
2. 可能な時点で、アプリケーションが SQLCODE ではなく SQLSTATE をチェックするようにする。

   アプリケーションが DB2 Connect を使用する場合に SQLCODE 値を使用する際は、 DB2 Connect によって提供されるマッピング機能を用いて、異なるデータベース間の SQLCODE 変換をマップするようにする。
3. サポートする計画のホスト、AS/400、または iSeries データベース (DB2 Universal Database for z/OS and OS/390、OS/400、または DB2 Server for VSE & VM) でアプリケーションをテストする。

関連概念:

- 757 ページの『ホストまたは iSeries 環境におけるアプリケーション』

並行トランザクション

以下の節では、並行トランザクションと、それに伴う問題を避ける方法について説明しています。

並行トランザクション

アプリケーションが、*並行トランザクション* と呼ぶ複数の独立した接続を持っていると便利である場合がしばしばあります。トランザクションを使用すると、アプリケーションは一度に複数のデータベースに接続でき、また同じデータベースへの複数の独立した接続を確立することもできます。

マルチスレッドのデータベース・アクセスで使用されるコンテキスト API を組み込むと、アプリケーションで並行トランザクションを使用することができます。アプリケーション内で作成されるコンテキストは、それぞれ他のコンテキストとは独立しています。ということは、他のコンテキストによって実行される COMMIT または ROLLBACK ステートメントなどの活動に影響されずに、新たにコンテキストを作成し、そのコンテキストを使用してデータベースに接続し、データベースに SQL ステートメントを実行できるということです。

たとえば、ユーザーがあるデータベースに対して SQL ステートメントを実行し、同時に別のデータベースで実行している活動のログを保存するというアプリケーションを作成するとしましょう。ログは最新のものでなければなりませんから、ログ

を更新するたびに COMMIT ステートメントを発行することが必要ですが、ログに対して発行したコミットがユーザーの SQL ステートメントに影響することのないようにしたいと思います。 このようなときにこそ、並行トランザクションを使います。アプリケーション内で、次の 2 つのコンテキストを作成します。 1 つのコンテキストはユーザーのデータベースに接続し、ユーザーの SQL すべてに対して使用するもので、別のコンテキストはログ・データベースに接続し、ログの更新に使用します。このように設計することにより、ログ・データベースへの変更をコミットしても、ユーザーの現在の作業単位には影響は及びません。

並行トランザクションのもう 1 つの利点は、ある接続でカーソルに対する処理がロールバックされても、他の接続のカーソルには何の影響もないということです。 1 つの接続でロールバックが行われた後も、他の接続では終了した処理とカーソル位置はそのまま保持されます。

関連概念:

- 193 ページの『マルチスレッド・データベース・アクセスの目的』

並行トランザクションを使用する際に気を付けるべき問題

並行トランザクションを使用するアプリケーションでは、単一接続を使用するアプリケーションを作成する場合には起こりえない問題にいくつか直面することがあります。並行トランザクションを用いたアプリケーションを作成する場合には、以下の注意が必要です。

- 2 つ以上のコンテキスト間でのデータベースの従属関係

  アプリケーション内の各コンテキストには、データベース・オブジェクトに対するロックなど、それぞれ固有のデータベース・リソースがあります。これらの異なるリソース・セットにより、2 つのコンテキストが同じデータベース・オブジェクトにアクセスしている場合、デッドロックを引き起こす可能性があります。データベース・マネージャーはデッドロックを検出し、一方のコンテキストが SQLCODE -911 を受け取ると、その作業単位がロールバックされます。

- 2 つ以上のコンテキスト間におけるアプリケーションの従属関係

  単一のスレッド内で複数のコンテキストを切り換えると、そのコンテキスト間に従属関係が作成されます。コンテキストにデータベースの従属関係もある場合は、デッドロックが起こる可能性があります。一部の従属関係はデータベース・マネージャーの管理範囲外にあるため、デッドロックが検出されず、アプリケーションが中断してしまうこともあります。

  この種の問題の例として、次のアプリケーションを検討してみましょう。

```
context 1
UPDATE TAB1 SET COL = :new_val

context 2
SELECT * FROM TAB1
COMMIT

context 1
COMMIT
```

最初のコンテキスト (context 1) が UPDATE ステートメントを正常に実行したとします。 UPDATE は、TAB1 のすべての行をロックします。次に、コンテキスト 2 (context 2) は TAB1 のすべての行を選択しようとします。 2 つのコンテキストは独立しているので、コンテキスト 2 はコンテキスト 1 がロックを保持する間待機します。しかし、コンテキスト 1 はコンテキスト 2 が実行を終了するまでロックを解放できません。こうして、アプリケーションはデッドロック状態になりますが、データベース・マネージャーはコンテキスト 1 がコンテキスト 2 を待機していることを知らないため、どちらか一方のコンテキストを強制的にロールバックすることはしません。未解決の従属関係のため、アプリケーションは延期状態になってしまいます。

関連概念:

- 696 ページの『並行トランザクションのデッドロックの回避』

並行トランザクションのデッドロックの回避

データベース・マネージャーはコンテキスト間のデッドロックを検出できないため、デッドロックしないように (または少なくともデッドロックを回避できるように) アプリケーションを設計してコーディングしなければなりません。デッドロック状態になる以下の例を考慮してみましょう。

```
context 1
UPDATE TAB1 SET COL = :new_val

context 2
SELECT * FROM TAB1
COMMIT

context 1
COMMIT
```

最初のコンテキスト (context 1) が UPDATE ステートメントを正常に実行したとします。 UPDATE は、TAB1 のすべての行をロックします。次に、コンテキスト 2 (context 2) は TAB1 のすべての行を選択しようとします。 2 つのコンテキストは独立しているので、コンテキスト 2 はコンテキスト 1 がロックを保持する間待機します。しかし、コンテキスト 1 はコンテキスト 2 が実行を終了するまでロックを解放できません。こうして、アプリケーションはデッドロック状態になりますが、データベース・マネージャーはコンテキスト 1 がコンテキスト 2 を待機していることを知らないため、どちらか一方のコンテキストを強制的にロールバックすることはしません。未解決の従属関係のため、アプリケーションは延期状態になってしまいます。

この例では、いくつかの方法でデッドロックを回避することができます。

- コンテキストを切り換える前に保持しているロックをすべて解除する。

  コンテキスト 2 に切り換える前にコンテキスト 1 がコミットを実行するように、コードを変更します。
- 一度に 2 つ以上のコンテキストから同じオブジェクトにアクセスしない。

  同じコンテキストから更新と選択の両方が実行されるように、コードを変更します。
- *locktimeout* データベース構成パラメーターを -1 以外の値に設定する。

-1 以外の値ではデッドロックを防ぐことはできませんが、実行を再開することはできます。コンテキスト 2 は、要求されたロックを獲得できないため、結局はロールバックされます。コンテキスト 2 がロールバックされれば、コンテキスト 1 は実行を継続でき (これによってロックは解除され)、コンテキスト 2 は処理を再試行します。

デッドロックを回避する技法を上記の例を参考にして説明しましたが、この方法は並行トランザクションを使用するすべてのアプリケーションに適用できます。

関連概念:

- 695 ページの『並行トランザクションを使用する際に気を付けるべき問題』

関連資料:

- 「*管理ガイド: パフォーマンス*」の『locktimeout - 「ロック・タイムアウト」構成パラメーター』

セーブポイントとトランザクション

以下の節では、セーブポイントと、セーブポイントを使用してトランザクションを管理する方法について説明しています。

セーブポイントによるトランザクションの管理

アプリケーションのセーブポイントは、トランザクションまたは作業単位において、 SQL ステートメントのサブセットによって実行される作業に対して制御を実施します。アプリケーション内でセーブポイントを設定し、後でそのセーブポイントを解放するか、またはセーブポイントを設定した後で実行された作業をロールバックすることができます。 1 つのトランザクション内で任意の数のセーブポイントを使用することができます。次の例では、1 つのトランザクション内で 2 つのセーブポイントを使用して、アプリケーションの振る舞いを制御しています。

アプリケーションのセーブポイントを使用したオーダーの例:

```
INSERT INTO order ...
INSERT INTO order_item ... lamp

-- set the first savepoint in the transaction
SAVEPOINT before_radio ON ROLLBACK RETAIN® CURSORS
  INSERT INTO order_item ... Radio
  INSERT INTO order_item ... Power Cord
  -- Pseudo-SQL:
  IF SQLSTATE = "No Power Cord"
    ROLLBACK TO SAVEPOINT before_radio
RELEASE SAVEPOINT before_radio

-- set the second savepoint in the transaction
SAVEPOINT before_checkout ON ROLLBACK RETAIN CURSORS
  INSERT INTO order ... Approval
  -- Pseudo-SQL:
  IF SQLSTATE = "No approval"
    ROLLBACK TO SAVEPOINT before_checkout

-- commit the transaction, which releases the savepoint
COMMIT
```

前述の例では、最初のセーブポイントは 2 つのデータ・オブジェクトの間の従属関係を強制します。その従属関係はオブジェクト自体に組み込まれているわけではありません。ラジオと電源コードの間の関係を説明するために参照保全は使用しません。一方が存在して他方が存在しないことがあるからです。しかし、電源コードを付けないでラジオを顧客に出荷することはありませんし、ラジオの電源コードがないからといってトランザクション全体をロールバックし、電気スタンドのオーダーまでキャンセルしようとも思わないでしょう。アプリケーションのセーブポイントは、このオーダーを完了するために必要な事細かな制御を提供します。

ROLLBACK TO SAVEPOINT ステートメントを発行するときに、対応するセーブポイントは自動的に解放されるわけではありません。 RELEASE SAVEPOINT ステートメントを使用してセーブポイントが明示的に解放されるか、トランザクションまたは作業単位を終了することによって暗黙的に解放されるまで、後続の SQL ステートメントがそのセーブポイントに関連付けられます。セーブポイントは、現在のセーブポイント・レベルの最後か、現在のセーブポイントが古くなってしまう、前のアクティブなセーブポイントが解放されるかロールバックされるときまでに、暗黙的に解放することもできます。これは、1 つのセーブポイントに対して複数の ROLLBACK TO SAVEPOINT ステートメントを発行できることを意味します。

セーブポイントによって、複数の COMMIT および ROLLBACK ステートメントを使用するよりもパフォーマンスは向上し、アプリケーション設計の見栄えが良くなります。 COMMIT ステートメントを発行するときに、DB2® は付加的な作業を行って、現行トランザクションをコミットし、新規のトランザクションを開始する必要があります。セーブポイントを使用すると、複数の COMMIT ステートメントの発行という追加の作業を必要とせずに、トランザクションを小さい単位またはステップに分割することができます。次の例は、セーブポイントの代わりに複数のトランザクションを使用することによって発生する、パフォーマンス上の問題点を示します。

複数のトランザクションを使用したオーダーの例:

```
INSERT INTO order ...
INSERT INTO order_item ... lamp
-- commit current transaction, start new transaction
COMMIT

INSERT INTO order_item ... Radio
INSERT INTO order_item ... Power Cord
-- Pseudo-SQL:
IF SQLSTATE = "No Power Cord"
  -- roll back current transaction, start new transaction
  ROLLBACK
ELSE
  -- commit current transaction, start new transaction
  COMMIT

INSERT INTO order ... Approval
-- Pseudo-SQL:
IF SQLSTATE = "No approval"
  -- roll back current transaction, start new transaction
  ROLLBACK
ELSE
  -- commit current transaction, start new transaction
  COMMIT
```

複数のコミット・ポイントを使用する別の問題は、オブジェクトがコミットされている可能性があるため、それが完全に完了する前に他のアプリケーションに見えてしまうことです。 2 番目の例では、すべてのアイテムが追加される前に、またさらに悪いことにオーダーが承認される前に、そのオーダーを別のユーザーが利用できるようになっています。アプリケーションのセーブポイントを使用すると、「不正データ」に対するこのような公開を避けながら、操作に対して事細かな制御を行うことができます。

関連サンプル:

- 『tbsavept.sqc -- How to use external savepoints (C)』

アプリケーションのセーブポイントとコンパウンド SQL ブロックの比較

セーブポイントは、コンパウンド SQL ブロックと比較して、次の利点を提供しています。
- トランザクションの制御の拡張
- ロック競合の減少化
- アプリケーション・ロジックとの統合の向上

コンパウンド SQL ブロックは ATOMIC の場合と NOT ATOMIC の場合があります。 ATOMIC コンパウンド SQL ブロック内のステートメントが失敗すると、コンパウンド SQL ブロック全体がロールバックされます。 NOT ATOMIC コンパウンド SQL ブロック内のステートメントが失敗すると、トランザクションのコミットまたはロールバック (コンパウンド SQL ブロック全体を含む) がアプリケーションによって制御されます。それと比較して、セーブポイントの有効範囲内の 1 つのステートメントが失敗すると、アプリケーションはセーブポイントの有効範囲内のすべてのステートメントをロールバックできます。しかし、セーブポイントの有効範囲外にあるステートメントによって実行された作業はコミットできません。このオプションは、次の例で示されています。セーブポイントの作業がロールバックされると、セーブポイントの前に 2 つの INSERT ステートメントの作業がコミットされます。あるいは、アプリケーションは、セーブポイントの有効範囲内にあるステートメントを含む、トランザクション内のすべてのステートメントによって実行された作業をコミットできます。

アプリケーションのセーブポイントを使用したオーダーの例:

```
INSERT INTO order ...
INSERT INTO order_item ... lamp

-- set the first savepoint in the transaction
SAVEPOINT before_radio ON ROLLBACK RETAIN® CURSORS
  INSERT INTO order_item ... Radio
  INSERT INTO order_item ... Power Cord
  -- Pseudo-SQL:
  IF SQLSTATE = "No Power Cord"
    ROLLBACK TO SAVEPOINT before_radio
RELEASE SAVEPOINT before_radio

-- set the second savepoint in the transaction
SAVEPOINT before_checkout ON ROLLBACK RETAIN CURSORS
  INSERT INTO order ... Approval
  -- Pseudo-SQL:
  IF SQLSTATE = "No approval"
```

```
    ROLLBACK TO SAVEPOINT before_checkout

-- commit the transaction, which releases the savepoint
COMMIT
```

コンパウンド SQL ブロックを発行すると同時に、DB2® はステートメントのコンパウンド SQL ブロック全体に必要なロックを獲得します。アプリケーションのセーブポイントを設定すると、セーブポイントの有効範囲内のステートメントが発行されるたびに、 DB2 はロックを獲得します。セーブポイントのロック動作は、コンパウンド SQL ブロックの場合よりもロック競合の大幅な減少につながります。そのため、アプリケーションがコンパウンド SQL ステートメントによって実行されたロックを必要としない場合は、セーブポイントを使用するのが最善です。

コンパウンド SQL ブロックはステートメントの完全セットを単一のステートメントとして実行します。アプリケーションは、ステートメントをコンパウンド SQL ブロックに追加するために、制御構造または関数を使用することはできません。それと比較して、アプリケーションのセーブポイントを設定すると、アプリケーションは、他のアプリケーション関数またはメソッドを呼び出すことにより、 while ループなどの制御構造を介して、または動的 SQL ステートメントを使用して、セーブポイントの有効範囲内の SQL ステートメントを発行することができます。アプリケーションのセーブポイントは、 SQL ステートメントをアプリケーション・ロジックと直観的な方法で自由に統合できるようにします。

たとえば、次の例では、アプリケーションはセーブポイントを設定し、セーブポイントの有効範囲内で 2 つの INSERT ステートメントを発行します。アプリケーションは IF ステートメントを使用し、それが真であれば関数 add_batteries() を呼び出します。 add_batteries() 関数は、このコンテキストでセーブポイントの有効範囲に含まれている SQL ステートメントを発行します。最後に、アプリケーションは、セーブポイント内で実行された作業 (add_batteries() 関数によって発行された SQL ステートメントを含む) をロールバックするか、またはトランザクション全体で実行された作業をコミットします。

セーブポイントと SQL ステートメントをアプリケーション・ロジック内で統合する例:

```
void add_batteries()
{
  -- the work performed by the following statement
  -- is controlled by the savepoint set in main()
  INSERT INTO order_item ... Batteries
}

void main(int argc, char[] *argv)
{
  INSERT INTO order ...
  INSERT INTO order_item ... lamp

  -- set the first savepoint in the transaction
  SAVEPOINT before_radio ON ROLLBACK RETAIN CURSORS
    INSERT INTO order_item ... Radio
    INSERT INTO order_item ... Power Cord

    if (strcmp(Radio..power_source(), "AC/DC"))
    {
      add_batteries();
    }
```

```
    -- Pseudo-SQL:
    IF SQLSTATE = "No Power Cord"
      ROLLBACK TO SAVEPOINT before_radio
   COMMIT
}
```

セーブポイントの作成や制御に使用する SQL ステートメント

次の SQL ステートメントでセーブポイントを作成および制御することができます。

SAVEPOINT

セーブポイントを設定するには、 SAVEPOINT SQL ステートメントを発行します。ネストされたセーブポイントの特定のセーブポイントを識別し、コードをより鮮明なものにするため、セーブポイントのために分かりやすい名前を選ぶことができます。これらの名前は、セーブポイント参照として知られています。以下に例を示します。

```
SAVEPOINT before_sales ON ROLLBACK RETAIN CURSORS
```

RELEASE SAVEPOINT

セーブポイントを解放するには、 RELEASE SAVEPOINT SQL ステートメントを発行します。以下に例を示します。

```
RELEASE SAVEPOINT before_sales
```

RELEASE SAVEPOINT SQL ステートメントを使用してセーブポイントを明示的に解放しない場合には、現在のセーブポイント・レベルの終わりで解放されます。

ROLLBACK TO SAVEPOINT

セーブポイントへロールバックするには、 ROLLBACK TO SAVEPOINT という SQL ステートメントを発行します。以下に例を示します。

```
ROLLBACK TO SAVEPOINT before_sales
```

セーブポイント・レベルとは、セーブポイント関連ステートメントの参照範囲を指します。セーブポイント・レベルを開始する際は、新規セーブポイント・レベル外で作成されたセーブポイントを参照できるセーブポイント関連ステートメントはありません。同様に、セーブポイント参照は、現在のセーブポイント・レベル内で解決されるもので、現在のセーブポイント・レベル外のセーブポイント参照を考慮に入れることはありません。

新規セーブポイント・レベルは、以下のいずれかが生じる場合にのみ開始されます。

- 新規作業単位が開始される場合
- ストアード・プロシージャーの定義に NEW SAVEPOINT LEVEL 文節が含まれる場合
- アトミックなコンパウンド SQL ステートメントが開始される場合

セーブポイント・レベルの作成が終了または除去されるイベントが生じる場合、セーブポイント・レベルは終了します。

セーブポイント・レベルの範囲内のアクションには、以下の規則が適用されます。

- セーブポイントは、セーブポイントが確立されたセーブポイント・レベル内でのみ参照可能です。現在のセーブポイント・レベル外で確立されたセーブポイントに解放またはロールバックすることはできません。
- 現在のセーブポイント・レベル内で確立されたすべてのアクティブなセーブポイントは、セーブポイント・レベルが終了すると、自動的に解放されます。
- セーブポイント・ユニークな名前は、現在のセーブポイント・レベル内でのみ実施されます。周辺のセーブポイント・レベルでアクティブなセーブポイントの名前は、これらのセーブポイントに影響を与えずに、現在のセーブポイント・レベルで再利用できます。

関連資料:

- 「*SQL リファレンス 第 2 巻*」の『ROLLBACK ステートメント』
- 「*SQL リファレンス 第 2 巻*」の『RELEASE SAVEPOINT ステートメント』
- 「*SQL リファレンス 第 2 巻*」の『SAVEPOINT ステートメント』

関連サンプル:

- 『tbsavept.sqc -- How to use external savepoints (C)』

セーブポイントの使用に関する制約事項

DB2® Universal Database では、アプリケーションでのセーブポイントの使用に関して次のような制約があります。

トリガー

DB2 では、トリガーの定義本体内でのセーブポイント関連 SQL ステートメントはサポートされていません。しかし、トリガーを使用して、セーブポイントを含むストアード・プロシージャーを呼び出すことはできます。これらのストアード・プロシージャーは、呼び出し時に開始する新規セーブポイント・レベルとして定義する必要があります (このことは、 CREATE PROCEDURE ステートメントの NEW SAVEPOINT LEVEL 文節で指定されます)。

SET INTEGRITY ステートメント

セーブポイント内では、DB2 は SET INTEGRITY ステートメントを DDL ステートメントとして扱います。

関連概念:

- 702 ページの『セーブポイントおよびデータ定義言語 (DDL)』

セーブポイントおよびデータ定義言語 (DDL)

DB2® はセーブポイント内に DDL ステートメントを組み込むことを可能にします。 DDL ステートメントを実行するセーブポイントをアプリケーションが正常に解放すると、そのアプリケーションは DDL によって作成された SQL オブジェクトを継続的に使用することができます。ただし、DDL ステートメントを実行するセーブポイントに対してアプリケーションが ROLLBACK TO SAVEPOINT ステートメントを発行すると、 DB2 はこれらの DDL ステートメントの効力に依存するカーソルをすべて無効とマーク付けします。

次の例では、アプリケーションは ROLLBACK TO SAVEPOINT ステートメントを発行した後に、以前に開かれた 3 つのカーソルからフェッチすることを試みます。

```
SAVEPOINT savepoint_name;
   PREPARE s1 FROM 'SELECT FROM t1';
   --issue DDL statement for t1
      ALTER TABLE t1 ADD COLUMN...
   PREPARE s2 FROM 'SELECT FROM t2';
   --issue DDL statement for t3
      ALTER TABLE t3 ADD COLUMN...
   PREPARE s3 FROM 'SELECT FROM t3';
   OPEN c1 USING s1;
   OPEN c2 USING s2;
   OPEN c3 USING s3;
ROLLBACK TO SAVEPOINT
FETCH c1; --invalid (SQLCODE -910)
FETCH c2; --successful
FETCH c3; --invalid (SQLCODE -910)
```

ROLLBACK TO SAVEPOINT ステートメントでは、 DB2 は “c1” および “c3” カーソルを無効としてマーク付けします。なぜなら、それらのカーソルが依存していた SQL オブジェクトがセーブポイント内の DDL ステートメントによって操作されたからです。しかし、この例の “c2” カーソルを使用した FETCH は、ROLLBACK TO SAVEPOINT ステートメントの後でも正常に実行されます。

無効なカーソルをクローズするには、 CLOSE ステートメントを発行します。無効なカーソルに対して FETCH を発行すると、 DB2 によって SQLCODE -910 が戻されます。無効なカーソルに対して OPEN ステートメントを発行すると、 DB2 によって SQLCODE -502 が戻されます。無効なカーソルに対して UPDATE または DELETE WHERE CURRENT OF ステートメントを発行すると、 DB2 によって SQLCODE -910 が戻されます。

セーブポイント内では、 DB2 は NOT LOGGED INITIALLY 特性を持つ表と一時表を次のように扱います。

NOT LOGGED INITIALLY 表
NOT LOGGED INITIALLY プロパティーを使用した表を作成するか、セーブポイント内で表を変更して、 NOT LOGGED INITIALLY プロパティーを指定する場合、DB2 は、 ROLLBACK TO SAVEPOINT ステートメントを ROLLBACK WORK ステートメントとして扱い、作業単位全体をロールバックします。

セーブポイント内での DECLARE TEMPORARY TABLE
セーブポイント内で一時表が宣言されると、 ROLLBACK TO SAVEPOINT ステートメントによって一時表がドロップされます。

セーブポイント外での DECLARE TEMPORARY TABLE
セーブポイント外で一時表が宣言されると、 ROLLBACK TO SAVEPOINT ステートメントは一時表をドロップしません。一時表がログ済みと宣言される場合 (DECLARE GLOBAL TEMPORARY TABLE ステートメントで NOT LOGGED 文節が使用されない場合)、セーブポイント内で表に加えた変更がロールバックされます。一時表が NOT LOGGED と宣言され、表内のデータがセーブポイント内で変更されている場合、すべての行が削除されます。そうではない場合、データが変更されていない場合には、すべての行が保持されます。

セーブポイントのネスト

DB2® では、ネストされたセーブポイントがサポートされていて、セーブポイントを他のセーブポイントの中にセットアップできます。セーブポイントはいくつでも設定でき、ネストされたセーブポイントの段階数にも制限はありません。

次の例は、ネストされたセーブポイントの使用法を示しています。

```
CREATE TABLE Department (deptno CHAR(6), deptname VARCHAR(20), mgrno INT)
INSERT INTO Department VALUES ('A20', 'Marketing', 301)
SAVEPOINT savepoint1 ON ROLLBACK RETAIN® CURSORS
INSERT INTO Department VALUES ('B30', 'Finance', 520)
SAVEPOINT savepoint2 ON ROLLBACK RETAIN CURSORS
INSERT INTO Department VALUES ('C40', 'IT Support', 430)
SAVEPOINT savepoint3 ON ROLLBACK RETAIN CURSORS
INSERT INTO Department VALUES ('R50', 'Research', 150)
ROLLBACK TO SAVEPOINT savepoint3
ROLLBACK TO SAVEPOINT savepoint1
```

この例には、2 段階のネストがあります。 savepoint3 が savepoint2 の中にネストされ、 savepoint2 が savepoint1 の中にネストされています。 savepoint3 で、部門 'R50' の追加が行われれば、合計で 4 行がこの例の Department 表に追加されたことになります。 savepoint3 ステートメントに対して ROLLBACK を発行すると、 savepoint3 の追加のみがロールバックされ、3 行は Department 表内に残ります。 savepoint1 ステートメントに対して ROLLBACK を発行すると、savepoint2、および savepoint1 内のすべての処理がロールバックされ、 1 行が Department 表内に残ります。

セーブポイントおよびバッファー化挿入

DB2® アプリケーションのパフォーマンスを向上させるには、 INSERT BUF オプションを使用してバインドを行うことによって、アプリケーションでバッファー化挿入を使用できます。アプリケーションがバッファー化挿入およびセーブポイントの両方を利用する場合、 DB2 は SAVEPOINT、RELEASE SAVEPOINT、または ROLLBACK TO SAVEPOINT ステートメントを実行する前にバッファーをフラッシュします。

関連概念:

- 717 ページの『パーティション・データベース環境におけるバッファー化挿入』

関連資料:

- 「コマンド・リファレンス」の『BIND コマンド』
- 「コマンド・リファレンス」の『PRECOMPILE コマンド』

セーブポイントとカーソル・ブロック化

アプリケーションがセーブポイントを使用する場合、BLOCKING NO というプリコンパイル・オプションを指定してアプリケーションをプリコンパイルまたはバインドすることにより、カーソル・ブロック化を防止することを考慮してください。カーソルをブロック化すると、複数の行が事前に取り出されるのでアプリケーションのパフォーマンスは向上しますが、セーブポイントおよびブロック化カーソルを使用するアプリケーションによって戻されたデータは、データベースにコミットされたデータを反映しない場合があります。

BLOCKING NO を使用してアプリケーションをプリコンパイルしないで、ROLLBACK TO SAVEPOINT が発生した後に FETCH ステートメントを発行する場合には、FETCH ステートメントは削除されたデータを取り出すかもしれません。たとえば、次のような SQL を含んだアプリケーションが BLOCKING NO オプションなしでプリコンパイルされたとします。

```
CREATE TABLE t1(c1 INTEGER);
DECLARE CURSOR c1 AS 'SELECT c1 FROM t1 ORDER BY c1';
INSERT INTO t1 VALUES (1);
SAVEPOINT showFetchDelete;
   INSERT INTO t1 VALUES (2);
   INSERT INTO t1 VALUES (3);
   OPEN CURSOR c1;
   FETCH c1; --get first value and cursor block
   ALTER TABLE t1... --add constraint
ROLLBACK TO SAVEPOINT;
FETCH c1; --retrieves second value from cursor block
```

アプリケーションが “t1” 表に対して最初の FETCH を発行すると、 DB2® サーバーは列値 (1、2、および 3) のブロックをクライアント・アプリケーションに送ります。これらの列値はクライアントによってローカルで保管されます。アプリケーションが ROLLBACK TO SAVEPOINT SQL ステートメントを発行すると、列値 '2' および '3' が表から削除されます。 ROLLBACK TO SAVEPOINT ステートメント後は、表で次の FETCH が行われると、表にその値がないとしても、列値 '2' が戻されます。 アプリケーションはカーソル・ブロック化オプションを利用してパフォーマンスを向上させ、ローカルに保管したデータにアクセスするために、この値を受け取ります。

関連資料:

- 「コマンド・リファレンス」の『BIND コマンド』
- 「コマンド・リファレンス」の『PRECOMPILE コマンド』

セーブポイントおよび XA に準拠したトランザクション・マネージャー

XA に準拠するトランザクション・マネージャーが XA_END 要求を発行する際にアプリケーションでアクティブなセーブポイントが存在する場合には、 DB2® は RELEASE SAVEPOINT ステートメントを発行します。

X/Open XA インターフェース・プログラミングに関する考慮事項

X/Open XA インターフェースは、複数のリソースの変更を調整すると同時にこれらの変更の保全性を確保するさいの標準とされています。 トランザクション・プロセス・モニター として知られるソフトウェア製品は一般に XA インターフェースを使用し、DB2 はこのインターフェースをサポートするので、そのような環境ではリソースとして 1 つ以上の DB2 データベースに同時にアクセスできます。

TP モニターから独立して実行されているアプリケーションと異なるモデルがトランザクション・プロセスに使用されるため、XA インターフェースを使用する分散トランザクション処理 (DTP) 環境で操作を行う場合、DB2 には特別な考慮が必要となります。このトランザクション・プロセスのモデルの特性を次に示します。

- 複数の種類のリカバリー可能なリソース (DB2 データベースなど) をトランザクション内で変更できます。
- 実行されているトランザクションの保全性を確保するため、2 フェーズ・コミットを使用してリソースが更新されます。
- アプリケーション・プログラムは、トランザクションのコミットまたはロールバックの要求を、リソースの管理プログラムではなく TP モニター製品に送ります。 たとえば、CICS® 環境では、アプリケーションは `EXEC CICS SYNCPOINT` を発行してトランザクションをコミットするので、`EXEC SQL COMMIT` を DB2 に発行することは無効かつ不要です。
- トランザクションの実行許可は、TP モニターと関連するソフトウェアにより事前に選別されるため、DB2 などのリソース管理プログラムは TP モニターを 1 つの許可ユーザーと見なします。たとえば、CICS トランザクションの使用には必ず CICS による確認が必要であり、データベースへアクセスする特権は、CICS アプリケーションを呼び出すエンド・ユーザーではなく CICS に付与されなければなりません。
- 複数のプログラム (トランザクション) は通常待機させられ、データベース・サーバーで実行されます (DB2 では、単一で、長時間にわたり実行されるアプリケーション・プログラムとなります)。

この環境のユニークな性質により、ここで実行するようにコーディングされたアプリケーションに対する DB2 の動作および要件は、特殊なものとなります。

- 分散作業単位のプリコンパイラー・オプションまたはクライアントの設定をしなくても、複数のデータベースを作業単位内で更新および接続することができる。
- DISCONNECT ステートメントは許可されないため、使用した場合は SQLCODE -30090 (SQLSTATE 25000) が戻されて拒否される。
- RELEASE ステートメントはサポートされておらず、使用した場合は -30090 が戻されて拒否される。
- COMMIT および ROLLBACK ステートメントは、TP モニター・トランザクションがアクセスするストアード・プロシージャー内では使用できない。
- 2 フェーズ・コミットの流れがトランザクションでは明示的に無効の場合 (これらは、XA インターフェースの専門用語で *LOCAL* トランザクションと呼ばれます)、このトランザクション内でアクセスできるデータベースは 1 つだけである。このデータベースは、SNA 接続を使用してアクセスするホスト、AS/400®、または iSeries データベースにすることはできません。 TCP/IP 接続を使用する DB2® (OS/390® 版) バージョン 5 へのローカル・トランザクションがサポートされています。
- LOCAL トランザクションは、各トランザクションの最後に SQL COMMIT または SQL ROLLBACK を出す。出さない場合は、そのトランザクションは次に処理されるトランザクションの一部と判断されてしまいます。
- 現行データベース接続間の交換は、SQL CONNECT または SQL SET CONNECTION を使用することにより行われる。接続に使用される許可は、CONNECT ステートメントのパスワードまたはユーザー ID を指定しても変更できません。
- 表、ビュー、または索引などのデータベース・オブジェクトが動的 SQL ステートメントにおいて完全には修飾されない場合、ユーザー ID ではなく TP モニターが動作している場合の確認 ID で暗黙的に修飾される。

- LOCAL ではないトランザクションに対する DB2 COMMIT または ROLLBACK ステートメントは拒否される。次のコードが戻されます。
  - 静的 COMMIT では SQLCODE -925 (SQLSTATE 2D521)
  - 静的 ROLLBACK では SQLCODE -926 (SQLSTATE 2D521)
  - 動的 COMMIT では SQLCODE -426 (SQLSTATE 2D528)
  - 動的 ROLLBACK では SQLCODE -427 (SQLSTATE 2D529)
- COMMIT または ROLLBACK への CLI 要求もまた拒否される。
- データベースにより開始されたロールバックの処理

  DTP 環境において、RM がグローバル・トランザクションの自らのブランチを終了させるためにロールバック (システム・エラーまたはデッドロックなどのため) を開始した場合、トランザクション・マネージャーが同期点要求を開始するまで、同アプリケーションのプロセスからの要求をそれ以上処理させないでください。これには、ストアード・プロシージャー中で発生したデッドロックも含まれます。データベース・マネージャーでは、CICS 環境で CICS SYNCPOINT ROLLBACK コマンドを使用するようなトランザクション・マネージャーの同期点サービスを利用して、グローバル・トランザクションをロールバックしなければならないことを通知する SQLCODE -918 (SQLSTATE 51021) を戻し、後に続く SQL 要求をすべて拒否することを意味します。何らかの理由で TM がトランザクションをコミットするように要求すると、RM は TM にそのロールバックについて通知し、TM が他の RM をロールバックするようにします。
- WITH HOLD と宣言されたカーソル

  WITH HOLD と宣言されたカーソルは、CICS トランザクション・プロセス・モニターの XA/DTP 環境でサポートされます。

  WITH HOLD と宣言されたカーソルがサポートされない場合は OPEN ステートメントが拒否され、SQLCODE -30090 (SQLSTATE 25000)、理由コード 03 が戻されます。

  トランザクションは、WITH HOLD と指定されるカーソルがもう必要なくなった場合に、明示的にクローズされるようにしなければなりません。クローズされない場合は他のトランザクションにより継承され、リソースの不必要な使用または衝突を引き起こす結果となります。

  TP モニターが WITH HOLD カーソルをサポートしている場合は、`xa_commit`、`xa_rollback`、および `xa_prepare` をグローバル・トランザクションとして同じ接続で発行する必要があります。
- データベースを更新または変更するステートメントは、2 フェーズ・コミット要求の流れをサポートしないデータベースに対しては実行できない。たとえば、DRDA® プロトコル (DRDA2) のレベル 2 の環境での、ホスト、AS/400、または iSeries データベース・サーバーのアクセスはサポートされていません。
- XA 環境でデータベースが更新をサポートするかどうかは、CONNECT ステートメントを出すことによりランタイムに決定される。データベースが更新可能の場合、3 番目の SQLERRD トークンの値は 1 です。その他の場合、このトークンの値は 2 となります。
- 更新が制限される場合に使用が認められる SQL ステートメントは、以下のものだけである。

```
CONNECT
DECLARE
DESCRIBE
EXECUTE IMMEDIATE (where the first token or keyword is SET but
                   not SET INTEGRITY)
OPEN CURSOR
FETCH CURSOR
CLOSE CURSOR
PREPARE (where the first token or keyword that is not blank or
         left parenthesis is SET (other than SET INTEGRITY),
         SELECT, WITH, or VALUES)
SELECT...INTO
VALUES...INTO
```

他の処理はすべて SQLCODE -30090 (SQLSTATE 25000) が戻されて拒否されます。

PREPARE ステートメントは、SELECT ステートメントの作成時にのみ使用可能です。 EXECUTE IMMEDIATE ステートメントは、DB2 Universal Database for z/OS and OS/390 からの SET SQLID ステートメントのように、出力値を戻さない SQL SET ステートメントの実行が可能です。

- API 制限:

  データベースにおいて内部的にコミットを出し、2 フェーズ・コミット・プロセスを回避する API は、SQLCODE -30090 (SQLSTATE 25000) が戻されて拒否されます。これらの API のリストについては、マルチサイト更新アプリケーションでの制限に関するトピックを参照してください。これらの API はマルチサイト更新ではサポートされません (Connect Type 2)。
- DB2 はマルチスレッドの XA/DTP 環境をサポートしています。

上記の制限は、XA インターフェースを使用した TP モニター環境で実行されるアプリケーションに適用されることに注意してください。 DB2 データベースが XA インターフェースによる使用を定義されていない場合、これらの制限は適用されませんが、DB2 が次のトランザクションの実行に悪影響を与えることのないようにトランザクションをコーディングする必要があることはいうまでもありません。

関連概念:

- 「*管理ガイド: プランニング*」の『XA トランザクション・マネージャーのセキュリティーに関する考慮事項』
- 「*管理ガイド: プランニング*」の『XA トランザクション・マネージャーの構成に関する考慮事項』
- 「*管理ガイド: プランニング*」の『DB2 Universal Database によってサポートされる XA 機能』
- 768 ページの『DB2 Connect によるマルチサイト更新』

関連タスク:

- 「*管理ガイド: プランニング*」の『XA 準拠のトランザクション・マネージャーを使用したホストまたは iSeries データベース・サーバーの更新』

アプリケーション・リンケージと X/Open XA インターフェース

実行可能なアプリケーションを作成するには、アプリケーション・オブジェクトを、言語ライブラリー、オペレーティング・システム・ライブラリー、通常のデータベース・マネージャー・ライブラリー、および TP モニターとトランザクション・マネージャー製品のライブラリーとリンクする必要があります。

MTS および COM+ トランザクション管理

トランザクション・マネージャーとしての Microsoft Transaction Server (MTS) および Microsoft Component Services (COM+)

DB2® UDB は Microsoft® Transaction Server (MTS) バージョン 2.0 (Windows® NT の場合) または Microsoft Component Services (COM+) (Windows 2000 および Windows XP の場合) と完全に統合され、複数の DB2 UDB、zSeries®、および iSeries™ データベース・サーバー、さらに MTS や COM+ の仕様に準拠する他のリソース・マネージャーとの間で、 2 フェーズ・コミットを整合できるようになりました。

前提条件:

MTS または COM+ の分散トランザクション・サポートを使用するには、 DB2 クライアントがインストールされている Windows マシンが以下の要件を満たしている必要があります。

- Windows NT® で MTS バージョン 2.0 を使用する場合: Microsoft Hotfix 0772 以降

  MTS Version 2.0 for Windows NT は Windows NT 4.0 Option Pack の一部として入手できます。 Option Pack は以下のサイトからダウンロードできます。

  ```
  http://www.microsoft.com/ntserver/nts/downloads/recommended/NT4OptPk/
  ```

- Windows 2000: Service Pack 3 以降

MTS または COM+ を使用する DB2 CLI アプリケーションについて:

- SQL_ATTR_CONNECTION_POOLING CLI 環境属性のデフォルト値 (SQL_CP_OFF) を変更しないでください。
- DB2 ODBC ドライバーを Windows オペレーティング・システムにインストールすると、以下の新しいキーワードがレジストリーに自動的に追加されます。

  ```
  HKEY_LOCAL_MACHINE¥software¥ODBC¥odbcinit.ini¥IBM DB2 ODBC Driver:
  Keyword Value Name: CPTimeout
  Data Type: REG_SZ
  Value: 60
  ```

サポートされる DB2 データベース・サーバー:

MTS または COM+ を使用してトランザクションを整合する場合、以下のサーバーがマルチサイト更新用にサポートされます。

- DB2 Universal Database™ Enterprise Server Edition (ESE)

注: MTS または COM+ 用の疎結合グローバル・トランザクションは、超並列処理 (MPP) 環境ではサポートされません。疎結合のグローバル・トランザクションとは、複数のアプリケーション・プロセスがトランザクション・マネージャーの調整下にあるにもかかわらず、個々のアプリケーション・プロセスがあたかも別々のグローバル・トランザクションに属するかのように複数のリソース・マネージャーにアクセスする状況です。個々のアプリケーション・プロセスごとに、リソース・マネージャー内にそれぞれ固有のトランザクション分岐があります。いずれかのアプリケーション・プロセス、トランザクション・マネージャー、またはリソース・マネージャーによってコミットまたはロールバックが要求されると、トランザクション分岐は完了します。分岐間でリソース・デッドロックが発生しないように担当するのは、アプリケーションです。

(複数のアプリケーション・プロセスが 1 つのリソース・マネージャー内の同じトランザクション分岐の下で操作を分担している状況は、密結合グローバル・トランザクションです。これら 2 つのアプリケーション・プロセスは、リソース・マネージャーからは単一のエンティティーと見なされます。リソース・マネージャーは、トランザクション分岐の中でリソースのデッドロックが発生しないようにします。)

- DB2 Universal Database for z/OS™
- DB2 Universal Database for iSeries
- DB2 Server for VSE & VM

インストールおよび構成の考慮事項:

MTS を使用する場合のインストールおよび構成の考慮事項は、以下のとおりです (COM+ はデフォルトで Windows 2000 の一部としてインストールされます)。

- MTS と DB2 クライアントの両方を、 MTS アプリケーションが実行される同じマシン上にインストールします。
- マルチサイト更新でホストまたは iSeries データベース・サーバーを使用する場合、以下のようにします。
  1. ローカル・マシンまたはリモート・マシン上に、DB2 Connect™ 機能 (DB2 Connect Enterprise Edition (EE)、または DB2 Connect 機能がインストールされている DB2 UDB Enterprise Server Edition (ESE) のいずれか) をインストールします。 DB2 Connect 機能によって、ホストまたは iSeries サーバーはマルチサイト更新トランザクションに参加できるようになります。
  2. DB2 Connect サーバーでマルチサイト更新が使用可能になっていることを確認します。

関連概念:

- 「*管理ガイド: プランニング*」の『X/Open 分散トランザクション処理のモデル』
- 269 ページの『MTS および COM+ 分散トランザクションのサポートと IBM OLE DB Provider』
- 712 ページの『Microsoft Transaction Server (MTS) および Microsoft Component Services (COM+) のトランザクション・タイムアウト』
- 711 ページの『Microsoft Component Services (COM+) での疎結合サポート』
- 713 ページの『Microsoft Transaction Server (MTS) および Microsoft Component Services (COM+) を使用した ODBC および ADO 接続プール』

関連タスク:

- 「*DB2 Connect Enterprise Edition 概説およびインストール*」の『DB2 Connect Enterprise Edition のインストール (Windows)』

関連資料:

- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『SQLSetConnectAttr 関数 (CLI) - 接続属性の設定』
- 「*DB2 Connect ユーザーズ・ガイド*」の『DB2 Connect の製品オファリング』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『接続属性 (CLI) リスト』
- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」の『環境属性 (CLI) リスト』

Microsoft Component Services (COM+) での疎結合サポート

疎結合のグローバル・トランザクションとは、複数のアプリケーション・プロセスがトランザクション・マネージャーの調整下にあるにもかかわらず、個々のアプリケーション・プロセスがあたかも別々のグローバル・トランザクションに属するかのように複数のリソース・マネージャーにアクセスする状況です。個々のアプリケーション・プロセスごとに、リソース・マネージャー内にそれぞれ固有のトランザクション分岐があります。いずれかのアプリケーション・プロセス、トランザクション・マネージャー、またはリソース・マネージャーによってコミットまたはロールバックが要求されると、トランザクション分岐は完了します。分岐間でリソース・デッドロックが発生しないように担当するのは、アプリケーションです。

DB2® Universal Database バージョン 8 は、 COM+ オブジェクトに対して疎結合グローバル・トランザクションをサポートし、ロック・タイムアウトやデッドロックはありません。ただし、以下のような制約事項があります。

