IBM® DB2 Universal Database™

管理 API リファレンス

バージョン 8.2

SC88-9136-01
(英文原典：SC09-4824-01)

IBM® DB2 Universal Database™

管理 API リファレンス

バージョン 8.2

SC88-9136-01
(英文原典：SC09-4824-01)

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： SC09–4824–01
IBM® DB2 Universal Database™
Administrative API Reference
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

本書では、アプリケーション・プログラミング・インターフェース (API) を使用して、データベース管理機能を実行する方法を示しています。以下に挙げるプログラミング言語で作成されたアプリケーションにおける、データベース・マネージャー API 呼び出しの使用法について詳しく説明しています。

- C
- C++
- COBOL
- FORTRAN
- REXX

コンパイルする必要のある言語の場合、ステートメントを処理するため、適切なプリコンパイラーが前もって使えるようになっていなければなりません。サポートされているどの言語にも、プリコンパイラーが備えられています。

本書の対象読者

本書の読者は、データベースの管理やアプリケーション・プログラミングについて理解していることに加えて、以下の知識を有していることを前提としています。

- 構造化照会言語 (SQL)
- C、C++、COBOL、FORTRAN、または REXX プログラミング言語
- アプリケーション・プログラム設計

第 1 章 アプリケーション・プログラミング・インターフェース

このセクションでは、 DB2 アプリケーション・プログラミング・インターフェースについてアルファベット順で説明しています。アプリケーション・プログラム内からの管理機能のほとんどは、 API によって可能になります。

注: ディレクトリー・パスのスラッシュ (/) は、UNIX ベースのシステムに固有のものです。これは、Windows オペレーティング・システムのディレクトリー・パスで使われる円記号 (¥) と同等のものです。

DB2 API

以下の表では、DB2 API を DB2 のサンプル付きで示しています。最初の表は、DB2 API をリストしています。これは、機能カテゴリー、それぞれの組み込みファイル、およびそれを例示するサンプル・プログラムごとにグループ分けしています (組み込みファイルの詳細については、表の後にある注を参照してください)。 2 番目の表は C/C++ サンプル・プログラムをリストしています。ここでは、各 C/C++ プログラムで例示されている DB2 API を示します。 3 番目の表は COBOL サンプル・プログラムおよび 各 COBOL プログラムで例示されている DB2 API を示します。

DB2 API、組み込みファイル、およびサンプル・プログラム
表 1.

DB2 API を使用した C/C++ サンプル・プログラム
8 ページの表 2.

DB2 API を使用した COBOL サンプル・プログラム
11 ページの表 3.

表 1. DB2 API、組み込みファイル、およびサンプル・プログラム

DB2 API	組み込みファイル	サンプル・プログラム
データベース・マネージャーの制御		
db2DatabaseQuiesce - データベースの静止	db2ApiDf	n/a
db2DatabaseUnquiesce - データベースの静止解除	db2ApiDf	n/a
db2InstanceStart - インスタンスの開始	db2ApiDf	C: instart.c C++: instart.C
db2InstanceStop - インスタンスの停止	db2ApiDf	C: instart.c C++: instart.C
db2InstanceQuiesce - インスタンスの静止	db2ApiDf	n/a
db2InstanceUnquiesce - インスタンスの静止解除	db2ApiDf	n/a
sqlesdeg - ランタイム次数の設定	sqlenv	C: ininfo.c C++: ininfo.C
データベースの制御		
db2DatabaseRestart - データベースの再始動	db2ApiDf	C: dbconn.sqc C++: dbconn.sqC

表 1. DB2 API、組み込みファイル、およびサンプル・プログラム (続き)

DB2 API	組み込みファイル	サンプル・プログラム
sqlecrea - データベースの作成	sqlenv	C: dbcreate.c dbrecov.sqc dbsample.sqc C++: dbcreate.C dbrecov.sqC COBOL: db_udcs.cbl dbconf.cbl ebcdicdb.cbl
sqlecran - ノードでのデータベースの作成	sqlenv	n/a
sqledrpd - データベースのドロップ	sqlenv	C: dbcreate.c C++: dbcreate.C COBOL: dbconf.cbl
sqledpan - ノードでのデータベースのドロップ	sqlenv	n/a
sqlemgdb - データベースの移行	sqlenv	C: dbmigrat.c C++: dbmigrat.C COBOL: migrate.cbl
db2XaListIndTrans - 未確定トランザクションのリスト	db2ApiDf	n/a
sqle_activate_db - データベースの活動化	sqlenv	n/a
sqle_deactivate_db - データベースの非活動化	sqlenv	n/a
sqlcspqy - DRDA 未確定トランザクションのリスト	sqlxa	n/a
データベース・マネージャーおよびデータベース構成		
db2CfgGet - 構成パラメーターの入手	db2ApiDf	C: dbinfo.c dbrecov.sqc inauth.sqc ininfo.c tscreate.sqc C++: dbinfo.C dbrecov.sqC inauth.sqC ininfo.C tscreate.sqC
db2CfgSet - 構成パラメーターの設定	db2ApiDf	C: dbinfo.c dbrecov.sqc ininfo.c C++: dbinfo.C dbrecov.sqC ininfo.C
データベース・ディレクトリーの管理		
sqlecadb - データベースのカタログ	sqlenv	C: ininfo.c C++: ininfo.C COBOL: dbcat.cbl
sqleuncd - データベースのアンカタログ	sqlenv	C: ininfo.c C++: ininfo.C COBOL: dbcat.cbl
sqlegdad - DCS データベースのカタログ	sqlenv	C: ininfo.c C++: ininfo.C COBOL: dcscat.cbl
sqlegdel - DCS データベースのアンカタログ	sqlenv	C: ininfo.c C++: ininfo.C COBOL: dcscat.cbl
sqledcgd - データベース・コメントの変更	sqlenv	C: ininfo.c C++: ininfo.C COBOL: dbcmt.cbl
db2DbDirOpenScan - データベース・ディレクトリー・スキャンのオープン	db2ApiDf	C: ininfo.c C++: ininfo.C COBOL: dbcat.cbl dbcmt.cbl
db2DbDirGetNextEntry - データベース・ディレクトリーの次項目の入手	db2ApiDf	C: ininfo.c C++: ininfo.C COBOL: dbcat.cbl dbcmt.cbl
db2DbDirCloseScan - データベース・ディレクトリー・スキャンのクローズ	db2ApiDf	C: ininfo.c C++: ininfo.C COBOL: dbcat.cbl dbcmt.cbl
sqlegdsc - DCS ディレクトリー・スキャンのオープン	sqlenv	C: ininfo.c C++: ininfo.C COBOL: dcscat.cbl
sqlegdgt - DCS ディレクトリー項目の入手	sqlenv	C: ininfo.c C++: ininfo.C COBOL: dcscat.cbl
sqlegdcl - DCS ディレクトリー・スキャンのクローズ	sqlenv	C: ininfo.c C++: ininfo.C COBOL: dcscat.cbl