- データ定義言語 (DDL) は、単一の分岐で実行されており、他の疎結合トランザクションがアクティブでない場合にのみサポートされます。 DDL を実行している単一分岐がアクティブ状態のとき、他の疎結合分岐を開始しようとすると、その疎結合分岐は拒否されます。反対に、アクティブな疎結合のトランザクションが少なくとも 1 つ存在する場合、別の分岐において DDL を実行しようとしても拒否されます。
- 疎結合グローバル・トランザクションは、超並列処理 (MPP) 環境ではサポートされません。 MPP 環境ではそれぞれのグローバル・トランザクションが独立して処理され、デッドロックやタイムアウトが発生する可能性があります。
- セーブポイントの処理および SQL ステートメントは、複数の接続の間で連続的に実行されます。
- ある接続において暗黙的にロールバックが実行された場合、疎結合トランザクションに参加しているその他の接続のすべての分岐は、 SQL0998N とともに理由コード 225、サブコード 4「このトランザクションには、ロールバックのみが許されています」を戻します。

関連概念:

- 「*管理ガイド: プランニング*」の『X/Open 分散トランザクション処理のモデル』

- 269 ページの『MTS および COM+ 分散トランザクションのサポートと IBM OLE DB Provider』
- 709 ページの『トランザクション・マネージャーとしての Microsoft Transaction Server (MTS) および Microsoft Component Services (COM+)』
- 712 ページの『Microsoft Transaction Server (MTS) および Microsoft Component Services (COM+) のトランザクション・タイムアウト』
- 713 ページの『Microsoft Transaction Server (MTS) および Microsoft Component Services (COM+) を使用した ODBC および ADO 接続プール』

Microsoft Transaction Server (MTS) および Microsoft Component Services (COM+) のトランザクション・タイムアウト

MTS または COM+ を使用している場合、以下のツールを使用してトランザクション・タイムアウトを設定できます。

- MTS (Microsoft Windows® NT): MTS Explorer ツール
- COM+ (Microsoft Windows 2000 および XP): Component Services (Windows コントロール パネルの管理ツールの下にある)

トランザクションの時間がトランザクション・タイムアウト値 (デフォルトは 60 秒) を超えた場合、 MTS または COM+ は関係するすべてのリソース・マネージャーに対してアボートを非同期的に出し、トランザクション全体がアボートされます。

アボートは、サーバー側では DB2® ロールバック要求として解釈されます。 DB2 for z/OS™ および DB2 for iSeries™ 以外のサーバーでは、データベース・サーバー上のデータ保全性を保証するために、ロールバック要求が接続においてシリアライズされます。 DB2 for z/OS サーバーまたは DB2 for iSeries サーバーの場合、DCS カタログ項目の INTERRUPT_ENABLED オプションを使って接続を定義します。これによって、タイムアウト発生時に DB2 Connect™ サーバーから z/OS または iSeries サーバーへの接続が切断され、 z/OS または iSeries サーバー上でロールバックが強制的に実行されます。

その結果、以下のようになります。

- 接続がアイドル状態の場合、ただちにロールバックが実行されます。
- SQL ステートメントが長時間にわたって処理されている場合、その SQL ステートメントの終了までロールバック要求は実行されません。

関連概念:

- 「*管理ガイド: プランニング*」の『X/Open 分散トランザクション処理のモデル』
- 「*DB2 Connect ユーザーズ・ガイド*」の『DCS ディレクトリーの値』
- 269 ページの『MTS および COM+ 分散トランザクションのサポートと IBM OLE DB Provider』
- 761 ページの『割り込み要求の処理』
- 709 ページの『トランザクション・マネージャーとしての Microsoft Transaction Server (MTS) および Microsoft Component Services (COM+)』

- 711 ページの『Microsoft Component Services (COM+) での疎結合サポート』
- 713 ページの『Microsoft Transaction Server (MTS) および Microsoft Component Services (COM+) を使用した ODBC および ADO 接続プール』

Microsoft Transaction Server (MTS) および Microsoft Component Services (COM+) を使用した ODBC および ADO 接続プール

接続プールを使用すれば、アプリケーションは複数の接続からなるプールの中から 1 つの接続を使用できます。これによって、接続を使用するたびに再確立する必要がなくなります。いったん接続が作成されてプールに入れられると、アプリケーションは完全な接続プロセスを実行しなくても、その接続を再使用できます。アプリケーションがデータ・ソースから切断したときに接続はプールに入れられ、同じ属性の新しい接続で使用されます。

ODBC 接続プール:

接続プールは ODBC 2.x 以来、ODBC Driver Manager 機能に含まれてきました。Microsoft® Data Access Components (MDAC) ダウンロードの一部として利用できる最新の ODBC Driver Manager (バージョン 3.5) では、接続プールの構成の一部が変更され、トランザクション MTS COM+ オブジェクトの ODBC 接続に関する新しい機能が追加されました。

ODBC Driver Manager 3.5 では、接続プールを活動化する前に、 ODBC ドライバーが新しいキーワードをレジストリーに登録しなければなりません。以下のようなキーワードです。

```
Key Name: SOFTWARE¥ODBC¥ODBCINST.INI¥IBM DB2® ODBC DRIVER
Name: CPTimeout
Type: REG_SZ
Data: 60
```

Windows® オペレーティング・システム用の DB2 ODBC ドライバーは接続プールを完全にサポートするので、このキーワードは登録されます。

デフォルト値 60 は、接続が切断される前に 60 秒間プールされることを意味しています。

通信量の多い環境では、物理的な接続および切断の回数が多くなりすぎないよう、CPTimeout 値をより大きくする必要があります。接続や切断が多数発生した場合、システム・メモリーや通信スタック・リソースなどのシステム・リソースが大量に消費されるためです。

さらに、マルチプロセッサー・マシンの同じトランザクションに含まれるオブジェクトの間で同じ接続が使用されるようにするために、「プロセッサーあたりの複数プール」サポートをオフにする必要があります。これを行うには、以下のレジストリー設定値を `odbcpool.reg` というファイルにコピーし、それをプレーン・テキスト・ファイルとして保存した後、 **odbcpool.reg** コマンドを発行します。 Windows オペレーティング・システムでは、このレジストリー設定値がインポートされます。

```
REGEDIT4

[HKEY_LOCAL_MACHINE¥SOFTWARE¥ODBC¥ODBCINST.INI¥ODBC Connection Pooling]
"NumberOfPools"="1"
```

このキーワードを 1 に設定しないと、 MTS や COM+ は同一トランザクション用の複数の接続をさまざまなプールに入れる可能性があり、その場合、同じ接続が再使用されません。

ADO 接続プール:

MTS または COM+ オブジェクトが ADO を使ってデータベースにアクセスする場合、 OLE DB リソース・プールをオフにする必要があります。こうすれば、Microsoft OLE DB Provider for ODBC (MSDASQL) が ODBC 接続プールと競合しなくなります。この機能は ADO 2.0 では初期設定で `OFF` でしたが、 ADO 2.1 では初期設定が `ON` になっています。 OLE DB リソース・プールをオフにするには、以下の行を `oledb.reg` というファイルにコピーし、それをプレーン・テキスト・ファイルとして保存した後、 **oledb.reg** コマンドを発行します。 Windows オペレーティング・システムでは、これらのレジストリー設定値がインポートされます。

```
REGEDIT4

[HKEY_CLASSES_ROOT¥CLSID¥{c8b522cb-5cf3-11ce-ade5-00aa0044773d}]
@="MSDASQL"
"OLEDB_SERVICES"=dword:fffffffc
```

関連概念:

- 「*管理ガイド: プランニング*」の『X/Open 分散トランザクション処理のモデル』
- 269 ページの『MTS および COM+ 分散トランザクションのサポートと IBM OLE DB Provider』
- 709 ページの『トランザクション・マネージャーとしての Microsoft Transaction Server (MTS) および Microsoft Component Services (COM+)』
- 712 ページの『Microsoft Transaction Server (MTS) および Microsoft Component Services (COM+) のトランザクション・タイムアウト』
- 711 ページの『Microsoft Component Services (COM+) での疎結合サポート』

第 31 章 パーティション・データベース環境におけるプログラミング上の考慮事項

パーティション・データベース環境における FOR READ ONLY カーソル 715
指示 DSS およびローカル・バイパス 715
パーティション・データベース環境における指示 DSS およびローカル・バイパス 715
パーティション・データベース環境における指示 DSS 716
パーティション・データベース環境におけるローカル・バイパス. 717
バッファー化挿入 717
パーティション・データベース環境におけるバッファー化挿入 717
バッファー化挿入の使用に関する考慮事項. . . 720
バッファー化挿入の使用に関する制限 722
パーティション・データベース環境における大量データの抽出の例. 723
シミュレートされたパーティション・データベース環境の作成 728
トラブルシューティング. 728
パーティション・データベース環境におけるエラー処理. 728
パーティション・データベース環境における重大エラー. 729
マージされた複数の SQLCA 構造 730
エラーを戻すパーティション 730
ループ状態または延期状態のアプリケーション 731

パーティション・データベース環境における FOR READ ONLY カーソル

読み取り専用のカーソルを宣言する場合は、 FOR READ ONLY または FOR FETCH ONLY を、OPEN CURSOR 宣言の中に含めてください。 (FOR READ ONLY と FOR FETCH ONLY は同等のステートメントです。) FOR READ ONLY カーソルによって、コーディネーター・パーティションが一度に複数の行を取り出すことができるようになり、その後の FETCH ステートメントのパフォーマンスが大幅に向上します。 FOR READ ONLY カーソルを明示的に宣言しない場合は、コーディネーター・パーティションがそれらを更新可能カーソルとして扱います。更新可能カーソルは、コーディネーター・パーティションに 1 回の FETCH につき 1 行だけの取り出しを要求するので、パフォーマンスはかなり低下します。

指示 DSS およびローカル・バイパス

以下の節では、パーティション・データベース環境で指示 DSS およびローカル・バイパスを使用する際の考慮事項について説明しています。

パーティション・データベース環境における指示 DSS およびローカル・バイパス

オンライン・トランザクション処理 (OLTP) アプリケーションを最適化するために、すべてのデータベース・パーティションの処理を要求する単純 SQL ステートメントの使用を避けたい場合があるかもしれません。その場合、アプリケーションを設計して、 SQL ステートメントがただ 1 つのデータベース・パーティションからデータを取り出せるようにします。指示分散サブセクション (DSS) およびローカル・バイパスによるこの技法は、コーディネーター・パーティションが、関連する 1 つまたはすべてのパーティションと通信する際のパフォーマンスの低下を防ぎます。

関連概念:

- 716 ページの『パーティション・データベース環境における指示 DSS』
- 717 ページの『パーティション・データベース環境におけるローカル・バイパス』

パーティション・データベース環境における指示 DSS

分散サブセクション (DSS) は、並列照会に応じた処理を一部必要とするデータベース・パーティションにサブセクションを送信するアクションです。また、DSS は呼び出し固有の値 (OLTP 環境にある変数の値など) を使って、サブセクションの始まりも記述します。*指示 DSS* は、表パーティション・キーを使用して、単一のパーティションを照会するように指示します。このタイプの照会をアプリケーションで使用して、すべてのパーティションに照会をブロードキャストした場合に発生するコーディネーター・パーティションのオーバーヘッドを防ぎます。

指示 DSS の利点を生かした SELECT ステートメントの部分例は、以下のようになります。

```
SELECT ... FROM t1
  WHERE PARTKEY=:hostvar
```

コーディネーター・パーティションは照会を受け取ると、 *:hostvar* に対するデータのサブセットを保持しているデータベース・パーティションを判別して、その特定のデータベース・パーティションに照会するように指示します。

指示 DSS を使ってアプリケーションを最適化するには、複合照会を複数の単純照会に分けてください。たとえば以下の照会では、コーディネーター・パーティションが、パーティション・キーを複数の値と突き合わせます。この照会を満たすデータは複数のデータベース・パーティションにあるため、コーディネーター・パーティションは、照会をすべてのデータベース・パーティションにブロードキャストします。

```
SELECT ... FROM t1
  WHERE PARTKEY IN (:hostvar1, :hostvar2)
```

この代わりに、照会を複数の SELECT ステートメント (各 SELECT ステートメントには 1 つのホスト変数がある) に分けたり、 UNION を指定した 1 つの SELECT ステートメントを使用しても、これと同じ結果を得ることができます。コーディネーター・パーティションはさらに単純な SELECT ステートメントを利用して、指示 DSS を使って必要なデータベース・パーティションとだけ通信します。最適化された照会は次のようになります。

```
SELECT ... AS res1 FROM t1
  WHERE PARTKEY=:hostvar1
  UNION
SELECT ... AS res2 FROM t1
  WHERE PARTKEY=:hostvar2
```

上の手法では、UNION 内 SELECT の数がパーティションの数よりも大幅に少ない場合に限って、パフォーマンスが向上することに注意してください。

パーティション・データベース環境におけるローカル・バイパス

特別な形式の指示 DSS 照会は、コーディネーター・パーティションにあるデータにのみアクセスします。コーディネーター・パーティションは他のパーティションと通信することなく照会を完了するため、この形式は*ローカル・バイパス* と呼ばれます。

ローカル・バイパスは可能な場合には自動的に使用可能になります。しかし、トランザクションに用いるデータがあるデータベース・パーティションにそのトランザクションをルーティングすれば、ローカル・バイパスをより一層活用できます。これを行うための 1 つの手法として、 1 つのリモート・クライアントに各データベース・パーティションとの接続を維持させる方法があります。そうすると、トランザクションは、入力パーティション・キーに基づいた適切な接続を使えるようになります。別の手法として、トランザクションをデータベース・パーティションごとにグループ化して、各データベース・パーティションに対して別個のアプリケーション・サーバーを割り当てる方法もあります。

トランザクション・データのあるデータベース・パーティションの番号を判別するには、 `sqlugrpn` API (行パーティション番号の入手) を使用します。この API を使うと、アプリケーションは、特定のパーティション・キーの行のパーティション番号を効果的に計算できます。

もう 1 つの手法として、 `db2atld` ユーティリティーを使って入力データをパーティション番号に分け、各データベース・パーティションに対してアプリケーションのコピーを実行する方法があります。

関連資料:

- 「*管理 API リファレンス*」の『sqlugrpn - 行パーティション番号の入手』
- 「*コマンド・リファレンス*」の『db2atld - オートローダー・コマンド』

バッファー化挿入

以下の節では、パーティション・データベース環境でバッファー化挿入を使用する際の考慮事項について説明しています。

パーティション・データベース環境におけるバッファー化挿入

バッファー化挿入は、表キューを利用して、挿入する行をバッファーに蓄積することによって、パフォーマンスを大幅に向上させる挿入ステートメントです。バッファー化挿入を使用するには、アプリケーションは INSERT BUF オプションを使用して準備またはバインドされている必要があります。

バッファー化挿入を行うと、挿入を実行するアプリケーションのパフォーマンスを大幅に向上させることができます。一般に、バッファー化挿入を使用できるのは、単一挿入ステートメント (他のデータベース修正ステートメントはない) をループ内で使用して多数の行を挿入し、データのソースが INSERT ステートメントの VALUES 文節にあるアプリケーションにおいてです。通常、INSERT ステートメントは 1 つ以上のホスト変数を参照し、ループを連続して実行するうちにその値を変更します。 VALUES 文節は、単一行または複数行を指定することができます。

一般的な意思決定支援アプリケーションでは、新規データのロードと定期的な挿入が必要となります。このデータは、膨大な数の行になることがあります。表をロードするときに、バッファー化挿入を使用するアプリケーションを準備しバインドすることができます。

アプリケーションがバッファー化挿入を使用するには、PREP コマンドを使用して、アプリケーション・プログラムのソース・ファイルを処理するか、または生成されたバインド・ファイルに対して BIND コマンドを使用します。いずれの場合も、INSERT BUF オプションを指定する必要があります。

注: バッファー化挿入により、以下のステップが生じます。

1. データベース・マネージャーが、表の常駐する各データベース・パーティションにつき、4 KB のバッファーを 1 つオープンします。
2. アプリケーションが INSERT ステートメントを VALUES 文節を指定して発行すると、行 (1 つまたは複数) が該当するバッファー (1 つまたは複数) に置かれます。
3. データベース・マネージャーが制御をアプリケーションに戻します。
4. バッファーがいっぱいになるときにバッファー内の行がパーティションに送信されるか、または部分的に満たされたバッファー内の行が送信されます。部分的に満たされたバッファーは、次の事柄が発生すると、フラッシュされます。
   - アプリケーションが COMMIT (アプリケーションの終了により暗黙または明示的に) または ROLLBACK を発行したとき。
   - アプリケーションが、セーブポイントを取る別のステートメントを発行するとき。 OPEN、FETCH、および CLOSE カーソル・ステートメントは、セーブポイントを取らせず、オープンしているバッファー化挿入のクローズもしません。

     以下の SQL ステートメントは、オープンしているバッファー化挿入をクローズします。
     - BEGIN COMPOUND SQL
     - COMMIT
     - DDL
     - DELETE
     - END COMPOUND SQL
     - EXECUTE IMMEDIATE
     - GRANT
     - 別の表への INSERT
     - データ変更ステートメントの全選択に対する OPEN CURSOR
     - バッファー化挿入を実行するのと名前が同じ動的ステートメントの PREPARE
     - REDISTRIBUTE DATABASE PARTITION GROUP
     - RELEASE SAVEPOINT
     - REORG
     - REVOKE
     - ROLLBACK
     - ROLLBACK TO SAVEPOINT
     - RUNSTATS

– SAVEPOINT
– SELECT INTO
– UPDATE
– 他の任意のステートメントの実行、ただし、バッファー化 INSERT の繰り返しの実行 (ループ) ではない
– アプリケーションの終了

次の API は、オープンしているバッファー化挿入をクローズします。
– BIND (API)
– REBIND (API)
– RUNSTATS (API)
– REORG (API)
– REDISTRIBUTE (API)

これらの状況のいずれにおいても、別のステートメントがバッファー化挿入をクローズすると、すべてのデータベース・パーティションがバッファーを受け取って行が挿入されるのを、コーディネーター・パーティションが待機します。すべての行が正常に挿入されたならば、バッファー化挿入をクローズするステートメントを実行します。

パーティション環境での標準インターフェース (バッファー化挿入を使用しない) は、次のステップを行って一度に 1 行をロードします (アプリケーションがデータベース・パーティションの 1 つでローカルに実行していることを前提とします)。

1. コーディネーター・パーティションが、同じパーティションにあるデータベース・マネージャーに行を渡します。
2. データベース・マネージャーは、行を挿入するデータベース・パーティションを判別するために間接的なハッシュを使用します。
   - ターゲット・パーティションが行を受け取ります。
   - ターゲット・パーティションが行をローカルに挿入します。
   - ターゲット・パーティションが応答をコーディネーター・パーティションに送信します。
3. コーディネーター・パーティションがターゲット・パーティションから応答を受け取ります。
4. コーディネーター・パーティションがアプリケーションに応答を与えます。

   挿入は、アプリケーションが COMMIT を発行するまでコミットされません。
5. VALUES 文節を含む INSERT ステートメントは、行数や行のエレメントのタイプに関係なく、バッファー化挿入の候補となります。つまり、エレメントとして、定数、特殊レジスター、ホスト変数、式、関数などを使用できます。

指定された VALUES 文節のある INSERT ステートメントに対して、DB2® SQL コンパイラーは、セマンティック、パフォーマンス、または実現の考慮事項に基づいて、挿入をバッファー化することはできません。 INSERT BUF オプションを使用してアプリケーションを準備またはバインドする場合は、それがバッファー化挿入に従属していないことを確認してください。つまり、

- エラーは、バッファー化挿入については非同期に、通常の挿入については同期して報告することができます。非同期に報告される場合、挿入エラーはバッファー内の次の挿入に関してか、またはバッファーをクローズする*他の* ステートメント

に関して報告することができます。 エラーを報告するステートメントは実行されません。たとえば、COMMIT ステートメントを使用してバッファー化挿入ループをクローズする場合を考えてみましょう。コミットは、以前の挿入との重複キーに関する SQLCODE -803 (SQLSTATE 23505) を報告します。このシナリオでは、コミットは実行されません。アプリケーションに、たとえば、バッファー化挿入ループに入る前に実行される何らかの更新を実際にコミットさせたい場合、COMMIT ステートメントを再発行する必要があります。

- 挿入される行は、バッファー化挿入を使用せずにカーソルを使用する SELECT ステートメントでは、すぐに表示できます。しかしバッファー化挿入では、行はすぐには表示されません。 INSERT BUF オプションを指定してアプリケーションをプリコンパイルまたはバインドする場合は、これらのカーソル選択行に従属するようにアプリケーションを作成しないでください。

バッファー化挿入によって、次のパフォーマンス上の利点が生じます。

- ターゲット・パーティションが受け取るバッファーごとに 1 つのメッセージのみがターゲット・パーティションからコーディネーター・パーティションに送信されます。
- バッファーには、多数の行が含まれることがあります (特に、行が短い場合)。
- コーディネーター・パーティションが新しい行を受信している間にパーティション間で挿入が行われている場合、並列処理が発生します。

INSERT BUF でバインドするアプリケーションは、バッファー化挿入をクローズするステートメントまたは API が発行される前に、VALUES 文節を持つ同じ INSERT ステートメントが繰り返されるように作成すべきです。

注: バッファー化挿入がトランザクション・ログを一杯にするのを防ぐためには、定期的にコミットを実行する必要があります。

関連概念:

- 65 ページの『ソース・ファイルの作成と準備』
- 73 ページの『BIND コマンドを使用したパッケージの作成』
- 720 ページの『バッファー化挿入の使用に関する考慮事項』
- 722 ページの『バッファー化挿入の使用に関する制限』

バッファー化挿入の使用に関する考慮事項

バッファー化挿入は、アプリケーション・プログラムに影響する可能性のある振る舞いを示します。この振る舞いは、バッファー化挿入の非同期特性により生じます。行のパーティション・キーの値に基づいて、挿入された各行は、正しいパーティションを指すバッファーに入れられます。これらのバッファーは、いっぱいになるか、またはフラッシュを引き起こすイベントが発生すると、宛先パーティションに送信されます。次の事柄に注意して、アプリケーションの設計およびコーディングの際にそれらを考慮に入れる必要があります。

- 行の挿入に関する特定のエラー条件は、INSERT ステートメントの実行時には報告されません。それらは後ほど、INSERT (または別の表への INSERT) 以外のステートメント、たとえば DELETE、UPDATE、COMMIT、または ROLLBACK などのうちの最初のものが実行されるときに報告されます。バッファー化挿入ス

テートメントをクローズするステートメントまたは API は、エラー・レポートを参照できます。また、挿入自体の呼び出しは、それ以前の行の挿入のエラーを参照できます。さらに、バッファー化挿入エラーが別のステートメント、たとえば UPDATE や COMMIT などによって報告されると、 DB2® はそのステートメントの実行をしません。

- *行のグループ* の挿入中に検出されたエラーは、そのグループのすべての行をバックアウトさせます。行のグループとは、バッファー化挿入ステートメントの実行によって挿入されるすべての行のことです。
  - 作業単位の開始から
  - ステートメントが準備されて以降 (それが動的の場合)
  - 別の更新ステートメントの直前の実行以降。バッファー化挿入をクローズ (またはフラッシュ) するステートメントのリストは、パーティション・データベース環境におけるバッファー化挿入についての説明を参照してください。
- 挿入される行は、SELECT をカーソルを使用して実行する場合、同じアプリケーション・プログラムによる INSERT の後で発行される SELECT ステートメントにすぐに表示することはできません。

バッファー化 INSERT ステートメントは、オープンまたはクローズのいずれかの状態になっています。ステートメントの最初の呼び出しで、バッファー化 INSERT がオープンし、行が該当するバッファーに追加され、制御がアプリケーションに戻されます。その後の呼び出しでは、行がバッファーに追加され、ステートメントはオープンしたままにされます。ステートメントがオープンしている間、バッファーがその宛先パーティションに送信されることがあります。その場合、行がターゲット表のパーティションに挿入されます。バッファー化挿入をクローズするステートメントまたは API が、バッファー化 INSERT ステートメントのオープン中に呼び出された場合 (*別の* バッファー化 INSERT ステートメントの呼び出しも含む)、またはオープンしているバッファー化 INSERT ステートメントに対して PREPARE ステートメントが発行された場合、オープン・ステートメントは、新規の要求を処理する前に、クローズします。バッファー化 INSERT ステートメントがクローズすると、残ったバッファーはフラッシュされます。次に、行がターゲット・パーティションに送信され、挿入されます。すべてのバッファーが送信されて、すべての行が挿入された後でのみ、新規の要求が処理を開始します。

INSERT ステートメントのクローズ中にエラーが検出されると、新規要求の SQLCA が、エラーの記述でいっぱいになり、新規要求は実行されません。また、*そのオープン以来*、バッファー化 INSERT ステートメントによって挿入された行のグループ全体が、データベースから除去されます。 アプリケーションの状態は、検出された特定のエラーに定義された状態になります。たとえば、以下のとおりです。

- エラーがデッドロックの場合、トランザクションはロールバックされます (バッファー化挿入セクションのオープン前に行った変更も含む)。
- エラーがユニーク・キー違反の場合、データベースの状態は、ステートメントのオープン前と同じです。トランザクションはアクティブなままで、ステートメントのオープン前に行った変更は影響されません。

たとえば、バッファー化挿入オプションを指定してバインドする、次のアプリケーションを例にして考えてみましょう。

```
EXEC SQL UPDATE t1 SET COMMENT='about to start inserts';
DO UNTIL EOF OR SQLCODE < 0;
  READ VALUE OF hv1 FROM A FILE;
  EXEC SQL INSERT INTO t2 VALUES (:hv1);
  IF 1000 INSERTS DONE, THEN DO
     EXEC SQL INSERT INTO t3 VALUES ('another 1000 done');
     RESET COUNTER;
  END;
END;
EXEC SQL COMMIT;
```

ファイルに 8 000 個の値が含まれているが、値 3 258 は正しくない (たとえば、ユニーク・キー違反) とします。 1 000 個の行を挿入すると、次の SQL ステートメントの実行を引き起こし、 INSERT INTO t2 ステートメントをクローズします。 4 回目に 1 000 個の行の挿入を実行しているときに、値 3 258 のエラーが検出されます。さらに値を挿入した後で (必ずしも次の値とは限らない)、検出されることもあります。この状況では、エラー・コードは INSERT INTO t2 ステートメントに対して戻されます。

また、表 t3 に対して挿入しようとすると、エラーが検出されることもあります。この動作は、INSERT INTO t2 ステートメントをクローズします。 この状況では、エラーは表 t2 に適用されるとしても、エラー・コードは INSERT INTO t3 ステートメントに対して戻されます。

代わりに、3 900 行を挿入するとします。行番号 3 258 に関するエラーが通知される前に、アプリケーションはループを終了して、COMMIT を発行しようとします。ユニーク・キー違反戻りコードが、COMMIT ステートメントに対して発行され、COMMIT は実行されません。アプリケーションがデータベース (すなわち遠端) にある 3 000 行をコミットするには (EXEC SQL INSERT INTO t3 ... の最後の実行は、それら 3 000 行のセーブポイントを終了する)、COMMIT を*再発行*する 必要があります。同様の考慮事項が、ROLLBACK にもあてはまります。

注: バッファー化挿入を使用する際には、戻される SQLCODES を注意深くモニターして、表が未決状態にならないようにすべきです。たとえば、上の例で THEN DO ステートメントから SQLCODE < 0 文節を除去すると、表は行数が定まらないまま終了してしまいます。

関連概念:

- 717 ページの『パーティション・データベース環境におけるバッファー化挿入』

バッファー化挿入の使用に関する制限

バッファー化挿入には以下の制約事項が適用されます。

- バッファー化挿入を利用するアプリケーションは、次のいずれか 1 つが真でなければなりません。
  - アプリケーションは PREP によって準備されているか、または BIND コマンドによってバインドされている必要があり、 INSERT BUF オプションを指定する。
  - アプリケーションは BIND または SQL_INSERT_BUF オプションを指定した PREP API を使用してバインドする必要がある。

- VALUES 文節のある INSERT ステートメントで、長いフィールドまたは LOBS が明示または暗黙の列リストに含まれる場合、そのステートメントでは INSERT BUF オプションが無視されて、バッファー化挿入ではなく通常の挿入操作セクションが実行されます。これは、エラー条件ではないので、エラーまたは警告メッセージは発行されません。
- 全選択を行う INSERT は INSERT BUF によって影響されません。バッファー化挿入は、このタイプの INSERT のパフォーマンスを改善しません。
- バッファー化挿入は、アプリケーションでのみ使用でき、CLP の発行する挿入では使用できません。後者の挿入は、EXECUTE IMMEDIATE ステートメントによって実行されるからです。

アプリケーションは、サポートされる任意のクライアント・プラットフォームから実行することができます。

パーティション・データベース環境における大量データの抽出の例

DB2 Universal Database は、並列照会処理のための優れた機能を提供しますが、アプリケーションまたは EXPORT コマンドの接続の単一点は、大量のデータを抽出する際に、障害になる可能性があります。この障害が生じるのは、データベース・マネージャーからアプリケーションへのデータの受け渡しが、単一パーティション (通常はシングル・プロセッサー) 上で実行される CPU 集中処理であるからです。

DB2 Universal Database は、障害を解決するために、プロセッサーの数を増やして、抽出したデータのボリュームが時間単位に正比例するような、いくつかの手段を提供しています。次の例は、これらの手法の背後にある基本的な考えを説明しています。

EMPLOYEE という表があるとします。この表は、20 個のデータベース・パーティションに保管されています。正規の部門に属する (すなわち、WORKDEPT は NULL ではない) すべての従業員の郵送リスト (FIRSTNME (名)、 LASTNAME (姓)、JOB (担当作業)) を生成するとします。

次の照会は、各パーティションで実行し、その後、単一パーティション (コーディネーター・パーティション) で全体の応答セットを生成します。

```
SELECT FIRSTNME, LASTNAME, JOB FROM EMPLOYEE WHERE WORKDEPT IS NOT NULL
```

しかし、次の照会をデータベース内の各パーティションで実行できます (つまり、パーティションが 5 つあれば、各パーティションに 1 つずつ、合計 5 つの別個の照会が必要となる)。それぞれの照会は、照会を実行した特定のパーティションにレコードがあるすべての従業員名のセットを生成します。それぞれのローカル結果セットは、ファイルにリダイレクトすることができます。その後、結果セットは、単一の結果セットにマージする必要があります。

AIX® では、ネットワーク・ファイル・システム (NFS) ファイルの特性を使用して、そのマージを自動化することができます。すべてのパーティションが、応答セットを NFS マウント上の同じファイルに送信する場合、結果はマージされます。応答を大きなバッファーにブロック化せずに NFS を使用すると、パフォーマンスがかなり低下する原因になることに注意してください。

```
   SELECT FIRSTNME, LASTNAME, JOB FROM EMPLOYEE WHERE WORKDEPT IS NOT NULL
                    AND NODENUMBER(NAME) = CURRENT NODE
```

結果は、ローカル・ファイル (つまり、最終的な結果は 20 個のファイルとなり、それぞれに応答セット全体の一部が含まれる) か、単一の NFS マウント・ファイルのどちらかに保管することができます。

次の例では、2 番目の手法を使用するので、結果は、20 個のノードにわたってマウントされた NFS である単一ファイルに保管されます。 NFS ロック・メカニズムによって、異なるパーティションから結果ファイルへの書き込みが確実にシリアル化されます。この例は、明示されているように、 NFS ファイル・システムをインストールした AIX プラットフォームでのみ実行されることに注意してください。

```
#define _POSIX_SOURCE
#define INCL_32

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>
#include <sqlenv.h>
#include <errno.h>
#include <sys/access.h>
#include <sys/flock.h>
#include <unistd.h>

#define BUF_SIZE 1500000  /* Local buffer to store the fetched records */
#define MAX_RECORD_SIZE 80 /* >= size of one written record */

int main(int argc, char *argv[]) {

    EXEC SQL INCLUDE SQLCA;
    EXEC SQL BEGIN DECLARE SECTION;
       char dbname[10];  /* Database name (argument of the program) */
       char userid[9];
       char passwd[19];
       char first_name[21];
       char last_name[21];
       char job_code[11];
    EXEC SQL END DECLARE SECTION;

       struct flock unlock ;  /* structures and variables for handling */
       struct flock lock ;  /* the NFS locking mechanism */
       int lock_command ;
       int lock_rc ;
       int iFileHandle ;  /* output file */
       int iOpenOptions = 0 ;
       int iPermissions ;
       char * file_buf ;  /* pointer to the buffer where the fetched
                             records are accumulated */
       char * write_ptr ;  /* position where the next record is written */
       int buffer_len = 0 ;  /* length of used portion of the buffer */

    /* Initialization */

       lock.l_type = F_WRLCK;  /* An exclusive write lock request */
       lock.l_start = 0;  /* To lock the entire file */
       lock.l_whence = SEEK_SET;
       lock.l_len = 0;
       unlock.l_type = F_UNLCK;  /* An release lock request */
       unlock.l_start = 0;  /* To unlock the entire file */
       unlock.l_whence = SEEK_SET;
       unlock.l_len = 0;
       lock_command = F_SETLKW;  /* Set the lock */
```

```
    iOpenOptions = O_CREAT;  /* Create the file if not exist */
    iOpenOptions |= O_WRONLY;  /* Open for writing only */

  /* Connect to the database */

    if (argc == 3) {
       strcpy( dbname, argv[2] ); /* get database name from the argument */
       EXEC SQL CONNECT TO :dbname IN SHARE MODE ;
       if ( SQLCODE != 0 ) {
          printf( "Error: CONNECT TO the database failed. SQLCODE = %ld¥n",
            SQLCODE );
    exit(1);
 }
    }
    else if ( argc == 5 ) {
      strcpy( dbname, argv[2] ); /* get  database name from the argument */
strcpy (userid, argv[3]);
strcpy (passwd, argv[4]);
EXEC SQL CONNECT TO :dbname IN SHARE MODE USER :userid USING :passwd;
if ( SQLCODE != 0 ) {
   printf( "Error: CONNECT TO the database failed. SQLCODE = %ld¥n",
           SQLCODE );
    exit( 1 );
      }
    }
    else {
  printf ("¥nUSAGE: largevol txt_file database [userid passwd]¥n¥n");
  exit( 1 ) ;
    } /* endif */

   /* Open the input file with the specified access permissions */

   if ( ( iFileHandle = open(argv[1], iOpenOptions, 0666 ) ) == -1 ) {
     printf( "Error: Could not open %s.¥n", argv[2] ) ;
     exit( 2 ) ;
  }

  /* Set up error and end of table escapes */

  EXEC SQL WHENEVER SQLERROR GO TO ext ;
  EXEC SQL WHENEVER NOT FOUND GO TO cls ;

  /* Declare and open the cursor */

  EXEC SQL DECLARE c1 CURSOR FOR
           SELECT firstnme, lastname, job FROM employee
           WHERE workdept IS NOT NULL
           AND NODENUMBER(lastname) = CURRENT NODE;
  EXEC SQL OPEN c1 ;

  /* Set up the temporary buffer for storing the fetched result */

  if ( ( file_buf = ( char * ) malloc( BUF_SIZE ) ) == NULL ) {
     printf( "Error: Allocation of buffer failed.¥n" ) ;
     exit( 3 ) ;
  }
  memset( file_buf, 0, BUF_SIZE ) ; /* reset the buffer */
  buffer_len = 0 ;  /* reset the buffer length */
  write_ptr = file_buf ;  /* reset the write pointer */
  /* For each fetched record perform the following    */
  /*  - insert it into the buffer following the       */
  /*    previously stored record                      */
  /*  - check if there is still enough space in  the  */
  /*    buffer for the next record and lock/write/    */
  /*    unlock the file and initialize the buffer     */
  /*    if not                                        */
```

```
    do {
       EXEC SQL FETCH c1 INTO :first_name, :last_name, :job_code;
        buffer_len += sprintf( write_ptr, "%s %s %s¥n",
                               first_name, last_name, job_code );
        buffer_len = strlen( file_buf ) ;
       /* Write the content of the buffer to the file if */
       /* the buffer reaches the limit                   */
       if ( buffer_len >= ( BUF_SIZE - MAX_RECORD_SIZE ) ) {
        /*  get excl. write lock */
        lock_rc = fcntl( iFileHandle, lock_command, &lock );
   if ( lock_rc != 0 ) goto file_lock_err;
   /*  position at the end of file */
   lock_rc = lseek( iFileHandle, 0, SEEK_END );
  if ( lock_rc < 0 ) goto file_seek_err;
  /* write the buffer */
  lock_rc = write( iFileHandle,
                               ( void * ) file_buf, buffer_len );
  if ( lock_rc < 0 ) goto file_write_err;
    /* release the lock */
   lock_rc = fcntl( iFileHandle, lock_command, &unlock );
   if ( lock_rc != 0 ) goto file_unlock_err;
   file_buf[0] = '¥0' ;  /* reset the buffer */
    buffer_len = 0 ;  /* reset the buffer length */
    write_ptr = file_buf ;  /* reset the write pointer */
       }
       else {
          write_ptr = file_buf + buffer_len ;  /* next write position */
       }
    } while (1) ;

cls:
    /* Write the last piece of data out to the file */
    if (buffer_len > 0) {
       lock_rc = fcntl(iFileHandle, lock_command, &lock);
       if (lock_rc != 0) goto file_lock_err;
      lock_rc = lseek(iFileHandle, 0, SEEK_END);
       if (lock_rc < 0) goto file_seek_err;
       lock_rc = write(iFileHandle, (void *)file_buf, buffer_len);
       if (lock_rc < 0) goto file_write_err;
       lock_rc = fcntl(iFileHandle, lock_command, &unlock);
      if (lock_rc != 0) goto file_unlock_err;
    }
    free(file_buf);
 close(iFileHandle);
    EXEC SQL CLOSE c1;
    exit (0);
 ext:
    if ( SQLCODE != 0 )
       printf( "Error:  SQLCODE = %ld.¥n", SQLCODE );
    EXEC SQL WHENEVER SQLERROR CONTINUE;
    EXEC SQL CONNECT RESET;
    if ( SQLCODE != 0 ) {
       printf( "CONNECT RESET Error:  SQLCODE = %ld¥n", SQLCODE );
       exit(4);
    }
    exit (5);
 file_lock_err:
    printf("Error: file lock error = %ld.¥n",lock_rc);
      /* unconditional unlock of the file */
    fcntl(iFileHandle, lock_command, &unlock);
    exit(6);
file_seek_err:
    printf("Error: file seek error = %ld.¥n",lock_rc);
      /* unconditional unlock of the file */
    fcntl(iFileHandle, lock_command, &unlock);
   exit(7);
file_write_err:
```

```
    printf("Error: file write error = %ld.¥n",lock_rc);
      /* unconditional unlock of the file */
    fcntl(iFileHandle, lock_command, &unlock);
    exit(8);
file_unlock_err:
    printf("Error: file unlock error = %ld.¥n",lock_rc);
      /* unconditional unlock of the file */
   fcntl(iFileHandle, lock_command, &unlock);
    exit(9);
}
```

この手法は、単一表からの選択だけでなく、さらに複雑な照会にも適用されます。ただし、照会が配列されていない操作を必要とする (つまり、 Explain がコーディネーター・サブセクションの他に複数のサブセクションを表示する) 場合に、照会をすべてのパーティションで並列して実行すると、いくつかのパーティションであまりにも多くのプロセスが発生することになります。この状況では、必要なだけのパーティションで照会の結果を一時表 TEMP に保管した後、最終的な抽出を TEMP から並列して実行できます。

選択した担当作業種別のみに従って、すべての従業員を抽出する場合は、次のようにして、FIRSTNME、LASTNAME、JOB という名前の列のある TEMP 表を定義することができます。

```
INSERT INTO TEMP
SELECT FIRSTNME, LASTNAME, JOB FROM EMPLOYEE WHERE WORKDEPT IS NOT NULL
       AND EMPNO NOT IN (SELECT EMPNO FROM EMP_ACT WHERE
                         EMPNO<200)
```

次いで、TEMP に対して並列抽出を実行します。

TEMP 表を定義するときには、次の事柄を考慮します。

- 照会が集約 GROUP BY を指定する場合は、 TEMP のパーティション・キーを GROUP BY 列のサブセットとして定義する必要があります。
- TEMP 表のパーティション・キーは、表が定義されるパーティション間で均等に分散されるように、十分なカーディナリティー (つまり、応答セット内の別個の値の数) を持っている必要があります。
- TEMP 表を NOT LOGGED INITIALLY 属性を指定して作成し、その表を作成した作業単位を COMMIT して、獲得されたカタログ・ロックをすべて解放してください。
- TEMP 表を使用する時は、 1 つの作業単位内で次のステートメントを出してください。
  1. ALTER TABLE TEMP ACTIVATE NOT LOGGED INITIALLY WITH EMPTY TABLE (TEMP 表を空にしてロギングをオフにする)
  2. INSERT INTO TEMP SELECT FIRSTNAME...
  3. COMMIT

  このような手法によって、ロギングを取らずに、またカタログ競合なしで、大きな応答セットを表に挿入できます。ただし、表で NOT LOGGED INITIALLY 属性が活動化されていて、ログに記録されていないアクティビティーが発生し、以下のいずれかの状態になると、
  - ステートメントが失敗し、ロールバックが発生する
  - ROLLBACK TO SAVEPOINT が実行される

作業単位全体はロールバックされ (SQL1476N)、TEMP 表が使用できなくなります。このことが起きた場合は、TEMP 表をドロップして再作成しなければなりません。この理由から、再作成が困難と思われる表にデータを追加する際は、この手法を使わないで ください。

最終的な応答セット (すべてのパーティションからマージされる部分的な応答セット) をソートする必要がある場合は、次のようにすることができます。

- 最終的な SELECT に SORT BY 文節を指定します。
- それぞれのパーティション上の別のファイルに抽出を行います。
- それぞれのファイルを 1 つの出力セットにマージします。そのためには、たとえば、**sort -m** AIX コマンドを使用します。

シミュレートされたパーティション・データベース環境の作成

パーティション・データベース環境をセットアップしなくても、パーティション・データベース環境アプリケーションのためのテスト環境を作成することができます。

手順:

シミュレートされたパーティション・データベース環境を作成するには、以下のようにします。

1. DB2 Enterprise Server Edition を使ってデータベース設計のモデルを作成します。
2. 実稼働環境でパーティションを超えてデータを分散するのに使用するサンプル表を、 PARTITIONING KEY 文節を使って作成します。
3. テスト・データベースに対してアプリケーションを開発して実行します。

DB2 Enterprise Server Edition は、パーティション・データベース環境で適用されるパーティション・キー制限を強制し、ご使用のアプリケーションに役立つテスト環境を提供します。

トラブルシューティング

以下の節では、パーティション・データベース環境でアプリケーションのトラブルシューティングを行う方法について説明しています。

パーティション・データベース環境におけるエラー処理

パーティション・データベース環境では、DB2® は SQL ステートメントをサブセクションに区切ります。各サブセクションは関係するデータがあるパーティションで処理されます。結果として、アプリケーションにアクセスできないパーティションで、エラーが発生することがあります。この状態は、パーティション・データベース環境以外では生じません。

次の事柄を考慮する必要があります。

- 非 CURSOR (EXECUTE) 非重大エラー
- CURSOR 非重大エラー

- 重大エラー
- マージされた複数の SQLCA 構造
- エラーを戻したパーティションを識別する方法

重大エラーが原因でアプリケーションが異常終了した場合、未確定トランザクションがデータベースに残っている可能性があります。 (1 つのフェーズが正常に完了すると、未確定トランザクションはグローバル・トランザクションに関係付けられますが、後続のフェーズが完了する前にシステムに障害が起き、データベースは不整合状態のままになります。)

関連概念:

- 729 ページの『パーティション・データベース環境における重大エラー』
- 730 ページの『マージされた複数の SQLCA 構造』
- 730 ページの『エラーを戻すパーティション』

関連タスク:

- 「*管理ガイド: プランニング*」の『未確定トランザクションの手動での解決』

パーティション・データベース環境における重大エラー

重大エラーがパーティション・データベース環境で発生すると、以下のいずれかが発生します。

- エラーが発生したパーティション上のデータベース・マネージャーが遮断します。

  アクティブな作業単位がロールバックされません。

  この状況では、パーティション、およびシャットダウンが生じたときにパーティションでアクティブだったデータベースを回復する必要があります。
- すべてのエージェントが、エラーの発生したパーティションでデータベースを強制的に切断します。

  そのデータベースのすべての作業単位がロールバックされます。

  この状況では、エラーの発生したデータベース・パーティションが、不整合とマークされます。それにアクセスしようとすると、SQLCODE -1034 (SQLSTATE 58031) または SQLCODE -1015 (SQLSTATE 55025) のいずれかが戻されます。ユーザーまたは別のパーティションの他のアプリケーションが、このデータベース・パーティションにアクセスできるようにするには、そのデータベースに対して RESTART DATABASE コマンドを実行する必要があります。

重大エラー SQLCODE -30081 (SQLSTATE 08001) は、さまざまな理由で発生します。このメッセージを受け取ったならば、SQLCA を調べてみます。そこに、障害が起きたパーティションが示されています。その後、管理通知ログ・ファイルで、詳細を調べます。

関連概念:

- 730 ページの『エラーを戻すパーティション』

関連資料:

- 「コマンド・リファレンス」の『RESTART DATABASE コマンド』

マージされた複数の SQLCA 構造

1 つの SQL ステートメントは、異なるデータベース・パーティション上の多数のエージェントが実行することができ、各エージェントが異なるエラーまたは警告のために異なる SQLCA を戻すことがあります。コーディネーター・エージェントも独自の SQLCA を持っています。 さらに、SQLCA には、グローバル数を示すフィールドもあります (行カウントを示す *sqlerrd* フィールドなど)。 各種アプリケーションの整合性のあるビューを提供するため、すべての SQLCA 値を 1 つの構造にマージします。

エラー報告は次のようになされます。

- 重大エラー条件は必ず報告されます。重大エラーを報告するとすぐに、重大エラーが SQLCA に追加されます。
- 重大エラーが発生していなければ、デッドロック・エラーが他のエラーに優先します。
- 他のすべてのエラーについては、最初の負の SQLCODE の SQLCA がアプリケーションに戻されます。
- 負の SQLCODE 値が検出されなければ、最初の警告 (つまり正の SQLCODE) の SQLCA がアプリケーションに戻されます。ただし、表の 1 つのパーティションは空でも別のパーティションにデータが入っている場合、その表でデータ処理操作を行っても、上記のようにはなりません。 SQLCODE +100 がアプリケーションに戻されるのは、表のすべてのパーティションで空であるか、または UPDATE ステートメントの WHERE 文節を満たす行が表にないために、すべてのパーティションのエージェントが SQL0100W を戻した場合だけです。
- すべてのエラーおよび警告において、*sqlwarn* フィールドに、すべてのエージェントから受け取った警告標識が入っています。
- 行カウントを示す *sqlerrd* フィールドの値は、すべてのエージェントから累計されます。

アプリケーションは、最初のエラーまたは警告の原因となった問題を修正した後、続いて次のエラーまたは警告を受け取ることがあります。エラーは SQLCA に報告されて、検出した最初のエラーが他のエラーに優先するかを確認します。これは、以前のエラーが引き起こしたエラーが、修正したばかりのエラーを重ね書きしていないか確認するためです。重大エラーおよびデッドロック・エラーは、最高の優先順位を与えられています。なぜなら、それらについてはコーディネーター・エージェントがただちに処置を行う必要があるからです。

関連資料:

- 「*管理 API リファレンス*」の『SQLCA』

エラーを戻すパーティション

パーティションがエラーまたは警告を戻す場合、その番号が SQLCA の SQLERRD(6) フィールドに示されます。 このフィールド内の番号は、`db2nodes.cfg` ファイルでそのパーティションに指定されたものと同じです。

SQL ステートメントまたは API 呼び出しが成功した場合は、このフィールドのパーティション番号は無効です。

関連資料:

- 「*管理 API リファレンス*」の『SQLCA』

ループ状態または延期状態のアプリケーション

照会またはアプリケーションを開始した後、それが延期状態 (活動を表示しない) またはループ状態 (活動は表示するが、結果をアプリケーションに戻していない) であることに気付く場合があります。 ロック・タイムアウトをオンにしているかを確認してください。 しかし、ある状況では、エラーが戻されません。 これらの状況では、DB2® に備わっている診断ツールや、データベース・システム・モニター・スナップショットが役に立ちます。

アプリケーションのデバッグに役立つデータベース・システム・モニターの機能の 1 つは、すべてのアクティブ・エージェントの状況の表示です。 スナップショットを最大限に活用するには、アプリケーションの実行前に (できれば DB2START の実行直後に)、ステートメントのコレクションが実行されているかを、次のように確認してください。

```
db2_all "db2 UPDATE MONITOR SWITCHES USING STATEMENT ON"
```

アプリケーションまたは照会が、停止またはループしているように思える場合は、次のコマンドを発行します。

```
db2_all "db2 GET SNAPSHOT FOR AGENTS ON database
```

関連概念:

- 「*システム・モニター ガイドおよびリファレンス*」の『データベース・システム・モニター』
- 「*管理ガイド: パフォーマンス*」の『データベース・システム・モニター情報』

関連資料:

- 「コマンド・リファレンス」の『GET SNAPSHOT コマンド』
- 「コマンド・リファレンス」の『UPDATE MONITOR SWITCHES コマンド』
- 「コマンド・リファレンス」の『db2trc - トレース・コマンド』
- 「コマンド・リファレンス」の『db2support - 問題分析および環境収集ツール・コマンド』

第 32 章 一般的な DB2 アプリケーションの技法

Windows Local System Account からのアプリケーションの実行 733
生成列. 733
ID 列 734
| SQL データ変更ステートメントからの結果セットの検索. 735
| 中間結果表 736
| ターゲット表とビュー 737
| INPUT SEQUENCE をベースにした結果セットのソート 737
| カーソルを使用した SQL データ変更ステートメントからの結果セットの検索 738
| INCLUDE 列 739
| INSERT 操作における INCLUDE 列. 739
| UPDATE および DELETE 操作における INCLUDE 列 740
全選択に対する検索
UPDATE、INSERT、DELETE、および MERGE 操作 740
順次値とシーケンス・オブジェクト 741
順次値の生成 741
シーケンスの振る舞いの管理 742
アプリケーションのパフォーマンスとシーケンス・オブジェクト 743
シーケンス・オブジェクトと ID 列の比較. . . 744
宣言済み一時表とアプリケーションのパフォーマンス 744
ネットワークを通じた大量データの伝送 746

Windows Local System Account からのアプリケーションの実行

Windows プラットフォームでは、DB2 は、暗黙的なローカル・データベース接続を使用した、 Local System Account (LSA) からのアプリケーションの実行をサポートします。明示的なデータベース接続やリモート・データベース接続を、LSA を使用して行うことはできません。

DB2 では、LSA のスキーマ名は SYSTEM です。 DB2 には、″SYS″ で始まるスキーマ名を持つオブジェクトに対する制約事項があるので、 LSA から実行されるデータベース・アプリケーションは、暗黙的な接続スキーマを使用して DB2 オブジェクトを作成することができません。 LSA から実行するアプリケーションから DB2 オブジェクトを作成するには、次のようにする必要があります。

- 静的 SQL の場合は、アプリケーションを ″SYSTEM″ 以外の QUALIFIER 値でバインドします。
- 動的 SQL 場合は、作成するオブジェクトを ″SYSTEM″ 以外のスキーマ名で明示的に修飾するか、 CURRENT SCHEMA レジスターに ″SYSTEM″ 以外のスキーマ名を設定する必要があります。

関連資料:

- 「*SQL リファレンス 第 2 巻*」の『SET SCHEMA ステートメント』
- 「コマンド・リファレンス」の『BIND コマンド』
- 「*SQL リファレンス 第 1 巻*」の『CURRENT SCHEMA 特殊レジスター』

生成列

DB2® では、GENERATED ALWAYS AS 文節を使用して、生成列を表に組み込むことができます。これにより、厄介な挿入トリガーおよび更新トリガーを使用する必要はありません。 生成列とは、各行の値を挿入操作または更新操作からではなく、式から派生する列です。更新トリガーおよび挿入トリガーを組み合わせて使用すると同様のことが行えますが、生成列を使用すると、派生した値が式と一貫したものであることを保証できます。

表で生成列を作成するには、列で GENERATED ALWAYS AS 文節を使用して、列の値が派生する式を含めてください。 GENERATED ALWAYS AS 文節は、ALTER TABLE および CREATE TABLE ステートメントに含められます。次の例では、“c1” および “c2” という通常の 2 つの列と、表の通常の列から派生した “c3” および “c4” という 2 つの生成された列の入った表を作成します。

```
CREATE TABLE T1(c1 INT, c2 DOUBLE,
                c3 DOUBLE GENERATED ALWAYS AS (c1 + c2),
                c4 SMALLINT GENERATED ALWAYS AS
                  (CASE
                     WHEN c1 > c2 THEN 1
                     ELSE NULL
                   END)
               );
```

関連タスク:

- 「*管理ガイド: インプリメンテーション*」の『新しい表に生成列を定義する』
- 「*管理ガイド: インプリメンテーション*」の『既存の表での生成列の定義』

関連資料:

- 「*SQL リファレンス 第 2 巻*」の『ALTER TABLE ステートメント』
- 「*SQL リファレンス 第 2 巻*」の『CREATE TABLE ステートメント』

関連サンプル:

- 『TbGenCol.java -- How to use generated columns (JDBC)』

ID 列

DB2® アプリケーション開発者は、ID 列を使用して表の各行に対して数値列の値を自動的に生成することができます。この値をユニーク値として生成し、ID 列を主キーとして定義できます。 ID 列を作成するには、CREATE TABLE に IDENTITY 文節を含めてください。

アプリケーションがデータベースの外に独自のカウンターを生成する際に生じる、並行性およびパフォーマンス上の問題を回避するため、アプリケーション内で ID 列を使用してください。ユニークな主キーを自動生成するのに ID 列を使用しない場合には、単一行の表にカウンターを保管するのが一般的な設計方法です。各トランザクションはこの表をロックして、数を増分してからトランザクションをコミットして、カウンターをアンロックします。しかし、残念ながら、この設計では、カウンターを増分できるのは一度に 1 つのトランザクションのみです。

それとは対照的に、ID 列を使用して主キーを自動的に生成すると、アプリケーションでより高度なレベルの並行性を実現できます。 ID 列では、トランザクションがカウンターをロックしなくてもいいように DB2 はカウンターを保持します。カウンターを増分したコミットされていないトランザクションが、他の後続するトランザクションによるカウンターの増分を阻止しないので、ID 列を使用するアプリケーションはパフォーマンスが向上します。

ID 列のカウンターは、トランザクションに関係なく増分されたり減分されたりします。あるトランザクションが識別カウンターを 2 回増分する場合、他のトランザクションが同一の識別カウンターを並行して増分することがあるので、生成される 2 つの数の値には差があるかもしれません。

トランザクションがロールバックされたため、またはキャッシュに入れられた値すべてが割り当てられる前に、非活動化された (正常にまたは異常に) 値がデータベースによってキャッシュに入れられたため、ID 列のカウンターの数が離れたものになる場合があります。

ID 列を持つ表に新規行を挿入した後で、生成された値を検索するには、`identity_val_local()` 関数を使用します。

IDENTITY 文節は、CREATE TABLE および ALTER TABLE ステートメントの両方で使用できます。

関連概念:

- 「*管理ガイド: プランニング*」の『ID 列』

関連タスク:

- 「*管理ガイド: インプリメンテーション*」の『新しい表での ID 列の定義』
- 「*管理ガイド: インプリメンテーション*」の『ID 列定義の変更』
- 「*管理ガイド: インプリメンテーション*」の『ID 列の変更』

関連資料:

- 「*SQL リファレンス 第 2 巻*」の『ALTER TABLE ステートメント』
- 「*SQL リファレンス 第 2 巻*」の『CREATE TABLE ステートメント』

関連サンプル:

- 『tbident.sqc -- How to use identity columns (C)』
- 『TbIdent.java -- How to use Identity Columns (JDBC)』
- 『TbIdent.sqlj -- How to use Identity Columns (SQLj)』

SQL データ変更ステートメントからの結果セットの検索

INSERT、UPDATE、または DELETE ステートメントによって表を変更するアプリケーションは、変更行に追加の処理を必要とする場合があります。この処理を簡単にするために、SELECT および SELECT INTO ステートメントの FROM 文節に、SQL データ変更操作を組み込むことができます。単一作業単位内で、アプリケーションは、SQL データ変更操作によって変更された表またはビューからの変更行を含む結果セットを検索できます。

たとえば、次のステートメントは、SAMPLE データベースにある EMPLOYEE 表の給料のすべての記録を更新し、次いですべての更新済みの行に従業員番号および新規給料を戻します。

```
SELECT empno, salary FROM FINAL TABLE
 (UPDATE employee SET salary = salary * 1.10 WHERE job = 'CLERK')
```

正常にデータを戻すために、FROM SQL データ変更操作の結果セットを検索する SELECT ステートメントは、正常に実行するために SQL データ変更操作を必要とします。正常な SQL データ変更操作では、該当する場合、すべての制約およびトリガーの処理を組み込みます。

例えば、EMPLOYEE 表に対して SELECT 特権はあるが INSERT 特権はないユーザーが、EMPLOYEE 表に対して SELECT FROM INSERT ステートメントを試行すると想定します。 INSERT 操作は特権がないために失敗し、結果として SELECT ステートメント全体が失敗します。

次の照会を考慮してください。EMPLOYEE 表からの記録が選択され、EMP という名前の別の表に挿入されます。この SELECT ステートメントは失敗します。

```
SELECT empno FROM FINAL TABLE
 (INSERT INTO emp(name, salary)
 SELECT firstnme || midinit || lastname, salary
 FROM employee)
```

EMPLOYEE 表に行が 100 あり、90 行目に $9,999,000.00 の SALARY 値がある場合、$10,000.00 の追加は 10 進数のオーバーフローを引き起こす原因となります。オーバーフローが発生すると、データベース・マネージャーは EMP 表への追加をロールバックすることになります。

中間結果表

SELECT ステートメントの FROM 文節で SQL データ変更操作によってターゲットにされた表またはビューの変更行は、中間結果表を構成します。中間結果表は、SQL データ変更操作で定義された列を組み込むことに加えて、ターゲット表またはビューのすべての列を組み込みます。選択リスト、ORDER BY 文節、または WHERE 文節内の名前によって中間結果表のすべての列を参照できます。

中間結果表の内容は、FROM 文節で指定された修飾子に依存しています。中間結果表として結果セットを検索する SELECT ステートメント内の、次の FROM 文節修飾子の 1 つを組み込む必要があります。

OLD TABLE
: 中間結果表の行は、BEFORE トリガーおよび SQL データ変更操作の実行の直前のターゲット表の行の値を含みます。 OLD TABLE 修飾子は UPDATE および DELETE 操作を適用します。

NEW TABLE
: 中間結果表の行には、SQL データ変更ステートメントが実行された直後、および参照保全評価といずれかの AFTER トリガーの実行前のターゲット表の行の値が入ります。 NEW TABLE 修飾子は UPDATE および INSERT 操作を適用します。

FINAL TABLE
: この修飾子は、NEW TABLE と同じ中間結果表に戻ります。さらに、AFTER トリガーまたは参照保全制約がない FINAL TABLE を使用すると、 UPDATE または INSERT 操作のターゲットがさらに変更されることが保証されます。 FINAL TABLE 修飾子は UPDATE および INSERT 操作を適用します。

FROM 文節の修飾子は、中間結果表にどのバージョンのターゲット・データがあるかを判別します。これらの修飾子は、ターゲット表の行の追加、削除、または更新には影響しません。

ターゲット表とビュー

FROM SQL データ変更操作の結果セットを選択する場合、ターゲットは表またはビューにすることができます。

ビューに対する SQL データ変更操作で、結果表には、NEW TABLE と FINAL TABLE のビュー定義を満たさなくなった行を組み込むことはできません。SELECT ステートメントのビューを参照する INSERT または UPDATE ステートメントを組み込む場合、ビューは WITH CASCADED CHECK OPTION として定義される必要があります。代替として、ビューは、WITH CASCADED CHECK OPTION としてのビューの定義を許可する制限を満たす必要があります。

SELECT ステートメントの FROM 文節に組み込まれた SQL データ変更操作のターゲットが全選択である場合、結果表には、全選択に該当しない行が組み込まれることがあります。これは、WHERE 文節の述部が更新済みの値に対して再評価されないためです。

INPUT SEQUENCE をベースにした結果セットのソート

ターゲット表またはビューに挿入されたのと同じ ORDER BY で行を SELECT するには、 ORDER BY 文節で INPUT SEQUENCE キーワードを使用します。INPUT SEQUENCE キーワードの使用によって、行がその提供と同じ順序で挿入されることはありません。

以下に示すのは、INSERT 操作の結果セットをソートするため、 ORDER BY 文節で INPUT SEQUENCE キーワードを使用する例です。

```
CREATE TABLE orders (purchase_date DATE,
                     sales_person VARCHAR(16),
                     region VARCHAR(10),
                     quantity SMALLINT,
                     order_num INTEGER NOT NULL
                       GENERATED ALWAYS AS IDENTITY (START WITH 100,
                                     INCREMENT BY 1))

SELECT * FROM FINAL TABLE
         (INSERT INTO orders
          (purchase_date, sales_person, region, quantity)
          VALUES (CURRENT DATE,'Judith','Beijing',6),
                 (CURRENT DATE,'Marieke','Medway',5),
                 (CURRENT DATE,'Hanneke','Halifax',5))
         ORDER BY INPUT SEQUENCE

PURCHASE_DATE SALES_PERSON     REGION     QUANTITY ORDER_NUM
------------- ---------------- ---------- -------- -----------
07/18/2003    Judith           Beijing           6         100
07/18/2003    Marieke          Medway            5         101
07/18/2003    Hanneke          Halifax           5         102
```

列の組み込みを使用して結果セットの行をソートすることもできます。

関連概念:

- 738 ページの『カーソルを使用した SQL データ変更ステートメントからの結果セットの検索』
- 739 ページの『INCLUDE 列』

関連資料:

- 「*SQL リファレンス 第 2 巻*」の『DELETE ステートメント』
- 「*SQL リファレンス 第 2 巻*」の『INSERT ステートメント』
- 「*SQL リファレンス 第 2 巻*」の『SELECT ステートメント』
- 「*SQL リファレンス 第 2 巻*」の『SELECT INTO ステートメント』
- 「*SQL リファレンス 第 2 巻*」の『UPDATE ステートメント』

カーソルを使用した SQL データ変更ステートメントからの結果セットの検索

SQL データ変更操作から結果セットを検索する照会に対して、カーソルを宣言することができます。たとえば、次のようにします。

```
DECLARE C1 CURSOR FOR SELECT salary FROM FINAL TABLE
        (INSERT INTO employee (name, salary, level)
            SELECT name, income, band FROM old_employee)
```

その定義に SQL データ変更操作を含むカーソルからの取り出しで生じるエラーは、変更行のロールバックの原因となります。このエラーによってカーソルが閉じたとしても、アプリケーションがカーソルを開くときには、それらは完了しているので、行の変更は何も行われないままになります。

そのようにカーソルが開く時に、データベース・マネージャーは完全に SQL データ変更操作を実行し、結果セットは一時表に保管されます。カーソルが開いている間にエラーが生じた場合、SQL データ変更操作によって加えられた変更はロールバックされます。ターゲット表またはビューのそれ以降の更新は、SQL データ変更操作からの結果セットを検索するカーソルの結果表の行には表示されません。たとえば、アプリケーションはカーソルを宣言し、カーソルを開き、取り出しを実行し、表を更新し、追加の行を取り出します。 UPDATE ステートメント後の取り出しは、UPDATE ステートメント前のオープン・カーソル処理中に決定されたこれらの値を戻します。

SQL データ変更操作からの結果セットを検索する照会に対して、両方向スクロール・カーソルを宣言することができます。カーソルを OPEN にする場合、結果表が生成されるので、データの変更はすでにターゲット表またはビューに書き込まれています。 SQL データ変更操作から行を選択する照会を持つカーソルは、INSENSITIVE または ASENSITIVE として定義される必要があります。

注: 両方向スクロール・カーソルは、CLI、JDBC、および SQLJ アプリケーションによってのみサポートされます。

WITH HOLD オプションと、COMMIT を実行するアプリケーションがあるカーソルを宣言する場合、すべてのデータ変更はコミットされます。 WITH HOLD として宣言しないカーソルは、同じ方法で振る舞います。すべてのカーソルに対して、照会に組み込まれた SQL データ変更操作は、どの行が取り出されるよりも前に完全に評価されます。

OPEN CURSOR ステートメントに対して明示的なロールバックを実行する場合、または OPEN CURSOR ステートメントより前のセーブポイントまでロールバックする場合、カーソルに対するすべてのデータ変更は取り消されます。 SQL データ変更操作からの結果セットを検索する照会付きのカーソルの場合、すべてのデータ変

更はロールバック後に取り消されますが、カーソルは保存され、事前に挿入されていた行は引き続き取り出すことができます。

関連概念:

- 735 ページの『SQL データ変更ステートメントからの結果セットの検索』
- 739 ページの『INCLUDE 列』

関連資料:

- 「*SQL リファレンス 第 2 巻*」の『DECLARE CURSOR ステートメント』
- 「*SQL リファレンス 第 2 巻*」の『DELETE ステートメント』
- 「*SQL リファレンス 第 2 巻*」の『FETCH ステートメント』
- 「*SQL リファレンス 第 2 巻*」の『OPEN ステートメント』
- 「*SQL リファレンス 第 2 巻*」の『SELECT ステートメント』
- 「*SQL リファレンス 第 2 巻*」の『UPDATE ステートメント』

INCLUDE 列

INCLUDE 列によって、ターゲット表またはビューに存在していない列を、中間結果表に導入することができます。中間結果表で行のハンドルとして使用する列を組み込むために、値を割り当てることができます。 INCLUDE 列は SQL データ変更操作には影響せず、ターゲット表またはビューの定義を変更することもありません。

INCLUDE 列は、任意のデータ・タイプの列とすることができ、NULL 可能であり、 SQL データ変更操作におけるターゲット表またはビューにあるどの列とも異なるユニークな名前を持っている必要があります。 SELECT ステートメントの選択リスト、ORDER BY 文節、または WHERE 文節で、列の組み込みを参照することができます。 結果セットでは、列の組み込みは右端の列として表示されます。

INSERT 操作における INCLUDE 列

INSERT 操作で INCLUDE 列の値を割り当てるには、VALUES 文節を使用できます。 INSERT 操作での INCLUDE 列の一般的な使用目的は、結果セットの配列をカスタマイズすることです。たとえば、次のようにします。

```
SELECT * FROM FINAL TABLE
         (INSERT INTO sales INCLUDE (sortkey integer) VALUES
          (CURRENT DATE,'Judith', 'Halifax',6,1),
          (CURRENT DATE,'Marieke', 'Medway',5,3),
          (CURRENT DATE,'Hanneke', 'Halifax',5,2))
         ORDER BY sortkey

SALES_DATE SALES_PERSON    REGION          SALES       SORTKEY
---------- --------------- --------------- ----------- -----------
07/16/2003 Judith          Amsterdam                 6           1
07/16/2003 Hanneke         Halifax                   5           2
07/16/2003 Marieke         Medway                    5           3
```

全選択を使用することによっても、INSERT 操作で INCLUDE 列に値を割り当てることができます。

UPDATE および DELETE 操作における INCLUDE 列

UPDATE または DELETE 操作で INCLUDE 列の値を割り当てるには、SET 文節を使用します。 UPDATE または DELETE ステートメントの SET 文節で INCLUDE 列に値が割り当てられない場合、NULL 値はその列に戻されます。

UPDATE ステートメントで、INCLUDE 列を使用して、行のために新旧両方の列の値を戻すことができます。たとえば、次のようにします。

```
SELECT salary, oldSalary FROM FINAL TABLE
       (UPDATE employee INCLUDE (oldSalary decimal(9,2))
        SET oldSalary = salary, salary = salary * 1.05
        WHERE job = 'CLERK')

SALARY      OLDSALARY
----------- -----------
   30712.50    29250.00
   23289.00    22180.00
   30198.00    28760.00
   20139.00    19180.00
   18112.50    17250.00
   28749.00    27380.00
```

関連概念:

- 735 ページの『SQL データ変更ステートメントからの結果セットの検索』
- 738 ページの『カーソルを使用した SQL データ変更ステートメントからの結果セットの検索』

関連資料:

- 「*SQL リファレンス 第 2 巻*」の『DELETE ステートメント』
- 「*SQL リファレンス 第 2 巻*」の『INSERT ステートメント』
- 「*SQL リファレンス 第 2 巻*」の『SELECT ステートメント』
- 「*SQL リファレンス 第 2 巻*」の『UPDATE ステートメント』

全選択に対する検索 UPDATE、INSERT、DELETE、および MERGE 操作

DB2® バージョン 8.1.4 現在で、全選択の結果に対して、検索 INSERT、UPDATE、 DELETE、および MERGE ステートメントを発行できます。このフィーチャーを使用する場合は、そうでない場合に 2 つのステートメント (全選択と、全選択の結果に対する INSERT、UPDATE、DELETE、または MERGE) を必要とする処理を、単一のステートメントに減らすことができます。この処理を単一のステートメントにまとめれば、デッドロックの可能性を減らすことができ、ビューおよびカーソルの定義の必要もなくなる可能性があります。追加、更新、または削除できるビューを生成するために使用できる照会はどれも、検索 INSERT、 UPDATE、DELETE、または MERGE ステートメントのターゲットにできます。

たとえば、次のステートメントでは、EMPLOYEE 表内の最も学歴の低い 10 人の従業員が削除されます。

```
DELETE FROM (SELECT edlevel FROM employee
               ORDER BY edlevel
               FETCH FIRST 10 ROWS ONLY)
```

関連資料:

- 「*SQL リファレンス 第 2 巻*」の『DELETE ステートメント』

- 「*SQL リファレンス 第 2 巻*」の『INSERT ステートメント』
- 「*SQL リファレンス 第 2 巻*」の『SELECT ステートメント』
- 「*SQL リファレンス 第 2 巻*」の『SELECT INTO ステートメント』
- 「*SQL リファレンス 第 2 巻*」の『UPDATE ステートメント』
- 「*SQL リファレンス 第 2 巻*」の『MERGE ステートメント』

順次値とシーケンス・オブジェクト

以下の節では、順次値とシーケンス・オブジェクトに関する考慮事項について説明します。

順次値の生成

順次値を生成することは、一般的なデータベース・アプリケーション開発の問題です。この問題を解決する最善の方法は、SQL でシーケンス・オブジェクトとシーケンス式を使用することです。各シーケンス・オブジェクト は、固有の名前が付けられたデータベース・オブジェクトであり、シーケンス式によってのみアクセスできます。シーケンス式 には、 PREVVAL 式と NEXTVAL 式の 2 つがあります。PREVVAL 式は、アプリケーション・プロセスで、指定されたシーケンス・オブジェクトについて生成された最新の値を返します。 PREVVAL 式と同じステートメントで発生する NEXTVAL 式は、そのステートメントの PREVVAL 式で生成された値に対して影響を与えません。 NEXTVAL シーケンス式は、シーケンス・オブジェクトの値を増やして、そのシーケンス・オブジェクトの新しい値を返します。

シーケンス・オブジェクトを作成するには、CREATE SEQUENCE ステートメントを発行します。たとえば、デフォルトの属性を使用して id_values というシーケンス・オブジェクトを作成するには、次のステートメントを発行します。

```
CREATE SEQUENCE id_values
```

アプリケーション・セッションで、シーケンス・オブジェクトの最初の値を生成するには、次のように NEXTVAL 式を使用して VALUES ステートメントを発行します。

```
VALUES NEXTVAL FOR id_values

1
-----------
          1

  1 record(s) selected.
```

シーケンス・オブジェクトの現行値を表示するには、 PREVVAL 式を使用して VALUES ステートメントを発行します。

```
VALUES PREVVAL FOR id_values

1
-----------
          1

  1 record(s) selected.
```

シーケンス・オブジェクトの現行値は繰り返し検索することができます。シーケンス・オブジェクトが返す値は、NEXTVAL 式を発行するまで変わりません。以下の例では、現行接続の NEXTVAL 式がシーケンス・オブジェクトの値を増やすまで、PREVVAL は値 1 を返します。

```
VALUES PREVVAL FOR id_values

1
-----------
          1

            1 record(s) selected.

VALUES PREVVAL FOR id_values

1
-----------
          1

            1 record(s) selected.

VALUES NEXTVAL FOR id_values

1
-----------
          2

            1 record(s) selected.

VALUES PREVVAL FOR id_values

1
-----------
          2

            1 record(s) selected.
```

シーケンス・オブジェクトの次の値で列の値を更新するには、次のように UPDATE ステートメントに NEXTVAL 式を組み込みます。

```
UPDATE staff
  SET id = NEXTVAL FOR id_values
  WHERE id = 350
```

シーケンス・オブジェクトの次の値を使用して新しい行を表に挿入するには、次のように INSERT ステートメントに NEXTVAL 式を組み込みます。

```
INSERT INTO staff (id, name, dept, job)
  VALUES (NEXTVAL FOR id_values, 'Kandil', 51, 'Mgr')
```

関連資料:

- 「*SQL リファレンス 第 2 巻*」の『CREATE SEQUENCE ステートメント』

関連サンプル:

- 『DbSeq.java -- How to create, alter and drop a sequence in a database (JDBC)』

シーケンスの振る舞いの管理

アプリケーションの要求を満たすようにシーケンス・オブジェクトの振る舞いを調整することができます。 CREATE SEQUENCE ステートメントを発行して新しいシーケンス・オブジェクトを作成する場合、および既存のシーケンス・オブジェクト

に対して ALTER SEQUENCE を発行する場合は、シーケンス・オブジェクトの属性を変更します。 指定可能なシーケンス・オブジェクトの属性のいくつかを以下に示します。

データ・タイプ
CREATE SEQUENCE ステートメントの AS 文節は、シーケンス・オブジェクトの数値データ・タイプを指定します。このデータ・タイプはシーケンス・オブジェクトの使用可能な最小値と最大値を決定します (データ・タイプごとの最小値と最大値は、SQL 制限を説明しているトピックにリストされています)。シーケンス・オブジェクトのデータ・タイプを変更することはできません。代わりに、DROP SEQUENCE ステートメントを発行してから新しいデータ・タイプで CREATE SEQUENCE ステートメントを発行することによりシーケンス・オブジェクトをドロップする必要があります。

開始値 CREATE SEQUENCE ステートメントの START WITH 文節は、シーケンス・オブジェクトの初期値を設定します。 ALTER SEQUENCE ステートメントの RESTART WITH 文節は、シーケンス・オブジェクトの値を指定値にリセットします。

最小値 MINVALUE 文節は、シーケンス・オブジェクトの最小値を設定します。

最大値 MAXVALUE 文節は、シーケンス・オブジェクトの最大値を設定します。

増分値 INCREMENT BY 文節は、各 NEXTVAL 式がシーケンス・オブジェクトの現行値に追加する値を設定します。シーケンス・オブジェクトの値を減らすには、負の値を指定します。

シーケンス循環
CYCLE 文節は、シーケンス・オブジェクトの値が最小値または最大値に達したとき、次の NEXTVAL 式でそれぞれ最小値または最大値を初期値に戻します。

たとえば、各 NEXTVAL 式で開始時の最小値が 0、最大値が 1000、増分値が 2 で、最大値に達したときに最小値に戻る id_values というシーケンス・オブジェクトを作成するには、次のステートメントを発行します。

```
CREATE SEQUENCE id_values
  START WITH 0
  INCREMENT BY 2
  MAXVALUE 1000
  CYCLE
```

関連資料:

- 「*SQL リファレンス 第 1 巻*」の『SQL の制限値』
- 「*SQL リファレンス 第 2 巻*」の『ALTER SEQUENCE ステートメント』
- 「*SQL リファレンス 第 2 巻*」の『CREATE SEQUENCE ステートメント』

アプリケーションのパフォーマンスとシーケンス・オブジェクト

ID 列のように、シーケンス・オブジェクトを使用して値を生成する場合、一般に、他の方法と比べてアプリケーションのパフォーマンスが向上します。シーケンス・オブジェクトを制御する別の方法として、現行値を保管する単一列表を作成し、トリガーを使用して、またはアプリケーションの制御下でその値を増やす方法があり

ます。単一列表にアプリケーションが並行してアクセスする分散環境では、順番に表にアクセスすることを強制するために必要になるロックが、パフォーマンスに大きく影響します。

シーケンス・オブジェクトは、単一列表を使用するために必要になるロック発行を行わずに、シーケンス値をメモリーにキャッシュして DB2® の応答時間を改善することができます。 シーケンス・オブジェクトを使用するアプリケーションのパフォーマンスを最大にするには、シーケンス・オブジェクトが適切な量のシーケンス値を確実にキャッシュするようにします。 CREATE SEQUENCE ステートメントおよび ALTER SEQUENCE ステートメントの CACHE 文節は、 DB2 が生成してメモリーに保管するシーケンス値の最大数を指定します。

シーケンス・オブジェクトが順序正しく値を生成する必要があり、システム障害またはデータベース非活動化でその順序が途切れないようにする場合、 ORDER および NO CACHE 文節を CREATE SEQUENCE ステートメントで使用します。 NO CACHE 文節は、生成された値が途切れないことを保証します。この場合、シーケンス・オブジェクトが新しい値を生成するたびにデータベース・ログに書き込むため、アプリケーションのパフォーマンスが低下します。トランザクションがロールバックし、要求したシーケンス値を実際には使用しないために、依然としてギャップが存在する場合があることに注意してください。

シーケンス・オブジェクトと ID 列の比較

シーケンス・オブジェクトと ID 列は DB2® アプリケーションに対して同じような目的を果たすために使用されているように見えますが、重要な違いがあります。 ID 列は、単一表の列の値を自動的に生成します。シーケンス・オブジェクトは、SQL ステートメントで使用可能な順次値を要求時に生成します。

宣言済み一時表とアプリケーションのパフォーマンス

宣言済み一時表 とは、一時表を作成したアプリケーションによって発行された SQL ステートメントにのみアクセス可能な一時表を表します。宣言済み一時表は、アプリケーションがデータベースに接続している間しか有効ではありません。

宣言済み一時表を使用して、アプリケーションの潜在的パフォーマンスの向上を図ってください。宣言済み一時表を作成する場合には、 DB2® はシステム・カタログ表に項目を挿入しないため、サーバーでカタログの競合による問題が起きることはありません。通常の表の場合とは異なり、 DB2 は宣言済み一時表またはその行をロックせず、作成時に NOT LOGGED パラメーターが指定されていれば、宣言済み一時表またはその表の内容をログに記録しません。現行のアプリケーションで多量のデータを処理するために表を作成し、アプリケーションによるデータの操作が終了する際にそれらの表をドロップする場合には、通常の表のかわりに宣言済み一時表を使用することを検討してください。

並行ユーザー用にアプリケーションを開発する場合には、宣言済み一時表が役立ちます。通常の表とは異なり、宣言済み一時表では名前が重複しても問題は起こりません。アプリケーションの各インスタンスでは、 DB2 は同じ名前の宣言済み一時表を作成することができます。たとえば、多量の一時データを処理するのに通常の表を使用する並行ユーザーのためにアプリケーションを作成する場合には、アプリ

ケーションの各インスタンスが一時データを保持する通常の表に対してユニークな名前を使用する必要があります。一般的には、ある時点で使用されている表の名前をたどる別の表を作成します。しかし、宣言済み一時表を使用すると、一時データに対して 1 つの宣言済み一時表を指定するだけで済みます。 DB2 は、アプリケーションの各インスタンスでユニークな表が使用されていることを保証します。

宣言済み一時表を使用するには、次のようなステップを実行します。

ステップ 1. USER TEMPORARY TABLESPACE が存在することを確認する。 USER TEMPORARY TABLESPACE が存在しない場合には、 CREATE USER TEMPORARY TABLESPACE ステートメントを発行します。

ステップ 2. アプリケーションで DECLARE GLOBAL TEMPORARY TABLE ステートメントを発行する。

宣言済み一時表のスキーマは必ず SESSION になります。 SQL ステートメントで宣言済み一時表を使用するには、 SESSION スキーマ修飾子を使用して表を明示的に参照するか、 SESSION の DEFAULT スキーマを使用して修飾されていない参照を修飾してください。次の例では、以下のステートメントで TT1 という宣言済み一時表を作成すると、表の名前は必ず SESSION というスキーマ名で修飾されます。

```
DECLARE GLOBAL TEMPORARY TABLE TT1
```

前述の例で作成された宣言済み一時表から *column1* 列の内容を選択するには、次のようなステートメントを使用します。

```
SELECT column1 FROM SESSION.TT1;
```

DB2 では、SESSION スキーマを使用して持続する表も作成できることに注目してください。 SESSION.TT3 という修飾名で持続する表を作成すると、 SESSION.TT3 という修飾名の宣言済み一時表を作成できます。この場合、同一の修飾名を持つ持続表および宣言済み一時表への参照は、 DB2 によって宣言済み一時表に解決されます。持続表と宣言済み一時表の混同を避けるには、 SESSION スキーマを使用して持続表を作成しないでください。

SESSION スキーマで修飾された表、ビュー、または別名への静的 SQL 参照を含むアプリケーションを作成する場合、 DB2 プリコンパイラーはバインド実行時にはそのステートメントをコンパイルせずに、そのステートメントに対して「コンパイルが必要」なことを記します。そして、ランタイムに DB2 はそのステートメントをコンパイルします。この動作は、*追加バインド* として知られています。DB2 は、SESSION スキーマによって修飾されている表、ビュー、および別名への静的 SQL 参照の追加バインドを自動的に実行します。これらのステートメントで追加バインドが使用可能になるように、 BIND または PRECOMPILE コマンドで VALIDATE RUN オプションを指定する必要はありません。

トランザクションに対して DECLARE GLOBAL TEMPORARY TABLE ステートメントを含む ROLLBACK ステートメントを出すと、 DB2 は宣言済み一時表をドロップします。宣言済み一時表に DROP TABLE ステートメントを出すと、そのトランザクションに ROLLBACK ステートメントを出しても空の宣言済み一時表がリストアされるだけです。 DROP TABLE ステートメントの ROLLBACK は、宣言済み一時表の行をリストアしません。

宣言済み一時表のデフォルトの振る舞いでは、トランザクションがコミットされる際に表からすべての行が削除されます。しかし、1 つまたは複数の WITH HOLD カーソルが宣言済み一時表でまだオープンされている場合には、トランザクションをコミットしても DB2 は表から行を削除しません。トランザクションをコミットする際にすべての行が削除されないようにするには、 DECLARE GLOBAL TEMPORARY TABLE で ON COMMIT PRESERVE ROWS 文節を使用して一時表を作成してください。

トランザクション内で INSERT、UPDATE、または DELETE ステートメントを使用して宣言済み一時表の内容を変更した後に、そのトランザクションをロールバックした場合、 DB2 は宣言済み一時表のすべての行を削除します。 INSERT、UPDATE、または DELETE ステートメントを使用して宣言済み一時表の内容を変更しようして、そのステートメントが失敗すると、 DB2 は以下のように動作します。

- その表が NOT LOGGED パラメーターを指定せずに作成された (つまり、その表がログに記録されている) 場合、失敗した INSERT、UPDATE、または DELETE ステートメントによる変更内容だけがロールバックされます。
- その表が NOT LOGGED パラメーターを指定して作成された場合、DB2 は、宣言済み一時表のすべての行を削除します。

パーティション・データベース環境で障害が発生すると、その障害が発生したデータベース・パーティションに存在するすべての宣言済み一時表は使用できなくなります。その後、これらの使用不能な宣言済み一時表にアクセスしようとすると、エラー (SQL1477N) が起きます。アプリケーションによって使用不能な宣言済み一時表が検出されると、アプリケーションはその表をドロップするか、 DECLARE GLOBAL TEMPORARY TABLE ステートメントで WITH REPLACE 文節を指定して、その表を再作成することができます。

宣言済み一時表には、いくつかの制限があります。たとえば、宣言済み一時表では別名またはビューを定義することができません。 IMPORT および LOAD を使用して宣言済み一時表を移植することもできません。宣言済み一時表では索引は作成できますが、いくつかの制限があります。さらに、宣言済み一時表に対して RUNSTATS を実行すれば、その宣言済み一時表およびその索引の統計を更新することができます。

関連資料:

- 「*SQL リファレンス 第 2 巻*」の『DECLARE GLOBAL TEMPORARY TABLE ステートメント』

関連サンプル:

- 『tbtemp.sqc -- How to use a declared temporary table (C)』
- 『TbTemp.java -- How to use Declared Temporary Table (JDBC)』

ネットワークを通じた大量データの伝送

ストアード・プロシージャーの技法と行のブロック化を組み合わせることにより、ネットワークを通じて大量のデータを受け渡す必要のあるアプリケーションのパフォーマンスを大幅に向上させることができます。

ネットワークを通じて、配列、大量のデータ、データのパッケージを受け渡すアプリケーションは、 SQLDA データ構造体または転送メカニズムとしてのホスト変数を使用して、データをブロック化して受け渡すことができます。この技法は、構造をサポートするホスト言語ではかなりの効果があります。

クライアントのアプリケーションまたはサーバー・プロシージャーのいずれもネットワークを通じてデータを受け渡すことができます。次のデータ・タイプのいずれかを使用してデータを受け渡すことができます。

- VARCHAR
- LONG VARCHAR
- CLOB
- BLOB

次のグラフィック・タイプを使用してもデータを受け渡すことができます。

- VARGRAPHIC
- LONG VARGRAPHIC
- DBCLOB

注: この技法を使用する際は、文字変換の機能を考慮してください。
VARCHAR、LONG VARCHAR、または CLOB などの文字ストリング・データ、または VARGRAPHIC、LONG VARGRAPHIC、または DBCLOB などのグラフィック・データ・タイプのいずれかのデータを受け渡し、アプリケーションのコード・ページがデータベースのコード・ページと異なる場合、文字データ以外のデータも文字データであるかのように変換されます。文字変換を避けるには、BLOB のデータ・タイプの変数でデータを受け渡してください。

関連概念:

- 666 ページの『異なるコード・ページ間での文字変換』
- 21 ページの『DB2 ストアード・プロシージャー』

関連タスク:

- 「*管理ガイド: パフォーマンス*」の『オーバーヘッド削減のための行ブロッキングの指定』

第 8 部 付録

付録 A. サポートされる SQL ステートメント

次の表は、Linux、UNIX、および Windows の DB2 Universal Database でサポートされる SQL ステートメントをすべてリストしています。「SQL ステートメント」列のステートメントはすべて、静的 SQL アプリケーションでサポートされています。残りの列は、他の DB2 アプリケーション開発コンテキストで、それぞれのステートメントがサポートされるかどうかを示します。

表 93. SQL ステートメント (DB2 Universal Database)

SQL ステートメント	動的[1]	コマンド行プロセッサー (CLP)	トリガー[4]	SQL ユーザー定義関数 (UDF) およびメソッド	SQL プロシージャー
ALLOCATE CURSOR					X
ALTER { BUFFERPOOL、 DATABASE PARTITION GROUP、 FUNCTION、 METHOD、 NICKNAME、[8] PROCEDURE、 SEQUENCE、 SERVER、[8] TABLE、 TABLESPACE、 TYPE、 USER MAPPING、[8] VIEW }	X				
ASSOCIATE LOCATORS					X
BEGIN DECLARE SECTION[2]					
CALL		X[7]	X	X	X
CASE ステートメント					X
CLOSE		X			X
COMMENT ON	X	X			X
COMMIT	X	X			X
コンパウンド SQL (組み込み)					
コンパウンド・ステートメント					X
CONNECT (タイプ 1)		X			
CONNECT (タイプ 2)		X			
CREATE { ALIAS、 BUFFERPOOL、 DATABASE PARTITION GROUP、 DISTINCT TYPE、 EVENT MONITOR、 FUNCTION、 FUNCTION MAPPING、[8] INDEX、 INDEX EXTENSION、 METHOD、 NICKNAME、[8] PROCEDURE、 SCHEMA、 SEQUENCE、 SERVER、 TABLE、 TABLESPACE、 TRANSFORM、 TRIGGER、 TYPE、 TYPE MAPPING、[8] USER MAPPING、[8] VIEW、 WRAPPER[8] }	X	X			X[9]
DECLARE CURSOR[2]		X			X