表 1. DB2 API、組み込みファイル、およびサンプル・プログラム (続き)

DB2 API	組み込みファイル	サンプル・プログラム
sqlegdge - データベース用 DCS ディレクトリー項目の入手	sqlenv	C: ininfo.c C++: ininfo.C COBOL: dcscat.cbl
クライアント/サーバーのディレクトリーの管理		
sqlectnd - ノードのカタログ	sqlenv	C: ininfo.c C++: ininfo.C COBOL: nodecat.cbl
sqleuncn - ノードのアンカタログ	sqlenv	C: ininfo.c C++: ininfo.C COBOL: nodecat.cbl
sqlenops - ノード・ディレクトリー・スキャンのオープン	sqlenv	C: ininfo.c C++: ininfo.C COBOL: nodecat.cbl
sqlengne - ノード・ディレクトリー次項目の入手	sqlenv	C: ininfo.c C++: ininfo.C COBOL: nodecat.cbl
sqlencls - ノード・ディレクトリー・スキャンのクローズ	sqlenv	C: ininfo.c C++: ininfo.C COBOL: nodecat.cbl
ネットワーク・サポート		
sqleregs - 登録	sqlenv	n/a
sqledreg - 登録解除	sqlenv	n/a
db2LdapRegister - LDAP 登録サーバー	db2ApiDf	n/a
db2LdapUpdate - LDAP 更新サーバー	db2ApiDf	n/a
db2LdapDeregister - LDAP 登録解除サーバー	db2ApiDf	n/a
db2LdapCatalogNode - ノード LDAP 項目のカタログ	db2ApiDf	n/a
db2LdapUncatalogNode - ノード LDAP 項目のアンカタログ	db2ApiDf	n/a
db2LdapCatalogDatabase - データベース LDAP 項目のカタログ	db2ApiDf	n/a
db2LdapUncatalogDatabase - データベース LDAP 項目のアンカタログ	db2ApiDf	n/a
リカバリー		
db2Backup - データベースのバックアップ	db2ApiDf	C: dbrecov.sqc C++: dbrecov.sqC
sqlurcon - 調整	sqlutil	n/a
db2Restore - データベースのリストア	db2ApiDf	C: dbrecov.sqc C++: dbrecov.sqC
db2Rollforward - データベースのロールフォワード	db2ApiDf	C: dbrecov.sqc C++: dbrecov.sqC
db2HistoryOpenScan - 履歴ファイルのスキャンのオープン	db2ApiDf	C: dbrecov.sqc C++: dbrecov.sqC
db2HistoryGetEntry - 履歴ファイルの次項目の入手	db2ApiDf	C: dbrecov.sqc C++: dbrecov.sqC
db2HistoryCloseScan - 履歴ファイルのスキャンのクローズ	db2ApiDf	C: dbrecov.sqc C++: dbrecov.sqC
db2Prune - 履歴ファイルの整理	db2ApiDf	C: dbrecov.sqc C++: dbrecov.sqC
db2HistoryUpdate - 履歴ファイルの更新	db2ApiDf	C: dbrecov.sqc C++: dbrecov.sqC
高可用性		

表 1. DB2 API、組み込みファイル、およびサンプル・プログラム (続き)

DB2 API	組み込みファイル	サンプル・プログラム
db2HADRStart - HADR の開始	db2ApiDf	n/a
db2HADRStop - HADR の停止	db2ApiDf	n/a
db2HADRTakeover - 1 次データベースとしてのテークオーバー	db2ApiDf	n/a
操作ユーティリティー		
sqlefrce - アプリケーションの強制終了	sqlenv	C: dbconn.sqc dbsample.sqc instart.c C++: dbconn.sqC instart.C COBOL: dbstop.cbl
db2Reorg - 再編成	db2ApiDf	C: tbreorg.sqc C++: tbreorg.sqC COBOL: dbstat.sqb
db2Runstats - 統計の実行	db2ApiDf	C: tbreorg.sqc C++: tbreorg.sqC COBOL: dbstat.sqb
データベース・モニター		
db2GetSnapshotSize - db2GetSnapshot 出力バッファーに必要なサイズの見積もり	db2ApiDf	n/a
db2MonitorSwitches - モニター・スイッチの入手 / 更新	db2ApiDf	C: utilsnap.c C++: utilsnap.C
db2GetSnapshot - スナップショットの入手	db2ApiDf	C: utilsnap.c C++: utilsnap.C
db2ResetMonitor - モニターのリセット	db2ApiDf	n/a
db2ConvMonStream - モニター・ストリームの変換	db2ApiDf	n/a
ヘルス・モニター		
db2AddContact - 連絡先の追加	db2ApiDf	n/a
db2AddContactGroup - 連絡先グループの追加	db2ApiDf	n/a
db2DropContact - 連絡先のドロップ	db2ApiDf	n/a
db2DropContactGroup - 連絡先グループのドロップ	db2ApiDf	n/a
db2GetAlertCfg - アラート構成の入手	db2ApiDf	n/a
db2GetContactGroup - 連絡先グループの入手	db2ApiDf	n/a
db2GetContactGroups - 連絡先グループの入手	db2ApiDf	n/a
db2GetContacts - 連絡先の入手	db2ApiDf	n/a
db2GetHealthNotificationList - ヘルス通知リストの入手	db2ApiDf	n/a
db2ResetAlertCfg - アラート構成のリセット	db2ApiDf	n/a
db2UpdateAlertCfg - アラート構成の更新	db2ApiDf	n/a
db2UpdateContact - 連絡先の更新	db2ApiDf	n/a
db2UpdateContactGroup - 連絡先グループの更新	db2ApiDf	n/a
db2UpdateHealthNotificationList - ヘルス通知リストの更新	db2ApiDf	n/a