表 93. SQL ステートメント (DB2 Universal Database) (続き)

SQL ステートメント	動的[1]	コマンド行プロセッサー (CLP)	トリガー[4]	SQL ユーザー定義関数 (UDF) およびメソッド	SQL プロシージャー
DECLARE GLOBAL TEMPORARY TABLE	X	X			X
DELETE	X	X	X[3]	X	X
DESCRIBE[6]		X			
DISCONNECT		X			
DROP	X	X			X[9]
END DECLARE SECTION[2]					
EXECUTE					X
EXECUTE IMMEDIATE					X
EXPLAIN	X	X			X
FETCH		X			X
FLUSH EVENT MONITOR	X	X			
FLUSH PACKAGE CACHE	X	X			X
FOR ステートメント			X	X	X
FREE LOCATOR					
GET DIAGNOSTICS			X	X	X
GOTO ステートメント					X
GRANT	X	X			X
IF ステートメント			X	X	X
INCLUDE[2]					
INSERT	X	X	X[3]	X	X
ITERATE			X	X	X
LEAVE ステートメント			X	X	X
LOCK TABLE	X	X			X
LOOP ステートメント					X
MERGE	X	X	X[3]	X	X
OPEN		X			X
PREPARE					X
REFRESH TABLE	X	X			
RELEASE		X			
RELEASE SAVEPOINT	X	X			X
RENAME TABLE	X	X			
RENAME TABLESPACE	X	X			
REPEAT ステートメント					X
RESIGNAL ステートメント					X
RETURN ステートメント				X	X
REVOKE	X	X			

表 93. SQL ステートメント (DB2 Universal Database) (続き)

SQL ステートメント	動的[1]	コマンド行プロセッサー (CLP)	トリガー[4]	SQL ユーザー定義関数 (UDF) およびメソッド	SQL プロシージャー
ROLLBACK	X	X			X
SAVEPOINT	X	X			X
SELECT ステートメント	X	X	X	X	
SELECT INTO					X
SET CONNECTION		X			
SET CURRENT DEFAULT TRANSFORM GROUP	X	X			X
SET CURRENT DEGREE	X	X			X
SET CURRENT EXPLAIN MODE	X	X			X
SET CURRENT EXPLAIN SNAPSHOT	X	X			X
SET CURRENT ISOLATION	X	X			X
SET CURRENT LOCK TIMEOUT	X	X			X
SET CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION	X	X			X
SET CURRENT PACKAGE PATH					
SET CURRENT PACKAGESET					
SET CURRENT QUERY OPTIMIZATION	X	X			X
SET CURRENT REFRESH AGE	X	X			X
SET ENCRYPTION PASSWORD	X	X			X
SET EVENT MONITOR STATE	X	X			X
SET INTEGRITY	X	X			
SET PASSTHRU[8]	X	X			
SET PATH	X	X			X
SET SCHEMA	X	X			X
SET SERVER OPTION[8]	X	X			
SET SESSION AUTHORIZATION[8]	X	X			
変数の設定			X	X	X
SIGNAL ステートメント			X	X	X
SIGNAL SQLSTATE[5]	X	X			
UPDATE	X	X	X[3]	X	X
VALUES INTO					X
WHENEVER[2]					
WHILE ステートメント			X	X	X

表 93. SQL ステートメント (DB2 Universal Database) (続き)

SQL ステートメント	動的[1]	コマンド行プロセッサー (CLP)	トリガー[4]	SQL ユーザー定義関数 (UDF) およびメソッド	SQL プロシージャー

注:

1. このリストのすべてのステートメントは静的 SQL としてコーディングできますが、動的 SQL としてコーディングできるのは X マークの付いたステートメントだけです。
2. このステートメントは実行できません。
3. トリガーの前に、表データを変更することはできません。したがって、MODIFIES SQL DATA で定義された CALL プロシージャーを呼び出したり、トリガーの前に INSERT、UPDATE、DELETE、または MERGE ステートメントを使用できません。
4. トリガーの SQL ステートメントは、未定義の遷移変数、フェデレーテッドされたオブジェクト、または宣言済み一時表を参照することはできません。また、トリガー前の SQL ステートメントは、 REFRESH IMMEDIATE で定義されたマテリアライズ照会表を参照することができません。

   トリガーの制約事項についての完全なリストは、「*SQL リファレンス 第 2 巻*」の『CREATE TRIGGER ステートメント』を参照してください。
5. このステートメントは、CREATE FUNCTION、CREATE METHOD、CREATE PROCEDURE、または CREATE TRIGGER ステートメント内でのみ使用できます。
6. DESCRIBE SQL ステートメントの構文は、CLP DESCRIBE コマンドとは違っています。
7. CALL がコマンド行プロセッサーから発行された場合、以下のプロシージャーとその個別パラメーターだけがサポートされます。
8. ステートメントがサポートされるのは、フェデレーテッド・データベース・サーバーの場合だけです。
9. SQL プロシージャーが発行できるのは、索引、表、およびビュー用の CREATE および DROP ステートメントだけです。

関連資料:

- 「*SQL リファレンス 第 2 巻*」の『DESCRIBE ステートメント』
- 「*アプリケーション開発ガイド サーバー・アプリケーションのプログラミング*」の『データベース・サーバーでの JAR ファイル管理 』

付録 B. セキュリティー・プラグインのデプロイメントに関する制限

以下は、セキュリティー・プラグインの使用に関連した制限です。

DB2 Universal JDBC ドライバーのサポートに関する制限:

DB2® Universal JDBC ドライバーは、クライアント・サイドのプラグイン認証モデルをサポートしません。したがって、GSS-API 認証プラグインを使用して DB2 Universal JDBC ドライバー・クライアントから DB2 Universal Database (DB2 UDB) for Linux, UNIX®, and Windows® サーバーに接続することはできません。DB2 Universal JDBC ドライバー・クライアントは、サポートされているオペレーティング・システム・レベルの認証メカニズムか、 Kerberos 認証方式しか使用できません。この制限は、タイプ 2 とタイプ 4 の両方の接続に当てはまります。

特に、データベース・マネージャー構成パラメーター *srvcon_gssplugin_list* の値に Kerberos ベースの GSS-API プラグインの名前が同時に含まれていない場合、サーバーのデータベース・マネージャー構成パラメーター *srvcon_auth* に GSSPLUGIN を設定することはできません。

ただし *srvcon_auth* パラメーターには、 CLIENT、 SERVER、 SERVER_ENCRYPT、 KERBEROS、 KRB_SERVER_ENCRYPT、または GSS_SERVER_ENCRYPT、 DATA_ENCRYPT、または DATA_ENCRYPT_CMP のいずれかを設定できます。

DB2 UDB ファミリーのサポートに関する制限:

GSS-API プラグインを使用して、DB2 UDB for Linux, UNIX, and Windows クライアントと、他の DB2 UDB ファミリー・サーバーとの間の接続を認証することはできません。また、クライアントとして機能する他の DB2 UDB ファミリー・サーバーから DB2 UDB for Linux, UNIX, and Windows サーバーへの接続も認証できません。

ただし、DB2 UDB for Linux, UNIX, and Windows クライアントを使用して他の DB2 UDB ファミリー・サーバーに接続する場合には、 IBM® 提供のオペレーティング・システム認証プラグインなどのクライアント・サイドのユーザー ID/パスワード・プラグインを使用することや、独自のユーザー ID/パスワード・プラグインを作成することは可能です。さらに Kerberos プラグインを使用することや、独自の Kerberos プラグインをインプリメントすることもできます。

DB2 UDB for Linux, UNIX, and Windows クライアントでは、 GSSPLUGIN 認証タイプを使用してデータベースをカタログすることはできません。

DB2 Information Integrator のサポートに関する制限:

DB2 II は、GSS_API プラグインからの委任証明書を使用して、データ・ソースへのアウトバウンド接続を確立することをサポートしていません。データ・ソースへの接続には、引き続き CREATE USER MAPPING コマンドを使用する必要があります。

Database Administration Server のサポートに関する制限:

DB2 Administration Server (DAS) はセキュリティー・プラグインをサポートしていません。 DAS はオペレーティング・システムの認証メカニズムのみをサポートします。

付録 C. ホストまたは iSeries 環境でのプログラミング

ホストまたは iSeries 環境におけるアプリケーション 757
ホストまたは iSeries 環境におけるデータ定義言語 758
ホストまたは iSeries 環境におけるデータ操作言語 759
ホストまたは iSeries 環境におけるデータ制御言語 760
DB2 Connect によるデータベース接続の管理 . . . 760
割り込み要求の処理 761
パッケージ属性、PREP、および BIND 761
IBM のリレーショナル・データベース・システム間でのパッケージ属性の相違点. 761
C ヌル終了ストリング用の CNULREQD BIND オプション 762
スタンドアロン SQLCODE および SQLSTATE 変数 762
DB2 Connect でサポートされている分離レベル 763
ユーザー定義のソート順序 764
IBM のリレーショナル・データベース・システム間での参照保全の相違点. 764
ロックとアプリケーションの移植性 764
IBM のリレーショナル・データベース・システム間での SQLCODE と SQLSTATE の相違点 . . . 765
IBM のリレーショナル・データベース・システム間でのシステム・カタログの相違点 765
検索割り当て時の数値変換のオーバーフロー . . . 765
ホストまたは iSeries 環境におけるストアード・プロシージャー 766
DB2 Connect によるコンパウンド SQL のサポート 767
DB2 Connect によるマルチサイト更新 768
DB2 Connect がサポートするホストおよび iSeries サーバー SQL ステートメント 769
DB2 Connect が拒否するホストおよび iSeries サーバー SQL ステートメント 769

ホストまたは iSeries 環境におけるアプリケーション

DB2® Connect により、アプリケーション・プログラムは、System/390® サーバー、zSeries® サーバー、iSeries™ サーバー上にある DB2 データベース内のデータにアクセスできます。たとえば、Windows® 上で稼働しているアプリケーションは、DB2 Universal Database for z/OS and OS/390 データベース内のデータにアクセスできます。 iSeries 環境で稼働する新しいアプリケーションを作成するか、または既存のアプリケーションをこの環境で稼働するように修正できます。 ある環境でアプリケーションを開発し、それを別の環境に移植することもできます。

DB2 Connect™ を使用すると、 DB2 Universal Database for z/OS and OS/390 などのホスト・データベース製品でサポートされている場合、以下の API をホストとともに使用することができます。

- 静的および動的 SQL
- DB2 コール・レベル・インターフェース
- Microsoft® ODBC API
- JDBC

SQL ステートメントの中には、リレーショナル・データベース製品により異なるものもあります。以下のような SQL ステートメントに遭遇する場合があります。

- どの標準に準拠しているかにかかわらず、使用するデータベース製品すべてで同一の SQL ステートメント
- すべての IBM® リレーショナル・データベース製品で使用可能な SQL ステートメント (詳しくは、お手持ちの SQL リファレンスの情報を参照)
- アクセスするデータベース・システムにユニークな SQL ステートメント

初めの 2 つのカテゴリーの SQL ステートメントには高い移植性がありますが、 3 番目のカテゴリーのものは変更を加える必要があります。一般に、データ定義言語 (DDL) 内の SQL ステートメントは、データ操作言語 (DML) 内の SQL ステートメントよりも移植性が低くなります。

DB2 Connect は、DB2 Universal Database ではサポートされていない一部の SQL ステートメントを受け入れます。 DB2 Connect は、これらのステートメントを iSeries サーバーに渡します。最大列長のような別のプラットフォームの制限に関する情報は、 SQL 制限に関するトピックを参照してください。

別の CICS 製品 (たとえば、CICS for AIX) の下で実行するために CICS® アプリケーションを OS/390® または VSE から移動する場合、 DB2 Connect を使用して OS/390 または VSE データベースにアクセスすることもできます。詳細については、「*CICS/6000 Application Programming Guide*」および「*CICS Customization and Operation*」を参照してください。

注: DB2 Universal Database バージョン 8 データベースで DB2 Connect を使用することができます。ただし、必要なのは DB2 クライアントです。 DB2 Connect を DB2 Universal Database バージョン 8 データベースに対して使用した場合、以下のトピックにリストする非互換性の問題の多くは該当しません。ただし、DB2 Connect 自体の制限により制約事項が生じる場合は除きます。

関連タスク:

- 「*アプリケーション開発ガイド アプリケーションの構築および実行*」の『ホスト・サーバーまたは AS/400 および iSeries サーバーでのサンプル・データベースの作成』

関連資料:

- 「*SQL リファレンス 第 1 巻*」の『SQL の制限値』

ホストまたは iSeries 環境におけるデータ定義言語

それぞれのシステムでストレージの処理方法が違っているため、 DDL ステートメントは IBM® データベース製品により異なります。 iSeries™ サーバー・システム上では、データベースを設計してから CREATE TABLE ステートメントを発行するまでにいくつかの手順を踏むことが必要な場合もあります。 たとえば論理オブジェクトの設計が、一連のステートメントによりそれらのオブジェクトのストレージでの物理表現に変換されます。

iSeries サーバー・データベースにプリコンパイルを行う場合、プリコンパイラーは iSeries サーバーに多数の DDL ステートメントを渡します。 アプリケーションを実行中のシステムのデータベースには、同一ステートメントはプリコンパイルされません。たとえば、Windows® アプリケーションでは、 CREATE STORGROUP ステートメントは DB2 Universal Database for z/OS and OS/390 データベースを正常にプリコンパイルしますが、 DB2® (Windows 版) データベースはプリコンパイルしません。

ホストまたは iSeries 環境におけるデータ操作言語

一般に、DML ステートメントは高い移植性を持ちます。 SELECT、 INSERT、 UPDATE、および DELETE ステートメントは、 IBM® リレーショナル・データベース製品すべてで同様に使用されます。ほとんどのアプリケーションは主に DML SQL ステートメントを使用しています。それらは DB2® Connect によりサポートされています。

以下に示すのは、ホストおよび iSeries™ 環境で DML を使用する際の考慮事項です。

- 数値™データ・タイプ

  数値データが DB2 Universal Database に転送されると、データ・タイプが変更されることがあります。数値およびゾーン 10 進数 SQLTYPE 値 (OS/400® でサポート) は、固定 (パック) 10 進数 SQLTYPE 値に変換されます。

- 混合バイト・データ

  混合バイト・データは、同一の列にある拡張 UNIX® コード (EUC) 文字セット、 2 バイト文字セット (DBCS)、および 1 バイト文字セット (SBCS) の文字からなります。 EBCDIC でデータを保管するシステム (OS/390、z/OS™、 OS/400、 VSE、および VM) では、シフトアウト文字とシフトイン文字により 2 バイト・データの始点と終点がマークされます。データを ASCII 形式で保管するシステム (UNIX など) では、シフトイン文字およびシフトアウト文字は必要ありません。

  アプリケーションが混合バイト・データを ASCII システムから EBCDIC システムに転送する場合、シフト文字を入れる余地を確保してください。それぞれのデータを SBCS データから DBCS データへ切り替える際に、データ長に 2 バイトが追加されます。移植性を高めるためには、混合バイト・データを使用するアプリケーションにおいて可変長文字ストリングを使用してください。

- 長フィールド

  長フィールド (254 文字より長いストリング) は、別のシステムでは扱いが異なります。ホストまたは iSeries サーバーでは長形式フィールドにはスカラー関数のサブセットしかサポートしていません。たとえば、DB2 Universal Database for z/OS and OS/390 では長形式フィールドには **LENGTH** および **SUBSTR** 関数しか使用できません。また、iSeries サーバーでは特定の SQL ステートメントに対して異なる処理が必要になります。たとえば、DB2 Server for VSE & VM では INSERT ステートメントにはホスト変数、SQLDA、または NULL 値しか使用できません。

- ラージ・オブジェクト (LOB) データ・タイプ

  LOB データ・タイプは DB2 Connect によってサポートされます。

- ユーザー定義タイプ

  DB2 Connect は、ユーザー定義特殊タイプだけをサポートしています。構造タイプ (抽象データ・タイプともいう) は、 DB2 Connect ではサポートされていません。

- ROWID データ・タイプ

  ROWID データ・タイプは、 DB2 Connect によってビット・データ用の VARCHAR として扱われます。
- BIGINT データ・タイプ

  8 バイト (64 ビット) 整数は、DB2 Connect によってサポートされます。 BIGINT 内部データ・タイプは、データの精度を保ちながら大規模データベースのカーディナリティーをサポートするために使われます。

ホストまたは iSeries 環境におけるデータ制御言語

それぞれの IBM® リレーショナル・データベース管理システムは、 GRANT および REVOKE SQL ステートメントにさまざまなレベルの細分性を提供しています。各データベース管理システムに合った SQL ステートメントを確かめるには、それぞれの製品の資料を調べてみてください。

DB2 Connect によるデータベース接続の管理

DB2® Connect は、パラメーターなしの CONNECT だけでなく、 CONNECT TO および CONNECT RESET もサポートしています。 アプリケーションが明示的な CONNECT TO ステートメントを実行せずに SQL ステートメントを呼び出すと、デフォルトのアプリケーション・ サーバー (定義されている場合) への*暗黙の* 接続が実行されます。

データベースに接続すると、 SQLCA の SQLERRP フィールドにリレーショナル・データベース管理システムを識別する情報が戻されます。アプリケーション・サーバーが IBM® のリレーショナル・データベースである場合、 SQLERRP の先頭の 3 文字には次のいずれかが含まれます。

DSN DB2 Universal Database for z/OS and OS/390

ARI DB2 Server for VSE & VM

QSQ DB2 UDB for iSeries™

SQL DB2 Universal Database

DB2 Connect™ を使用しているときに CONNECT TO またはヌル CONNECT ステートメントを発行した場合、 SQLCA の SQLERRMC フィールド内にはテリトリー・コードまたはテリトリー・トークンがブランクとして戻されます。アプリケーション・サーバーの CCSID はコード・ページまたはコード・セット・トークンに戻されます。

CONNECT RESET ステートメント (タイプ 1 の接続の場合)、 RELEASE および COMMIT ステートメント (タイプ 2 の接続の場合)、または DISCONNECT ステートメント (いずれのタイプの接続にも使用できるが、 TP モニター環境では使用できない) を使用して、明示的に切断を行えます。

注: アプリケーションがエラーを示す SQLCODE 値を受け取っても、正常に終了することができます。この場合、DB2 Connect はデータをコミットします。データをコミットしたくない場合は、ROLLBACK コマンドを使用してください。

FORCE コマンドを使用すると、選択したユーザーまたはすべてのユーザーをデータベースから切断することができます。これは、ホストおよび iSeries サーバー・データベースでサポートされています。ユーザーは、DB2 Connect ワークステーションから強制的に切断されます。

関連資料:

- 「*SQL リファレンス 第 2 巻*」の『CONNECT (タイプ 1) ステートメント』
- 「*SQL リファレンス 第 2 巻*」の『CONNECT (タイプ 2) ステートメント』

割り込み要求の処理

DB2® Connect は、DB2 クライアントからの割り込み要求を、以下の 2 つの方法のいずれかで処理します。

- DCS カタログ項目の PARMS フィールドにキーワード INTERRUPT_ENABLED が存在していれば、 DB2 Connect™ は割り込み要求の受信時にホストまたは iSeries™ サーバーへの接続をドロップします。接続の消失により、少なくとも DB2 UDB for OS/390® および z/OS™ サーバーでは、現在の要求に対する割り込みがサーバーで行われます。
- DCS カタログ項目の PARMS フィールドにキーワード INTERRUPT_ENABLED が存在していなければ、割り込み要求は無視されます。

パッケージ属性、PREP、および BIND

以下の節では、 IBM のリレーショナル・データベース・システム間でのパッケージ属性の相違と、 PREPCOMPILE および BIND コマンドに関する考慮事項について説明します。

IBM のリレーショナル・データベース・システム間でのパッケージ属性の相違点

パッケージには、以下の属性があります。

コレクション ID
パッケージの ID です。 PREP コマンドで指定できます。

所有者 パッケージの所有者の許可 ID です。 PREP または BIND コマンドで指定できます。

作成者 パッケージをバインドするユーザー名です。

修飾子 パッケージ内のオブジェクトの暗黙修飾子です。 PREP または BIND コマンドで指定できます。

それぞれの iSeries™ サーバー・システムには、これらの属性の使用に制限があります。

DB2 Universal Database for z/OS and OS/390

4 つの属性すべてが異なってもかまいません。異なる修飾子を使用するには、特別の管理特権が必要です。これらの属性の使用条件の詳細については、DB2 Universal Database for z/OS and OS/390 用の「コマンド・リファレンス」を参照してください。

DB2 Server for VSE & VM

すべての属性が一致しなければなりません。 USER1 がバインド・ファイルを作成し (PREP を使用して)、 USER2 が実際にバインドを実行する場合、USER2 は USER1 のバインドに対する DBA 権限が必要です。属性に使用できるのは USER1 のユーザー名だけです。

DB2® UDB for iSeries

修飾子が集合名と一致しなければなりません。修飾子と所有権の関係は、オブジェクトに対する特権の付与および取り消しに影響します。ログオンされたユーザー名はコレクション ID で修飾されていない限り、作成者または所有者になります。コレクション ID で修飾されている場合は、コレクション ID が所有者です。コレクション ID は、修飾子として使用される以前にすでに存在していなければなりません。

DB2 Universal Database

4 つの属性すべてが異なってもかまいません。異なる所有者の使用には管理権限が必要であり、バインド・プログラムは (スキーマがすでに存在すれば) スキーマに対する CREATEIN 特権を持っていなければなりません。

C ヌル終了ストリング用の CNULREQD BIND オプション

CNULREQD BIND オプションは、LANGLEVEL オプションを使用して指定されたヌル終了ストリングの処理を一時変更します。

CNULREQD デフォルト設定は YES です。このオプションが YES に設定されている場合、ヌル終了ストリングは MIA 規格に準拠して解釈されます。 DB2 Universal Database for z/OS and OS/390 サーバーに接続している場合には、CNULREQD を YES に設定するよう強くお勧めします。 (ヌル終了ストリングに関して) SAA1 規格でコーディングされたアプリケーションをバインドする場合は、 CNULREQD オプションを NO に設定する必要があります。 NO に設定しなかった場合、LANGLEVEL を SAA1 に設定して入力したとしても、ヌル終了ストリングは MIA 規格に準拠して解釈されてしまいます。

関連概念:

• 175 ページの『C および C++ でのヌル終了ストリング』

スタンドアロン SQLCODE および SQLSTATE 変数

ISO/ANS SQL92 で定義されているスタンドアロンの SQLCODE および SQLSTATE 変数は、 LANGLEVEL SQL92E プリコンパイル・オプションによってサポートされています。 LANGLEVEL がサポートされていなければ、プリコンパイル時にそのことを示す SQL0020W 警告が出されます。この警告は LANGLEVEL SQL92E のサブセットである LANGLEVEL MIA の下にリストされている機能にのみ適用されます。

関連資料:

- 「コマンド・リファレンス」の『PRECOMPILE コマンド』

DB2 Connect でサポートされている分離レベル

DB2 Connect は、アプリケーションを準備 (prep) またはバインド (bind) するときに、以下の分離レベルを受け入れます。

RR 反復可能読み取り

RS 読み取り固定

CS カーソル固定

UR 非コミット読み取り

NC コミットなし

分離レベルは、最も厳重な保護から最も緩い保護へという順番でリストされています。ユーザーが指定した分離レベルを iSeries™ サーバーがサポートしていない場合、サポートされているレベルの中で、指定の分離レベルの次に保護が厳重なものが使用されます。

以下の表は、各ホストまたは iSeries アプリケーション・サーバーにおけるそれぞれの分離レベルの結果を示しています。

表 94. 分離レベル

DB2 Connect	DB2 Universal Database for z/OS and OS/390	DB2 Server for VSE & VM	DB2® UDB for iSeries	DB2 Universal Database
RR	RR	RR	注 1	RR
RS	注 2	RR	COMMIT(*ALL)	RS
CS	CS	CS	COMMIT(*CS)	CS
UR	注 3	CS	COMMIT(*CHG)	UR
NC	注 4	注 5	COMMIT(*NONE)	UR

注:

1. DB2 UDB for iSeries には RR に一致する同等の COMMIT オプションはありません。 DB2 UDB for iSeries は表全体をロックすることにより RR をサポートします。
2. バージョン 3.1 では RR であったものは、バージョン 4.1 (APAR PN75407) またはバージョン 5.1 では RS になります。
3. バージョン 3.1 では CS であったものは、バージョン 4.1 またはバージョン 5.1 では UR になります。
4. バージョン 3.1 では CS であったものは、バージョン 4.1 (APAR PN60988) またはバージョン 5.1 では UR になります。
5. DB2 Server for VSE & VM では、分離レベル NC はサポートされません。

DB2 UDB for iSeries では、分離レベルが UR でブロック化が ALL にセットされていてアプリケーションがバインドされているならば、もしくは分離レベルが NC にセットされているならば、ジャーナルされていない表にアクセスすることができます。

ユーザー定義のソート順序

EBCDIC と ASCII の違いは、さまざまなデータベース製品においてソート順序の違いの原因となり、また ORDER BY および GROUP BY 文節に影響を与えます。この差を最小化するための 1 つの方法は、 EBCDIC のソート順序を模倣したユーザー定義照合順序を作成することです。照合順序を指定できるのは新しいデータベースの作成時のみです。

注: データベース表は、DB2 Universal Database for z/OS and OS/390 に ASCII 形式で保管されています。このため、DB2 Connect と DB2 Universal Database for z/OS and OS/390 との間の変換が高速になり、データを変換して並べ直すときに使用しなければならないフィールド手順を実行する必要はなくなります。

IBM のリレーショナル・データベース・システム間での参照保全の相違点

システムにより、参照保全の処理方法が異なります。

DB2 Universal Database for z/OS and OS/390
主キーを使用して外部キーを作成できるように、主キーに索引を付けなければなりません。表は自己参照が可能です。

DB2 Server for VSE & VM
外部キーには自動的に索引が付けられます。表は自己参照を行えません。

DB2® UDB for iSeries™
外部キーには自動的に索引が付けられます。表は自己参照が可能です。

DB2 Universal Database
DB2 Universal Database のデータベースの場合、索引は、主キーを含むユニーク制約に対して自動的に作成されます。表は自己参照が可能です。

その他の規則は、カスケードのレベルによって異なります。

ロックとアプリケーションの移植性

データベース・サーバーがロックを実行する方法は、一部のアプリケーションに影響を与えることがあります。たとえば、行レベルのロックおよびカーソル固定の分離レベルで設計されているアプリケーションは、ページ・レベルのロックを実行しているシステムに直接移植することはできません。このような基礎的な差があるため、アプリケーションを調整する必要があります。

DB2 Universal Database for z/OS and OS/390 および DB2 Universal Database 製品には、ロックをタイムアウトにし、待機中のアプリケーションにエラー戻りコードを送信する機能があります。

IBM のリレーショナル・データベース・システム間での SQLCODE と SQLSTATE の相違点

それぞれの IBM® リレーショナル・データベース製品は、類似したエラーに対して必ずしも同一の SQLCODE 値を出すとは限りません。この問題は次の 2 通りの方法で解決できます。

- アプリケーション・エラーに対し、SQLCODE ではなく SQLSTATE を使用する。

  SQLSTATE 値は、さまざまなデータベース製品間でほぼ同じ意味を持ち、これらの製品は SQLCODE 値に対応した SQLSTATE 値を出します。
- SQLCODE 値をあるシステムから別のシステムへマップする。

  デフォルト設定では、DB2® Connect は SQLCODE 値とトークンを、それぞれの IBM ホストまたは iSeries™ サーバー・システムからユーザーの DB2 Universal Database システムにマップします。デフォルト・マッピングを一時変更したい場合、または SQLCODE マッピングを持たないデータベース・サーバー (IBM 以外のデータベース・サーバー) を使用している場合に、独自の SQLCODE マッピング・ファイルを指定することができます。 SQLCODE マッピングを作動させないでおくこともできます。

関連概念:

- 「*DB2 Connect ユーザーズ・ガイド*」の『SQLCODE マッピング』

IBM のリレーショナル・データベース・システム間でのシステム・カタログの相違点

システム・カタログは IBM® のデータベース製品により異なります。 たいていの差異は、ビューを使用することによりマスクされます。詳しくは、使用しているデータベース・サーバーの資料を参照してください。

CLI にあるカタログ機能は、同じ API にサポートされることにより、この問題を克服し、 DB2® ファミリー全体のカタログ照会を設定していきます。

関連概念:

- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 1 巻*」の『CLI アプリケーションでのシステム・カタログ情報の照会のためのカタログ関数』

検索割り当て時の数値変換のオーバーフロー

検索割り当て時に数値変換のオーバーフローがある場合、 IBM® リレーショナル・データベース製品によってその処理は異なります。たとえば、DB2 Universal Database for z/OS and OS/390 および DB2 Universal Database から、浮動列を整数のホスト変数として取り出すとします。浮動値を整数値に変換するときに、変換のオーバーフローが生じることがあります。デフォルトには、DB2 Universal Database for z/OS and OS/390 は警告の SQLCODE と NULL 値をアプリケーションに戻し

ます。対照的に、DB2 Universal Database は変換オーバーフロー・エラーを戻します。適切なサイズのホスト変数にして取り出すことにより、アプリケーション側で検索割り当て時の数値変換のオーバーフローを避けるようにお勧めします。

ホストまたは iSeries 環境におけるストアード・プロシージャー

ホストおよび iSeries™ 環境におけるストアード・プロシージャーに関する考慮事項は以下のとおりです。

- 呼び出し

  クライアント側のプログラムは、SQL CALL ステートメントを出して、サーバー側のプログラムを呼び出すことができます。この場合、各サーバーの作業はその他のサーバーの作業と若干異なります。

  z/OS™ および OS/390®
  スキーマ名は 8 バイト以下の長さでなければならず、プロシージャー名は 18 バイト以下の長さでなければならず、ストアード・プロシージャーはサーバー上の SYSIBM.SYSPROCEDURES カタログに定義しなければなりません。

  VSE または VM
  プロシージャー名は 18 バイト以上の長さでなければならず、サーバー上の SYSTEM.SYSROUTINES カタログに定義しなければなりません。

  OS/400®
  プロシージャー名は SQL の ID でなければなりません。 DECLARE PROCEDURE または CREATE PROCEDURE ステートメントを使用し、ストアード・プロシージャーを突き止めるために実際のパス名 (スキーマ名または集合名) を指定することもできます。

  REXX/SQL から DB2® UDB for iSeries に対する CALL ステートメントはすべて、 CALL USING DESCRIPTOR 形式の REXX/SQL マップで設定された CALL ステートメントとしてアプリケーションにより動的に作成され、実行される必要があります。

  DB2 Universal Database 上のサーバー・プログラムは、DB2 Universal Database for z/OS and OS/390、DB2 UDB for iSeries、または DB2 Server for VSE & VM で使用されるサーバー・プログラムと同じパラメーター規則を指定して起動できます。他のプラットフォームのパラメーターの規則の詳細については、そのプラットフォームの DB2 製品の資料を参照してください。

  ストアード・プロシージャー内の全 SQL ステートメントは、クライアント側の SQL プログラムにより開始される SQL 作業単位の一部として実行されます。
- ストアード・プロシージャーとの間で、特別の意味を持つインディケーター値をやりとりしない。

  DB2 Universal Database との間で、システムはなんであれ標識変数に指定されたものを渡します。ただし、DB2 Connect™ を使用している場合は、0、-1、および -128 以外の値を標識変数に渡すことはできません。
- サーバー側のアプリケーションで発生したエラーまたは警告を戻すパラメーターを定義する。

DB2 Universal Database 上のサーバー・プログラムは、 SQLCA を更新してエラーや警告をすべて返すことができますが、 DB2 Universal Database for z/OS and OS/390 または DB2 UDB for iSeries 上のストアード・プロシージャーにはそのような機能はありません。ストアード・プロシージャーからエラー・コードを戻したい場合は、これをパラメーターとして渡してください。サーバーが SQLCODE および SQLCA にセットできるエラーは、システムが検出したエラーだけです。

- 現時点では、ストアード・プロシージャーの結果セットを返すことができるホストまたは iSeries アプリケーション・サーバーは、 DB2 Server for VSE & VM バージョン 7 以上、DB2 Universal Database for z/OS and OS/390 バージョン 5.1 以上、DB2 (AS/400® 版) V5R1、および DB2 for iSeries バージョン 7 以上だけです。

関連概念:

- 21 ページの『DB2 ストアード・プロシージャー』

関連資料:

- 「*SQL リファレンス 第 2 巻*」の『CALL ステートメント』

DB2 Connect によるコンパウンド SQL のサポート

コンパウンド SQL を使用すると、複数の SQL ステートメントをグループ化して単一の実行可能ブロックにすることができます。これによりネットワークのオーバーヘッドが減少し、応答時間が短縮されます。

NOT ATOMIC コンパウンド SQL を使用した場合、エラーの発生した後も、コンパウンド SQL の処理が継続されます。 ATOMIC コンパウンド SQL を使用した場合、エラーになったときにコンパウンド SQL のグループ全体がロールバックされてしまいます。

ステートメントは、アプリケーション・サーバーによって終了されるまで継続して実行されます。一般に、コンパウンド SQL ステートメントの実行は重大エラーの場合にのみサポートされます。

NOT ATOMIC コンパウンド SQL は、サポートされる iSeries™ アプリケーション・サーバーすべてで使用できます。 ATOMIC コンパウンド SQL は、サポートされるホスト・アプリケーション・サーバーすべてで使用できます。

複数の SQL エラーが発生した場合、先頭から順に 7 つまでの失敗したステートメントの SQLSTATE 値が、複数のエラーが発生したことを示すメッセージとともに SQLCA の SQLERRMC フィールドに戻されます。

関連資料:

- 「*管理 API リファレンス*」の『SQLCA』

DB2 Connect によるマルチサイト更新

DB2® Connect を使用すると、マルチサイト更新 (2 フェーズ・コミットとしても知られる) を実行できます。 マルチサイト更新とは、 1 つの分散作業単位 (DUOW) 内で複数のデータベースを更新することです。この機能を使用できるかどうかには、以下のいくつかの要素が関係しています。

- ご使用のアプリケーション・プログラムは、 CONNECT 2 および SYNCPOINT TWOPHASE オプションを指定してプリコンパイルされていなければなりません。
- SNA ネットワーク接続の場合、DB2 Connect™ Enterprise Edition (AIX® 版および Windows® NT 版) の DB2 同期点マネージャー (SPM) 機能で提供される、 2 フェーズ・コミット・サポートを使用することができます。この機能を使用すると、以下のホスト・データベース・サーバーが 1 つの分散作業単位に参加できるようになります。
  - DB2 (AS/400® 版) バージョン 3.1 またはそれ以降
  - DB2 UDB (iSeries™ 版) バージョン 5.1 またはそれ以降
  - DB2 (OS/390® 版) バージョン 5.1 またはそれ以降
  - DB2 UDB (OS/390 および z/OS™ 版) バージョン 7 またはそれ以降
  - DB2 (VSE および VM 版) バージョン 5.1 またはそれ以降

  以上はネイティブの DB2 UDB アプリケーション、および外部の TP モニター (IBM® TXSeries®、CICS® for Open Systems、 BEA Tuxedo、Encina® Monitor、および Microsoft® Transaction Server など) によって調整されているアプリケーションに当てはまります。
- TCP/IP ネットワークに接続している場合は、 DB2 (OS/390 版) バージョン 5.1 またはそれ以降のサーバーが分散作業単位に参加できます。 IBM TXSeries、CICS for Open Systems、 Encina Monitor、または Microsoft Transaction Server などのトランザクション処理モニターを使ってアプリケーションを制御している場合は、 SPM を使用する必要があります。

  TCP/IP 接続でホスト・データにアクセスするのにネイティブ DB2 アプリケーションと TP モニター・アプリケーションの両方が共通 DB2 Connect Enterprise Edition サーバーを使用しているなら、同期点マネージャーは必ず使用しなければなりません。

  1 つの DB2 Connect Enterprise Edition サーバーから SNA と TCP/IP ネットワーク・プロトコルの両方を使ってホスト・データにアクセスし、 2 フェーズ・コミットが必要な場合は、SPM を使用しなければなりません。これは、DB2 アプリケーションと TP モニター・アプリケーションの両方に当てはまります。

関連概念:

- 「*管理ガイド: プランニング*」の『DB2 Universal Database によってサポートされる XA 機能』
- 「*DB2 Connect ユーザーズ・ガイド*」の『DB2 Connect と XA 準拠トランザクション・マネージャーとの構成』

関連タスク:

- 「*管理ガイド: プランニング*」の『BEA Tuxedo の構成』

- 「*管理ガイド: プランニング*」の『XA 準拠のトランザクション・マネージャーを使用したホストまたは iSeries データベース・サーバーの更新』

DB2 Connect がサポートするホストおよび iSeries サーバー SQL ステートメント

以下のステートメントはホストおよび iSeries™ サーバー処理では正常にコンパイルされますが、 DB2 Universal Database システムでの処理用には正常にコンパイルされません。

- ACQUIRE
- DECLARE (modifier.(qualifier.)table_name TABLE ...
- LABEL ON

これらのステートメントはコマンド行プロセッサーでもサポートされています。

以下のステートメントは、ホストおよび iSeries サーバー処理についてはサポートされていますが、バインド・ファイルまたはパッケージには追加されません。また、コマンド行プロセッサーはこれらのステートメントをサポートしません。

- DESCRIBE statement_name INTO descriptor_name USING NAMES
- PREPARE statement_name INTO descriptor_name USING NAMES FROM ...

プリコンパイラーは、以下の条件を前提事項としています。

- ホスト変数に変数が入力されている。
- ステートメントにユニークなセクション番号が割り当てられている。

DB2 Connect が拒否するホストおよび iSeries サーバー SQL ステートメント

以下の SQL ステートメントは、DB2® Connect でもコマンド行プロセッサーでもサポートされていません。

- COMMIT WORK RELEASE
- DECLARE state_name, statement_name STATEMENT
- DESCRIBE statement_name INTO descriptor_name USING xxxx (xxxx は ANY、BOTH、または LABELS です)
- PREPARE statement_name INTO descriptor_name USING xxxx FROM :host_variable (xxxx は ANY、BOTH、または LABELS です)
- PUT ...
- ROLLBACK WORK RELEASE
- SET :host_variable = CURRENT ...