表 1. DB2 API、組み込みファイル、およびサンプル・プログラム (続き)

DB2 API	組み込みファイル	サンプル・プログラム
データ・ユーティリティー		
db2Export - エクスポート	sqlutil	C: tbmove.sqc C++: tbmove.sqC COBOL: expsamp.sqb impexp.sqb tload.sqb
db2Import - インポート	db2ApiDf	C: dtformat.sqc tbmove.sqc C++: tbmove.sqC COBOL: expsamp.sqb impexp.sqb
db2Load - ロード	db2ApiDf	C: dtformat.sqc tbmove.sqc C++: tbmove.sqC
db2LoadQuery - ロードの照会	db2ApiDf	C: tbmove.sqc C++: tbmove.sqC COBOL: loadqry.sqb
一般的なアプリケーション・プログラミング		
db2AutoConfig - 自動構成	db2AuCfg	C: dbcfg.sqc C++: dbcfg.sqC
db2AutoConfigFreeMemory - 自動構成メモリーの解放	db2AuCfg	C: dbcfg.sqc C++: dbcfg.sqC
sqlaintp - エラー・メッセージの入手	sql	C: dbcfg.sqc utilapi.c C++: dbcfg.sqC utilapi.C COBOL: checkerr.cbl
sqlogstt - SQLSTATE メッセージの入手	sql	C: utilapi.c C++: utilapi.C COBOL: checkerr.cbl
sqleisig - シグナル・ハンドラーのインストール	sqlenv	COBOL: dbcmt.cbl
sqleintr - 割り込み	sqlenv	n/a
sqlgdref - アドレスの参照解除	sqlutil	n/a
sqlgmcpy - メモリーのコピー	sqlutil	n/a
sqlefmem - メモリーの解放	sqlenv	C: dbrecov.sqc tsinfo.sqc C++: dbrecov.sqC tsinfo.sqC COBOL: tabscont.sqb tabspace.sqb tspace.sqb
sqlgaddr - アドレスの入手	sqlutil	n/a
アプリケーションの準備		
sqlaprep - プログラムのプリコンパイル	sql	C: dbpkg.sqc C++: dbpkg.sqC
sqlabndx - バインド	sql	C: dbpkg.sqc dbsample.sqc C++: dbpkg.sqC
sqlarbnd - 再バインド	sql	C: dbpkg.sqc dbsample.sqc C++: dbpkg.sqC COBOL: rebind.sqb
リモート・サーバー・ユーティリティー		
sqleatin - アタッチ	sqlenv	C: inattach.c utilapi.c C++: inattach.C utilapi.C COBOL: dbinst.cbl
sqleatcp - パスワードのアタッチと変更	sqlenv	C: inattach.c C++: inattach.C COBOL: dbinst.cbl
sqledtin - 切り離し	sqlenv	C: inattach.c utilapi.c C++: inattach.C utilapi.C COBOL: dbinst.cbl
テーブル・スペースの管理		
sqlbtcq - 表スペース・コンテナー照会	sqlutil	C: dbrecov.sqc tsinfo.sqc C++: dbrecov.sqC tsinfo.sqC COBOL: tabscont.sqb tspace.sqb
sqlbotcq - 表スペース・コンテナー照会のオープン	sqlutil	C: tsinfo.sqc C++: tsinfo.sqC COBOL: tabscont.sqb tspace.sqb

表 1. DB2 API、組み込みファイル、およびサンプル・プログラム (続き)

DB2 API	組み込みファイル	サンプル・プログラム
sqlbftcq - 表スペース・コンテナー照会の取り出し	sqlutil	C: tsinfo.sqc C++: tsinfo.sqC COBOL: tabscont.sqb tspace.sqb
sqlbctcq - 表スペース・コンテナー照会のクローズ	sqlutil	C: tsinfo.sqc C++: tsinfo.sqC COBOL: tabscont.sqb tspace.sqb
sqlbstsc - 表スペース・コンテナーの設定	sqlutil	C: dbrecov.sqc C++: dbrecov.sqC COBOL: tabscont.sqb tspace.sqb
sqlbmtsq - 表スペース照会	sqlutil	C: dbrecov.sqc tsinfo.sqc C++: dbrecov.sqC tsinfo.sqC COBOL: tabspace.sqb tspace.sqb
sqlbstpq - 単一の表スペース照会	sqlutil	C: tsinfo.sqc C++: tsinfo.sqC COBOL: tabspace.sqb tspace.sqb
sqlbotsq - 表スペース照会のオープン	sqlutil	C: tsinfo.sqc C++: tsinfo.sqC COBOL: tabspace.sqb tspace.sqb
sqlbftpq - 表スペース照会の取り出し	sqlutil	C: tsinfo.sqc C++: tsinfo.sqC COBOL: tabspace.sqb tspace.sqb
sqlbctsq - 表スペース照会のクローズ	sqlutil	C: tsinfo.sqc C++: tsinfo.sqC COBOL: tabspace.sqb tspace.sqb
sqlbgtss - 表スペース統計の入手	sqlutil	C: tsinfo.sqc C++: tsinfo.sqC COBOL: tabspace.sqb tspace.sqb
sqluvqdp - 表の表スペースの静止	sqlutil	C: tbmove.sqc C++: tbmove.sqC COBOL: tload.sqb
ノードの管理		
sqleaddn - ノードの追加	sqlenv	n/a
sqledrpn - ノード・ドロップの検査	sqlenv	n/a
サテライト		
db2GetSyncSession - サテライト同期セッションの入手	db2ApiDf	n/a
db2QuerySatelliteProgress - サテライト同期の照会	db2ApiDf	n/a
db2SetSyncSession - サテライト同期セッションの設定	db2ApiDf	n/a
db2SyncSatellite - サテライトの同期	db2ApiDf	n/a
db2SyncSatelliteStop - サテライト同期の停止	db2ApiDf	n/a
db2SyncSatelliteTest - サテライト同期のテスト	db2ApiDf	n/a
データベース・パーティション・グループの管理		
sqludrdt - データベース・パーティション・グループの再分散	sqlutil	n/a
追加の API		
sqluadau - 許可の入手	sqlutil	C: dbauth.sqc inauth.sqc C++: dbauth.sqC inauth.sqC
sqlegins - インスタンスの入手	sqlenv	C: ininfo.c C++: ininfo.C COBOL: dbinst.cbl