DB2 (VSE および VM 版) の拡張動的 SQL ステートメントは拒否され、 -104 および構文エラーの SQLCODE 値が出されます。

付録 D. EBCDIC バイナリー照合のシミュレート

DB2® では、ユーザー定義の照合順序に従って文字ストリングを照合できます。この機能は、EBCDIC バイナリー照合のシミュレートに使えます。

EBCDIC 照合をシミュレートする方法の一例として、コード・ページ 850 の ASCII データベースを作成し、コード・ページ 500 の EBCDIC データベースに実際にデータが存在することを前提として文字ストリングを照合するとしましょう。コード・ページ 500 およびコード・ページ 850 の定義については、以下の図を参照してください。

EBCDIC コード・ページ 500 のデータベースにある 4 つの文字を相対的に照合することについて考えます。これらの文字は次のようにバイナリーで照合されています。

文字	コード・ページ 500 コード・ポイント
'a'	X'81'
'b'	X'82'
'A'	X'C1'
'B'	X'C2'

コード・ページ 500 のバイナリー照合順序 (希望する順序) は、次のとおりです。

```
'a' < 'b' < 'A' < 'B'
```

データベースを ASCII コード・ページ 850 で作成する場合、バイナリー照合の結果は以下のようになります。

文字	コード・ページ 850 コード・ポイント
'a'	X'61'
'b'	X'62'
'A'	X'41'
'B'	X'42'

コード・ページ 850 の バイナリー照合 (希望する順序ではない) は、次のとおりです。

```
'A' < 'B' < 'a' < 'b'
```

希望する順序を実現するには、ユーザー定義照合順序でデータベースを作成する必要があります。 `sqle850a.h` 組み込みファイルの DB2 には、まさにこの目的のために照合順序のサンプルが付属しています。 `sqle850a.h` の内容は、以下に示されています。

```
#ifndef SQL_H_SQLE850A
#define SQL_H_SQLE850A

#ifdef __cplusplus
extern "C" {
#endif

unsigned char sqle_850_500[256] = {
0x00,0x01,0x02,0x03,0x37,0x2d,0x2e,0x2f,0x16,0x05,0x25,0x0b,0x0c,0x0d,0x0e,0x0f,
0x10,0x11,0x12,0x13,0x3c,0x3d,0x32,0x26,0x18,0x19,0x3f,0x27,0x1c,0x1d,0x1e,0x1f,
0x40,0x4f,0x7f,0x7b,0x5b,0x6c,0x50,0x7d,0x4d,0x5d,0x5c,0x4e,0x6b,0x60,0x4b,0x61,
0xf0,0xf1,0xf2,0xf3,0xf4,0xf5,0xf6,0xf7,0xf8,0xf9,0x7a,0x5e,0x4c,0x7e,0x6e,0x6f,
0x7c, 0xc1 , 0xc2 ,0xc3,0xc4,0xc5,0xc6,0xc7,0xc8,0xc9,0xd1,0xd2,0xd3,0xd4,0xd5,0xd6,
0xd7,0xd8,0xd9,0xe2,0xe3,0xe4,0xe5,0xe6,0xe7,0xe8,0xe9,0x4a,0xe0,0x5a,0x5f,0x6d,
0x79, 0x81 , 0x82 ,0x83,0x84,0x85,0x86,0x87,0x88,0x89,0x91,0x92,0x93,0x94,0x95,0x96,
0x97,0x98,0x99,0xa2,0xa3,0xa4,0xa5,0xa6,0xa7,0xa8,0xa9,0xc0,0xbb,0xd0,0xa1,0x07,
0x68,0xdc,0x51,0x42,0x43,0x44,0x47,0x48,0x52,0x53,0x54,0x57,0x56,0x58,0x63,0x67,
0x71,0x9c,0x9e,0xcb,0xcc,0xcd,0xdb,0xdd,0xdf,0xec,0xfc,0x70,0xb1,0x80,0xbf,0xff,
0x45,0x55,0xce,0xde,0x49,0x69,0x9a,0x9b,0xab,0xaf,0xba,0xb8,0xb7,0xaa,0x8a,0x8b,
0x2b,0x2c,0x09,0x21,0x28,0x65,0x62,0x64,0xb4,0x38,0x31,0x34,0x33,0xb0,0xb2,0x24,
0x22,0x17,0x29,0x06,0x20,0x2a,0x46,0x66,0x1a,0x35,0x08,0x39,0x36,0x30,0x3a,0x9f,
0x8c,0xac,0x72,0x73,0x74,0x0a,0x75,0x76,0x77,0x23,0x15,0x14,0x04,0x6a,0x78,0x3b,
0xee,0x59,0xeb,0xed,0xcf,0xef,0xa0,0x8e,0xae,0xfe,0xfb,0xfd,0x8d,0xad,0xbc,0xbe,
0xca,0x8f,0x1b,0xb9,0xb6,0xb5,0xe1,0x9d,0x90,0xbd,0xb3,0xda,0xfa,0xea,0x3e,0x41
};
#ifdef __cplusplus
}
#endif

#endif /* SQL_H_SQLE850A */
```

図 66. ユーザー定義の照合順序 - sqle_850_500

コード・ページ 500 の バイナリー照合をコード・ページ 850 の文字で実現する方法を確かめるには、sqle_850_500 にある照合順序のサンプルを調べてください。コード・ページ 850 のそれぞれの文字の照合順序における重要度は、コード・ページ 500 の対応するコード・ポイントに相当する以上のものではありません。

たとえば、文字 'a' を考えてみましょう。この文字は、コード・ページ 850 ではコード・ポイント X'61' になります。配列 sqle_850_500 で、文字 'a' には X'81' の重要度 (つまり、配列 sqle_850_500 の 98 番目のエレメント) が割り当てられています。

データベースが上記のサンプルに示したユーザー定義の照合順序で作成されている場合、これら 4 つの文字の照合方法はどうなるか考慮してみます。

文字	コード・ページ 850 コード・ポイント/重要度 (sqle_850_500 からの)
'a'	X'61' /X'81'
'b'	X'62' /X'82'
'A'	X'41' /X'C1'
'B'	X'42' /X'C2'

重要度別のコード・ページ 850 のユーザー定義照合 (希望する照合) は、次のとおりです。

```
'a' < 'b' < 'A' < 'B'
```

この例では、正確な重要度を指定して希望する照合を実現し、希望する動作をシミュレートしています。

実際の照合順序を注意深く観察すると、順序そのものは単に遷移表になっているだけで、ソース・コード・ページがデータベースのコード・ページ (850)、ターゲット・コード・ページが希望するバイナリー照合コード・ページ (500) であることが分かります。 DB2 提供の他の照合順序のサンプルにより、さまざまな変換が可能になります。必要な変換テーブルが DB2 で提供されていない場合は、 IBM® 資料「*Character Data Representation Architecture, Reference and Registry* (SC09-2190)」から追加の変換テーブルを入手できます。 追加の変換テーブルは、この資料に同梱されている CD-ROM に収録されています。

HEX DIGITS 1ST → 2ND ↓	4-	5-	6-	7-	8-	9-	A-	B-	C-	D-	E-	F-
-0	(SP) SP010000	& SM030000	- SP010000	ø LO610000	Ø LO620000	° SM190000	µ SM170000	¢ SC040000	{ SM110000	} SM140000	\ SM070000	0 ND100000
-1	(RSP) SP300000	é LE110000	/ SP120000	É LE120000	a LA010000	j LJ010000	~ SD190000	£ SC020000	A LA020000	J LJ020000	÷ SA060000	1 ND010000
-2	â LA150000	ê LE150000	Â LA160000	Ê LE160000	b LB010000	k LK010000	s LS010000	¥ SC050000	B LB020000	K LK020000	S LS020000	2 ND020000
-3	ä LA170000	ë LE170000	Ä LA180000	Ë LE180000	c LC010000	l LL010000	t LT010000	· SD630000	C LC020000	L LL020000	T LT020000	3 ND030000
-4	à LA130000	è LE130000	À LA140000	È LE140000	d LD010000	m LM010000	u LU010000	© SM520000	D LD020000	M LM020000	U LU020000	4 ND040000
-5	á LA110000	í LI110000	Á LA120000	Í LI120000	e LE010000	n LN010000	v LV010000	§ SM240000	E LE020000	N LN020000	V LV020000	5 ND050000
-6	ã LA190000	î LI150000	Ã LA200000	Î LI160000	f LF010000	o LO010000	w LW010000	¶ SM250000	F LF020000	O LO020000	W LW020000	6 ND060000
-7	å LA270000	ï LI170000	Å LA280000	Ï LI180000	g LG010000	p LP010000	x LX010000	¼ NF040000	G LG020000	P LP020000	X LX020000	7 ND070000
-8	ç LC410000	ì LI130000	Ç LC420000	Ì LI140000	h LH010000	q LQ010000	y LY010000	½ NF010000	H LH020000	Q LQ020000	Y LY020000	8 ND080000
-9	ñ LN190000	ß LS610000	Ñ LN200000	` SD130000	i LI010000	r LR010000	z LZ010000	¾ NF050000	I LI020000	R LR020000	Z LZ020000	9 ND090000
-A	[SM060000	] SM080000	¦ SM650000	: SP130000	« SP170000	ª SM210000	¡ SP030000	¬ SM660000	(SHY) SP320000	1 ND011000	2 ND021000	3 ND031000
-B	. SP110000	$ SC030000	, SP080000	# SM010000	» SP180000	º SM200000	¿ SP160000	‾ SM130000	ô LO150000	û LU150000	Ô LO160000	Û LU160000
-C	< SA030000	* SM040000	% SM020000	@ SM050000	ð LD630000	æ LA510000	Ð LD620000	¯ SM150000	ö LO170000	ü LU170000	Ö LO180000	Ü LU180000
-D	(SP060000	) SP070000	_ SP090000	' SP050000	ý LY110000	¸ SD410000	Ý LY120000	¨ SD170000	ò LO130000	ù LU130000	Ò LO140000	Ù LU140000
-E	+ SA010000	; SP140000	> SA050000	= SA040000	þ LT630000	Æ LA520000	Þ LT640000	´ SD110000	ó LO110000	ú LU110000	Ó LO120000	Ú LU120000
-F	! SP020000	^ SD150000	? SP150000	" SP040000	± SA020000	¤ SC010000	® SM530000	× SA070000	õ LO190000	ÿ LY170000	Õ LO200000	(EO)

コード・ページ 00500

図 67. コード・ページ 500

HEX DIGITS 1ST → 2ND ↓	0-	1-	2-	3-	4-	5-	6-	7-	8-	9-	A-	B-	C-	D-	E-	F-
-0		► SM590000	(SP) SP010000	0 ND100000	@ SM050000	P LP020000	` SD130000	p LP010000	Ç LC420000	É LE120000	á LA110000	▒ SF140000	SF020000	ð LD630000	Ó LO120000	(SHY) SP320000
-1	☺ SS000000	◄ SM630000	! SP020000	1 ND010000	A LA020000	Q LQ020000	a LA010000	q LQ010000	ü LU170000	æ LA510000	í LI110000	▓ SF150000	┴ SF070000	Ð LD620000	ß LS610000	± SA020000
-2	☻ SS010000	↕ SM760000	" SP040000	2 ND020000	B LB020000	R LR020000	b LB010000	r LR010000	é LE110000	Æ LA520000	ó LO110000	▐ SF160000	┬ SF060000	Ê LE160000	Ô LO160000	═ SM100000
-3	♥ SS020000	!! SP330000	# SM010000	3 ND030000	C LC020000	S LS020000	c LC010000	s LS010000	â LA150000	ô LO150000	ú LU110000	┤ SF110000	├ SF080000	Ë LE180000	Ò LO140000	¾ NF050000
-4	♦ SS030000	¶ SM250000	$ SC030000	4 ND040000	D LD020000	T LT020000	d LD010000	t LT010000	ä LA170000	ö LO170000	ñ LN190000	╣ SF090000	─ SF100000	È LE140000	õ LO190000	¶ SM250000
-5	♣ SS040000	§ SM240000	% SM020000	5 ND050000	E LE020000	U LU020000	e LE010000	u LU010000	à LA130000	ò LO130000	Ñ LN200000	Á LA120000	┼ SF050000	ı LI610000	Õ LO200000	§ SM240000
-6	♠ SS050000	▬ SM700000	& SM030000	6 ND060000	F LF020000	V LV020000	f LF010000	v LV010000	å LA270000	û LU150000	ª SM210000	Â LA160000	ã LA190000	Í LI120000	µ SM170000	÷ SA060000
-7	• SM570000	↨ SM770000	' SP050000	7 ND070000	G LG020000	W LW020000	g LG010000	w LW010000	ç LC410000	ù LU130000	º SM200000	À LA140000	Ã LA200000	Î LI160000	þ LT630000	¸ SD410000
-8	◘ SM570001	↑ SM320000	(SP060000	8 ND080000	H LH020000	X LX020000	h LH010000	x LX010000	ê LE150000	ÿ LY170000	¿ SP160000	© SM520000	╚ SF380000	Ï LI180000	Þ LT640000	° SM190000
-9	○ SM750000	↓ SM330000	) SP070000	9 ND090000	I LI020000	Y LY020000	i LI010000	y LY010000	ë LE170000	Ö LO180000	® SM530000	╣ SF230000	╔ SF390000	SF040000	Ú LU120000	¨ SD170000
-A	◙ SM750002	→ SM310000	* SM040000	: SP130000	J LJ020000	Z LZ020000	j LJ010000	z LZ010000	è LE130000	Ü LU180000	¬ SM660000	╚ SF240000	╩ SF400000	SF010000	Û LU160000	. SD630000
-B	♂ SM280000	← SM300000	+ SA010000	; SP140000	K LK020000	[SM060000	k LK010000	{ SM110000	ï LI170000	ø LO610000	½ NF010000	╗ SF250000	╦ SF410000	■ SF610000	Ù LU140000	1 ND011000
-C	♀ SM290000	└ SA420000	, SP080000	< SA030000	L LL020000	\ SM070000	l LL010000	| SM130000	î LI150000	£ SC020000	¼ NF040000	SF260000	╠ SF420000	■ SP570000	ý LY110000	3 ND031000
-D	♪ SM930000	↔ SM780000	- SP100000	= SA040000	M LM020000	] LM080000	m LM010000	} SM140000	ì LI130000	Ø LO620000	¡ SP030000	¢ SC040000	═ SF430000	¦ SM650000	Ý LY120000	2 ND021000
-E	♫ SM910000	▲ SM600000	. SP110000	> SA050000	N LN020000	^ SD150000	n LN010000	~ SD190000	Ä LA180000	× SA070000	« SP170000	¥ SC050000	╬ SF440000	Ì LI140000	¯ SM150000	■ SM470000
-F	☼ SM690000	▼ SV040000	/ SP120000	? SP150000	O LO020000	_ SP090000	o LO010000	⌂ SM790000	Å LA280000	ƒ SC070000	» SP180000	SF030000	¤ SC010000	■ SF600000	´ SD110000	(RSP) SP300000

コード・ページ 00850

図 68. コード・ページ 850

関連概念:

- 653 ページの『照合順序』

関連資料:

- 「*管理 API リファレンス*」の『sqlecrea - データベースの作成』

付録 E. DB2 Universal Database 技術情報

DB2 資料とヘルプ

DB2® 技術情報は、以下のツールと方法を介して利用できます。

- DB2 インフォメーション・センター
  - トピック
  - DB2 ツールのヘルプ
  - サンプル・プログラム
  - チュートリアル
- ダウンロード可能な PDF ファイル、CD 上の PDF ファイル、および印刷された資料
  - ガイド
  - リファレンス・マニュアル
- コマンド行ヘルプ
  - コマンド・ヘルプ
  - メッセージ・ヘルプ
  - SQL 状態ヘルプ
- インストール済みソース・コード
  - サンプル・プログラム

ibm.com® にある技術資料、白書、 Redbooks™ その他の DB2 Universal Database™ 技術情報にオンラインでアクセスできます。 DB2 Information Management ソフトウェア・ライブラリー・サイト (www.ibm.com/software/data/pubs/) にアクセスしてください。

DB2 資料の更新

IBM® は、 DB2 インフォメーション・センターの資料のフィックスパックやその他の資料更新を定期的に発行しています。 DB2 インフォメーション・センター (http://publib.boulder.ibm.com/infocenter/db2help/) にアクセスすれば、常に最新の情報が掲載されます。 DB2 インフォメーション・センターをローカル・インストールしている場合、更新記事を表示するには、まず手動で更新をインストールしてください。新しい情報が発表されたときに資料を更新することにより、 *DB2 インフォメーション・センター CD* からインストールした情報を更新することができます。

インフォメーション・センターの方が、PDF 資料やハードコピー資料よりも頻繁に更新されます。 DB2 の最新の技術情報を入手するには、資料更新が発行されたときにそれをインストールするか、または www.ibm.com サイトの DB2 インフォメーション・センターにアクセスしてください。

関連概念:

- 「*コール・レベル・インターフェース ガイドおよびリファレンス 第 1 巻*」の『CLI サンプル・プログラム』

- 「*アプリケーション開発ガイド アプリケーションの構築および実行*」の『Java サンプル・プログラム』
- 776 ページの『DB2 インフォメーション・センター』

関連タスク:

- 797 ページの『DB2 ツールからコンテキスト・ヘルプを呼び出す』
- 787 ページの『コンピューターまたはイントラネット・サーバーへの DB2 インフォメーション・センターの更新インストール』
- 798 ページの『コマンド行プロセッサーからメッセージ・ヘルプを呼び出す』
- 798 ページの『コマンド行プロセッサーからコマンド・ヘルプを呼び出す』
- 799 ページの『コマンド行プロセッサーから SQL 状態ヘルプを呼び出す』

関連資料:

- 789 ページの『DB2 PDF 資料および印刷された資料』

DB2 インフォメーション・センター

DB2® インフォメーション・センターを使用すると、 DB2 Universal Database™、 DB2 Connect™、 DB2 Information Integrator および DB2 Query Patroller™ などの DB2 ファミリー製品を最大限に活用するのに必要なすべての情報にアクセスできます。また、DB2 インフォメーション・センターは、 DB2 の主な機能とコンポーネントに関する情報を提供します (レプリケーション、データウェアハウジング、および DB2 の種々の Extender など)。

Mozilla 1.0 以上または Microsoft® Internet Explorer 5.5 以上で表示する場合、 DB2 インフォメーション・センターには以下の機能があります。以下のいくつかの機能では、 JavaScript™ のサポートを使用可能にする必要があります:

柔軟なインストール・オプション
以下の中から、 ご使用の環境に最も適したオプションを使って DB2 資料を表示できます。

- 最新の資料を常に自動的に利用できるようにするには、 IBM® の Web サイト (http://publib.boulder.ibm.com/infocenter/db2help/) にある DB2 インフォメーション・センターからすべての資料に直接アクセスします。
- 更新処理を最小化し、イントラネット内のネットワーク・トラフィックだけに制限するには、イントラネット上の 1 つのサーバーに DB2 資料をインストールします。
- 柔軟性を改善し、ネットワーク接続への依存を軽減するには、個々のコンピューターに DB2 資料をインストールします。

検索 「**検索**」テキスト・フィールドに検索語を入力することにより、 DB2 インフォメーション・センターのすべてのトピックを検索できます。複数の語句を引用符で囲めば、完全一致を検索できます。また、ワイルドカード演算子 (*、?) とブール演算子 (AND、 NOT、OR) を使用して検索を絞り込むことができます。

タスク指向の目次
単一の目次の中から、DB2 資料のトピックを見付けることができます。目

次は、主に実行するタスクの種類に従って編成されていますが、そのほかに製品概要、特定のゴール (目的) の情報、参照情報、索引、および用語集も含まれます。

- 製品概要では、DB2 ファミリーで使用可能な製品間の関係、そうした各製品で提供される機能、および各製品の最新リリース情報について説明されています。
- インストール、管理および開発などのゴール・カテゴリーには、タスクを迅速に完了し、そのための背景情報をよく理解できるようにするトピックが含まれています。
- 「参照」トピックでは、その対象に関する詳細な情報 (ステートメントとコマンドの構文、メッセージ・ヘルプ、構成パラメーターなど) が説明されています。

現在のトピックを目次に表示する

現在のトピックが目次のどの部分に該当するかを表示するには、目次フレーム内の**「リフレッシュ/現在のトピックの表示 (Refresh/Show Current Topic)」**ボタンをクリックするか、コンテンツ・フレーム内の**「目次に表示 (Show in Table of Contents)」**ボタンをクリックします。幾つかのファイルで関連トピックへの複数のリンクをたどった場合、または検索結果からトピックにアクセスした場合には、この機能が役立ちます。

索引 索引から、すべての資料にアクセスすることができます。索引では、用語が 50 音順に編成されています。

用語集 用語集を見れば、DB2 資料で使われているさまざまな用語の定義を調べることができます。用語集では、用語が 50 音順に編成されています。

組み込まれているローカライズ情報

DB2 インフォメーション・センターは、ブラウザーで設定された言語でトピックを表示します。設定された言語のトピックが利用できない場合、DB2 インフォメーション・センターにはそのトピックの英語版が表示されます。

iSeries™ 技術情報については、 IBM eServer™ iSeries Information Center (www.ibm.com/eserver/iseries/infocenter/) を参照してください。

関連概念:

- 778 ページの『DB2 インフォメーション・センターのインストール・シナリオ』

関連タスク:

- 787 ページの『コンピューターまたはイントラネット・サーバーへの DB2 インフォメーション・センターの更新インストール』
- 788 ページの『DB2 インフォメーション・センターにおける特定の言語でのトピックの表示』
- 786 ページの『DB2 インフォメーション・センターの呼び出し』
- 780 ページの『DB2 セットアップ・ウィザードを使用した DB2 インフォメーション・センターのインストール (UNIX)』
- 783 ページの『DB2 セットアップ・ウィザードを使用した DB2 インフォメーション・センターのインストール (Windows)』

DB2 インフォメーション・センターのインストール・シナリオ

さまざまに異なる業務環境のもとでは、 DB2® 情報にどのようにアクセスするかの要件もそれぞれ異なります。 DB2 インフォメーション・センターにアクセスするには、 IBM® の Web サイト、サーバーまたは組織のネットワーク、あるいはコンピューターへのインストールという 3 つの方法が可能です。この 3 つのケースのいずれも、資料は DB2 インフォメーション・センター内に置かれます。インフォメーション・センターは、ブラウザーを使って表示できるように設計されたトピック・ベースの情報の Web サイトです。デフォルトでは、DB2 製品から、 IBM Web サイト上の DB2 インフォメーション・センターにアクセスします。これに対して、 イントラネット・サーバーまたはご自分のコンピューターから DB2 インフォメーション・センターにアクセスしたい場合、製品メディア・パック内にある DB2 インフォメーション・センター CD から DB2 インフォメーション・センターをインストールする必要があります。以下では、DB2 資料へのアクセス・オプションの要約、および 3 つのインストール・シナリオを示します。これを参考にして、お客様の業務環境で DB2 インフォメーション・センターにアクセスするにはどの方法が最適か、どのようなインストール上の問題に配慮する必要があるかを判別してください。

DB2 資料にアクセスするオプションの要約:

以下の表は、お客様の実際の業務環境で、 DB2 インフォメーション・センターの DB2 製品情報にアクセスする方法としてどんなオプションが推奨されるかを示します。

インターネット・アクセス	イントラネット・アクセス	推奨されるアクション
はい	はい	IBM Web サイト上の DB2 インフォメーション・センターへのアクセス、またはイントラネット・サーバーにインストール済みの DB2 インフォメーション・センターへのアクセス
はい	いいえ	IBM Web サイト上の DB2 インフォメーション・センターへのアクセス
いいえ	はい	イントラネット・サーバーにインストール済みの DB2 インフォメーション・センターへのアクセス
いいえ	いいえ	ローカル・コンピューター上の DB2 インフォメーション・センターへのアクセス

シナリオ: コンピューター上の DB2 インフォメーション・センターへのアクセス:

Tsu-Chen 氏は小さな町で工場を経営していますが、その町には、インターネット・アクセスを提供する地元のインターネット・サービス・プロバイダーがありません。彼は、在庫、製品オーダー、銀行口座情報、および営業経費を管理するために DB2 Universal Database™ を購入しました。 Tsu-Chen 氏は以前に DB2 製品を利用したことがないので、 DB2 の使用方法を習得するために、DB2 製品資料を参照する必要があります。

Tsu-Chen 氏は 標準インストール・オプションを使って DB2 Universal Database を自分のコンピューターにインストールした後、 DB2 資料にアクセスしようとします。しかし、開こうとしているページが見つからないというエラー・メッセージがブラウザーから通知されました。 Tsu-Chen 氏は DB2 製品のインストール・マニュアルを調べた結果、 DB2 資料を自分のコンピューター上で利用するには、 DB2 インフォメーション・センターをインストールしなければならないことに気付きます。そしてメディア・パックの中にあった *DB2 インフォメーション・センター CD* を見つけ出して、インストールしました。

これで、Tsu-Chen 氏はオペレーティング・システムのアプリケーション・ランチャーから DB2 インフォメーション・センターにアクセスできるようになり、より良い業務成果をあげるために DB2 製品を利用する方法を習得できます。

シナリオ: IBM Web サイト上の DB2 インフォメーション・センターへのアクセス:

Colin は、あるセミナー企業に所属する情報技術コンサルタントです。彼の専門はデータベース・テクノロジーおよび SQL で、 DB2 Universal Database を使って北米一帯の企業を対象にこれらの科目のセミナーを開催しています。 Colin のセミナーでは、教材として DB2 資料も使用されます。たとえば、SQL の講習コースでは、データベース照会の基本構文と拡張構文を教えるために SQL に関する DB2 資料が使用されます。

Colin が教えている企業の大半はインターネット・アクセスを配備しています。このような状況から判断して、Colin は、最新バージョンの DB2 Universal Database を自分のモバイル・コンピューターにインストールしたとき、 IBM Web サイト上の DB2 インフォメーション・センターにアクセスするよう構成しました。この構成によって、Colin はセミナーで教えるときに最新の DB2 資料にオンライン・アクセスすることができます。

しかし、時折、Colin は移動中にインターネット・アクセスを利用できないことがあります。これは問題となります。担任するセミナーの準備のために DB2 資料にアクセスする必要のある場合には、とくにそうです。このような事態が起きないようにするために、 Colin は自分のモバイル・コンピューターに DB2 インフォメーション・センターのコピーをインストールしました。

こうして、Colin は常に DB2 資料のコピーを自在に活用できるようになりました。**db2set** コマンドを使って自分のモバイル・コンピューターのレジストリー変数を簡単に構成し、どこにいるかに応じて、IBM Web サイトまたは自分のモバイル・コンピューターから DB2 インフォメーション・センターにアクセスできます。

シナリオ: イントラネット・サーバー上の DB2 インフォメーション・センターへのアクセス:

Eva は、生命保険会社のデータベース上級管理者です。彼女は管理業務の一環として、会社の UNIX® データベース・サーバーに最新バージョンの DB2 Universal Database をインストールおよび構成します。彼女の会社は最近、セキュリティー上の理由から、インターネット・アクセスをもはや業務で利用できないようにすると社員に通知しました。同社はネットワーク環境を装備しているため、 Eva は DB2 インフォメーション・センターのコピーをイントラネット・サーバー上にインスト

ールして、社内のデータウェアハウスを定期的に利用するすべての社員 (営業担当者、営業部長、および業務分析担当者) から DB2 資料へのアクセスを可能にすることにしました。

Eva は、応答ファイルを使って全社員のコンピューター上に最新バージョンの DB2 Universal Database をインストールするようデータベース・チームに指示します。その際、イントラネット・サーバーのホスト名とポート番号を使って DB2 インフォメーション・センターにアクセスできるよう、確実に各コンピューターを構成します。

しかし、Eva のチームの下級データベース管理者である Migual の誤解によって、数人の社員のコンピューター上で、イントラネット・サーバーの DB2 インフォメーション・センターにアクセスするよう DB2 Universal Database を構成する代わりに、 DB2 インフォメーション・センターのコピーをそれらのコンピューターにインストールしてしまいました。これを訂正するために、Eva は、 **db2set** コマンドを使ってこれらのコンピューター上の DB2 インフォメーション・センターのレジストリー変数 (ホスト名は DB2_DOCHOST、ポート番号は DB2_DOCPORT) を変更するよう Migual に指示しました。これで、ネットワーク上の適切なすべてのコンピューターが DB2 インフォメーション・センターにアクセスできるようになり、社員は DB2 に関する質問の答えを DB2 資料から見つけることができます。

関連概念:

- 776 ページの『DB2 インフォメーション・センター』

関連タスク:

- 787 ページの『コンピューターまたはイントラネット・サーバーへの DB2 インフォメーション・センターの更新インストール』
- 780 ページの『DB2 セットアップ・ウィザードを使用した DB2 インフォメーション・センターのインストール (UNIX)』
- 783 ページの『DB2 セットアップ・ウィザードを使用した DB2 インフォメーション・センターのインストール (Windows)』
- 『DB2 インフォメーション・センターへのアクセスのロケーションの設定: Common GUI help』

関連資料:

- 「コマンド・リファレンス」の『db2set - DB2 プロファイル・レジストリー・コマンド』

DB2 セットアップ・ウィザードを使用した DB2 インフォメーション・センターのインストール (UNIX)

DB2 製品資料にアクセスする方法として、 IBM Web サイト、イントラネット・サーバー、またはコンピューターにインストールしたバージョンの 3 つがあります。デフォルトでは、DB2 製品は IBM Web サイト上の DB2 資料にアクセスします。イントラネット・サーバーまたはコンピューター上の DB2 資料にアクセスしたい場合には、 *DB2 インフォメーション・センター CD* から資料をインストールする必要があります。 DB2 セットアップ・ウィザードを使用すれば、インストール設

定を定義し、 UNIX オペレーティング・システムを使用するコンピューターに DB2 インフォメーション・センターをインストールできます。

前提条件:

このセクションでは、UNIX コンピューターに DB2 インフォメーション・センターをインストールするためのハードウェア、オペレーティング・システム、ソフトウェア、および通信の諸要件を一覧で示します。

- **ハードウェア要件**

  以下のいずれかのプロセッサーが必要です。

  - PowerPC (AIX)
  - HP 9000 (HP-UX)
  - Intel 32 ビット (Linux)
  - Solaris UltraSPARC コンピューター (Solaris オペレーティング環境)

- **オペレーティング・システム要件**

  以下のいずれかのオペレーティング・システムが必要です。

  - IBM AIX 5.1 (PowerPC 上)
  - HP-UX 11i (HP 9000 上)
  - Red Hat Linux 8.0 (Intel 32 ビット上)
  - SuSE Linux 8.1 (Intel 32 ビット上)
  - Sun Solaris バージョン 8 (Solaris オペレーティング環境の UltraSPARC コンピューター上)

  注: DB2 インフォメーション・センターは、 DB2 クライアントをサポートする UNIX オペレーティング・システム上で稼動します。このため、IBM Web サイトから DB2 インフォメーション・センターにアクセスするか、イントラネット・サーバーに DB2 インフォメーション・センターをインストールしてそれにアクセスすることをお勧めします。

- **ソフトウェア要件**
  - 以下のブラウザーがサポートされています。
    - Mozilla バージョン 1.0 以上
- DB2 セットアップ・ウィザードは、グラフィック・インストーラーです。ご使用のマシンで DB2 セットアップ・ウィザードのグラフィカル・ユーザー・インターフェースを表示可能にする X Window システム・ソフトウェアをインプリメントする必要があります。 DB2 セットアップ・ウィザードを実行する前に、ディスプレイを正しくエクスポートしたことを確認してください。たとえば、コマンド・プロンプトで

  ```
  export DISPLAY=9.26.163.144:0.
  ```

  というコマンドを入力します。

- **通信要件**
  - TCP/IP

手順:

DB2 セットアップ・ウィザードを使用して DB2 インフォメーション・センターをインストールするには、以下のようにします。

1. システムにログオンします。
2. DB2 インフォメーション・センター製品 CD を挿入してシステムにマウントします。
3. 次のコマンドを入力して、CD がマウントされているディレクトリーに移動します。

   ```
   cd /cd
   ```

   /cd は、CD のマウント・ポイントを表します。
4. **./db2setup** コマンドを入力して、 DB2 セットアップ・ウィザードを開始します。
5. IBM DB2 セットアップ・ランチパッドが開きます。 DB2 インフォメーション・センターのインストールに直接進むには、**「製品のインストール」**をクリックします。残りのステップについて説明しているオンライン・ヘルプを利用できます。オンライン・ヘルプを呼び出すには、**「ヘルプ」**をクリックします。**「キャンセル」**をクリックすれば、いつでもインストールを終了できます。
6. **「インストールしたい製品を選択します」**ページでは、**「次へ」**をクリックします。
7. **「DB2 セットアップ・ウィザードにようこそ (Welcome to the DB2 Setup wizard)」**ページで、**「次へ」**をクリックします。 DB2 セットアップ・ウィザードは、プログラムのセットアップ操作を案内します。
8. インストールを続行するには、使用許諾条件に同意する必要があります。**「ご使用条件」**ページで、**「ご使用条件に同意します (I accept the terms in the license agreement)」** をクリックして、**「次へ」**をクリックします。
9. **「インストール・アクションの選択」**で、**「このコンピューターに DB2 インフォメーション・センターをインストールする (Install DB2 Information Center on this computer)」**を選択します。応答ファイルを使用して、このコンピューターまたは他のコンピューターに DB2 インフォメーション・センターをあとでインストールしたい場合には、**「設定を応答ファイルに保管する」**を選択します。**「次へ」**をクリックします。
10. **「インストールする言語の選択」**ページでは、 DB2 インフォメーション・センターをインストールする言語を選択します。**「次へ」**をクリックします。
11. **「DB2 インフォメーション・センター・ポートの指定」**ページでは、 DB2 インフォメーション・センターへの着信通信を構成します。**「次へ」**をクリックしてインストールを続けます。
12. **「ファイルのコピーの開始」**ページでは、インストールの選択項目を確認します。設定を変更するには、**「戻る」**をクリックします。**「インストール」**をクリックすると、 DB2 インフォメーション・センターのファイルがコンピューターにコピーされます。

このほか、応答ファイルを使って DB2 インフォメーション・センターをインストールすることもできます。

インストール・ログ db2setup.his、 db2setup.log、および db2setup.err は、デフォルトでは /tmp ディレクトリーに置かれます。

db2setup.log ファイルは、エラーも含めた DB2 製品のインストール情報をすべてキャプチャーします。 db2setup.his ファイルは、コンピューター上の DB2 製品インストール内容をすべて記録します。 DB2 は、db2setup.log ファイルを db2setup.his に付加します。 db2setup.err ファイルは、 Java から戻されるすべてのエラー出力 (例外やトラップの情報など) をキャプチャーします。

インストールが完了したら、ご使用の UNIX オペレーティング・システムに応じて、 DB2 は以下のいずれかのディレクトリーにインストールされます。

- AIX: /usr/opt/db2_08_01
- HP-UX: /opt/IBM/db2/V8.1
- Linux: /opt/IBM/db2/V8.1
- Solaris オペレーティング環境: /opt/IBM/db2/V8.1

関連概念:

- 776 ページの『DB2 インフォメーション・センター』
- 778 ページの『DB2 インフォメーション・センターのインストール・シナリオ』

関連タスク:

- 「*インストールおよび構成 補足*」の『応答ファイルによる DB2 のインストール (UNIX)』
- 787 ページの『コンピューターまたはイントラネット・サーバーへの DB2 インフォメーション・センターの更新インストール』
- 788 ページの『DB2 インフォメーション・センターにおける特定の言語でのトピックの表示』
- 786 ページの『DB2 インフォメーション・センターの呼び出し』
- 783 ページの『DB2 セットアップ・ウィザードを使用した DB2 インフォメーション・センターのインストール (Windows)』

DB2 セットアップ・ウィザードを使用した DB2 インフォメーション・センターのインストール (Windows)

DB2 製品資料にアクセスする方法として、 IBM Web サイト、イントラネット・サーバー、またはコンピューターにインストールしたバージョンの 3 つがあります。デフォルトでは、DB2 製品は IBM Web サイト上の DB2 資料にアクセスします。イントラネット・サーバーまたはコンピューター上の DB2 資料にアクセスしたい場合には、 *DB2 インフォメーション・センター CD* から DB2 資料をインストールする必要があります。 DB2 セットアップ・ウィザードを使用すれば、インストール設定を定義し、 Windows オペレーティング・システムを使用するコンピューターに DB2 インフォメーション・センターをインストールできます。

前提条件:

このセクションでは、Windows に DB2 インフォメーション・センターをインストールするためのハードウェア、オペレーティング・システム、ソフトウェア、および通信の諸要件を一覧で示します。

- **ハードウェア要件**

  以下のいずれかのプロセッサーが必要です。

  – 32 ビット・コンピューター: Pentium または Pentium 互換の CPU

- **オペレーティング・システム要件**

  以下のいずれかのオペレーティング・システムが必要です。

  – Windows 2000

  – Windows XP

  注: DB2 インフォメーション・センターは、 DB2 クライアントをサポートする Windows オペレーティング・システム上で稼動します。このため、IBM Web サイトの DB2 インフォメーション・センターにアクセスするか、イントラネット・サーバーに DB2 インフォメーション・センターをインストールしてそれにアクセスすることをお勧めします。

- **ソフトウェア要件**

  – 以下のブラウザーがサポートされています。

    - Mozilla 1.0 以上

    - Internet Explorer バージョン 5.5 または 6.0 (Windows XP の場合はバージョン 6.0)

- **通信要件**

  – TCP/IP

制約事項:

- DB2 インフォメーション・センターをインストールするには、管理権限をもつアカウントが必要です。

手順:

DB2 セットアップ・ウィザードを使用して DB2 インフォメーション・センターをインストールするには、以下のようにします。

1. DB2 インフォメーション・センターのインストールで定義したアカウントで、システムにログオンします。
2. CD をドライブに挿入します。自動実行機能が使用可能になっていれば、IBM DB2 セットアップ・ランチパッドが起動します。
3. DB2 セットアップ・ウィザードは、システム言語を判別して、その言語用のセットアップ・プログラムを立ち上げます。英語以外の言語でセットアップ・プログラムを実行したい場合、またはセットアップ・プログラムの自動始動が失敗した場合には、 DB2 セットアップ・ウィザードを手動で開始できます。

   次のようにして、DB2 セットアップ・ウィザードを手動で開始します。

   a. **「スタート」**をクリックし、**「ファイル名を指定して実行」**を選択します。

   b. **「開く」**フィールドで、以下のコマンドを入力します。

      ```
      x:¥setup.exe /i 2-letter language identifier
      ```

ここで、*x:* は CD ドライブ、 *2-letter language identifier* (2 文字の言語識別子) はセットアップ・プログラムを実行する言語を表します。

c. **「OK」**をクリックします。

4. IBM DB2 セットアップ・ランチパッドが開きます。 DB2 インフォメーション・センターのインストールに直接進むには、**「製品のインストール」**をクリックします。残りのステップについて説明しているオンライン・ヘルプを利用できます。オンライン・ヘルプを呼び出すには、**「ヘルプ」**をクリックします。**「キャンセル」**をクリックすれば、いつでもインストールを終了できます。
5. **「インストールしたい製品を選択します」**ページでは、**「次へ」**をクリックします。
6. **「DB2 セットアップ・ウィザードにようこそ (Welcome to the DB2 Setup wizard)」**ページで、**「次へ」**をクリックします。 DB2 セットアップ・ウィザードは、プログラムのセットアップ操作を案内します。
7. インストールを続行するには、使用許諾条件に同意する必要があります。**「ご使用条件」**ページで、**「ご使用条件に同意します (I accept the terms in the license agreement)」** をクリックして、**「次へ」**をクリックします。
8. **「インストール・アクションの選択」**で、**「このコンピューターに DB2 インフォメーション・センターをインストールする (Install DB2 Information Center on this computer)」**を選択します。応答ファイルを使用して、このコンピューターまたは他のコンピューターに DB2 インフォメーション・センターをあとでインストールしたい場合には、**「設定を応答ファイルに保管する」**を選択します。**「次へ」**をクリックします。
9. **「インストールする言語の選択」**ページでは、 DB2 インフォメーション・センターをインストールする言語を選択します。**「次へ」**をクリックします。
10. **「DB2 インフォメーション・センター・ポートの指定」**ページでは、 DB2 インフォメーション・センターへの着信通信を構成します。**「次へ」**をクリックしてインストールを続けます。
11. **「ファイルのコピーの開始」**ページでは、インストールの選択項目を確認します。設定を変更するには、**「戻る」**をクリックします。**「インストール」**をクリックすると、 DB2 インフォメーション・センターのファイルがコンピューターにコピーされます。

応答ファイルを使って DB2 インフォメーション・センターをインストールすることができます。また、**db2rspgn** コマンドを使って、既存のインストール内容に基づく応答ファイルを生成することもできます。

インストール時に検出されるエラーの詳細については、 「マイ ドキュメント」¥DB2LOG¥ ディレクトリー内の db2.log ファイルと db2wi.log ファイルを参照してください。 「マイ ドキュメント」 ディレクトリーの場所は、ご使用のコンピューターの設定によって異なります。

db2wi.log ファイルは、 DB2 の最新のインストール情報をキャプチャーします。db2.log は、DB2 製品のインストールの履歴をキャプチャーします。

関連概念:

• 776 ページの『DB2 インフォメーション・センター』

- 778 ページの『DB2 インフォメーション・センターのインストール・シナリオ』

関連タスク:

- 「*インストールおよび構成 補足*」の『応答ファイルによる DB2 製品のインストール (Windows)』
- 787 ページの『コンピューターまたはイントラネット・サーバーへの DB2 インフォメーション・センターの更新インストール』
- 788 ページの『DB2 インフォメーション・センターにおける特定の言語でのトピックの表示』
- 786 ページの『DB2 インフォメーション・センターの呼び出し』
- 780 ページの『DB2 セットアップ・ウィザードを使用した DB2 インフォメーション・センターのインストール (UNIX)』

関連資料:

- 「コマンド・リファレンス」の『db2rspgn - 応答ファイル生成プログラム・コマンド』

DB2 インフォメーション・センターの呼び出し

DB2 インフォメーション・センターは、Linux、UNIX、および Windows オペレーティング・システム用の DB2 製品 (DB2 Universal Database、 DB2 Connect、DB2 Information Integrator、 DB2 Query Patroller など) を使用するために必要なすべての情報を提供します。

DB2 インフォメーション・センターは、以下の場所から呼び出すことができます。

- DB2 UDB クライアントまたはサーバーがインストールされているコンピューター
- DB2 インフォメーション・センターがインストールされているイントラネット・サーバーまたはローカル・コンピューター
- IBM の Web サイト

前提条件:

DB2 インフォメーション・センターを呼び出すための要件は、以下のとおりです。

- *オプショナル*: 希望する言語でトピックを表示するようブラウザーを構成する
- *オプショナル*: コンピューターまたはイントラネット・サーバーにインストール済みの DB2 インフォメーション・センターを使用するよう DB2 クライアントを構成する

手順:

DB2 UDB クライアントまたはサーバーがインストールされているコンピューターから DB2 インフォメーション・センターを呼び出すには、以下のようにします。

- (Windows オペレーティング・システムの)「スタート」メニューから: **「スタート」→「プログラム」→「IBM DB2」→「情報」→「インフォメーション・センター」**をクリックします。
- コマンド行プロンプトから:
  - Linux および UNIX オペレーティング・システムの場合、 **db2icdocs** コマンドを発行します。

– Windows オペレーティング・システムの場合、 **db2icdocs.exe** コマンドを発行します。

イントラネット・サーバーまたはローカル・コンピューターにインストール済みの DB2 インフォメーション・センターを Web ブラウザーで開くには、以下のようにします。

- Web ページ http://<host-name>:<port-number>/ を開きます (<host-name> はホスト名、 <port-number> は DB2 インフォメーション・センターを利用可能なポート番号)。

IBM Web サイトにある DB2 インフォメーション・センターを Web ブラウザーで開くには、以下のようにします。

- Web ページ publib.boulder.ibm.com/infocenter/db2help/ を開きます。

関連概念:

- 776 ページの『DB2 インフォメーション・センター』
- 778 ページの『DB2 インフォメーション・センターのインストール・シナリオ』

関連タスク:

- 788 ページの『DB2 インフォメーション・センターにおける特定の言語でのトピックの表示』
- 797 ページの『DB2 ツールからコンテキスト・ヘルプを呼び出す』
- 787 ページの『コンピューターまたはイントラネット・サーバーへの DB2 インフォメーション・センターの更新インストール』
- 798 ページの『コマンド行プロセッサーからコマンド・ヘルプを呼び出す』
- 『DB2 インフォメーション・センターへのアクセスのロケーションの設定: Common GUI help』

関連資料:

- 「*コマンド・リファレンス*」の『HELP コマンド』

コンピューターまたはイントラネット・サーバーへの DB2 インフォメーション・センターの更新インストール

http://publib.boulder.ibm.com/infocenter/db2help/ から利用できる DB2 インフォメーション・センターは、資料の新規追加または変更によって定期的に更新されます。さらに、更新された DB2 インフォメーション・センターをコンピューターまたはイントラネット・サーバーにダウンロードしてインストールできる場合もあります。DB2 インフォメーション・センターを更新しても、 DB2 クライアント製品またはサーバー製品は更新されません。

前提条件:

インターネットに接続されたコンピューターへのアクセスが必要です。

手順:

DB2 インフォメーション・センターの更新をコンピューターまたはイントラネット・サーバーにインストールするには、以下のようにします。

1. IBM の Web サイト (http://publib.boulder.ibm.com/infocenter/db2help/) にある DB2 インフォメーション・センターを開きます。
2. 「DB2 インフォメーション・センターにようこそ」ページの見出し「サービスおよびサポート」の「ダウンロード」セクションで、**「DB2 資料」**リンクをクリックします。
3. 最新のドキュメンテーション・イメージのレベルと、インストール済みのドキュメンテーション・レベルを比較して、 DB2 インフォメーション・センターを更新する必要があるかどうかを確認します。「DB2 インフォメーション・センターにようこそ」ページに、インストール済みのドキュメンテーションのレベルがリストされます。
4. より新しいバージョンの DB2 インフォメーション・センターが存在する場合、ご使用のオペレーティング・システムに対応する最新の *DB2 インフォメーション・センター・イメージをダウンロードします。
5. 最新の *DB2 インフォメーション・センター・イメージをインストールするには、 Web ページの指示に従ってください。

関連概念:

- 778 ページの『DB2 インフォメーション・センターのインストール・シナリオ』

関連タスク:

- 786 ページの『DB2 インフォメーション・センターの呼び出し』
- 780 ページの『DB2 セットアップ・ウィザードを使用した DB2 インフォメーション・センターのインストール (UNIX)』
- 783 ページの『DB2 セットアップ・ウィザードを使用した DB2 インフォメーション・センターのインストール (Windows)』

DB2 インフォメーション・センターにおける特定の言語でのトピックの表示

DB2 インフォメーション・センターでは、ブラウザーの設定で指定した言語でのトピックの表示が試みられます。トピックがその指定言語に翻訳されていない場合は、 DB2 インフォメーション・センターでは英語でトピックが表示されます。

手順:

Internet Explorer Web ブラウザーで、指定どおりの言語でトピックを表示するには、以下のようにします。

1. Internet Explorer の**「ツール」**—> **「インターネット オプション」**—> **「言語...」**ボタンをクリックします。「言語の優先順位」ウィンドウがオープンします。
2. 該当する言語が、言語リストの先頭の項目に指定されていることを確認します。
   - リストに新しい言語を追加するには、**「追加...」**ボタンをクリックします。

     注: 言語を追加しても、特定の言語でトピックを表示するのに必要なフォントがコンピューターに備えられているとはかぎりません。
   - リストの先頭に新しい言語を移動するには、その言語を選択してから、その言語が言語リストに先頭に行くまで**「上へ」**ボタンをクリックします。

3. 使いたい言語で DB2 インフォメーション・センターを表示するには、ページをリフレッシュします。

Mozilla Web ブラウザーの場合に、使いたい言語でトピックを表示するには、以下のようにします。

1. Mozilla の**「編集」**—> **「設定」**—> **「言語」**ボタンをクリックします。「設定」ウィンドウに「言語」パネルが表示されます。
2. 該当する言語が、言語リストの先頭の項目に指定されていることを確認します。
   - リストに新しい言語を追加するには、**「追加...」**ボタンをクリックしてから、「言語を追加」ウィンドウで言語を選択します。
   - リストの先頭に新しい言語を移動するには、その言語を選択してから、その言語が言語リストに先頭に行くまで**「上に移動」**ボタンをクリックします。
3. 使いたい言語で DB2 インフォメーション・センターを表示するには、ページをリフレッシュします。

関連概念:

- 776 ページの『DB2 インフォメーション・センター』

DB2 PDF 資料および印刷された資料

以下の表は、正式な資料名、資料番号、および PDF ファイル名を示しています。ハードコピー版の資料を注文するには、正式な資料名を知っておく必要があります。 PDF ファイルを印刷するには、PDF ファイル名を知っておく必要があります。

DB2 資料は、以下のカテゴリーに分類されています。

- DB2 中核情報
- 管理情報
- アプリケーション開発情報
- ビジネス・インテリジェンス情報
- DB2 Connect 情報
- 入門情報
- チュートリアル情報
- オプショナル・コンポーネント情報
- リリース・ノート

以下の表は、DB2 ライブラリー内の各資料について、その資料のハードコピー版を注文したり、PDF 版を印刷または表示したりするのに必要な情報を示しています。DB2 ライブラリー内の各資料に関する詳細な説明については、www.ibm.com/shop/publications/order にある IBM Publications Center にアクセスしてください。

DB2 の基本情報

こうした資料の情報は、すべての DB2 ユーザーに基本的なもので、プログラマーおよびデータベース管理者にとって役立つ情報であるとともに、DB2 Connect、DB2 Warehouse Manager、または他の DB2 製品を使用するユーザーにとっても役

立つ内容です。

表 95. DB2 の基本情報

資料名	資料番号	PDF ファイル名
「*IBM DB2 Universal Database コマンド・リファレンス*」	SC88-9140	db2n0j81
「*IBM DB2 Universal Database 用語集*」	資料番号なし	db2t0j81
「*IBM DB2 Universal Database メッセージ・リファレンス 第 1 巻*」	GC88-9152 (ハードコピーなし)	db2m1j81
「*IBM DB2 Universal Database メッセージ・リファレンス 第 2 巻*」	GC88-9153 (ハードコピーなし)	db2m2j81
「*IBM DB2 Universal Database 新機能*」	SC88-9158	db2q0j81

管理情報

これらの資料の情報は、DB2 データベース、データウェアハウス、およびフェデレーテッド・システムを効果的に設計し、インプリメントし、保守するために必要なトピックを扱っています。

表 96. 管理情報

資料名	資料番号	PDF ファイル名
「*IBM DB2 Universal Database 管理ガイド: プランニング*」	SC88-9135	db2d1j81
「*IBM DB2 Universal Database 管理ガイド: インプリメンテーション*」	SC88-9133	db2d2j81
「*IBM DB2 Universal Database 管理ガイド: パフォーマンス*」	SC88-9134	db2d3j81
「*IBM DB2 Universal Database 管理 API リファレンス*」	SC88-9136	db2b0j81
「*IBM DB2 Universal Database データ移動ユーティリティー ガイドおよびリファレンス*」	SC88-9142	db2dmj81
「*IBM DB2 Universal Database データ・リカバリーと高可用性 ガイドおよびリファレンス*」	SC88-9143	db2haj81
「*IBM DB2 Universal Database データウェアハウス・センター 管理ガイド*」	SC88-9165	db2ddj81
「*IBM DB2 Universal Database SQL リファレンス 第 1 巻*」	SC88-9155	db2s1j81
「*IBM DB2 Universal Database SQL リファレンス 第 2 巻*」	SC88-9156	db2s2j81

表 96. 管理情報 (続き)

資料名	資料番号	PDF ファイル名
「*IBM DB2 Universal Database システム・モニター ガイドおよびリファレンス*」	SC88-9157	db2f0j81

アプリケーション開発情報

これらの資料の情報は、 DB2 Universal Database (DB2 UDB) のアプリケーション開発者またはプログラマーが特に関心を持つ内容です。サポートされるさまざまなプログラミング・インターフェース (組み込み SQL、ODBC、JDBC、SQLJ、CLI など) を使用して DB2 UDB にアクセスするのに必要な資料とともに、サポートされる言語およびコンパイラーについても紹介されています。また、DB2 インフォメーション・センターをご使用の場合には、サンプル・プログラムのソース・コードの HTML バージョンにアクセスすることもできます。

表 97. アプリケーション開発情報

資料名	資料番号	PDF ファイル名
「*IBM DB2 Universal Database アプリケーション開発ガイド アプリケーションの構築および実行*」	SC88-9137	db2axj81
「*IBM DB2 Universal Database アプリケーション開発ガイド クライアント・アプリケーションのプログラミング*」	SC88-9138	db2a1j81
「*IBM DB2 Universal Database アプリケーション開発ガイド サーバー・アプリケーションのプログラミング*」	SC88-9139	db2a2j81
「*IBM DB2 Universal Database コール・レベル・インターフェース ガイドおよびリファレンス 第 1 巻*」	SC88-9159	db2l1j81
「*IBM DB2 Universal Database コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻*」	SC88-9160	db2l2j81
「*IBM DB2 Universal Database データウェアハウス・センター アプリケーション統合ガイド*」	SC88-9166	db2adj81
「*IBM DB2 Universal Database XML Extender 管理およびプログラミングのガイド*」	SC88-9172	db2sxj81

ビジネス・インテリジェンス情報

これらの資料の情報は、さまざまなコンポーネントを使用して、 DB2 Universal Database のデータウェアハウジング機能および分析機能を拡張する方法を説明しています。

表 98. ビジネス・インテリジェンス情報

資料名	資料番号	PDF ファイル名
「*IBM DB2 Warehouse Manager Standard Edition インフォメーション・カタログ・センター 管理ガイド*」	SC88-9167	db2dij81
「*IBM DB2 Warehouse Manager Standard Edition インストール・ガイド*」	GC88-9164	db2idj81
「*IBM DB2 Warehouse Manager Standard Edition DB2 Warehouse Manager を使用時の ETI ソリューション・コンバージョン・プログラムの管理*」	SC88-9894	iwhe1mstx80

DB2 Connect 情報

このカテゴリーの情報は、 DB2 Connect Enterprise Edition または DB2 Connect Personal Edition を使用して、メインフレーム・サーバーおよびミッドレンジ・サーバー上のデータにアクセスする方法を説明しています。

表 99. DB2 Connect 情報

資料名	資料番号	PDF ファイル名
「*IBM コネクティビティー 補足*」	資料番号なし	db2h1j81
「*IBM DB2 Connect Enterprise Edition 概説およびインストール*」	GC88-9145	db2c6j81
「*IBM DB2 Connect Personal Edition 概説およびインストール*」	GC88-9146	db2c1j81
「*IBM DB2 Connect ユーザーズ・ガイド*」	SC88-9147	db2c0j81

入門情報

このカテゴリーの情報は、サーバー、クライアント、および他の DB2 製品をインストールして構成する場合に役立ちます。

表 100. 入門情報

資料名	資料番号	PDF ファイル名
「*IBM DB2 Universal Database DB2 クライアント機能 概説およびインストール*」	GC88-9144 (ハードコピーなし)	db2itj81
「*IBM DB2 Universal Database DB2 サーバー機能 概説およびインストール*」	GC88-9148	db2isj81
「*IBM DB2 Universal Database DB2 Personal Edition 概説およびインストール*」	GC88-9150	db2i1j81
「*IBM DB2 Universal Database インストールおよび構成 補足*」	GC88-9149 (ハードコピーなし)	db2iyj81
「*IBM DB2 Universal Database DB2 Data Links Manager 概説およびインストール*」	GC88-9141	db2z6j81

チュートリアル情報

チュートリアル情報は、DB2 機能を紹介し、さまざまなタスクを実行する方法を示します。

表 101. チュートリアル情報

資料名	資料番号	PDF ファイル名
「*ビジネス・インテリジェンス・チュートリアル: データウェアハウス・センターの紹介*」	資料番号なし	db2tuj81
「*ビジネス・インテリジェンス・チュートリアル: データウェアハウジングの上級者向けガイド*」	資料番号なし	db2taj81
「*インフォメーション・カタログ・センター チュートリアル*」	資料番号なし	db2aij81
「*Video Central for e-business チュートリアル*」	資料番号なし	db2twj81
「*Visual Explain チュートリアル*」	資料番号なし	db2tvj81

オプショナル・コンポーネント情報

このカテゴリーの情報は、 DB2 のオプショナル・コンポーネントを使用する方法について説明しています。

表 102. オプショナル・コンポーネント情報

資料名	資料番号	PDF ファイル名
「*IBM DB2 Cube Views Guide and Reference*」	SC18–7298	db2aax81
「*IBM DB2 Query Patroller インストール、管理、使用法のガイド*」	GC88–9154	db2dwj81
「*IBM DB2 Spatial Extender and Geodetic Extender ユーザーズ・ガイドおよびリファレンス*」	SC88-9171	db2sbj81
「*IBM DB2 Universal Database Data Links Manager 管理ガイドおよびリファレンス*」	SC88-9169	db2z0x82
「*DB2 Net Search Extender 管理およびユーザーズ・ガイド*」 **注:** この資料の HTML 版は、HTML ドキュメンテーション CD からインストールされません。	SH88-8546	N/A

リリース・ノート

リリース・ノートは、ご使用の製品のリリースおよびフィックスパック・レベルに特有の追加情報を紹介します。また、リリース・ノートには、各リリース、アップデート、およびフィックスパックで組み込まれた資料上の更新の要約も含まれています。

表 103. リリース・ノート

資料名	資料番号	PDF ファイル名
「*DB2 リリース・ノート*」	「注」を参照。	「注」を参照。
「*DB2 インストール情報*」	製品 CD-ROM でのみ参照可能。	使用できません。

注: リリース・ノートは以下の形式で入手できます。

- XHTML およびテキスト形式 (製品 CD 内)
- PDF 形式 (PDF ドキュメンテーション CD 内)

さらに、リリース・ノートの中で、『*既知の問題と予備手段*』および『*リリース間の非互換性*』に関する部分は DB2 インフォメーション・センターにも表示されます。

UNIX ベースのプラットフォームでテキスト形式でリリース・ノートを確認するには、 `Release.Notes` ファイルを参照してください。このファイルは、`DB2DIR/Readme/%L` ディレクトリーに収録されています。 `%L` はロケール名を表しています。 `DB2DIR` は以下になります。

- AIX オペレーティング・システムの場合: /usr/opt/db2_08_01
- その他のすべての UNIX ベースのオペレーティング・システムの場合: /opt/IBM/db2/V8.1

関連概念:

- 775 ページの『DB2 資料とヘルプ』

関連タスク:

- 795 ページの『PDF ファイルからの DB2 資料の印刷方法』
- 796 ページの『DB2 の印刷資料の注文方法』
- 797 ページの『DB2 ツールからコンテキスト・ヘルプを呼び出す』

PDF ファイルからの DB2 資料の印刷方法

DB2 PDF ドキュメンテーション CD に収録されている DB2 資料を印刷することができます。 Adobe Acrobat Reader を使用すれば、資料全体または特定のページを印刷できます。

前提条件:

Adobe Acrobat Reader がインストールされていることを確認してください。 Adobe Acrobat Reader をインストールする必要がある場合、 Adobe Web サイト (www.adobe.com) から入手できます。

手順:

PDF ファイルから DB2 資料を印刷するには以下のようにします。

1. *DB2 PDF* ドキュメンテーション CD をドライブに挿入します。 UNIX オペレーティング・システムの場合、 DB2 PDF ドキュメンテーション CD をマウントします。 UNIX オペレーティング・システムで CD をマウントする方法については、「*概説およびインストール*」を参照してください。
2. index.htm を開きます。ブラウザー・ウィンドウにファイルが開きます。
3. 参照したい PDF のタイトルをクリックします。 Acrobat Reader で PDF が開きます。
4. **「ファイル」 → 「印刷」**を選択して、所要の資料の任意の部分を印刷します。

関連概念:

- 776 ページの『DB2 インフォメーション・センター』

関連タスク:

- 「*DB2 Universal Database サーバー機能 概説およびインストール*」の『CD-ROM のマウント (AIX)』
- 「*DB2 Universal Database サーバー機能 概説およびインストール*」の『HP-UX 上での CD-ROM のマウント』
- 「*DB2 Universal Database サーバー機能 概説およびインストール*」の『CD-ROM のマウント (Linux)』
- 796 ページの『DB2 の印刷資料の注文方法』

- 「*DB2 Universal Database サーバー機能 概説およびインストール*」の『CD-ROM のマウント (Solaris)』

関連資料:

- 789 ページの『DB2 PDF 資料および印刷された資料』

DB2 の印刷資料の注文方法

ハードコピー版の資料を望む場合には、以下のいずれかの方法で注文できます。

印刷資料の注文方法:

一部の国または地域では、印刷された資料を注文することもできます。お客様がお住まいの国または地域でこのサービスが利用可能かどうかを確認するには、お住まいの国または地域の IBM Publications Web サイトをご覧ください。資料のご注文が可能な場合、以下のようにすることができます。

- 正規の IBM 製品販売業者または営業担当員に連絡してください。お客様がお住まいの地域の IBM 担当員の情報については、お手数ですが IBM の Web サイト (www.ibm.com/planetwide) の IBM Worldwide Directory of Contacts で確認してください。
- IBM Publications Center (http://www.ibm.com/shop/publications/order) にアクセスしてください。なお、IBM Publications Center から資料を注文できない国もあります。

DB2 製品がご利用可能になった時点で、印刷された資料は *DB2 PDF* ドキュメンテーション *CD* にある PDF 形式の資料と同じものです。さらに、 *DB2* インフォメーション・センター *CD* に収録されている印刷された資料の内容もまた、これらと同じです。ただし、DB2 インフォメーション・センター CD には、 PDF 資料にない追加情報も含まれます (たとえば、SQL 管理作業や HTML サンプル)。 DB2 PDF ドキュメンテーション CD に収録されている資料の中には、ハードコピーとしてご注文できない資料もあります。

注: DB2 インフォメーション・センターは、 PDF またはハードコピー の資料よりも頻繁に更新されます。ドキュメンテーションの更新が入手可能になった時点でインストールするか、 DB2 インフォメーション・センター (http://publib.boulder.ibm.com/infocenter/db2help/) を参照して最新の情報を入手してください。

関連タスク:

- 795 ページの『PDF ファイルからの DB2 資料の印刷方法』

関連資料:

- 789 ページの『DB2 PDF 資料および印刷された資料』

DB2 ツールからコンテキスト・ヘルプを呼び出す

コンテキスト・ヘルプは、特定のウィンドウ、ノートブック、ウィザード、またはアドバイザーに関連したタスクまたはコントロールの情報を提供します。コンテキスト・ヘルプは、グラフィカル・ユーザー・インターフェースのある DB2 管理ツールおよび開発ツールから利用できます。コンテキスト・ヘルプには、以下の 2 種類があります。

- それぞれのウィンドウまたはノートブックにある**「ヘルプ」**ボタンからアクセス可能なヘルプ
- infopop (ポップアップ情報ウィンドウ)。これは、マウス・カーソルを特定のフィールドまたはコントロール上に置いたとき、またはウィンドウ、ノートブック、ウィザード、アドバイザー内でフィールドまたはコントロールを選択して F1 を押すと表示されます。

「ヘルプ」ボタンを押すと、概説、前提条件、およびタスク情報が表示されます。infopop は、それぞれのフィールドおよびコントロールについて説明します。

手順:

コンテキスト・ヘルプを呼び出すには、以下のようにします。

- ウィンドウおよびノートブックのヘルプを表示するには、いずれかの DB2 ツールを開始して、任意のウィンドウまたはノートブックを開きます。ウィンドウまたはノートブックの右下隅にある**「ヘルプ」**ボタンをクリックして、コンテキスト・ヘルプを呼び出します。

  また、それぞれの DB2 ツール・センターの上部にある**「ヘルプ」**メニュー項目からコンテキスト・ヘルプにアクセスすることもできます。

  ウィザードおよびアドバイザーでは、最初のページの「タスクの概要」リンクをクリックすると、コンテキスト・ヘルプを表示できます。
- ウィンドウまたはノートブック上の各コントロールの infopop ヘルプを表示するには、コントロールをクリックしてから、**F1** を押します。コントロールの詳細情報を示すポップアップ情報が、黄色いウィンドウに表示されます。

  注: フィールドまたはコントロールにマウス・カーソルを置いておくだけで infopops が表示されるようにするには、「ツール設定」ノートブックの**「文書 (Documentation)」**ページの**「infopops の自動表示」**チェック・ボックスを選択します。

  infopop に似た別のコンテキスト・ヘルプに、診断ポップアップ情報があります。これにはデータ入力規則が示されます。診断ポップアップ情報は、無効または不十分なデータが入力されたとき、紫色のウィンドウに表示されます。診断ポップアップ情報は、以下に関して表示されます。
  - 必須フィールド。
  - 日付フィールドのように、正確なフォーマットを必要とするデータのフィールド。

関連タスク:

- 786 ページの『DB2 インフォメーション・センターの呼び出し』
- 798 ページの『コマンド行プロセッサーからメッセージ・ヘルプを呼び出す』

- 798 ページの『コマンド行プロセッサーからコマンド・ヘルプを呼び出す』
- 799 ページの『コマンド行プロセッサーから SQL 状態ヘルプを呼び出す』
- 『DB2 インフォメーション・センターへのアクセス: Concepts help』
- 『DB2 UDB ヘルプの使用法: Common GUI help』
- 『DB2 インフォメーション・センターへのアクセスのロケーションの設定: Common GUI help』
- 『DB2 コンテキスト・ヘルプと資料へのアクセスを設定する: Common GUI help』

コマンド行プロセッサーからメッセージ・ヘルプを呼び出す

メッセージ・ヘルプは、メッセージが出された原因と、エラーへの応答として実行すべきアクションを説明します。

手順:

メッセージ・ヘルプを呼び出すには、コマンド行プロセッサーを開いて以下のように入力します。

```
? XXXnnnnn
```

ここで、*XXXnnnnn* は有効なメッセージ ID を表します。

たとえば、`? SQL30081` と入力すると、メッセージ SQL30081 に関するヘルプを表示します。

関連概念:

- 「*メッセージ・リファレンス 第 1 巻*」の『メッセージの概要』

関連資料:

- 「コマンド・リファレンス」の『db2 - コマンド行プロセッサーの呼び出しコマンド』

コマンド行プロセッサーからコマンド・ヘルプを呼び出す

コマンド・ヘルプは、コマンド行プロセッサーでのコマンドの構文を説明します。

手順:

コマンド・ヘルプを呼び出すには、コマンド行プロセッサーを開いて以下のように入力します。

```
? command
```

ここで *command* はキーワードまたはコマンド全体を表します。

たとえば、`? catalog` と入力すると、すべての CATALOG コマンドに関するヘルプが表示され、 `? catalog database` と入力すると、 CATALOG DATABASE コマンドのヘルプだけが表示されます。

関連タスク:

- 797 ページの『DB2 ツールからコンテキスト・ヘルプを呼び出す』
- 786 ページの『DB2 インフォメーション・センターの呼び出し』
- 798 ページの『コマンド行プロセッサーからメッセージ・ヘルプを呼び出す』
- 799 ページの『コマンド行プロセッサーから SQL 状態ヘルプを呼び出す』

関連資料:

- 「コマンド・リファレンス」の『db2 - コマンド行プロセッサーの呼び出しコマンド』

コマンド行プロセッサーから SQL 状態ヘルプを呼び出す

DB2 Univerrsal Database は、 SQL ステートメントの結果の原因となったと考えられる条件の SQLSTATE 値を戻します。 SQLSTATE ヘルプは、SQL 状態および SQL 状態クラス・コードの意味を説明します。

手順:

SQL 状態ヘルプを呼び出すには、コマンド行プロセッサーを開いて以下のように入力します。

```
? sqlstate または ? class code
```

ここで、*sqlstate* は有効な 5 桁の SQL 状態を、*class code* は SQL 状態の最初の 2 桁を表します。

たとえば、`? 08003` を指定すると SQL 状態 08003 のヘルプが表示され、 `? 08` を指定するとクラス・コード 08 のヘルプが表示されます。

関連タスク:

- 786 ページの『DB2 インフォメーション・センターの呼び出し』
- 798 ページの『コマンド行プロセッサーからメッセージ・ヘルプを呼び出す』
- 798 ページの『コマンド行プロセッサーからコマンド・ヘルプを呼び出す』

DB2 チュートリアル

DB2® チュートリアルは、 DB2 Universal Database のさまざまな機能について学習するのを支援します。このチュートリアルでは、アプリケーションの開発、SQL 照会のパフォーマンス調整、データウェアハウスの処理、メタデータの管理、および DB2 を使用した Web サービスの開発の各分野で、段階的なレッスンが用意されています。

はじめに:

インフォメーション・センター (http://publib.boulder.ibm.com/infocenter/db2help/) から、このチュートリアルの XHTML 版を表示できます。

チュートリアルの中で、サンプル・データまたはサンプル・コードを使用する場合があります。個々のタスクの前提条件については、それぞれのチュートリアルを参照してください。

DB2 Universal Database チュートリアル:

以下に示すチュートリアルのタイトルをクリックすると、そのチュートリアルを表示できます。

ビジネス・インテリジェンス・チュートリアル: データウェアハウス・センターの紹介
データウェアハウス・センターを使用して簡単なデータウェアハウジング・タスクを実行します。

ビジネス・インテリジェンス・チュートリアル: データウェアハウジングの上級者向けガイド
データウェアハウス・センターを使用して高度なデータウェアハウジング・タスクを実行します。

インフォメーション・カタログ・センター・チュートリアル
インフォメーション・カタログを作成および管理して、インフォメーション・カタログ・センターを使用してメタデータを配置し使用します。

Visual Explain チュートリアル
Visual Explain を使用して、パフォーマンスを向上させるために SQL ステートメントを分析し、最適化し、調整します。

DB2 トラブルシューティング情報

DB2® 製品を使用する際に役立つ、トラブルシューティングおよび問題判別に関する広範囲な情報を利用できます。

DB2 ドキュメンテーション
トラブルシューティング情報は、DB2 インフォメーション・センター、および DB2 ライブラリーに含まれる PDF 資料の中でご利用いただけます。DB2 インフォメーション・センターで、 (ブラウザー・ウィンドウの左側の) ナビゲーション・ツリーの「サポートおよびトラブルシューティング (Support and troubleshooting)」ブランチを参照すると、 DB2 トラブルシューティング・ドキュメンテーションの詳細なリストが見つかります。

DB2 Technical Support の Web サイト
現在問題が発生していて、考えられる原因とソリューションを検索したい場合は、 DB2 Technical Support の Web サイトを参照してください。Technical Support サイトには、最新の DB2 出版物、TechNotes、プログラム診断依頼書 (APAR)、フィックスパック、DB2 内部エラー・コードの最新リスト、その他のリソースが用意されています。この知識ベースを活用して、問題に対する有効なソリューションを探し出すことができます。

DB2 Technical Support の Web サイト (http://www.ibm.com/software/data/db2/udb/winos2unix/support) にアクセスしてください。

DB2 Problem Determination Tutorial Series
DB2 製品で作業中に直面するかもしれない問題を素早く識別し、解決する方法に関する情報を見つけるには、 DB2 Problem Determination Tutorial Series の Web サイトを参照してください。あるチュートリアルでは、使用可能な DB2 問題判別機能およびツールを紹介し、それらをいつ使用すべきかを判断する助けを与えます。別のチュートリアルは、『データベース・エ

ンジン問題判別 (Database Engine Problem Determination)』、『パフォーマンス問題判別 (Performance Problem Determination)』、『アプリケーション問題判別 (Application Problem Determination)』などの関連トピックを扱っています。

DB2 Technical Support (http://www.ibm.com/software/data/support/pdm/db2tutorials.html) には、 DB2 問題判別チュートリアルがすべて揃っています。

関連概念:

- 776 ページの『DB2 インフォメーション・センター』
- 「*問題判別の手引き*」の『Introduction to Problem Determination - DB2 テクニカル・サポートのチュートリアル』

アクセス支援

アクセス支援機能は、身体に障害のある (身体動作が制限されている、視力が弱いなど) ユーザーがソフトウェア製品を十分活用できるように支援します。 DB2® バージョン 8 製品に備わっている主なアクセス支援機能は、以下のとおりです。

- すべての DB2 機能は、マウスの代わりにキーボードを使ってナビゲーションできます。詳細については、『キーボードによる入力およびナビゲーション』を参照してください。
- DB2 ユーザー・インターフェースのフォント・サイズおよび色をカスタマイズすることができます。詳細については、 802 ページの『アクセスしやすい表示』を参照してください。
- DB2 製品は、Java™ Accessibility API を使用するアクセス支援アプリケーションをサポートします。詳細については、 802 ページの『支援テクノロジーとの互換性』を参照してください。
- DB2 資料は、アクセスしやすい形式で提供されています。詳細については、 802 ページの『アクセスしやすい資料』を参照してください。

キーボードによる入力およびナビゲーション

キーボード入力

キーボードだけを使用して DB2 ツールを操作できます。マウスを使って実行できる操作は、キーまたはキーの組み合わせによっても実行できます。標準のオペレーティング・システム・キー・ストロークを使用して、標準のオペレーティング・システム操作を実行できます。

キーまたはキーの組み合わせによって操作を実行する方法について、詳しくは キーボード・ショートカットおよびアクセラレーター: Common GUI help を参照してください。

キーボード・ナビゲーション

キーまたはキーの組み合わせを使用して、 DB2 ツールのユーザー・インターフェースをナビゲートできます。

キーまたはキーの組み合わせによって DB2 ツールをナビゲートする方法の詳細については、キーボード・ショートカットおよびアクセラレーター: Common GUI help を参照してください。

キーボード・フォーカス

UNIX® オペレーティング・システムでは、アクティブ・ウィンドウの中で、キー・ストロークによって操作できる領域が強調表示されます。

アクセスしやすい表示

DB2 ツールには、視力の弱いユーザー、その他の視力障害をもつユーザーのためにアクセシビリティーを向上させる機能が備わっています。これらのアクセシビリティー拡張機能には、フォント・プロパティーのカスタマイズを可能にする機能も含まれています。

フォントの設定

「ツール設定」ノートブックを使用して、メニューおよびダイアログ・ウィンドウに使用されるテキストの色、サイズ、およびフォントを選択できます。

フォント設定に関する詳細情報は、メニューおよびテキストのフォントを変更する: Common GUI help を参照してください。

色に依存しない

本製品のすべての機能を使用するために、ユーザーは必ずしも色を識別する必要はありません。

支援テクノロジーとの互換性

DB2 ツールのインターフェースは、Java Accessibility API をサポートします。これによって、スクリーン・リーダーその他の支援テクノロジーを DB2 製品で利用できるようになります。

アクセスしやすい資料

DB2 形式は、ほとんどの Web ブラウザーで表示可能な XHTML 1.0 形式で提供されています。 XHTML により、ご使用のブラウザーに設定されている表示設定に従って資料を表示できます。さらに、スクリーン・リーダーや他の支援テクノロジーを使用することもできます。

シンタックス・ダイアグラムはドット 10 進形式で提供されます。この形式は、スクリーン・リーダーを使用してオンライン・ドキュメンテーションにアクセスする場合にのみ使用できます。

関連概念:

- 803 ページの『ドット 10 進シンタックス・ダイアグラム』

関連タスク:

- 『キーボード・ショートカットおよびアクセラレーター: Common GUI help』
- 『メニューおよびテキストのフォントを変更する: Common GUI help』

ドット 10 進シンタックス・ダイアグラム

スクリーン・リーダーを使用してインフォメーション・センターを利用するユーザーのために、シンタックス・ダイアグラムがドット 10 進形式で提供されます。

ドット 10 進形式では、各シンタックス・エレメントは別々の行に書き込まれます。複数のシンタックス・エレメントが常に同時に存在する (または常に同時に不在の) 場合、単一のコンパウンド・シンタックス・エレメントとみなせるので同一行に表示できます。

各行は、ドット 10 進数で開始します。たとえば、3 または 3.1 ないしは 3.1.1 です。こうした数を適切に聞き取るには、スクリーン・リーダーが句読点を読み取るように設定されていることを確認してください。同じドット 10 進数を持つすべてのシンタックス・エレメント (たとえば、 3.1 という数値を持つすべてのシンタックス・エレメント) は、相互に排他的な代替エレメントです。 3.1 USERID および 3.1 SYSTEMID という行を聞き取る場合、シンタックスには両方ではなく USERID または SYSTEMID のどちらかが含まれることが分かります。

ドット 10 進レベルは、ネストのレベルを表示します。たとえば、ドット 10 進数 3 のシンタックス・エレメントの後に、一連のドット 10 進数 3.1 のシンタックス・エレメントが続きます。 3.1 の番号が付されたシンタックス・エレメントすべては、番号 3 の付されたシンタックス・エレメントに従属します。

シンタックス・エレメントに関する情報を追加するため、ドット 10 進数の次に特定のワードおよびシンボルが使用されます。 時折、こうしたワードおよびシンボルはエレメントの最初に表示される場合もあります。 簡単に識別するため、ワードやシンボルがシンタックス・エレメントの一部である場合には、円記号 (¥) 文字が先頭に付きます。 * シンボルはドット 10 進数の次に使用でき、シンタックス・エレメントが反復することを示します。 たとえば、ドット 10 進数 3 のシンタックス・エレメント *FILE は、3 ¥* FILE という形式になります。 3* FILE という形式は、シンタックス・エレメント FILE が反復されることを示します。 3* ¥* FILE という形式は、シンタックス・エレメント * FILE が反復されることを示します。

シンタックス・エレメントのストリングを分離するのに使用されるコンマなどの文字は、シンタックス内の分離する項目の直前に表示されます。 こうした文字は、それぞれの項目と同一行に表示するか、同じドット 10 進数を持つ関連する項目のある別の行に表示できます。 またその行には、シンタックス・エレメントに関する情報を提供する別のシンボルを表示することも可能です。 たとえば、複数の LASTRUN および DELETE シンタックス・エレメントを使用している場合には、5.1*、5.1 LASTRUN、および 5.1 DELETE という行は、エレメントをコンマで区切る必要があります。区切り文字が指定されないと、各シンタックス・エレメントを区切るのにブランクが使用されると想定されます。

シンタックス・エレメントの前に % シンボルが付く場合、他の箇所で定義されている参照であることを示します。 % シンボルの後のストリングは、リテラルではなくシンタックス・フラグメントの名前です。 たとえば、2.1 %OP1 という行は別のシンタックス・フラグメント OP1 を参照すべきことを意味します。

以下のワードおよびシンボルが、ドット 10 進数の次に使用されます。

- ? は、オプションのシンタックス・エレメントであることを表します。? シンボルが後に続くドット 10 進数は、対応するドット 10 進数のシンタックス・エレメント、および任意の従属のシンタックス・エレメントがオプションであることを示します。ドット 10 進数の付いたシンタックス・エレメントが 1 つしかない場合、 ? シンボルはそのシンタックス・エレメントと同じ行に表示されます (たとえば、5? NOTIFY)。ドット 10 進数の付いたシンタックス・エレメントが複数ある場合、 ? シンボルだけで行に表示され、その後にオプションのシンタックス・エレメントが続きます。たとえば、「5 ?、5 NOTIFY、および 5 UPDATE」という行を聞き取る場合、シンタックス・エレメント NOTIFY および UPDATE がオプションである、つまりそのいずれかを選択でき、どちらも選択しないこともできることが分かります。 ? シンボルは、線路型ダイアグラムのバイパス線に相当します。
- ! は、デフォルトのシンタックス・エレメントであることを表します。! シンボルおよびシンタックス・エレメントが後に続くドット 10 進数は、そのシンタックス・エレメントが、同じドット 10 進数を共用するシンタックス・エレメントすべてのデフォルト・オプションであることを示します。同じドット 10 進数を共用するシンタックス・エレメントのうち 1 つだけに、 ! シンボルを指定できます。たとえば、「2? FILE、2.1! (KEEP)、および 2.1 (DELETE)」という行を聞き取る場合、 FILE キーワードのデフォルト・オプションは (KEEP) になります。この例では、FILE キーワードを含めてもオプションを指定しない場合には、デフォルト・オプション KEEP が適用されます。デフォルト・オプションは、次に高位のドット 10 進数にも適用されます。 この例の場合、FILE キーワードが省略されると、デフォルトの FILE(KEEP) が使用されます。しかし、「2? FILE、2.1、2.1.1! (KEEP)、および 2.1.1 (DELETE)」という行を聞き取る場合、デフォルト・オプション KEEP は次に高位のドット 10 進数 2.1 (関連キーワードを持っていない) にのみ適用され、 2? FILE には適用されません。キーワード FILE が省略されると、どれも使用されません。
- * は、0 回以上反復できるシンタックス・エレメントを示します。 * シンボルが後に続くドット 10 進数は、このシンタックス・エレメントが 0 回以上使用できること、つまりオプションであり、なおかつ反復できることを表します。たとえば、5.1* データ域という行を聞き取る場合、 1 つまたは複数のデータ域を含めるか、またはデータ域を全く含めないことが可能です。「3*、3 HOST、および 3 STATE」という行を聞き取る場合、 HOST、STATE をどちらか一方または両方同時に含めるか、どちらも含めないことができます。

  注:

  1. ドット 10 進数の後にアスタリスク (*) が付き、ドット 10 進数の付いた項目が 1 つしかない場合には、同じ項目を複数回反復できます。
  2. ドット 10 進数の後にアスタリスクが付き、ドット 10 進数の付いた項目が複数ある場合、リストから複数の項目を使用できますが、各項目を複数回使用することはできません。 前述の例では、HOST STATE と書くことはできますが、HOST HOST とは書けません。
  3. * シンボルは、線路型シンタックス・ダイアグラムのループバック線に相当します。
- + は、1 回以上含める必要のあるシンタックス・エレメントであることを示します。 + シンボルが後に続くドット 10 進数は、このシンタックス・エレメントを 1 回以上含める必要があること、つまり少なくとも 1 回は含める必要があり、反

復できることを表します。たとえば、「6.1+ データ域」という行を聞き取る場合、データ域を少なくとも 1 回は含めなければなりません。「2+、2 HOST、および 2 STATE」という行を聞き取る場合には、 HOST、STATE、またはその両方を含める必要があります。 * シンボルと同様に、+ シンボルは、ドット 10 進数の付いた項目が 1 つしかない場合に限り、その特定の項目のみを反復できます。 * シンボルと同様、 + シンボルは線路型シンタックス・ダイアグラムのループバック線に相当します。

関連概念:

- 801 ページの『アクセス支援』

関連タスク:

- 『キーボード・ショートカットおよびアクセラレーター: Common GUI help』

関連資料:

- 「*SQL リファレンス 第 2 巻*」の『構文図の見方』

DB2 Universal Database 製品の共通基準認証

DB2 Universal Database は、 Common Criteria の評価検定レベル 4 (EAL4) で認証の評価を受けています。 Common Criteria の詳細については、以下の Common Criteria の Web サイトを参照してください。 http://niap.nist.gov/cc-scheme/

付録 F. 特記事項

本書に記載の製品、サービス、または機能が日本においては提供されていない場合があります。日本で利用可能な製品、サービス、および機能については、日本 IBM の営業担当員にお尋ねください。本書で IBM 製品、プログラム、またはサービスに言及していても、その IBM 製品、プログラム、またはサービスのみが使用可能であることを意味するものではありません。これらに代えて、IBM の知的所有権を侵害することのない、機能的に同等の製品、プログラム、またはサービスを使用することができます。ただし、IBM 以外の製品とプログラムの操作またはサービスの評価および検証は、お客様の責任で行っていただきます。

IBM は、本書に記載されている内容に関して特許権 (特許出願中のものを含む) を保有している場合があります。本書の提供は、お客様にこれらの特許権について実施権を許諾することを意味するものではありません。実施権についてのお問い合わせは、書面にて下記宛先にお送りください。

〒106-0032
東京都港区六本木 3-2-31
IBM World Trade Asia Corporation
Licensing

以下の保証は、国または地域の法律に沿わない場合は、適用されません。 IBM およびその直接または間接の子会社は、本書を特定物として現存するままの状態で提供し、商品性の保証、特定目的適合性の保証および法律上の瑕疵担保責任を含むすべての明示もしくは黙示の保証責任を負わないものとします。国または地域によっては、法律の強行規定により、保証責任の制限が禁じられる場合、強行規定の制限を受けるものとします。

この情報には、技術的に不適切な記述や誤植を含む場合があります。本書は定期的に見直され、必要な変更は本書の次版に組み込まれます。 IBM は予告なしに、随時、この文書に記載されている製品またはプログラムに対して、改良または変更を行うことがあります。

本書において IBM 以外の Web サイトに言及している場合がありますが、便宜のため記載しただけであり、決してそれらの Web サイトを推奨するものではありません。それらの Web サイトにある資料は、この IBM 製品の資料の一部ではありません。それらの Web サイトは、お客様の責任でご使用ください。

IBM は、お客様が提供するいかなる情報も、お客様に対してなんら義務も負うことのない、自ら適切と信ずる方法で、使用もしくは配布することができるものとします。

本プログラムのライセンス保持者で、(i) 独自に作成したプログラムとその他のプログラム(本プログラムを含む)との間での情報交換、および (ii) 交換された情報の相互利用を可能にすることを目的として、本プログラムに関する情報を必要とする方は、下記に連絡してください。

IBM Canada Limited

Office of the Lab Director
8200 Warden Avenue
Markham, Ontario
L6G 1C7
CANADA

本プログラムに関する上記の情報は、適切な使用条件の下で使用することができますが、有償の場合もあります。

本書で説明されているライセンス・プログラムまたはその他のライセンス資料は、IBM 所定のプログラム契約の契約条項、IBM プログラムのご使用条件、またはそれと同等の条項に基づいて、 IBM より提供されます。

この文書に含まれるいかなるパフォーマンス・データも、管理環境下で決定されたものです。そのため、他の操作環境で得られた結果は、異なる可能性があります。一部の測定が、開発レベルのシステムで行われた可能性がありますが、その測定値が、一般に利用可能なシステムのものと同じである保証はありません。さらに、一部の測定値が、推定値である可能性があります。実際の結果は、異なる可能性があります。お客様は、お客様の特定の環境に適したデータを確かめる必要があります。

IBM 以外の製品に関する情報は、その製品の供給者、出版物、もしくはその他の公に利用可能なソースから入手したものです。 IBM は、それらの製品のテストは行っておりません。したがって、他社製品に関する実行性、互換性、またはその他の要求については確証できません。 IBM 以外の製品の性能に関する質問は、それらの製品の供給者にお願いします。

IBM の将来の方向または意向に関する記述については、予告なしに変更または撤回される場合があり、単に目標を示しているものです。

本書には、日常の業務処理で用いられるデータや報告書の例が含まれています。より具体性を与えるために、それらの例には、個人、企業、ブランド、あるいは製品などの名前が含まれている場合があります。これらの名称はすべて架空のものであり、名称や住所が類似する企業が実在しているとしても、それは偶然にすぎません。

著作権使用許諾:

本書には、様々なオペレーティング・プラットフォームでのプログラミング手法を例示するサンプル・アプリケーション・プログラムがソース言語で掲載されています。お客様は、サンプル・プログラムが書かれているオペレーティング・プラットフォームのアプリケーション・プログラミング・インターフェースに準拠したアプリケーション・プログラムの開発、使用、販売、配布を目的として、いかなる形式においても、IBM に対価を支払うことなくこれを複製し、改変し、配布することができます。このサンプル・プログラムは、あらゆる条件下における完全なテストを経ていません。従って IBM は、これらのサンプル・プログラムについて信頼性、利便性もしくは機能性があることをほのめかしたり、保証することはできません。

それぞれの複製物、サンプル・プログラムのいかなる部分、またはすべての派生した創作物には、次のように、著作権表示を入れていただく必要があります。

商標

以下は、IBM Corporation の商標です。

ACF/VTAM
AISPO
AIX
AIXwindows
AnyNet
APPN
AS/400
BookManager
C Set++
C/370
CICS
Database 2
DataHub
DataJoiner
DataPropagator
DataRefresher
DB2
DB2 Connect
DB2 Extenders
DB2 OLAP Server
DB2 Information Integrator
DB2 Query Patroller
DB2 Universal Database
Distributed Relational Database Architecture
DRDA
eServer
Extended Services
FFST
First Failure Support Technology
IBM
IMS
IMS/ESA
iSeries
LAN Distance
MVS
MVS/ESA
MVS/XA
Net.Data
NetView
OS/390
OS/400
PowerPC
pSeries
QBIC
QMF
RACF
RISC System/6000
RS/6000
S/370
SP
SQL/400
SQL/DS
System/370
System/390
SystemView
Tivoli
VisualAge
VM/ESA
VSE/ESA
VTAM
WebExplorer
WebSphere
WIN-OS/2
z/OS
zSeries

以下は、それぞれ各社の商標または登録商標です。

Microsoft、Windows、Windows NT および Windows ロゴは、Microsoft Corporation の米国およびその他の国における商標です。

Pentium は、Intel Corporation の米国およびその他の国における商標です。

Java およびすべての Java 関連の商標およびロゴは、Sun Microsystems, Inc. の米国およびその他の国における商標または登録商標です。

索引

日本語, 数字, 英字, 特殊文字の順に配列されています。なお, 濁音と半濁音は清音と同等に扱われています。

[ア行]

アクセス支援
  機能 801
  小数点付き 10 進数の構文図 803
アプリケーション
  延期 731
  開発用の DB2 ツール 3
  サポートされているプログラミング・インターフェース 6
  セーブポイントによるトランザクションの管理 697
  セーブポイントの制約事項 702
  データ・ソースへの接続、IBM OLE DB Provider 269
  ホスト環境と iSeries 環境 757
  マルチサイト更新のプリコンパイル 691
  ループ 731
  ADO
    更新可能な両方向スクロール・カーソル 264
    制限 264
  DB2 プログラミング機能 20
  IBM OLE DB Provider でサポートされている 248
  Java 2 Enterprise Edition でサポートされている 521
  MQSeries 関数 19
  Visual Basic、データ・ソースへの接続 264
  Web アプリケーションの構築のためのツール 17
  X/Open XA インターフェース、リンケージ 709
アプリケーション開発
  DB2 .NET Data Provider 16
  IBM DB2 Development Add-In 4
アプリケーション環境、プログラミングの 29
アプリケーション設計
  以前に検索されたデータのスクロール 108
  エラー処理の指針 37
アプリケーション設計 (続き)
  同じ名前を持つパッケージのバージョン 74
  カーソル処理 102
  可変リスト・ステートメントの処理 141
  疑似コード 44
  記述、SELECT ステートメントの 134
  組み込みファイル、COBOL 200
  コード・ポイント、特殊文字の 664
  サンプル・プログラム 112
  十分な SQLVAR エンティティーの宣言 129
  照合順序のガイドライン 653
  スタンドアロン・アプリケーションの構造 30
  静的 SQL の利点 88
  データ値の制御 46
  データの受け渡しの指針 139
  データ・オブジェクトのリレーションシップ 49
  テスト環境の設定 56
  動的 SQL キャッシュ 88
  動的 SQL の目的 119
  バインディング 65
  パラメーター・マーカーの使用 142
  必要なステートメント 31
  プリコンパイル 65
  並行ユーザー、宣言済み一時表 744
  変数のないステートメントの実行 120
  保管、エンド・ユーザー要求の 142
  マルチサイト更新の目的 688
  文字変換、ストアード・プロシージャーにおける 665
  文字変換、SQL ステートメントの 664
  文字変換に関する考慮事項 662
  ロジック、論理における 51
  2 度目のデータ検索 109
  2 バイト文字のサポート (DBCS) 664
  COBOL の日本語および中国語 (繁体字) EUC に関する考慮事項 219
  NULL 値の受け取り 95
  Perl でのプロトタイピング 535
  Perl の例 538
  REXX、ルーチンの登録 540
  SQLDA 構造体の作成の指針 136
アプリケーションのロジック
  サーバー 51
  ストアード・プロシージャー 51
アプリケーションのロジック (続き)
  データ値の制御 49
  データ・リレーションシップの制御 51
  トリガー 51
  ユーザー定義関数 51
アプリケーション・パフォーマンス
  シーケンス・オブジェクト 743
  シーケンス・オブジェクトと ID 列の比較 744
  宣言済み一時表 744
  データのブロックの受け渡し 746
  ローカル・バイパス 717
アプリケーション・プログラミング・インターフェース (API)
  概説 46
  許可についての考慮事項 55
  使用 46
  制約事項、XA 環境における 705
  設定、スレッド間のコンテキスト
    sqleAttachToCtx() 193
    sqleBeginCtx() 193
    sqleDetachFromCtx() 193
    sqleEndCtx() 193
    sqleGetCurrentCtx() 193
    sqleInterruptCtx() 193
    sqleSetTypeCtx() 193
  タイプ 46
  REXX の構文 553
アプリケーション・プログラム
  既存のリスト・ルーチン 116
  組み込み SQL の概説 8
  構造 30
  最適化 60
    パーティション・データベース環境 715
  順序の制御 742
  ストアード・プロシージャーの概説 21
  静的 SQL
    戻りコード 114
  静的 SQL の例 89
  前提条件 29
  データベースへの接続 38
  テスト環境の設定 56
  デバッグ 60
  トリガーの概説 26
  必要なステートメント 31
  COBOL
    ホスト変数の例 215