表 1. DB2 API、組み込みファイル、およびサンプル・プログラム (続き)

DB2 API	組み込みファイル	サンプル・プログラム
sqleqryc - クライアントの照会	sqlenv	C: cli_info.c C++: cli_info.C COBOL: client.cbl
sqleqryi - クライアント情報の照会	sqlenv	C: cli_info.c C++: cli_info.C
sqlesetc - クライアントの設定	sqlenv	C: cli_info.c dbcfg.sqc dbmcon.sqc C++: cli_info.C dbcfg.sqC dbmcon.sqC COBOL: client.cbl
sqleseti - クライアント情報の設定	sqlenv	C: cli_info.c C++: cli_info.C
sqlesact - 会計情報ストリングの設定	sqlenv	C: cli_info.c C++: cli_info.C COBOL: setact.cbl
db2ReadLog - ログの非同期読み取り	db2ApiDf	C: dbrecov.sqc C++: dbrecov.sqC
db2ReadLogNoConn - データベース接続なしのログの読み取り	db2ApiDf	n/a
主題に関するトピック集	db2ApiDf	n/a
db2ReadLogNoConnTerm - データベース接続なしのログ読み取りの終了	db2ApiDf	n/a
sqlugrpn - 行パーティション番号の入手	sqlutil	n/a
sqlugtpi - 表のパーティション情報の入手	sqlutil	n/a
db2AdminMsgWrite - 管理メッセージの書き込み	db2ApiDf	n/a
db2SetWriteForDB - 入出力の設定または再開	db2ApiDf	n/a
db2ArchiveLog - アクティブ・ログのアーカイブ	db2ApiDf	n/a
db2DatabasePing - データベースの PING	db2ApiDf	n/a
db2Inspect - データベースの検査	db2ApiDf	n/a

注: 組み込みファイルの拡張子はプログラミング言語ごとに異なります。 C/C++ 組み込みファイルのファイル拡張子は .h です。 COBOL 組み込みファイルのファイル拡張子は .cbl です。組み込みファイルは、以下のディレクトリーで検索できます。

C/C++ (UNIX):

```
sqllib/include
```

C/C++ (Windows):

```
sqllib¥include
```

COBOL (UNIX):

```
sqllib/include/cobol_a

sqllib/include/cobol_i

sqllib/include/cobol_mf
```

COBOL (Windows):

```
sqllib¥include¥cobol_a

sqllib¥include¥cobol_i

sqllib¥include¥cobol_mf
```

表 2. DB2 API を使用した C/C++ サンプル・プログラム

サンプル・プログラム	組み込まれた API
cli_info.c, cli_info.C	• sqlesact - 会計情報ストリングの設定 • sqlesetc - クライアントの設定 • sqleseti - クライアント情報の設定 • sqleqryc - クライアントの照会 • sqleqryi - クライアント情報の照会
dbauth.sqc, dbauth.sqC	• sqluadau - 許可の入手
dbcfg.sqc, dbcfg.sqC	• db2AutoConfig - 自動構成 • db2AutoConfigMemory - 自動構成メモリーの解放 • sqlesetc - クライアントの設定 • sqlaintp - エラー・メッセージの入手
dbconn.sqc, dbconn.sqC	• db2DatabaseRestart - データベースの再始動 • sqlefrce - アプリケーションの強制終了
dbcreate.c, dbcreate.C	• sqlecrea - データベースの作成 • sqledrpd - データベースのドロップ
dbinfo.c, dbinfo.C	• db2CfgGet - 構成の入手 • db2CfgSet - 構成の設定
dbmcon.sqc, dbmcon.sqC	• sqlesetc - クライアントの設定
dbmigrat.c, dbmigrat.C	• sqlemgdb - データベースの移行
dbpkg.sqc, dbpkg.sqC	• sqlaprep - プログラムのプリコンパイル • sqlabndx - バインド • sqlarbnd - 再バインド

表 2. DB2 API を使用した C/C++ サンプル・プログラム (続き)