アプリケーション・プログラム *(続き)*
COBOL *(続き)*
マルチスレッド・データベース・アクセスなし 199
DB2 CLI の概説 10
DB2 Connect によるマルチサイト更新 768
DB2 アプリケーション・プログラミング・インターフェース 8
DBCS 環境の考慮事項 672
FORTRAN
マルチスレッド・データベース・アクセスなし 222
Java 用組み込み SQL (SQLJ) の概説 14
Net.Data の概説 19
ODBC エンド・ユーザー・ツール 16
OLE DB 表関数 25
Perl
マルチスレッド・データベース・アクセスなし 535
Perl DBI 16
REXX
ストアード・プロシージャーの呼び出し、サーバーの考慮事項 556
マルチスレッド・データベース・アクセスなし 541
暗黙接続
プラットフォームによる違い 760
移植性、組み込み SQL ではなく CLI を使用する場合の 146
位置指定イテレーター
変数として渡される、SQLJ 396
SQLJ アプリケーション 370
位置指定更新
SQLJ 373
位置指定削除
SQLJ 373
一時表
宣言済み 744
一貫性
データの 39
イテレーター
位置指定、SQLJ 370
指定された、SQLJ 367
スクロール可能、SQLJ 399
JDBC 結果セットの取得 381
イテレーター変換文節、SQLJ 448
印刷
PDF ファイル 795
印刷資料、注文 796
インストール
インフォメーション・センター 778, 780, 783
Universal JDBC ドライバー 481
インフォメーション・センター
インストール 778, 780, 783
インプリメント文節、SQLJ 438
エラー
バッファー化挿入での検出 720
SQLJ での処理 380
エラー処理
エラーを戻したデータベース・パーティションの識別 730
延期状態のアプリケーション 731
組み込みファイル
COBOL 200
C/C++ 152
FORTRAN 222
パーティション・データベース環境 728, 729
プリコンパイル時 69
ループ状態のアプリケーション 731
レポート 730
C/C++ 言語プリコンパイラー 155
Perl 537
SQLCA 構造体 730
使用 36
マージされた複数の構造 730
SQLCODE 730
WHENEVER ステートメント 37
エラー・コード、JDBC
DB2 Universal JDBC ドライバーのエラー 479
エラー・メッセージ
エラー状態フラグ 114
エラー処理 36
警告状態フラグ 114
例外状態フラグ 114
SQLCA 構造体 114
SQLSTATE 114
SQLWARN 構造体 114
延期状態またはループ状態のアプリケーションの診断 731
オーバーフロー、数値 765
オープン状態、バッファー化挿入 720
オブジェクト指向 COBOL の制約事項 219
オプティマイザー
静的および動的 SQL の考慮事項 121
重みの定義 653
オンライン
ヘルプへのアクセス 797

[カ行]

カーソル
解放、ロック動作の 102
行
更新 105
削除 105
カーソル *(続き)*
行の検索 101
更新可能
定義 106
ADO アプリケーションでの 264
作業単位
完了 102
サンプル・プログラム 107
処理
サマリー 100
動的 SQL における 123
SQLDA 構造体での 135
静的 SQL の例 104
宣言
静的 SQL プログラムにおける 101
タイプ 106
動的 SQL のサンプル・プログラム 124
名前、REXX 541
表の末尾に置く 111
複数行を取り出す 100
複数使用、アプリケーションにおける 100
振る舞い
COMMIT ステートメントでの 102
ROLLBACK TO SAVEPOINT 後 702
ブロッキング、セーブポイントに関する考慮事項 704
未確定 106
無効なパッケージ、行の取り出し 102
目的 90, 100
読み取り専用
作業単位の考慮事項 102
宣言 101
定義 106
パーティション・データベース環境で 715
両方向スクロール
ADO アプリケーションでの 264
CLI (コール・レベル・インターフェース)
組み込み動的 SQL との比較 145
COMMIT に関する考慮事項 102
FOR FETCH ONLY 106
IBM OLE DB Provider 252
REXX 549
ROLLBACK に関する考慮事項 102
WITH HOLD
パッケージの再バインド、作業単位中の 102
振る舞い、COMMIT 後の 102
振る舞い、ROLLBACK 後の 102

カーソル *(続き)*
WITH HOLD *(続き)*
X/Open XA インターフェース 705
カーソル固定 (CS)
ホスト環境と iSeries 環境 764
外部キー
プラットフォームによる違い 764
拡張 UNIX コード (EUC)
アプリケーションでの拡張 677
拡張のサンプル 680
可変長データ・タイプ 683
クライアント・ベースのパラメーターの検証 680
グラフィック定数 675
グラフィック・データの処理 675
固定長データ・タイプ 683
異なるコード・ページ 677
混合しているコード・ページ 674
サーバーでの拡張 677
照合に関する考慮事項 675
ストアード・プロシージャー 675
中国語 (繁体字)
コード・セット 673
考慮事項 675
COBOL に関する考慮事項 219
C/C++ 183
FORTRAN 235
REXX 541
日本語
コード・セット 673
C/C++ 183
FORTRAN 235
REXX 541
日本語および中国語 (繁体字)
COBOL に関する考慮事項 219
文字ストリング長のオーバーフロー 683
文字セット 671
文字変換、ストアード・プロシージャー 683
文字変換によるオーバーフロー 683
2 バイト・コード・ページ 674
DBCLOB ファイル 675
DESCRIBE ステートメント 681
UDF (ユーザー定義関数) に関する考慮事項 675
拡張、データの
iSeries サーバー 759
OS/390 サーバー 759
拡張クライアント情報
DB2 Universal JDBC ドライバー 348
カスケード・レベル 764
型付きパラメーター・マーカー 142
カタログ統計
ユーザー更新可能 45
各国語サポート (NLS)
コード・ページ 666
混合バイト・データ 759
文字変換 666
環境 API
組み込みファイル
COBOL 200
C/C++ 152
FORTRAN 222
環境ハンドル
説明 145
環境変数
DB2INCLUDE 155, 225
管理通知ログ
パーティション・データベース環境 729
完了コード 36
キー
外部
プラットフォームによる違い 764
主 764
キーボード・ショートカット
サポート 801
記述子ハンドル
説明 145
行
位置決め、表内での 111
検索、カーソルによる 106
検索、SQLDA を用いた 134
取り出し、パッケージが無効になった後 102
複数の取り出し 100
2 度目の検索
行の順序 110
メソッド 109
行グループ、バッファー化挿入での 720
共通言語ランタイム
ルーチン
サポートされる SQL データ・タイプ 244
行のブロック化
パフォーマンス向上のためのカスタマイズ 746
行レベルのロック
ホスト環境と iSeries 環境 764
切り捨て
標識変数 95
ホスト変数 95
組み込み SQL
概説 8, 63
規則
C/C++ 156
FORTRAN 226
許可 53
構文規則 63
組み込み SQL *(続き)*
コメント
規則 226
COBOL 203
C/C++ 156
生成
順次値 741
生成された
列 733
ホスト変数の参照 91, 94
例 63
COBOL 203
C/C++ におけるコメント 156
DB2 CLI との比較 148
ID 列 734
組み込み SQL for Java (SQLJ)
概説 14
組み込み動的 SQL 11
組み込みファイル
位置
COBOL での 203
C/C++ における 155
FORTRAN での 225
要件
COBOL 200
C/C++ 152
FORTRAN 222
SQL
COBOL 用 200
C/C++ 用 152
FORTRAN の 222
SQL1252A
COBOL 200
FORTRAN 222
SQL1252B
COBOL 200
FORTRAN 222
SQLADEF、C/C++ の 152
SQLAPREP
COBOL 用 200
C/C++ 用 152
FORTRAN の 222
SQLCA
COBOL 用 200
C/C++ 用 152
FORTRAN の 222
SQLCACN
FORTRAN 222
SQLCACS
FORTRAN 222
SQLCA_92
COBOL 200
FORTRAN 222
SQLCLI1、C/C++ の 152
SQLCLI、C/C++ の 152

組み込みファイル *(続き)*
SQLCODES
COBOL 用 200
C/C++ 用 152
FORTRAN の 222
SQLDA
COBOL 200
C/C++ 用 152
FORTRAN の 222
SQLDACT
FORTRAN 222
SQLE819A
COBOL 用 200
C/C++ 用 152
FORTRAN の 222
SQLE819B
COBOL 用 200
C/C++ 用 152
FORTRAN の 222
SQLE850A
COBOL 用 200
C/C++ 用 152
FORTRAN の 222
SQLE850B
COBOL 用 200
C/C++ 用 152
FORTRAN の 222
SQLE932A
COBOL 用 200
C/C++ 用 152
FORTRAN の 222
SQLE932B
COBOL 用 200
C/C++ 用 152
FORTRAN の 222
SQLEAU
COBOL 用 200
C/C++ 用 152
FORTRAN の 222
SQLENV
COBOL 200
C/C++ 用 152
FORTRAN 222
SQLETSD
COBOL 200
SQLEXT、C/C++ の 152
SQLJACB、C/C++ の 152
SQLMON
COBOL 200
C/C++ 用 152
FORTRAN 222
SQLMONCT
COBOL 用 200
SQLSTATE
COBOL 用 200
C/C++ 用 152
組み込みファイル *(続き)*
SQLSTATE *(続き)*
FORTRAN の 222
SQLSYSTM、C/C++ の 152
SQLUDF、C/C++ の 152
SQLUTBCQ
COBOL 200
SQLUTBSQ
COBOL 200
SQLUTIL
COBOL 用 200
C/C++ 用 152
FORTRAN の 222
SQLUVEND、C/C++ の 152
SQLUV、C/C++ の 152
SQLXA、C/C++ の 152
組み込みファイルの位置、FORTRAN での 225
クライアント
REXX でのストアード・プロシージャーの呼び出し 556
クライアント転送サポート
DB2 Universal JDBC ドライバー 347
クライアント/サーバーのコード・ページ変換 666
クライアント・ベースのパラメーターの検証 680
クラス・データ・メンバー 177
グラフィック定数
中国語 (繁体字) コード・セット 675
日本語コード・セット 675
グラフィック・データ
中国語 (繁体字) コード・セット 673, 675
日本語コード・セット 673, 675
グラフィック・ホスト変数
COBOL 209
C/C++ 165
クリティカル・セクション 196
クローズ、バッファー化挿入の 717
クローズ状態
バッファー化挿入 720
警告メッセージ
切り捨て 95
結果コード 36
結果セットについての情報の検索
JDBC 333
結果セット・イテレーター
宣言に関する制約事項 371
別個のファイルでの public 宣言 382
検索
DB2 資料 776
検索割り当て
数値変換のオーバーフロー 765
コーディネーター・パーティション、バッファー化挿入なし 717
コード・セット
SQLCA の SQLERRMC フィールド 760
コード・ページ
アプリケーションの実行用 666
各国語サポート (NLS) 666
サポートされる変換 668
処理、アプリケーションでの拡張の 677
処理、異なる場合の 669, 677
処理、サーバーでの拡張の 677
ストレージの割り振り、異なる場合の 677
バインドの考慮事項 80
プリコンパイルおよびバインド用 666
変換
iSeries サーバー 759
OS/390 サーバー 759
文字変換 666
文字変換時 667
ロケールの導出 661
DB2CODEPAGE レジストリー変数 661
SQLCA の SQLERRMC フィールド 760
Windows コード・ページ 661
コード・ページ変換
文字置換 668
コード・ポイント 653
コール・レベル・インターフェース (CLI)
概説 145
組み込み SQL と DB2 CLI の比較 145
組み込み SQL との比較 148
サポートされる SQL ステートメント 751
利点 146
更新
DB2 インフォメーション・センター 787
DB2 表に対する、JDBC 310
更新可能な結果セット
JDBC 342
構成パラメーター
マルチサイト更新 692
locktimeout 196
構造化照会言語 (SQL)
サポートされているステートメント
コール・レベル・インターフェース (CLI) 751
コマンド行プロセッサー (CLP) 751
動的 SQL 751
SQL プロシージャー型言語 751

構造型
DB2 Connect によってサポートされていない 759
構文
組み込み SQL ステートメント
空白文字での置換 156
継続行 156
コメント、COBOL 203
コメント、C/C++ 156
コメント、FORTRAN 226
コメント、REXX 541
COBOL 203
C/C++ 156
FORTRAN 226
宣言セクション
COBOL 206
C/C++ 160
FORTRAN 228
文字ホスト変数 162
LOB 標識宣言、REXX 547
SQL ステートメントの組み込み
REXX 541
異なるコード・ページ 677
コマンド
FORCE 760
コマンド行プロセッサー (CLP)
キャッシュ設定、DB2INCLUDE 環境変数の 155
サポートされる SQL ステートメント 751
プロトタイピング 45
REXX アプリケーションからの呼び出し 553
コマンド・ヘルプ
呼び出し 798
コミット
トランザクション、JDBC 306
トランザクション、SQLJ 363
コミット、変更の
表 40
コメント
組み込み SQL ステートメント 541
SQL の規則 156, 203, 226
SQLJ アプリケーション 356
コレクション ID 属性
パッケージ 761
DB2 for iSeries 761
混合コード・ページ環境
パッケージ名 665
混合している拡張 UNIX コードの考慮事項 674
混合バイト・データ
iSeries サーバー 759
OS/390 サーバー 759
コンテキスト
その間でのアプリケーションの従属関係 196
その間でのデータベースの従属関係 196
デッドロック回避 196
マルチスレッド DB2 アプリケーションでの設定
説明 193
コンテキスト文節、SQLJ 444
コンテナー
Java 2 Enterprise Edition 522
コンパイル
概説 72
コンパイルされるアプリケーション、パッケージの作成 67
コンパウンド SQL
セーブポイントとの比較 699
DB2 Connect サポート 767

[サ行]

最適化
アプリケーション・プログラム 60
再バインド
説明 84
REBIND PACKAGE コマンド 84
作業単位 (UOW)
カーソル動作の完了 102
カーソルに関する考慮事項 102
コーディング 39
リモート 687
作成
コンパイルされるアプリケーション用のパッケージ 67
DB2 オブジェクト、JDBC 308
DB2 オブジェクト、SQLJ 366
参照制約
データ値の制御 48
参照保全
データ・リレーションシップについての考慮事項 49
プラットフォームによる違い 764
識別順序 653
シグナル・ハンドラー
インストール、サンプル・プログラム 112
目的 115
COMMIT と ROLLBACK に関する考慮事項 115
SQL ステートメントでの 115
システム要件
IBM 0LE DB Provider for DB2 247
システム・カタログ
ホスト環境と iSeries 環境 765
システム・カタログ・ビュー
プロトタイピング・ユーティリティー 45
実行
JDBC アプリケーションでの SQL 307
SQLJ アプリケーションでの SQL 365
実行可能文節、SQLJ 443
実行コンテキスト
SQLJ 390
実行振る舞い、DYNAMICRULES 126
シフトアウト文字、プラットフォームによる違い 759
修飾およびメンバー演算子、C/C++ における 178
重大エラー、パーティション・データベース環境 729
終了、暗黙にトランザクションを 43
主キー
プラットフォームによる違い 764
出力ファイル
C/C++ 152
出力ファイルの拡張子
C/C++ 152
順次値、順序を参照 741
順序
アプリケーション・パフォーマンス 743
目的 741
ID 列との比較 744
照会
更新可能 105
削除可能 105
照合
中国語 (繁体字) コード・セット 675
日本語コード・セット 675
照合順序
大文字小文字に無関係な比較 656
概説 653
組み込みファイル
COBOL 200
C/C++ 152
FORTRAN 222
コード・ポイント 653
サンプル 660
識別順序 653
事項、一般 653
指定 658
ソート順序の例 657
マルチバイト文字 653
文字比較 655
EBCDIC と ASCII のソート順序
説明 764
例 657

照合順序 *(続き)*
EBCDIC バイナリー照合のシミュレート 771
TRANSLATE 関数 656
小数点付き 10 進数の構文図 803
シリアライズ
データ構造 194
SQL ステートメント実行 193
資料
表示 786
身体障害 801
シンボル
置換、C/C++ 言語制限 171
数値データ・タイプ
プラットフォームによる違い 759
数値変換のオーバーフロー 765
数値ホスト変数
COBOL 206
C/C++ 160
FORTRAN 229
スキーマ行セット
IBM OLE DB Provider 249
スクロール可能イテレーター
データ・タイプの制約事項 399
SQLJ アプリケーションでの使用 399
スクロール可能な結果セット
データ・タイプの制約事項 342
JDBC 342
ステートメント
キャッシュ、WebSphere 577
準備、最小の SQLDA 構造体を使用するための 131
ACQUIRE、DB2 UDB ではサポートされていない 769
BEGIN DECLARE SECTION 31
CALL USING DESCRIPTOR 766
CALL、サポートされているプラットフォーム 766
COMMIT 40
COMMIT WORK RELEASE 769
CONNECT 760
CREATE SEQUENCE 741
DB2 Connect
サポートされていない 769
サポートされる 769
DECLARE CURSOR 38
DECLARE、DB2 UDB ではサポートされていない 769
DESCRIBE 769
END DECLARE SECTION 31
INCLUDE 38
INCLUDE SQLCA 36
INCLUDE SQLDA 38
LABEL ON、DB2 UDB ではサポートされていない 769
RELEASE SAVEPOINT 701
ステートメント *(続き)*
ROLLBACK
終了、トランザクションの 41
宣言済み一時表 744
プラットフォームによる違い 760
ROLLBACK TO SAVEPOINT 701
SAVEPOINT 701
ステートメント文節、SQLJ 445
ステートメント・ハンドル
説明 145
ストアード・プロシージャー
アプリケーションのロジックについての考慮事項 51
概説 21
結果セットの検索 391
サポートされているプラットフォーム 766
初期化
REXX 変数 554
中国語 (繁体字) コード・セット 675
日本語コード・セット 675
複数の結果セットの検索、JDBC 330
文字変換 665
文字変換、EUC 683
呼び出し
JDBC 313
REXX 555
SQLJ 379
REXX アプリケーション 554
ストリング
NULL 終了、C、CNULREQD BIND オプション 762
ストレージ
行を保持するための割り振り 134
異なるコード・ページに対する割り振り 677
十分な SQLVAR エンティティーの宣言 129
スレッド
複数
国別/地域別コードに関する考慮事項 195
コード・ページに関する考慮事項 195
コンテキスト間でのアプリケーションの従属関係 196
コンテキスト間でのデッドロックを避ける 196
コンテキスト間におけるデータベースの従属関係 196
推奨 194
潜在的な問題 196
DB2 アプリケーションでの使用 193
UNIX アプリケーションに関する考慮事項 195
スレッド *(続き)*
IBM OLE DB Provider 249
IBM OLE DB Provider for DB2 247
セーブポイント
アトミック・コンパウンド SQL 702
カーソル・ブロッキングに関する考慮事項 704
解放、JDBC 327
解放、SQLJ 363
コンパウンド SQL との比較 699
作成 701
作成、JDBC 327
作成、SQLJ 363
制御 701
制約事項 702
データ定義言語 (DDL) 702
トランザクション管理 697
トリガー 702
ネストされた 702
バッファー化挿入 704, 717
SET INTEGRITY ステートメント 702
XA トランザクション・マネージャー 705
制御ステートメント実行
SQLJ 390
成功コード 36
整合性トークン 82
整数データ・タイプ、64 ビット
DB2 Connect によってサポートされている 759
生成列
目的 733
静的 SQL
概説 87
許可 55
考慮事項 121
サンプル・カーソル・プログラム 104
サンプル・プログラム 89
静的更新プログラムの例 112
宣言、ホスト変数の 93
データの検索 90
動的 SQL
対比 87
比較 121
パフォーマンス 88
プリコンパイルの利点 82
ホスト変数の使用 91
DB2 Connect サポート 757
Perl、サポートされていない 535
制約事項
バッファー化挿入 722
COBOL 199
C/C++ の 171
FORTRAN 221
IBM OLE DB Provider 256

制約事項 *(続き)*
REXX 540
SQLJ 変数名 356
セキュリティー
暗号化されたユーザー ID または暗号化されたパスワード
DB2 Universal JDBC ドライバー 492
プラグイン 581
エラー・メッセージ 605
グループ検索プラグイン 592
グループ検索プラグイン用の API 612
制約事項 648
セキュリティー・プラグインの配置 595, 597
セキュリティー・プラグイン・インフラストラクチャーの概説 581
セキュリティー・プラグイン・ライブラリーに関する制約事項 601
デバッグ、問題判別の 590
配置 755
パスワードを検査する API 637
プラグインの配置 592, 593
プラグイン配置の制約事項 755
命名 586
戻りコード 603
ユーザー ID/パスワード・プラグイン用の API 622
呼び出しシーケンス、呼び出される順序の 606
ライブラリー、セキュリティー・プラグインの場所 585
ロード 599
2 パーツ・ユーザー ID のサポート 587
32 ビットの考慮事項 590
64 ビットの考慮事項 590
API 611, 615, 616, 620, 621, 622, 629, 630, 631, 633, 635, 639, 640, 641, 643, 644, 646
API、API のバージョン 649
GSS-API プラグイン用の API 647
SQLCODES、SQLSTATES 関連の 590
ユーザー ID およびパスワード
DB2 Universal JDBC ドライバー 489
ユーザー ID 専用
DB2 Universal JDBC ドライバー 491
DB2 JDBC Type 2 Driver 487
DB2 Universal JDBC ドライバー 488
Kerberos
DB2 Universal JDBC ドライバー 493
接続
暗黙
プラットフォームによる違い 760
データ・ソースへの、DataSource を使用した 303
データ・ソースへの、DriverManager を使用した
DB2 JDBC Type 2 Driver 298
DB2 Universal JDBC ドライバー 300
データ・ソースへの、SQLJ を使用した 357
プール
WebSphere 575, 576
CONNECT RESET ステートメント 760
CONNECT TO ステートメント 760
JDBC での使用 306
NULL CONNECT 760
接続宣言文節、SQLJ 441, 442
接続のクローズ
JDBC データ・ソース 307
SQLJ データ・ソース 364
接続ハンドル
説明 145
接続プール
ADO 713
ODBC 713
セマフォー 196
宣言
標識変数 95
ホスト変数、規則 91
JDBC アプリケーションでの変数 296
SQLJ アプリケーションでの変数 355
宣言済み一時表
目的 744
ROLLBACK ステートメント 744
宣言セクション
作成 31
ステートメントの規則 91
COBOL 206
COBOL での 215
C/C++ 184
C/C++ の 160
FORTRAN 228, 233
全選択
バッファー化挿入考慮事項 722
ソース
組み込み SQL アプリケーション 71
修正済みソース・ファイル 69
ファイル名拡張子 69
SQL ファイル拡張子 65
ソース・ファイル
作成 65
ソート
結果の配列 764
ソート *(続き)*
照合順序 658, 764
送信、大量データの 746
挿入、バッファー化挿入なし 717

[タ行]

ターゲット・パーティション
バッファー化挿入なしの振る舞い 717
タイム・スタンプ
プリコンパイル時 82
チュートリアル 799
トラブルシューティングおよび問題判別 800
中国語 (繁体字) コード・セット
拡張 UNIX コード
考慮事項 673
変換の考慮事項 675
2 バイトの考慮事項 675
COBOL に関する考慮事項 219
C/C++ の考慮事項 183
FORTRAN 235
REXX の考慮事項 541
UCS2 に関する考慮事項 673
抽出
大量データ 723
注文、DB2 資料 796
長フィールド
バッファー化挿入、制約事項 722
プラットフォームによる違い 759
ツール
開発用の 3
データ
以前に検索された
更新 112
スクロール 108
一貫性、トランザクション・レベルでの 39
更新 105
コミット、変更の 40
削除 105
大量データの送信 746
大量の抽出 723
テスト用の生成 58
展開
iSeries サーバー 759
OS/390 サーバー 759
取り出されたデータの保管 109
取り出し
静的 SQL での 90
2 度目の 109
パーティション・データベース環境 723
不整合 41
変更の取り消し、ROLLBACK ステートメントによる 41

データ *(続き)*
リレーションシップの制御 49
2 度目の検索 110
Microsoft 仕様でのアクセス 15
WebSphere でのエンタープライズへのアクセス 575
データ値の制御
アプリケーションのロジックと変数のタイプ 49
参照保全制約 48
データ・タイプ 47
表チェック制約 48
目的 46
ユニーク制約 47
CHECK OPTION のあるビュー 48
データ構造
宣言 31
ユーザー定義の、複数のスレッドにおける 194
SQLEDBDESC 658
データ制御言語 (DCL)
ホスト環境と iSeries 環境 760
データ操作言語 (DML)
ホスト環境と iSeries 環境 759
データ定義言語 (DDL)
セーブポイントでの発行 702
ホスト環境と iSeries 環境 758
データ転送
更新 112
データの検索
位置指定イテレーターの使用、SQLJ 370
イテレーターの複数インスタンスの使用、SQLJ 378
指定されたイテレーターの使用、SQLJ 367
静的 SQL 90
DB2 表から、JDBC 309, 312
DB2 表から、SQLJ 366
DB2 表での複数イテレーターの使用、SQLJ 377
Perl 536
データベース
アクセス
マルチスレッド 193
アプリケーションの接続 38
異なるコンテキストの使用 193
作成
照合順序 658
Perl での接続 536
データベース記述子ブロック (SQLEDBDESC)、照合順序の指定 658
データベース・マネージャー
定義 API、サンプル・プログラム 112
データ・ソース
接続
JDBC 297
データの検索、JDBC 334
データ・ソースについての情報の検索
JDBC 334
データ・タイプ
拡張 UNIX コードに関する考慮事項 683
クラス・データ・メンバー、C/C++ での宣言 177
互換性の問題 97
サポートされる 97
COBOL の規則 215
FORTRAN の規則 233
数値
プラットフォームによる違い 759
説明 31
選択、グラフィック・タイプの 179
データ値の制御 47
変換
DB2 と COBOL 間の 215
DB2 と FORTRAN 間の 233
DB2/REXX 間 549
変換、DB2 と C/C++ 間の 186
ポインター、C/C++ での宣言 176
ホスト言語と DB2 で対応するもの 97
文字変換によるオーバーフロー 683
BINARY
COBOL 218
COBOL 215
C/C++ 186
C/C++ における CLOB 190
C/C++ における FOR BIT DATA 190
C/C++ における VARCHAR 190
C/C++ における変換 186
DATALINK
ホスト変数の制約事項 233
DECIMAL
FORTRAN 233
FOR BIT DATA
COBOL 218
FORTRAN 233
ROWID
DB2 Connect によってサポートされている 759
データ・タイプおよび両方向スクロール・カーソル
制約事項 342, 399
データ・タイプの制約事項
スクロール可能イテレーター 399
スクロール可能な結果セット 342
データ・タイプ・マッピング
表 252
データ・タイプ・マッピング *(続き)*
Java、JDBC、および SQL 403
OLE DB/DB2 間 252
データ・リレーションシップの制御
アプリケーションのロジック 51
参照保全 49
トリガー 50
AFTER トリガー 51
BEFORE トリガー 50
定義振る舞い、DYNAMICRULES 126
出口リスト・ルーチン
使用上の制限 116
テスト環境
パーティション・データベース 728
テスト表の作成 57
テスト・データ
生成 58
テスト・ビューの作成 57
デッドロック
バッファー化挿入でのエラー 720
複数のコンテキストで避ける 196
並行トランザクションでの防止 696
マルチスレッドのアプリケーションでの 196
デバッグ
アプリケーション・プログラム 60
FORTRAN プログラム 222
テリトリー、SQLCA の SQLERRMC フィールド 760
テリトリー・コード
SQLCA の SQLERRMC フィールド 760
トークン
切り捨て、SQLCA 構造体 115
動的 SQL
影響、DYNAMICRULES の 126
カーソル処理 123
カーソルのサンプル・プログラム 124
キャッシュ 88
許可についての考慮事項 54
構文規則 120
考慮事項 121
削除、行の 105
サポートされているステートメント 120
サポートされる SQL ステートメント 751
処理、カーソルの 135
制限 120
静的 SQL との比較 87, 121
宣言、SQLDA の 129
定義 120
任意ステートメントの処理 140
バインディング 75
パフォーマンス 121
パラメーター・マーカー 142

動的 SQL *(続き)*
判別、任意のステートメント・タイプの 140
目的 119
DB2 Connect サポート 757
DB2 Connect ではサポートされていない 769
DESCRIBE ステートメント 120, 128
EXECUTE IMMEDIATE ステートメント 120
EXECUTE ステートメント 120
FETCH ステートメント 128
Perl サポート 535
PREPARE ステートメント 120, 128
特殊タイプ
DB2 Connect によってサポートされている 759
JDBC アプリケーションでの 326
SQLJ アプリケーションでの 389
特殊レジスター
CURRENT EXPLAIN MODE 75
CURRENT PATH 75
CURRENT QUERY OPTIMIZATION 75
トラブルシューティング
オンライン情報 800
チュートリアル 800
トランザクション
コーディング 39
コミット、作業の 40
終了
COMMIT ステートメント 42
CONNECT RESET ステートメント 42
ROLLBACK ステートメント 42
終了、暗黙的に 43
セーブポイント 697
疎結合 711
タイムアウト、MTS および COM+ での 712
データー貫性 39
並行
潜在的な問題 695
デッドロックの防止 696
目的 694
変更の取り消し、ROLLBACK ステートメントによる 41
トランザクション・プロセス・モニター
X/Open XA インターフェース 705
トランザクション・マネージャー
COM+ 709
MTS 709
トランザクション・ログ、バッファー化挿入 717
トリガー
アプリケーションのロジックについての考慮事項 51
概説 26
更新後 51
更新前 50
データ・リレーションシップの制御 50
トレース
CLI/ODBC/JDBC 504
DB2 Universal JDBC ドライバーの例 500

[ナ行]

日本語および中国語 (繁体字) EUC コード・セット
COBOL に関する考慮事項 219
日本語コード・セット
拡張 UNIX コードに関する考慮事項 673
C/C++ の考慮事項 183
FORTRAN 235
REXX 541
UCS2 に関する考慮事項 673
入力および出力ファイル
COBOL 200
FORTRAN 222
入力ファイル、C/C++ 152
入力ファイルの拡張子、C/C++ 152
認証
セキュリティー・プラグイン認証 581
プラグイン
クライアント認証プラグインを初期化する 629
クライアント認証プラグインを初期化する API 629
クライアント認証リソースを終結処理する API 630
認証 ID が存在するかどうか検査する API 646
パスワードを検査する API 637
ユーザー ID/パスワード認証 622
ライブラリーの場所 585
ヌル終了ストリング
CNULREQD BIND オプション 762
ヌル終了文字フォーム C/C++ データ・タイプ 186

[ハ行]

バージョン・レベル
IBM OLE DB Provider for DB2 247
パーティション・データベース環境
エラー処理 728
パーティション・データベース環境 *(続き)*
エラーを戻したパーティションの識別 730
延期状態またはループ状態のアプリケーション 731
重大エラー 729
大量データの抽出 723
テスト環境の作成 728
バッファー化挿入
考慮事項 720
制約事項 722
目的 717
分散サブセクション、指示 716
読み取り専用カーソル 715
ローカル・バイパス 717
OLTP アプリケーションの最適化 715
バインディング
オプション 73
概説 73
考慮事項 80
実行据え置き 81
動的ステートメント 75
バインド・ファイル記述ユーティリティー、db2bfd 81
バインド API
作成、パッケージの 73
実行据え置きバインディング 81
バインド振る舞い、DYNAMICRULES 126
バインド・ファイル
下位互換性 80
サポート、REXX アプリケーションに対する 552
プリコンパイル・オプション 69
REXX 552
派生した列
目的 733
パッケージ
同じ名前を持つバージョン 74
再バインド、作業単位中のカーソルの振る舞い 102
作成 67, 73
作動不能 84
説明 82
属性、プラットフォーム別 761
タイム・スタンプ・エラー 82
バージョン、特権 74
無効
状態 84
REXX アプリケーション・サポート 552
パッケージ名
混合コード・ページ環境 665
バッチ更新
JDBC アプリケーション 337

バッチ更新 *(続き)*
SQLJ アプリケーション 392
バッファー、バッファー化挿入のためのサイズ 717
バッファー化挿入
エラー検出 720
エラー・レポート 720
オープン状態 720
概説 717
行グループ 720
クローズ状態 720
クローズするステートメント 717
考慮事項 720
セーブポイント 704
セーブポイントに関する考慮事項 717
制約事項 722
デッドロック・エラー 720
トランザクション・ログ 717
長いフィールドの制限 722
バッファー・サイズ 717
非同期 720
部分的に満たされた 717
ユニーク・キー違反 720
利点 717
CLP ではサポートされていない 722
INSERT BUF BIND オプション 717
SELECT バッファー化挿入 720
パフォーマンス
影響する要因、静的 SQL 87
順序の制御 742
静的 SQL 88
静的 SQL ステートメントのプリコンパイル 82
動的 SQL 88, 121
パッケージによる最適化 82
バッファー化挿入 717
分散サブセクション、指示 716
読み取り専用カーソル 106, 715
リリース、ロックの 102
FOR UPDATE 文節 105
ID 列 734
パラメーター・マーカー
型付き 142
使用、SQLExecDirect での 145
情報の検索、JDBC 336
動的 SQL での使用 142
任意ステートメントの処理 140
プログラミング例 143
Perl 537
SQLVAR 項目 142
パラメーター・マーカーについての情報の検索
JDBC 336
ハンドル
環境 145
記述子 145
ハンドル *(続き)*
ステートメント 145
接続 145
反復可能読み取り (RR)
メソッド 109
非実行可能 SQL ステートメント
DECLARE CURSOR 38
INCLUDE 38
INCLUDE SQLDA 38
非同期イベント 193
非同期特性、バッファー化挿入の 720
ビュー
システム・カタログ 765
データ値の制御 48
表
カーソルを末尾に置く 111
解決、非修飾名の 79
行の取り出しの例 107
コミット、変更の 40
自己参照 764
初期状態ではログに記録されない表、セーブポイントでの作成 702
生成列 733
宣言済み一時
セーブポイント以外での作成 702
セーブポイントでの作成 702
チェック制約
データ値の制御 48
名前
解決、非修飾の 79
ID 列 734
TEMPORARY
宣言済み 744
標識表
COBOL サポート 214
C/C++ 174
標識変数
切り捨て 95
宣言 95
目的 95
COBOL 209
C/C++ 164
FORTRAN 231
INSERT または UPDATE の実行中 95
NULL 可能列での使用 99
REXX 544
ファイル
C/C++ における参照宣言 170
ファイル参照宣言、REXX における 548
複数の結果セット
ストアード・プロシージャーからの検索 391
不整合
状態 41
データ 41
プラグイン
セキュリティー
名前、命名規則 586
セキュリティー・プラグイン
エラー・メッセージ 605
セキュリティー・プラグインの配置 593, 595, 597
バージョン、バージョン付け 649
配置の制約事項 755
戻りコード 603
呼び出しシーケンス、プラグインが呼び出される順序 606
API 611
GSS-API プラグインの制約事項 648
認証、セキュリティー、グループ検索プラグイン 612, 622, 647
プリコンパイラー
オプション 69
概説 63
出力タイプ 69
セクション番号 769
COBOL 199
C/C++ 3 文字表記 151
C/C++ 言語 178
C/C++ 言語のデバッグ 155
C/C++ 文字セット 151
FORTRAN 221
LANGLEVEL SQL92E オプション 762
プリコンパイル 71
アクセス、 DB2 Connect を介したホストまたは AS/400 アプリケーション・サーバーへの 71
概説 69
更新可能カーソル・オプション 106
整合性トークン 82
タイム・スタンプ 82
動的 SQL ステートメントのサポート 120
複数のサーバーへのアクセス 71
例 69
flagger ユーティリティー 71
FORTRAN 222
プリコンパイル用の flagger ユーティリティー 71
プリプロセッサー機能
SQL プリコンパイラー 171
プログラミングに関する考慮事項
環境 29
疑似コードによるフレームワーク 44
サポートされているインターフェース 6
変数タイプ、データ値の制御 49
ホスト AS/400/iSeries サーバーへのアクセス 694

プログラミングに関する考慮事項 *(続き)*
COBOL 199
C/C++ 151
FORTRAN 221
REXX 539
X/Open XA インターフェース 705
プロパティー
サポートされている OLE DB プロパティー 259
DB2 Universal JDBC ドライバー 408
分散作業単位 688
分散サブセクション (DSS)
指示 716
分散トランザクション
例 525
分離レベル
サポートされているプラットフォーム 763
反復可能読み取り (RR) 109
JDBC アプリケーションの設定 305
SQLJ アプリケーションの設定 362
ページ・レベルのロック
ホスト環境と iSeries 環境 764
並行トランザクション
潜在的な問題 695
デッドロックの防止 696
目的 694
ヘルプ
コマンドの 798
メッセージの 798
変更
DB2 オブジェクト、JDBC 308
DB2 オブジェクト、SQLJ 366
変数
宣言 31
データベース・マネージャーと対話する 31
REXX、事前定義 545
SQL オブジェクトを表す 32
SQLCODE 192, 218, 235
SQLSTATE 192, 218, 235
ホスト環境と iSeries 環境
アプリケーションに関する考慮事項 757
カーソル固定 764
行レベルのロック 764
システム・カタログ 765
スタンドアロンの SQLCODE と SQLSTATE 762
ストアード・プロシージャー 766
データ制御言語 (DCL) 760
データ操作言語 (DML) 759
データ定義言語 (DDL) 758
ページ・レベルのロック 764
割り込み要求の処理 761
C ヌル終了ストリング 762
ホスト環境と iSeries 環境 *(続き)*
DB2 Connect 分離レベル 763
SQLCODE 値と SQLSTATE 値での違い 765
ホスト言語、SQL ステートメントの組み込み 63
ホスト構造体サポート
COBOL 212
C/C++ 172
ホスト式、SQLJ 355, 437
ホスト変数
関連付け、SQL ステートメントとの 35
切り捨て 95
クラス・データ・メンバー、C/C++ における 177
グラフィック
C/C++ 164
FORTRAN 235
固定長文字の構文、COBOL 207
参照
COBOL 205
C/C++ 158
FORTRAN 226
REXX 544
静的 SQL 91
宣言
規則 91
グラフィック、COBOL 209
サンプル・プログラム 112
使用、可変リスト・ステートメントの 141
静的 SQL プログラム 93
データ・タイプへのポインターとしての 176
COBOL 206
C/C++ 160
db2dclgn 宣言生成プログラムでの 34
FORTRAN 228
LOB ロケーター、COBOL 211
選択、グラフィック・データ・タイプの 179
データのブロックの受け渡し 746
定義 91
動的 SQL における 120
ヌル終了ストリング、C/C++ での処理 175
ファイル参照宣言
COBOL 211
C/C++ 170
FORTRAN 232
REXX 548
プリコンパイラーは C/C++ のモジュールに対してグローバルと見なす 159
ホスト変数 *(続き)*
ホスト言語ステートメント中の 91
マルチバイト文字のエンコード 179
命名
COBOL 205
C/C++ 159
FORTRAN 228
REXX 544
目的 158
列に対して使用するように定義する 35
COBOL データ・タイプ 215
C/C++ における初期化 171
DATALINK 制約事項 233
FORTRAN 227
LOB データ宣言
COBOL 210
C/C++ 167
FORTRAN 231
REXX 547
LOB ロケーター宣言
C/C++ 169
FORTRAN 232
REXX 547
LOB、REXX でのクリア 549
Perl ではサポートされていない 536
REXX 543
SQL からの参照 91, 94
SQL ステートメント中の 91
WCHARTYPE プリコンパイラー・オプション 180
ホスト・サーバーのアクセス 694
保留可能な結果セット、JDBC 342

[マ行]

マルチサイト更新
アプリケーションのプリコンパイル 691
概説 688
構成パラメーター 692
マルチサイト更新アプリケーションでの SQL ステートメント 689
目的 688
DB2 Connect サポート 768
マルチバイトの考慮事項
中国語 (繁体字) コード・セット
C/C++ 183
FORTRAN 235
REXX 541
日本語および中国語 (繁体字) EUC コード・セット
COBOL 219
日本語コード・セット
C/C++ 183
FORTRAN 235

マルチバイトの考慮事項 *(続き)*
日本語コード・セット *(続き)*
REXX 541
マルチバイト文字のサポート
コード・ポイント、特殊文字の 664
マルチバイト・コード・ページ
中国語 (繁体字) コード・セット 673
日本語コード・セット 673
メソッド
概説 22
メッセージ・ファイル
定義 69
メッセージ・ヘルプ
呼び出し 798
メモリー
異なるコード・ページに対する割り振り 677
メンバー演算子、C/C++ での制限 178
文字
コード・ページ変換時の置換 668
文字セット
拡張 UNIX コード (EUC) 671
マルチバイト、FORTRAN 235
2 バイト 670
文字比較 655
文字変換
アプリケーションの実行時 666
各国語サポート (NLS) 666
行われる時 667
コーディング、ストアード・プロシージャーの 665, 683
サポートされるコード・ページ 668
ストリング長のオーバーフロー 683
ストリング長のオーバーフロー、データ・タイプを超えた 683
展開 669
プリコンパイルおよびバインド時 666
プログラミングに関する考慮事項 662
SQL ステートメントのコーディング 664
Unicode (UCS2) 685
文字ホスト変数
固定および NULL 終了、C/C++ での 162
C/C++ 可変長 162
C/C++ における可変長 162
C/C++ の固定および NULL 終了 162
FORTRAN 229
モデル、DB2 プログラミングの 44
戻りコード
宣言、SQLCA の 36
SQLCA 構造体 114
問題判別
オンライン情報 800
チュートリアル 800

［ヤ行］

ユーザー ID
2 パーツ・ユーザー ID 587
ユーザー更新可能カタログ統計プロトタイピング・ユーティリティー 45
ユーザー定義関数 (UDF)
アプリケーションのロジックについての考慮事項 51
概説 22
中国語 (繁体字) コード・セット 675
日本語コード・セット 675
ユーザー定義タイプ (UDT)
アプリケーションに関する考慮事項 24
DB2 Connect によってサポートされている 759
ユーザー定義の照合順序 764, 771
ユーティリティー API
組み込みファイル
COBOL アプリケーション 200
FORTRAN アプリケーション 222
組み込みファイル、C/C++ アプリケーションの 152
ユニーク制約
データ値の制御 47
ユニーク・キー違反、バッファー化挿入 720
呼び出し
コマンド・ヘルプ 798
メッセージ・ヘルプ 798
SQL ステートメント・ヘルプ 799
呼び出し振る舞い、DYNAMICRULES 126