サンプル・プログラム	組み込まれた API
dbrecov.sqc, dbrecov.sqC	• db2HistoryCloseScan - 履歴ファイルのスキャンのクローズ • db2HistoryGetEntry - 履歴ファイルの次項目の入手 • db2HistoryOpenScan - 履歴ファイルのスキャンのオープン • db2HistoryUpdate - 履歴ファイルの更新 • db2Prune - 履歴ファイルの整理 • db2CfgGet - 構成パラメーターの入手 • db2CfgSet - 構成パラメーターの設定 • sqlbmtsq - 表スペース照会 • sqlbstsc - 表スペース・コンテナーの設定 • sqlbtcq - 表スペース・コンテナー照会 • sqlecrea - データベースの作成 • sqledrpd - データベースのドロップ • sqlefmem - メモリーの解放 • db2Backup - データベースのバックアップ • db2Restore - データベースのリストア • db2ReadLog - ログの非同期読み取り • db2ReadLogNoConn - データベース接続なしのログの読み取り • db2Rollforward - データベースのロールフォワード
dbsample.sqc	• db2DatabaseRestart - データベースの再始動 • sqlecrea - データベースの作成 • sqlefrce - アプリケーションの強制終了 • sqlabndx - パッケージのバインド
dbthrds.sqc, dbthrds.sqC	• sqleAttachToCtx - コンテキストへのアタッチ • 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 • sqleDetachFromCtx - コンテキストからの切り離し • sqleSetTypeCtx - アプリケーション・コンテキスト・タイプの設定
dtformat.sqc	• db2Load - ロード • db2Import - インポート
inattach.c, inattach.C	• sqleatcp - パスワードのアタッチと変更 • sqleatin - アタッチ • sqledtin - 切り離し
inauth.sqc, inauth.sqC	• db2CfgGet - 構成パラメーターの入手 • sqluadau - 許可の入手

表 2. DB2 API を使用した C/C++ サンプル・プログラム (続き)

サンプル・プログラム	組み込まれた API
ininfo.c, ininfo.C	• db2CfgGet - 構成パラメーターの入手 • db2CfgSet - 構成パラメーターの設定 • sqlegins - インスタンスの入手 • sqlectnd - ノードのカタログ • sqlenops - ノード・ディレクトリー・スキャンのオープン • sqlengne - ノード・ディレクトリー次項目の入手 • sqlencls - ノード・ディレクトリー・スキャンのクローズ • sqleuncn - ノードのアンカタログ • sqlecadb - データベースのカタログ • db2DbDirOpenScan - データベース・ディレクトリー・スキャンのオープン • db2DbDirGetNextEntry - データベース・ディレクトリーの次項目の入手 • sqledcgd - データベース・コメントの変更 • db2DbDirCloseScan - データベース・ディレクトリー・スキャンのクローズ • sqleuncd - データベースのアンカタログ • sqlegdad - DCS データベースのカタログ • sqlegdsc - DCS ディレクトリー・スキャンのオープン • sqlegdge - データベース用 DCS ディレクトリー項目の入手 • sqlegdgt - DCS ディレクトリー項目の入手 • sqlegdcl - DCS ディレクトリー・スキャンのクローズ • sqlegdel - DCS データベースのアンカタログ • sqlesdeg - ランタイム次数の設定
instart.c, instart.C	• sqlefrce - アプリケーションの強制終了 • db2InstanceStart - インスタンスの開始 • db2InstanceStop - インスタンスの停止
tbmove.sqc, tbmove.sqC	• db2Export - エクスポート • db2Import - インポート • sqluvqdp - 表の表スペースの静止 • db2Load - ロード • db2LoadQuery - ロードの照会
tbreorg.sqc, tbreorg.sqC	• db2Reorg - 再編成 • db2Runstats - 統計の実行
tscreate.sqc, tscreate.sqC	• db2CfgGet - 構成パラメーターの入手

表 2. DB2 API を使用した C/C++ サンプル・プログラム (続き)

サンプル・プログラム	組み込まれた API
tsinfo.sqc, tsinfo.sqC	• sqlbstpq - 単一の表スペース照会 • sqlbgtss - 表スペース統計の入手 • sqlbmtsq - 表スペース照会 • sqlefmem - メモリーの解放 • sqlbotsq - 表スペース照会のオープン • sqlbftpq - 表スペース照会の取り出し • sqlbctsq - 表スペース照会のクローズ • sqlbtcq - 表スペース・コンテナー照会 • sqlbotcq - 表スペース・コンテナー照会のオープン • sqlbftcq - 表スペース・コンテナー照会の取り出し • sqlbctcq - 表スペース・コンテナー照会のクローズ
utilapi.c, utilapi.C	• sqlaintp - エラー・メッセージの入手 • sqlogstt - SQLSTATE メッセージの入手 • sqleatin - アタッチ • sqledtin - 切り離し
utilsnap.c, utilsnap.C	• db2GetSnapshot - スナップショットの入手 • db2MonitorSwitches - モニター・スイッチの入手/更新

表 3. DB2 API を使用した COBOL サンプル・プログラム

サンプル・プログラム	組み込まれた API
checkerr.cbl	• sqlaintp - エラー・メッセージの入手 • sqlogstt - SQLSTATE メッセージの入手
client.cbl	• sqleqryc - クライアントの照会 • sqlesetc - クライアントの設定
db_udcs.cbl	• sqleatin - アタッチ • sqlecrea - データベースの作成 • sqledrpd - データベースのドロップ
dbcat.cbl	• sqlecadb - データベースのカタログ • db2DbDirCloseScan - データベース・ディレクトリー・スキャンのクローズ • db2DbDirGetNextEntry - データベース・ディレクトリーの次項目の入手 • db2DbDirOpenScan - データベース・ディレクトリー・スキャンのオープン • sqleuncd - データベースのアンカタログ

表 3. DB2 API を使用した COBOL サンプル・プログラム (続き)

サンプル・プログラム	組み込まれた API
dbcmt.cbl	• sqledcgd - データベース・コメントの変更 • db2DbDirCloseScan - データベース・ディレクトリー・スキャンのクローズ • db2DbDirGetNextEntry - データベース・ディレクトリーの次項目の入手 • db2DbDirOpenScan - データベース・ディレクトリー・スキャンのオープン • sqleisig - シグナル・ハンドラーのインストール
dbinst.cbl	• sqleatcp - パスワードのアタッチと変更 • sqleatin - アタッチ • sqledtin - 切り離し • sqlegins - インスタンスの入手
dbstat.sqb	• db2Reorg - 再編成 • db2Runstats - 統計の実行
dcscat.cbl	• sqlegdad - DCS データベースのカタログ • sqlegdcl - DCS ディレクトリー・スキャンのクローズ • sqlegdel - DCS データベースのアンカタログ • sqlegdge - データベース用 DCS ディレクトリー項目の入手 • sqlegdgt - DCS ディレクトリー項目の入手 • sqlegdsc - DCS ディレクトリー・スキャンのオープン
ebcdicdb.cbl	• sqleatin - アタッチ • sqlecrea - データベースの作成 • sqledrpd - データベースのドロップ
expsamp.sqb	• db2Export - エクスポート • db2Import - インポート
impexp.sqb	• db2Export - エクスポート • db2Import - インポート
loadqry.sqb	• db2LoadQuery - ロードの照会
migrate.cbl	• sqlemgdb - データベースの移行
nodecat.cbl	• sqlectnd - ノードのカタログ • sqlencls - ノード・ディレクトリー・スキャンのクローズ • sqlengne - ノード・ディレクトリー次項目の入手 • sqlenops - ノード・ディレクトリー・スキャンのオープン • sqleuncn - ノードのアンカタログ
rebind.sqb	• sqlarbnd - 再バインド

表 3. DB2 API を使用した COBOL サンプル・プログラム (続き)

サンプル・プログラム	組み込まれた API
tabscont.sqb	• sqlbctcq - 表スペース・コンテナー照会のクローズ • sqlbftcq - 表スペース・コンテナー照会の取り出し • sqlbotcq - 表スペース・コンテナー照会のオープン • sqlbtcq - 表スペース・コンテナー照会 • sqlefmem - メモリーの解放
tabspace.sqb	• sqlbctsq - 表スペース照会のクローズ • sqlbftpq - 表スペース照会の取り出し • sqlbgtss - 表スペース統計の入手 • sqlbmtsq - 表スペース照会 • sqlbotsq - 表スペース照会のオープン • sqlbstpq - 単一の表スペース照会 • sqlefmem - メモリーの解放
tload.sqb	• db2Export - エクスポート • sqluvqdp - 表の表スペースの静止
tspace.sqb	• sqlbctcq - 表スペース・コンテナー照会のクローズ • sqlbctsq - 表スペース照会のクローズ • sqlbftcq - 表スペース・コンテナー照会の取り出し • sqlbftpq - 表スペース照会の取り出し • sqlbgtss - 表スペース統計の入手 • sqlbmtsq - 表スペース照会 • sqlbotcq - 表スペース・コンテナー照会のオープン • sqlbotsq - 表スペース照会のオープン • sqlbstpq - 単一の表スペース照会 • sqlbstsc - 表スペース・コンテナーの設定 • sqlbtcq - 表スペース・コンテナー照会 • sqlefmem - メモリーの解放
setact.cbl	• sqlesact - 会計情報ストリングの設定

関連資料:

- 「*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*」の『C および C++ の組み込みファイル』
- 「*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*」の『COBOL の組み込みファイル』
- 「*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『C 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『COBOL のサンプル』

API の説明の編成方法

以下のサブセクションの一部またはすべての前には、各 API の簡潔な説明が記載されています。

有効範囲:

インスタンス内での API の操作の有効範囲です。単一パーティション・データベース環境では、有効範囲は単一データベース・パーティションのみになります。複数パーティション・データベース環境では、有効範囲はノード構成ファイル (db2nodes.cfg) に定義されたすべてのデータベース・パーティション・サーバーの集合、または API 呼び出し元のデータベース・パーティションです。

許可:

API を正常に呼び出すのに必要な権限です。

必要な接続:

データベース、インスタンス、なし、接続が確立される、のいずれかです。関数にデータベース接続またはインスタンス接続機構が必要かどうか、または正常に操作を行うのに接続は必要ないかを示します。

なし は、 API が正常に機能するためにデータベース接続が必要ないことを意味します。*接続が確立される* は、 API が呼び出されるとき、API がデータベースへの接続を確立することを意味します。

API によっては、それを呼び出す前に、データベースへの明示接続またはインスタンスへの接続機構が必要になる場合もあります。データベース接続またはインスタンス接続機構を必要とする API は、ローカルとリモートのどちらにおいても実行できます。データベース接続またはインスタンス接続機構のどちらも必要としない API は、リモートでは実行できません。クライアント側で呼び出されると、クライアント環境だけに影響が及びます。

API 組み込みファイル:

API プロトタイプを含む組み込みファイルの名前、および事前定義された定数およびパラメーターのうち必要なものです。

注: 組み込みファイルの拡張子はプログラミング言語ごとに異なります。 C/C++ 組み込みファイルのファイル拡張子は .h です。 COBOL 組み込みファイルのファイル拡張子は .cbl です。組み込みファイルは、以下のディレクトリーで検索できます。

C/C++ (UNIX):

```
sqllib/include
```

C/C++ (Windows):

```
sqllib¥include
```

COBOL (UNIX):

```
sqllib/include/cobol_a
```

```
sqllib/include/cobol_i

sqllib/include/cobol_mf
```

COBOL (Windows):

```
sqllib¥include¥cobol_a

sqllib¥include¥cobol_i

sqllib¥include¥cobol_mf
```

C API 構文:

API 呼び出しの C 構文です。

バージョン 6 以降、DB2 管理 API には新しい標準が適用されています。新しい API 定義のインプリメンテーションは、段階的に行われています。以下に、変更内容の概要を簡潔に示します。

- 新しい API 名では、接頭部「db2」の後ろに、意味を持つ大文字小文字混合のストリング (db2LoadQuery など) が続きます。関連する API には、これらを論理グループ化できるようにするための名前があります。たとえば、次のようなものがあります。

  ```
  db2HistoryCloseScan
  db2HistoryGetEntry
  db2HistoryOpenScan
  db2HistoryUpdate
  ```