［ラ行］

ラージ・オブジェクト (LOB)
DB2 Universal JDBC ドライバー、JDBC 321
DB2 Universal JDBC ドライバー、SQLJ 384
ラージ・オブジェクト (LOB) データ・タイプ
アプリケーションに関する考慮事項 24
データ宣言、C/C++ における 167
ロケーター宣言、C/C++ における 169
DB2 Connect によってサポートされている 759
IBM OLE DB Provider 249
ライブラリー
セキュリティー・プラグイン・ライブラリー 599
制約事項 601
ラッシュされたバッファー化挿入 717
ラッチ、状況、マルチスレッドにおける 193
ランタイム・サービス
マルチスレッド
影響、ラッチに対する 193
リモート作業単位
目的 687
リリース
接続、CMS アプリケーションの 40
リンク
説明 72
ルーチン
共通言語ランタイム・ルーチン
サポートされる SQL データ・タイプ 244
OLE オートメーション
概説 25
例
クラス・データ・メンバー、SQL ステートメントにおける 177
構文、文字ホスト変数、FORTRAN 229
サンプル SQL 宣言セクション、サポートされる SQL データ・タイプの 184
宣言、BLOB の
COBOL 210
FORTRAN 231
宣言、CLOB の
COBOL 210
FORTRAN 231
宣言する、BLOB ファイル参照を
COBOL 211
FORTRAN 232
宣言する、CLOB ファイル・ロケーターを
FORTRAN 232
パラメーター・マーカーの検索と更新における使用 143
BLOB データ宣言 167
BLOB ロケーターの宣言、COBOL 211
CLOB データ宣言 167
CLOB ファイル参照 170
CLOB ロケーター 169
DBCLOB データ宣言 167
DBCLOB の宣言、COBOL 210
Perl プログラム 538
REXX プログラム、SQLEXEC、SQLDBS、および SQLDB2 の登録 540
例外ハンドラー
目的 115
COMMIT と ROLLBACK に関する考慮事項 115

列
サポートされる SQL データ・タイプ 97
生成された 733
派生した 733
ID 734
INCLUDE 列 739
NULL 値の設定 95
NULL 可能データ列での標識変数の使用 99
列タイプ
作成
COBOL 215
C/C++ 186
FORTRAN 233
レポート、エラーの 730
ローカル
バイパス 717
ロールバック
セーブポイントまでの、JDBC 327
セーブポイントまでの、SQLJ 363
トランザクション、JDBC 306
トランザクション、SQLJ 363
ロールバック、変更の 41
ロケール
アプリケーション中の導出 661
DB2 の導出法 662
ロック
行レベル 764
タイムアウト 764
バッファー化挿入エラー 720
ページ・レベル 764
リリース、カーソルの 102

[ワ行]

渡す、スレッド間でコンテキストを 193
割り当て文節、SQLJ 447
割り込み、SIGUSR1 115
割り込みハンドラー
目的 115
COMMIT と ROLLBACK に関する考慮事項 115
割り込みハンドル、SQL ステートメントでの 115

[数字]

2 バイト文字セット (DBCS)
アプリケーション・プログラムの考慮事項 672
コード・ページ 674
コード・ポイント 670
異なるコード・ページ 677
照合に関する考慮事項 675
2 バイト文字セット (DBCS) *(続き)*
中国語 (繁体字) コード・セット 673
中国語 (繁体字) に関する考慮事項 675
日本語コード・セット 673
2 フェーズ・コミット
更新
複数のデータベース 688
3 文字表記、C/C++ 151
64 ビット整数 (BIGINT) データ・タイプ
DB2 Connect によってサポートされている 759

A

ACQUIRE ステートメント
DB2 UDB ではサポートされていない 769
ADO (ActiveX Data Object) 仕様
DB2 でサポートされる 15
DB2 .NET Data Provider 16
ADO アプリケーション
更新可能な両方向スクロール・カーソル 264
ストアード・プロシージャー 264
制限 264
接続ストリング・キーワード 263
接続プール、MTS および COM+ での 713
IBM OLE DB Provider での ADO メソッドおよびプロパティーのサポート 264
API
セキュリティー・プラグイン
API 611, 615, 616, 620, 621, 622, 629, 630, 631, 633, 635, 637, 639, 640, 641, 643, 644, 646
プラグイン API 612, 622
JDBC インプリメンテーションの比較 416
APPC (Advanced Program-to-Program Communication)
処理、割り込みの 115
ARI、SQLERRP フィールドの
DB2 for VSE VM 760
ASCII
混合バイト・データ 759
ソート順序 764
ATOMIC コンパウンド SQL
DB2 Connect サポート 767

B

BatchUpdateException
情報の検索、JDBC 339
BatchUpdateException からの情報の検索 339
BEGIN DECLARE SECTION ステートメント
宣言セクションの作成 31
BIGINT SQL データ・タイプ
COBOL 215
C/C++ における変換 186
DB2 Connect によってサポートされている 759
FORTRAN 233
BIGINT データ・タイプ
静的 SQL における 97
BINARY データ・タイプ、COBOL 218
BIND PACKAGE コマンド
再バインド 84
BIND オプション
EXPLSNAP 80
FUNCPATH 80
QUERYOPT 80
BIND コマンド
作成、パッケージの 73
INSERT BUF オプション 717
blob C/C++ タイプ 186
BLOB FORTRAN データ・タイプ 233
BLOB データ・タイプ
静的 SQL 97
COBOL 215
C/C++ における変換 186
FORTRAN 233
REXX 549
BLOB-FILE COBOL タイプ 215
BLOB-LOCATOR COBOL タイプ 215
blob_file C/C++ タイプ 186
BLOB_FILE FORTRAN データ・タイプ 233
blob_locator C/C++ タイプ 186
BLOB_LOCATOR FORTRAN データ・タイプ 233

C

C
ヌル終了ストリング 762
CALL ステートメント
サポートされているプラットフォーム 766
CALL USING DESCRIPTOR 766
char C/C++ データ・タイプ 186
CHAR データ・タイプ
標識変数 97
COBOL 215
C/C++ における変換 186
FORTRAN 233
REXX 549

CHARACTER*n FORTRAN データ・タイプ 233
CICS SYNCPOINT ROLLBACK コマンド 705
CICS (顧客情報管理システム)
プラットフォームによるアプリケーションの違い 757
CLI (コール・レベル・インターフェース)
組み込み動的 SQL との比較 145
トレース機能 504
トレース・ファイル 511
CLI/ODBC/JDBC
トレース
機能 504
ファイル 511
CLOB FORTRAN データ・タイプ 233
CLOB (文字ラージ・オブジェクト)
データ・タイプ
標識変数 97
COBOL 215
C/C++ 190
FORTRAN 233
REXX 549
C/C++ における変換 186
CLOB-FILE COBOL タイプ 215
CLOB-LOCATOR COBOL タイプ 215
clob_file C/C++ データ・タイプ 186
CLOB_FILE FORTRAN データ・タイプ 233
clob_locator C/C++ データ・タイプ 186
CLOB_LOCATOR FORTRAN データ・タイプ 233
COBOL 言語
オブジェクト指向に関する制約事項 219
規則、標識変数の 209
組み込み SQL ステートメント 63, 203
組み込みファイル 200
固定長文字ホスト変数の構文 207
参照、ホスト変数の 205
数値ホスト変数 206
制約事項 199
宣言、グラフィック・ホスト変数 209
宣言、ホスト変数の 206
中国語 (繁体字) EUC に関する考慮事項 219
データ・タイプ 215
日本語 EUC に関する考慮事項 219
入力および出力ファイル 200
標識表 214
ファイル参照宣言 211
プログラミングに関する考慮事項 199
ホスト構造 212
マルチスレッド・データベース・アクセスをサポートしない 199
COBOL 言語 *(続き)*
命名、ホスト変数の 205
FOR BIT DATA 218
LOB データ宣言 210
LOB ロケーター宣言 211
REDEFINES 214
SQLCODE 変数 218
SQLSTATE 変数 218
COBOL データ・タイプ
BINARY 218
BLOB 215
BLOB-FILE 215
BLOB-LOCATOR 215
CLOB 215
CLOB-FILE 215
CLOB-LOCATOR 215
COMP 218
COMP-1 215
COMP-3 215
COMP-4 218
COMP-5 215
DBCLOB 215
DBCLOB-FILE 215
DBCLOB-LOCATOR 215
PICTURE (PIC) 文節 215
USAGE 文節 215
COLLECTION パラメーター 79
COMMIT WORK RELEASE ステートメント、DB2 Connect ではサポートされていない 769
COMMIT ステートメント
カーソルとの関連 102
終了、トランザクションの 40, 42
COMP データ・タイプ、COBOL での 218
COMP-1 データ・タイプ、COBOL での 215
COMP-3 データ・タイプ、COBOL での 215
COMP-4 データ・タイプ、COBOL での 218
COMP-5 データ・タイプ、COBOL での 215
COM+
接続の再利用 711
疎結合サポート 711
トランザクション処理 711
トランザクション・タイムアウト 712
トランザクション・マネージャー 709
com.ibm.db2.jcc.DB2BaseDataSource
プロパティー 457
メソッド 457
com.ibm.db2.jcc.DB2DatabaseMetaData
メソッド 457
com.ibm.db2.jcc.DB2Diagnosable
メソッド 457
com.ibm.db2.jcc.DB2Driver
メソッド 457
com.ibm.db2.jcc.DB2ExceptionFormatter
メソッド 457
com.ibm.db2.jcc.DB2JccDataSource
メソッド 457
com.ibm.db2.jcc.DB2SimpleDataSource
プロパティー 457
メソッド 457
com.ibm.db2.jcc.DB2Sqlca
メソッド 457
CONNECT RESET ステートメント 42
CONNECT 時の SQLCA.SQLERRD 設定値 677
CONNECT ステートメント
サンプル・プログラム 112
SQLCA.SQLERRD 設定値 677
create database API
SQLEDBDESC 構造 658
CREATE SEQUENCE ステートメント
シーケンス・オブジェクトを作成するための 741
CURRENT EXPLAIN MODE 特殊レジスター
バインドされる動的 SQL への影響 75
CURRENT PATH 特殊レジスター
バインドされる動的 SQL への影響 75
CURRENT QUERY OPTIMIZATION 特殊レジスター
バインドされる動的 SQL への影響 75
CURVAL 式 741
C/C++ アプリケーション
コンパイルとリンク、IBM OLE DB Provider 269
データ・ソースへの接続、IBM OLE DB Provider 269
マルチスレッド・データベース・アクセス 193
C/C++ 言語
組み込み SQL ステートメント 156
組み込みファイル、必要とされる 152
クラス・データ・メンバー 177
グラフィック・ホスト変数 164, 165
サポートされるデータ・タイプ 186
修飾演算子、制限 178
出力ファイル 152
初期化、ホスト変数 171
処理、ヌル終了ストリング 175
数値ホスト変数 160
宣言、グラフィック・ホスト変数 164
中国語 (繁体字) EUC に関する考慮事項 183

C/C++ 言語 *(続き)*
データ・タイプ
関数用 190
サポートされる 186
ストアード・プロシージャー用 190
メソッド用 190
データ・タイプへのポインター 176
デバッグ 155
日本語 EUC に関する考慮事項 183
入力ファイル 152
標識表 174
標識変数 164
ファイル参照宣言 170
プログラミングに関する考慮事項 151
ホスト構造体サポート 172
ホスト変数
宣言 160
命名 159
目的 158
マクロ展開 171
マルチバイト文字のエンコード 179
メンバー演算子、制限 178
文字セット 151
3 文字表記 151
FOR BIT DATA 190
LOB データ宣言 167
LOB ロケーター宣言 169
SQL ステートメントの組み込み 63
SQLCODE 変数 192
sqldbchar データ・タイプ 179
SQLSTATE 変数 192
VARGRAPHIC 構造化フォームのグラフィック宣言、構文 166
WCHARTYPE プリコンパイラー・オプション 180
wchar_t データ・タイプ 179
#include マクロ、制約事項 155
#line マクロ、制約事項 155

D

DataSource インターフェース
SQLJ 359
DataSource オブジェクト、JDBC
作成およびデプロイ 345
DATE データ・タイプ 97
COBOL 215
C/C++ における変換 186
FORTRAN 233
REXX 549
DB2
ロケールの導出 662
DB2 Application Development Client 247
DB2 Connect
分離レベル 763
DB2 Connect *(続き)*
割り込み要求の処理 761
DB2 JDBC Type 2 Driver
セキュリティー 487
データ・ソースへの接続
DriverManager インターフェース 298
SQLException の処理 318
DB2 Universal JDBC ドライバー
暗号化されたユーザー ID または暗号化されたパスワードのセキュリティー 492
拡張クライアント情報 348
クライアント転送サポート 347
ここだけで定義されるメソッド 457
セキュリティー 488
データ・ソースへの接続
DriverManager インターフェース 300
ドライバー・エラーの SQLSTATE 479
ドライバー・エラーのエラー・コード 479
トレースの例 500
トレース・データの収集 497
プロパティー 408
ユーザー ID およびパスワード・セキュリティー 489
ユーザー ID 専用セキュリティー 491
JDBC 問題の診断 497
Kerberos セキュリティー 493
LOB サポート、JDBC 321
LOB サポート、SQLJ 384
ROWID、JDBC 325
ROWID、SQLJ 387
SQLException の処理 315
SQLJ 問題の診断 497
DB2 アプリケーション・プログラミング・インターフェース (API)
概説 8
DB2 インフォメーション・センター 776
更新 787
別の言語での表示 788
呼び出し 786
DB2 コール・レベル・インターフェース (DB2 CLI)
概説 10
動的 SQL との比較 11
DB2 資料
PDF ファイルの印刷 795
DB2 チュートリアル 799
DB2 プログラミング機能 20
DB2 .NET Data Provider 16
DB2ARXCS.BND REXX バインド・ファイル 552
DB2ARXNC.BND REXX バインド・ファイル 552
DB2ARXRR.BND REXX バインド・ファイル 552
DB2ARXRS.BND REXX バインド・ファイル 552
DB2ARXUR.BND REXX バインド・ファイル 552
db2bfd バインド・ファイル記述ユーティリティー 81
DB2CODEPAGE
レジストリー変数 661
db2dclgn 宣言生成プログラム
宣言、ホスト変数の 34
DB2INCLUDE
環境変数 203, 225
コマンド行プロセッサーキャッシュ設定 155
DBCLOB データ・タイプ
静的 SQL プログラムにおける 97
中国語 (繁体字) コード・セット 675
日本語コード・セット 675
COBOL 215
C/C++ における変換 186
REXX 549
DBCLOB-FILE COBOL タイプ 215
DBCLOB-LOCATOR COBOL タイプ 215
dbclob_file C/C++ データ・タイプ 186
dbclob_locator C/C++ データ・タイプ 186
DBCS (2 バイト文字セット)
日本語と中国語 (繁体字) のコード・セット 673
DCL (データ制御言語)
ホスト環境と iSeries 環境 760
DDL (データ定義言語)
動的 SQLのパフォーマンス 121
ホスト環境と iSeries 環境 758
DECIMAL データ・タイプ
静的 SQL における 97
COBOL 215
C/C++ における変換 186
FORTRAN 233
REXX 549
DECLARE CURSOR ステートメント
説明 101
追加、アプリケーションへの 38
DECLARE PROCEDURE ステートメント (OS/400) 766
DECLARE ステートメント
DB2 Connect ではサポートされていない 769
DB2 UDB ではサポートされていない 769
DESCRIBE ステートメント 769

DESCRIBE ステートメント *(続き)*
拡張 UNIX コードに関する考慮事項 681
処理、任意ステートメントの 140
DB2 Connect ではサポートされていない 769
Development Center
概説 22
機能 22
DML (データ操作言語)
動的 SQLのパフォーマンス 121
ホスト環境と iSeries 環境 759
DOUBLE データ・タイプ 97
double データ・タイプ
C および C++ プログラム 186
DriverManager インターフェース
SQLJ 357
DSN、SQLERRP フィールドの
DB2 UDB for OS/390 760
DSS (分散サブセクション)
指示 716
DYNAMICRULES プリコンパイル/ BIND オプション
影響、動的 SQL に対する 126

E

EBCDIC
混合バイト・データ 759
ソート順序 764
EBCDIC 照合順序
サンプル 660
END DECLARE SECTION ステートメント 31
Enterprise Java beans 530
EUC (拡張 UNIX コード)
考慮事項 673
文字セット 671
EXEC SQL INCLUDE SQLCA 194
EXEC SQL INCLUDE ステートメント 155
EXECUTE IMMEDIATE ステートメント
目的 120
EXECUTE ステートメント
目的 120
Explain 機能
プロトタイピング 45
Explain スナップショット
バインド時の 80
EXPLSNAP BIND オプション 80

F

FETCH ステートメント
反復データ・アクセス 109
FETCH ステートメント *(続き)*
ホスト変数 128
SQLDA 構造体 134
FIPS 127-2 標準
SQLSTATE と SQLCODE をホスト変数として宣言する 114
FLOAT データ・タイプ 97
COBOL 215
C/C++ における変換 186
FORTRAN 233
REXX 549
FOR BIT DATA データ・タイプ、C/C++ 190
FOR UPDATE 文節 105
FORCE コマンド
オペレーティング・システムによる違い 760
FORTRAN 言語
拡張の予定はない 29
組み込みファイル 222, 225
検索、組み込みファイルの 225
コメント行 222
条件行 222
数値ホスト変数 229
制約事項 221
中国語 (繁体字) に関する考慮事項 235
データ・タイプ 233
デバッグ 222
日本語に関する考慮事項 235
入力および出力ファイル 222
標識変数 231
ファイル参照宣言 232
プリコンパイル 222
プログラミングに関する考慮事項 221
ホスト変数
参照 226
宣言 228
命名 228
目的 227
マルチスレッド・データベース・アクセスをサポートしない 222
マルチバイト文字セット 235
LOB データ宣言 231
LOB ロケーター宣言 232
SQL ステートメントの組み込み 63, 226
SQL 宣言セクション 233
SQLCODE 変数 235
SQLSTATE 変数 235
FORTRAN データ・タイプ
BLOB 233
BLOB_FILE 233
BLOB_LOCATOR 233
CHARACTER*n 233
CLOB 233
FORTRAN データ・タイプ *(続き)*
CLOB_FILE 233
CLOB_LOCATOR 233
DB2 との変換 233
INTEGER*2 233
INTEGER*4 233
REAL*2 233
REAL*4 233
REAL*8 233
FUNCPATH BIND オプション 80

G

GET ERROR MESSAGE API
エラー・メッセージの検索 116
事前定義された REXX 変数 545
GRAPHIC ストリング
文字変換 669
GRAPHIC スペース 664
GRAPHIC データ・タイプ
選択 179
COBOL 215
C/C++ における変換 186
FORTRAN でサポートされていない 233
REXX 549
GROUP BY 文節
ソート順序 764
GSS-API
GSS-API 認証プラグイン 647
制約事項 647

H

help
表示 786, 788
SQL ステートメントの 799

I

IBM DB2 Development Add-In 4
IBM OLE DB Provider
カーソル 252
コンシューマー 247
サポートされている OLE DB プロパティー 259
サポートされているアプリケーション・タイプ 248
スキーマ行セット 249
スレッド化 249
制約事項 256
データ変換
DB2 タイプから OLE DB タイプへの 255
データ・ソースへの接続 262

IBM OLE DB Provider *(続き)*
プロバイダー 247
ADO アプリケーション 263
ADO アプリケーションに関する制限 264
ADO アプリケーションのカーソル 264
ADO のメソッドとプロパティーに関するサポート 264
C/C++ アプリケーション
データ・ソースへの接続 269
C/C++ アプリケーションのコンパイルとリンク 269
DB2 での MTS サポートの使用可能化 270
for DB2
インストール 247
LOB 249
MTS/COM 分散トランザクション・サポート 269
OLE DB サービスの自動使用可能化 251
OLE DB サポート 257
OLE DB タイプから DB2 タイプへのデータ変換 253
Visual Basic アプリケーションからデータ・ソースへの接続 264
ID 列
シーケンス・オブジェクトとの比較 744
データの検索、JDBC 328
目的 734
INCLUDE SQLCA ステートメント
疑似コード 36
INCLUDE SQLDA ステートメント 38
作成、SQLDA 構造体の 136
INCLUDE ステートメント 38
INCLUDE 文節 739
INSERT BUF BIND オプション
バッファー化挿入 717
INSERT ステートメント
CLP ではサポートされていない 722
VALUES 文節 717
INTEGER データ・タイプ 97
COBOL 215
C/C++ における変換 186
FORTRAN 233
REXX 549
INTEGER*2 FORTRAN データ・タイプ 233
INTEGER*4 FORTRAN データ・タイプ 233
iSeries 環境
ホスト・サーバーへのアクセス 694
ISO
10646 標準 673
ISO *(続き)*
2022 標準 673
ISO/ANS SQL92 標準
サポート 762

J

Java
Enterprise Java beans 530
Java 2 Enterprise Edition
概説 521
サーバー 523
データベース要件 523
SQL ステートメントの組み込み 63
WebSphere Studio の概説 17
Java 2 Enterprise Edition
アプリケーション・サポート 521
概説 521
コンテナー 522
サーバー 523
トランザクション管理 524
要件 523
Enterprise Java beans 530
Java Naming and Directory Interface (JNDI) 524
Java Transaction API (JTA) 524
Java Transaction Service (JTS) 524
Java アプリケーション・サポート 289
Java パッケージへのアクセス
JDBC 296
SQLJ 354
JDBC
更新可能な結果セット 342
サポートされるドライバー 289
スクロール可能な結果セット 342
ステートメント・パラメーターについての情報の検索 336
ストアード・プロシージャー、複数の結果セットの検索 330
ストアード・プロシージャーの呼び出し 313
接続の使用 306
違い、JDBC ドライバー 470, 477
データ・ソースへの接続、DataSource インターフェース 303
データ・ソースへの接続のクローズ 307
データ・タイプ・マッピング 403
特殊タイプの使用 326
トランザクションのコミット 306
トランザクションのロールバック 306
分散トランザクション 525
保留可能な結果セット 342
問題の診断、DB2 Universal JDBC ドライバー 497
JDBC *(続き)*
DataSource オブジェクト
作成およびデプロイ 345
DB2 JDBC Type 2 Driver
エラー処理 318
DB2 Universal JDBC ドライバー
エラー処理 315
DB2 ドライバー・サポートの比較 416
DB2 表からのデータの検索 309, 312
DB2 表のデータの更新 310
Java パッケージへのアクセス 296
ResultSet についての情報の検索 333
SQL 警告の処理 319, 320
JDBC (Java Database Connectivity)
概説 13
Universal JDBC ドライバー
インストール 481
JDBC ResultSet
DB2 Universal JDBC ドライバー 341
JDBC アプリケーション
基本ステップ 293
セーブポイントの処理 327
データ・ソースへの接続 297
バッチ更新 337
分離レベルの設定 305
変数の宣言 296
例 293
DB2 オブジェクトの作成および変更 308
ID 列からのデータの検索 328
SQL の実行 307
JDBC ドライバー・タイプ
定義 289
JNDI (Java Naming and Directory Interface) 524
JTA (Java Transaction API) 524
JTS (Java Transaction service) 524

L

LABEL ON ステートメント、サポートされていない 769
LANGLEVEL プリコンパイル・オプション
MIA 186
SAA1 186
SQL92E と SQLSTATE または SQLCODE 変数 192, 218, 235, 762
LOB サポート
JDBC 仕様を超えた 322
LOB ロケーター 322
LOB (ラージ・オブジェクト) データ・タイプ
アプリケーションに関する考慮事項 24

LOB (ラージ・オブジェクト) データ・タイプ *(続き)*
データ宣言、C/C++ における 167
ロケーター宣言、C/C++ における 169
DB2 Connect によってサポートされている 759
IBM OLE DB Provider 249
LOB 列
互換 Java データ・タイプの選択、JDBC 323
互換性のある Java データ・タイプの選択、SQLJ 385
locktimeout 構成パラメーター 196
long C/C++ データ・タイプ 186
long int C/C++ データ・タイプ 186
long long C/C++ データ・タイプ 186
long long int C/C++ データ・タイプ 186
LONG VARCHAR データ・タイプ
静的 SQL プログラムにおける 97
COBOL 215
C/C++ における変換 186
FORTRAN 233
REXX 549
LONG VARGRAPHIC データ・タイプ
静的 SQL プログラムにおける 97
COBOL 215
C/C++ における変換 186
FORTRAN 233
REXX 549

M

MIA LANGLEVEL プリコンパイル・オプション 186
Microsoft Component Services (COM+)
トランザクション・マネージャー 709
Microsoft OLE DB Provider for ODBC
OLE DB サポート 257
Microsoft Transaction Server (MTS)
トランザクション・タイムアウト 712
トランザクション・マネージャー 709
DB2 でのサポートの使用可能化 270
MTS/COM 分散トランザクション・サポート 269
Microsoft Transaction Server (MTS) 仕様
データのアクセス 15
Microsoft 仕様
データのアクセス 15
ADO (ActiveX Data Object) 15
MTS (Microsoft Transaction Server) 15
RDO (Remote Data Object) 15
Visual Basic 15
Visual C 15
MQSeries
アプリケーション用のサポート 19
MTS (Microsoft Transaction Server) サポート
トランザクション・タイムアウト 712
トランザクション・マネージャー 709
DB2 での使用可能化 270
MTS (Microsoft Transaction Server) 仕様
データのアクセス 15
MTS/COM 分散トランザクション・サポート
トランザクション・マネージャー 709
IBM OLE DB Provider 269

N

Net.Data
概説 19
NEXTVAL 式 741
NLS (各国語サポート)
コード・ページ 666
混合バイト・データ 759
文字変換 666
NOLINEMACRO プリコンパイル・オプション 155
NOT ATOMIC コンパウンド SQL
DB2 Connect サポート 767
NULL 値、SQL
標識変数が受け付ける 95
NULL 終止符、可変長グラフィック・データの処理 186
NUMERIC SQL データ・タイプ
COBOL 215
C/C++ における変換 186
FORTRAN 233
REXX 549

O

ODBC (Open Database Connectivity)
アプリケーション開発ツール 16
接続プール、MTS および COM+ での 713
OLE DB
行セットのサポート 257
コマンドのサポート 257
コンポーネントとインターフェースのサポート 257
サポートされているプロパティー 259
自動的に使用可能にされるサービス 251
セッションのサポート 257
データ変換
DB2 タイプから OLE DB タイプへの 255
OLE DB タイプから DB2 タイプへの 253
OLE DB *(続き)*
ビュー・オブジェクトのサポート 257
表関数
概説 25
BLOB サポート 257
DB2 でサポートされる 15
DB2 とのデータ・タイプ・マッピング 252
IBM OLE DB Provider によるデータ・ソースへの接続 262
OLE DB 表関数 247
OLE オートメーション・ルーチン 25
ORDER BY 文節
ソート順序 764

P

Perl
アプリケーションの例 538
制約事項 535
データの戻り 536
データベースへの接続 536
ドライバー 535
パラメーター・マーカー 537
プログラミングに関する考慮事項 535
マルチスレッド・データベース・アクセスをサポートしない 535
Database Interface (DBI) 仕様 16
SQLCODE 537
SQLSTATE 537
PICTURE (PIC) 文節、COBOL タイプにおける 215
PREP オプション、NOLINEMACRO 155
PREP コマンド (PRECOMPILE)
説明 69
例 69
PREPARE ステートメント
処理、任意ステートメントの 140
目的 120
DB2 Connect ではサポートされていない 769
PreparedStatement メソッド
パラメーター・マーカーを含まない SQL ステートメント 311
PUT ステートメント、DB2 Connect ではサポートされていない 769

Q

QSQ、SQLERRP フィールドの (iSeries) 760
queryopt プリコンパイル/ BIND オプション
コード・ページに関する考慮事項 80

R

RDO (Remote Data Object) 仕様
  DB2 でサポートされる 15
REAL SQL データ・タイプ
  リスト 97
  COBOL 215
  C/C++ における変換 186
  FORTRAN 233
  REXX 549
REAL*2 FORTRAN SQL データ・タイプ 233
REAL*4 FORTRAN SQL データ・タイプ 233
REAL*8 FORTRAN SQL データ・タイプ 233
REDEFINES 文節、COBOL 214
RELEASE SAVEPOINT ステートメント 701
REORGANIZE TABLE コマンド
  混合しているコード・ページ 674
RESULT REXX 事前定義変数 545
ResultSet
  DB2 Universal JDBC ドライバー 341
REXX アプリケーション 551
REXX 言語
  アプリケーションの実行 551
  カーソル 549
  カーソル ID 541
  事前定義変数 545
  ストアード・プロシージャー
    概説 554
    サーバーの考慮事項 556
    呼び出し 555
  ストアード・プロシージャーの呼び出し 556
  制約事項 540
  中国語 (繁体字) 541
  データ・タイプ 549
  日本語 541
  バインド・ファイル 552
  標識変数 544
  プログラミングに関する考慮事項 539, 540
  変数の初期化 554
  ホスト変数
    参照 544
    命名 544
    目的 543
  マルチスレッド・データベース・アクセスをサポートしない 541
  ルーチンの登録 540
  API
    SQLDB2 539
    SQLDBS 539
    SQLEXEC 539
REXX 言語 (続き)
  API 構文 553
  DB2 CLP の呼び出し 553
  LOB データ 547
  LOB ファイル参照宣言 548
  LOB ホスト変数、クリア 549
  LOB ロケーター宣言 547
  SQL ステートメント 541
  SQL ステートメントの組み込み 541
  SQLDA 10 進フィールド
    データの検索 556
  SQLEXEC、SQLDBS、および SQLDB2 の登録 540
REXX データ・タイプ 549
ROLLBACK TO SAVEPOINT ステートメント
  カーソルの振る舞い 702
ROLLBACK WORK RELEASE ステートメント
  DB2 Connect ではサポートされていない 769
ROLLBACK ステートメント
  カーソルとの関連 102
  終了、トランザクションの 42
  取り消し、変更の 41
  プラットフォームによる違い 760
  ロールバック、変更の 41
ROWID
  DB2 Universal JDBC ドライバー 325, 387
ROWID データ・タイプ
  DB2 Connect によってサポートされている 759
RUOW
  「リモート作業単位」を参照 687

S

SAA1 LANGLEVEL プリコンパイル・オプション 186
SAVEPOINT ステートメント
  トランザクションの制御 701
SELECT ステートメント
  可変リスト 141
  記述、SQLDA の割り振り後 134
  検索されたデータの更新 112
  宣言、SQLDA の 129
  データ変更ステートメントからの選択 735
  取り出し
    データの 2 度目の 109
    複数行 100
  バッファー化挿入 720
  DECLARE CURSOR ステートメント 101
SELECT ステートメント (続き)
  EXECUTE ステートメントとの関連 120
SET CURRENT PACKAGESET ステートメント 79
SET CURRENT ステートメント、DB2 Connect ではサポートされていない 769
SET-TRANSACTION 文節、SQLJ 446
short C/C++ データ・タイプ 186
short int C/C++ データ・タイプ 186
SIGUSR1 割り込み 115
SMALLINT データ・タイプ
  COBOL 215
  CREATE TABLE ステートメント 97
  C/C++ における変換 186
  FORTRAN 233
  REXX 549
SQL オブジェクト
  変数を使用して表す 32
SQL 組み込みファイル
  COBOL アプリケーション 200
  C/C++ アプリケーション 152
  FORTRAN アプリケーション 222
SQL 警告
  JDBC での処理 319, 320
  SQLJ での処理 380
SQL コードのプロトタイプ化 45
SQL (構造化照会言語)
  許可
    組み込み SQL 53
    静的 SQL 55
    動的 SQL 54
    API 55
  動的に作成された 145
SQL ステートメント
  シグナル・ハンドラー 115
  実行の直列化 193
  保管、エンド・ユーザー要求の 142
  マルチサイト更新アプリケーション 689
  例外ハンドラー 115
  割り込みハンドラー 115
  COBOL 構文 203
  CONNECT
    SQLCA.SQLERRD 設定値 677
  C/C++ 構文 156
  FORTRAN 構文 226
  REXX 541
  REXX 構文 541
SQL ステートメント・ヘルプ
  呼び出し 799
SQL データ・タイプ
  BIGINT 97
  BLOB 97
  CHAR 97

SQL データ・タイプ *(続き)*
CLOB 97
COBOL 215
C/C++ における変換 186
DATE 97
DBCLOB 97
DECIMAL 97
FLOAT 97
FORTRAN 233
INTEGER 97
LONG VARCHAR 97
LONG VARGRAPHIC 97
REAL 97
REXX 549
SMALLINT 97
TIME 97
TIMESTAMP 97
VARCHAR 97
VARGRAPHIC 97
SQL プロシージャー型言語 751
SQL 連絡域 (SQLCA) 36
SQL1252A 組み込みファイル
COBOL アプリケーション 200
FORTRAN アプリケーション 222
SQL1252B 組み込みファイル
COBOL アプリケーション 200
FORTRAN アプリケーション 222
SQL92 標準
サポート 762
SQLADEF 組み込みファイル
C/C++ アプリケーション 152
SQLAPREP 組み込みファイル
COBOL アプリケーション 200
C/C++ アプリケーション 152
FORTRAN アプリケーション 222
SQLCA 組み込みファイル
COBOL アプリケーション 200
C/C++ アプリケーション 152
FORTRAN アプリケーション 222
SQLCA (構造化照会言語連絡域)
エラー発生時の不完全な挿入 720
バッファー化挿入におけるエラー報告 720
マルチスレッドに関する考慮事項 194
SQLERRMC フィールド 760, 767
SQLERRP フィールドによる RDBMS の識別 760
SQLCA 構造体
概説 114
組み込みファイル
COBOL アプリケーション 200
FORTRAN アプリケーション 222
組み込みファイル、C/C++ の 152
警告 95
多重定義の回避 37
定義、サンプル・プログラム 112
SQLCA 構造体 *(続き)*
トークンの切り捨て 115
パーティション・データベース環境
マージされた複数の SQLCA 構造 730
マージされた複数の構造 730
要件 114
レポート、エラーの 730
SQLCODE フィールド 114
sqlerrd 730
SQLSTATE フィールド 114
SQLWARN1 フィールド 95
SQLCA 事前定義変数 545
SQLCA の SQLERRMC フィールド 669, 760, 767
SQLCA_92 組み込みファイル
COBOL アプリケーション 200
FORTRAN アプリケーション 222
SQLCA_92 構造 222
SQLCA_CN 組み込みファイル 222
SQLCA_CS 組み込みファイル 222
SQLCHAR 構造
データの受け渡し 139
SQLCLI 組み込みファイル 152
SQLCLI1 組み込みファイル 152
SQLCODE
エラー・コード 36
組み込み SQLCA 36
構造 114
スタンドアロン 762
フィールド、SQLCA 構造体の 114
プラットフォームによる違い 765
レポート、エラーの 730
SQLCODE -1015
パーティション・データベース環境 729
SQLCODE -1034
パーティション・データベース環境 729
SQLCODE -30081
パーティション・データベース環境 729
SQLCODES 組み込みファイル
COBOL アプリケーション 200
C/C++ アプリケーション 152
FORTRAN アプリケーション 222
SQLDA
データの検索
REXX アプリケーション・プログラム 556
SQLDA (SQL 記述子域)
マルチスレッドに関する考慮事項 194
SQLDA 組み込みファイル
COBOL アプリケーション 200
C/C++ アプリケーション 152
FORTRAN アプリケーション 222
SQLDA 構造体
作成 136
十分な SQLVAR エンティティーの宣言 133
準備、最小構造体を用いたステートメント 131
準備済みステートメントに関する情報を配置する 120
宣言 129
データの受け渡し 139
データのブロックの受け渡し 746
判別、任意のステートメント・タイプの 140
PREPARE ステートメントとの関連 120
SQLDACT 組み込みファイル 222
SQLDB2 REXX API 539, 553
SQLDB2 ルーチン、REXX の場合の登録 540
sqldbchar データ・タイプ
選択 179
対応する列タイプ 186
SQLDBS REXX API 539
SQLDBS ルーチン、REXX の場合の登録 540
SQLE819A 組み込みファイル
COBOL アプリケーション 200
C/C++ アプリケーション 152
FORTRAN アプリケーション 222
SQLE819B 組み込みファイル
COBOL アプリケーション 200
C/C++ アプリケーション 152
FORTRAN アプリケーション 222
SQLE850A 組み込みファイル
COBOL アプリケーション 200
FORTRAN アプリケーション 222
SQLE850B 組み込みファイル
COBOL アプリケーション 200
FORTRAN アプリケーション 222
SQLE859A 組み込みファイル
C/C++ アプリケーション 152
SQLE859B 組み込みファイル
C/C++ アプリケーション 152
SQLE932A 組み込みファイル
COBOL アプリケーション 200
C/C++ アプリケーション 152
FORTRAN アプリケーション 222
SQLE932B 組み込みファイル
COBOL アプリケーション 200
C/C++ アプリケーション 152
FORTRAN アプリケーション 222
sqleAttachToCtx API 193
SQLEAU 組み込みファイル
COBOL アプリケーション 200
C/C++ アプリケーション 152
FORTRAN アプリケーション 222

sqleBeginCtx API 193
sqleDetachFromCtx API 193
sqleEndCtx API 193
sqleGetCurrentCtx API 193
sqleInterruptCtx API 193
SQLENV 組み込みファイル
COBOL アプリケーション 200
C/C++ アプリケーション 152
FORTRAN アプリケーション 222
SQLERRD(1) 669, 677
SQLERRD(2) 669, 677
SQLERRD(3) 705
SQLERRP フィールド、SQLCA の 760
sqleSetTypeCtx API 193
SQLETSD 組み込みファイル 200
SQLException
処理 116
SQLEXEC REXX API
組み込み SQL 539
登録 540
SQL ステートメントの処理 541
SQLEXT 組み込みファイル
CLI アプリケーション 152
sqlint64 C/C++ データ・タイプ 186
SQLISL 事前定義変数 545
SQLJ
位置指定イテレーター、変数として渡される 396
イテレーターの複数インスタンス 378
実行コンテキスト 390
ストアード・プロシージャーの呼び出し 379
データ・ソースへの接続 357
データ・ソースへの接続のクローズ 364
デフォルト接続の使用 362
特殊タイプの使用 389
トランザクションのコミット 363
トランザクションのロールバック 363
表の複数のイテレーター 377
ホスト式 355
問題の診断、DB2 Universal JDBC ドライバー 497
DataSource インターフェースの使用 359
DriverManager インターフェースの使用 357
Java パッケージへのアクセス 354
SQL 警告の処理 380
SQLJ SET-TRANSACTION 文節 446
SQLJ with 文節 439
SQLJ アプリケーション
位置指定更新 373
位置指定削除 373
エラーの処理 380
基本ステップ 351
SQLJ アプリケーション *(続き)*
コメント 356
指定されたイテレーターの使用 367
スクロール可能イテレーターの使用 399
セーブポイントの処理 363
制御ステートメント実行 390
バッチ更新 392
分離レベルの設定 362
変数の宣言 355
例 351
DB2 オブジェクトの作成および変更 366
DB2 表からのデータの検索 366
SQL の実行 365
SQLJ アプリケーションで指定されたイテレーター 367
SQLJ イテレーター変換文節 448
SQLJ インプリメント文節 438
SQLJ コンテキスト文節 444
SQLJ 実行可能文節 443
SQLJ ステートメント文節 445
SQLJ 接続宣言文節 441, 442
SQLJ 代入文節 447
SQLJ 文節 437
SQLJ 変数名
制約事項 356
SQLJ ホスト式 437
SQLJACB 組み込みファイル
C/C++ アプリケーション 152
sqlj.runtime.ConnectionContext
メソッド 449
sqlj.runtime.ExecutionContext
メソッド 449
sqlj.runtime.ForUpdate
メソッド 449
sqlj.runtime.NamedIterator
メソッド 449
sqlj.runtime.PositionedIterator
メソッド 449
sqlj.runtime.ResultSetIterator
メソッド 449
sqlj.runtime.Scrollable
メソッド 449
SQLMON 組み込みファイル
COBOL アプリケーション 200
C/C++ アプリケーションの 152
FORTRAN アプリケーション 222
SQLMONCT 組み込みファイル 200
SQLMSG 事前定義変数 545
SQLRDAT 事前定義変数 545
SQLRIDA 事前定義変数 545
SQLRODA 事前定義変数 545
SQLSTATE
スタンドアロン 762
違い 765
SQLSTATE *(続き)*
CLI での 145
DB2 Universal JDBC ドライバーによって発行されたコード 479
SQLSTATE 組み込みファイル
COBOL アプリケーション 200
C/C++ アプリケーション 152
FORTRAN アプリケーション 222
SQLSTATE フィールド 114
SQLSYSTM 組み込みファイル 152
SQLUDF 組み込みファイル
C/C++ アプリケーション 152
SQLUTBCQ 組み込みファイル 200
SQLUTBSQ 組み込みファイル 200
SQLUTIL 組み込みファイル
COBOL アプリケーション 200
C/C++ アプリケーション 152
FORTRAN アプリケーション 222
SQLUV 組み込みファイル
C/C++ アプリケーション 152
SQLUVEND 組み込みファイル 152
SQLVAR エンティティー
宣言、十分な数の 133
不定数を宣言する 129
SQLWARN 構造体 114
SQLXA 組み込みファイル
C/C++ アプリケーション 152
SQL_WCHART_CONVERT プリプロセッサー・マクロ 180
SYSIBM.SYSPROCEDURES カタログ (OS/390) 766
SYSIBM.SYSROUTINES カタログ (VM/VSE) 766

T

TIME データ・タイプ
COBOL 215
CREATE TABLE ステートメントにおける 97
C/C++ における変換 186
FORTRAN 233
REXX 549
TIMESTAMP データ・タイプ
説明 97
COBOL 215
C/C++ における変換 186
FORTRAN 233
REXX 549

U

Unicode (UCS-2)
中国語 (繁体字) コード・セット 673
日本語コード・セット 673

Unicode (UCS-2) *(続き)*
文字変換 685
文字変換によるオーバーフロー 683
UDF (ユーザー定義関数) に関する考慮事項 675
Universal JDBC ドライバー
インストール 481
USAGE 文節、COBOL タイプにおける 215

V

VARCHAR データ・タイプ
構造化書式、C/C++ 186
表列における 97
C または C++ 190
COBOL 215
C/C++ における変換 186
FORTRAN 233
REXX 549
VARGRAPHIC 構造化フォームのグラフィック宣言
C/C++ の構文 166
VARGRAPHIC データ・タイプ
リスト 97
COBOL 215
C/C++ における変換 186
FORTRAN 233
REXX 549
Visual Basic
アプリケーション、データ・ソースへの接続 264
カーソルに関する考慮事項 264
データ制御サポート 264
DB2 でサポートされる 15
Visual C
DB2 でサポートされる 15

W

WCHARTYPE プリコンパイラー・オプション
指針 180
データ、NOCONVERT オプションで使用可能な 186
wchar_t データ・タイプ
選択 179
Web アプリケーション
構築用のツール 17
WebSphere
エンタープライズ・データへのアクセス 575
ステートメント・キャッシュ 577
接続プール
目的 575
WebSphere *(続き)*
接続プール *(続き)*
利点 576
データ・ソース 575
WebSphere Studio 17
WHENEVER ステートメント
エラー処理 37
Windows
コード・ページ 661
DB2CODEPAGE レジストリー変数 661
with 文節、SQLJ 439

X

XA インターフェース
アプリケーション・リンケージ 709
カーソル、WITH HOLD と宣言された 705
セーブポイント 705
単一スレッドのアプリケーション 705
トランザクション 705
トランザクション処理の特性 705
マルチスレッド・アプリケーション 705
目的 705
API 制限 705
CICS 環境 705
COMMIT ステートメント 705
DISCONNECT 705
RELEASE はサポートされていない 705
ROLLBACK ステートメント 705
SQL CONNECT 705
XA 環境 705
XASerialize 705
XML Extender
概説 18
XML (Extensible Markup Language)
説明 18

[特殊文字]

#ifdefs
C/C++ 制約事項 171
#include マクロ
C/C++ 制約事項 155
#line マクロ
C/C++ 制約事項 155

IBM と連絡をとる

技術上の問題がある場合は、お客様サポートにご連絡ください。

製品情報

DB2 Universal Database 製品に関する情報は、
http://www.ibm.com/software/data/db2/udb から入手できます。

このサイトには、技術ライブラリー、資料の注文方法、製品のダウンロード、ニュースグループ、フィックスパック、ニュース、および Web リソースへのリンクに関する最新情報が掲載されています。

米国以外の国で IBM に連絡する方法については、 IBM Worldwide ページ (www.ibm.com/planetwide) にアクセスしてください。

Printed in Japan

SC88-9138-01

日本アイ・ビー・エム株式会社
〒106-8711　東京都港区六本木3-2-12