- 汎用 API の接頭部「db2g」の後ろには、 C API 名と一致するストリングが続きます。汎用 API が使用するデータ構造の名前にも、接頭部「db2g」があります。
- 関数 (*versionNumber*) の最初のパラメーターは、コードをコンパイルするバージョン、リリース、または PTF レベルを表します。このバージョン番号は、 2 番目のパラメーターとして渡される構造のレベルの指定に使用します。
- この関数の 2 番目のパラメーターは、 API の基本インターフェース構造を指す void ポインターです。この構造のそれぞれのエレメントは、アトミック・タイプ (db2Long32 など) またはポインターです。それぞれのパラメーター名は、以下の命名規則に従っています。

  ```
  piCamelCase  - 入力データへのポインター
  poCamelCase  - 出力データへのポインター
  pioCamelCase - 入出力データへのポインター
  iCamelCase   - 入力データ
  ioCamelCase  - 入力/出力データ
  oCamelCase   - 出力データ
  ```

- 3 番目のパラメーターは SQLCA を指すポインターで、必須です。

汎用 API 構文:

COBOL および FORTRAN プログラミング言語を使用した場合の API の構文です。

重要: API に渡されるすべての文字ストリングごとに余分の 1 バイトを与えてください。そうしない場合、予期しないエラーが発生するおそれがあります。この余分のバイトはデータベース・マネージャーによって変更されます。

API パラメーター:

それぞれの API パラメーターおよびその値についての説明です。事前定義された値は適切なシンボルを使用してリストされています。シンボルの実際の値は適切な言語組み込みファイルから得ることができます。 COBOL プログラマーの方は、すべての記号の中で下線 (_) の代わりにハイフン (-) を使用する必要があります。各ホスト言語のパラメーター・データ・タイプに関する詳細については、サンプル・プログラムをご覧ください。

注: データベース・マネージャー API を呼び出すアプリケーションは、戻りコードと SQLCA 構造を検査して、正しくエラー条件をチェックする必要があります。ほとんどのデータベース・マネージャー API は、正常実行時に戻りコード 0 を戻します。一般に、0 以外の戻りコードは、 2 次エラー処理メカニズム (SQLCA 構造) が破壊されている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 構造が破壊された原因としては、この構造に無効なアドレスを渡したことが考えられます。

エラー情報は、SQLCA 構造の SQLCODE フィールドと SQLSTATE フィールドに戻され、ほとんどの API 呼び出しが実行された後に更新されます。データベース・マネージャー API を呼び出すソース・ファイルは、 1 つまたは複数の SQLCA 構造 (名前は任意) を提供できます。 SQLCODE 値が 0 の場合は、正常実行 (SQLWARN 警告条件が伴うこともある) を示します。正の値は、ステートメントがホスト変数の切り捨てなどの警告を伴って正常実行したことを示します。負の値は、エラー条件が発生したことを意味します。

追加フィールド 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・マネージャーで整合性を持つ標準化されたエラー・コードが含まれています。 SQLSTATE は多数のデータベース・マネージャーで共通なので、移植性が必要な場合には SQLSTATE を使用してください。

SQLWARN フィールドには、SQLCODE が 0 の場合でも警告標識の配列が含まれます。

REXX API 構文:

API 呼び出しの REXX 構文です (該当する場合)。

SQLDB2 インターフェースは REXX からの API 呼び出しをサポートします。SQLDB2 インターフェースは、新規の API、あるいは以前はサポートされていなかった、 SQLCA 以外の出力を行わない API を REXX でサポートするために作成されました。 SQLDB2 インターフェースを介してコマンドを呼び出す構文は、コマンド行プロセッサー (CLP) を介してコマンドを呼び出す構文と同じです。しかし、`call db2` というトークンが `CALL SQLDB2` に置き換えられる点が異なっています。REXX から `CALL SQLDB2` を使用すると、 CLP を直接呼び出す上で、以下のような利点があります。

- コンパウンド REXX 変数の SQLCA が設定されます。
- デフォルトでは、すべての CLP 出力メッセージがオフにされます。

REXX API パラメーター:

それぞれの REXX API パラメーターおよびその値についての説明です (該当する場合)。

使用上の注意:

その他の情報です。

db2AddContact - 連絡先の追加

連絡先を連絡先のリストに追加します。連絡先は、通知メッセージが送信されるユーザーです。連絡先は、システムでローカルに定義することも、グローバル・リストで定義することもできます。 DB2 Administration Server (DAS) の構成パラメーター *contact_host* の設定は、リストがローカルかグローバルかを判別します。

許可:

なし

必要な接続:

なし

API 組み込みファイル:

db2ApiDf.h

C API 構文:

```
/* File: db2ApiDf.h */
/* API: db2AddContact */
/* ... */
SQL_API_RC SQL_API_FN
db2AddContact (
  db2Uint32 versionNumber,
  void *pParmStruct,
  struct sqlca *pSqlca);

typedef SQL_STRUCTURE db2AddContactData
{
   char                     *piUserid;
   char                     *piPassword;
   char                     *piName;
   db2Uint32                iType;
   char                     *piAddress;
   db2Uint32                iMaxPageLength;
   char                     *piDescription;
} db2AddContactData;
/* ... */
```

API パラメーター:

versionNumber
入力。 2 番目のパラメーター *pParmStruct* として渡される構造のバージョンとリリースのレベルを指定します。

pParmStruct
入力。 *db2AddContactData* 構造を指すポインター。

pSqlca
出力。*sqlca* 構造へのポインター。

piUserid;
入力。ユーザー名。

piPassword
入力。 *piUserid* 用のパスワード。

piName
入力。連絡先名。

iType 入力。連絡先のタイプを指定します。有効な値は以下のとおりです。

- DB2CONTACT_EMAIL
- DB2CONTACT_PAGE

piAddress
入力。 *iType* パラメーターの電子メールまたはページャー・アドレス。

iMaxPageLength
入力。 *iType* が DB2CONTACT_PAGE に設定されたときの最大メッセージ長。

piDescription
入力。ユーザー提供の連絡先の説明。

関連資料:

- 453 ページの『SQLCA』
- 「*管理ガイド: パフォーマンス*」の『contact_host - 「連絡先リストのロケーション」構成パラメーター』
- 60 ページの『db2DropContact - 連絡先のドロップ』
- 83 ページの『db2GetContacts - 連絡先の入手』
- 288 ページの『db2UpdateContact - 連絡先の更新』

db2AddContactGroup - 連絡先グループの追加

連絡先グループのリストに、新しい連絡先グループを追加します。連絡先グループには、通知メッセージが送信されるユーザーのリストが入っています。連絡先グループは、システムでローカルに定義することも、グローバル・リストで定義することもできます。 DB2 Administration Server (DAS) の構成パラメーター *contact_host* の設定は、リストがローカルかグローバルかを判別します。

許可:

なし

必要な接続:

なし

API 組み込みファイル:

db2ApiDf.h

C API 構文:

```
/* File: db2ApiDf.h */
/* API: db2AddContactGroup */
/* ... */
```

```
SQL_API_RC SQL_API_FN
db2AddContactGroup (
  db2Uint32 versionNumber,
  void *pParmStruct,
  struct sqlca *pSqlca);

typedef SQL_STRUCTURE db2AddContactGroupData
{
   char                       *piUserid;
   char                       *piPassword;
   char                       *piGroupName;
   char                       *piDescription;
   db2Uint32                  iNumContacts;
   struct db2ContactTypeData *piContacts;
} db2AddContactGroupData;

typedef SQL_STRUCTURE db2ContactTypeData
{
   db2Uint32                  contactType;
   char                       *pName;
} db2ContactTypeData;
/* ... */
```

API パラメーター:

versionNumber
入力。 2 番目のパラメーター *pParmStruct* として渡される構造のバージョンとリリースのレベルを指定します。

pParmStruct
入力。 *db2AddContactGroupData* 構造を指すポインター。

pSqlca
出力。*sqlca* 構造へのポインター。

piUserid
入力。ユーザー名。

piPassword
入力。 *piUserid* 用のパスワード。

piGroupName
入力。検索されるグループ名。

piDescription
入力。グループの説明。

iNumContacts
入力。 *piContacts* の数。

piContacts
db2ContactTypeData 構造を指すポインター。

contactType
連絡先のタイプを指定します。有効な値は以下のとおりです。

- DB2CONTACT_SINGLE
- DB2CONTACT_GROUP

pName
連絡先グループ名、または *contactType* が DB2CONTACT_SINGLE に設定されている場合は連絡先の名前。

関連資料:

- 453 ページの『SQLCA』
- 「*管理ガイド: パフォーマンス*」の『contact_host - 「連絡先リストのロケーション」構成パラメーター』
- 61 ページの『db2DropContactGroup - 連絡先グループのドロップ』
- 80 ページの『db2GetContactGroup - 連絡先グループの入手』
- 81 ページの『db2GetContactGroups - 連絡先グループの入手』
- 289 ページの『db2UpdateContactGroup - 連絡先グループの更新』

---

db2AdminMsgWrite - 管理メッセージの書き込み

ユーザーとレプリケーションが db2diag.log と管理用通知ログに情報を書き込むためのメカニズムを提供します。

許可:

なし

必要な接続:

なし

API 組み込みファイル:

db2ApiDf.h

C API 構文:

```
/* File: db2ApiDf.h */
/* API: db2AdminMsgWrite */
/* ... */
SQL_API_RC SQL_API_FN
db2AdminMsgWrite (
  db2Uint32 versionNumber,
  void *pParmStruct,
  struct sqlca *pSqlca);

typedef struct
{
  db2Uint32 iMsgType;
  db2Uint32 iComponent;
  db2Uint32 iFunction;
  db2Uint32 iProbeID;
  char *piData_title;
  void *piData;
  db2Uint32 iDataLen;
  db2Uint32 iError_type;
} db2AdminMsgWriteStruct;
/* ... */
```

API パラメーター:

versionNumber
入力。 2 番目のパラメーター *pParmStruct* として渡される構造のバージョンとリリースのレベルを指定します。

pParmStruct
入力。 *db2AdminMsgWriteStruct* 構造を指すポインター。

pSqlca
出力。*sqlca* 構造へのポインター。

iMsgType
入力。記録するデータのタイプを指定します。有効な値は、BINARY_MSG (バイナリー・データの場合)、および STRING_MSG (ストリング・データの場合) です。

iComponent
入力。 0 を指定してください。

iFunction
入力。 0 を指定してください。

iProbeID
入力。数値のプローブ・ポイントを指定してください。

piData_title
入力。記録するデータを記述するタイトル・ストリングを指すポインター。タイトルが必要ない場合は、NULL に設定できます。

piData
入力。記録するデータを指すポインター。データの記録が必要ない場合は、NULL に設定できます。

iDataLen
入力。ロギングに使用する バイナリー・データのバイト数 (*iMsgType* が BINARY_MSG の場合)。 *iMsgType* が STRING_MSG の場合は使用されません。

iError_type
入力。有効な値は以下のとおりです。

```
DB2LOG_SEVERE_ERROR   (1) - 重大エラーが発生した
DB2LOG_ERROR          (2) - エラーが発生した
DB2LOG_WARNING        (3) - 警告が発生した
DB2LOG_INFORMATION    (4) - 通知
```

使用上の注意:

この API が管理用通知ログに記録を行うのは、指定されたエラー・タイプが *notifylevel* データベース・マネージャー構成パラメーターの値以下である場合だけです。この API が db2diag.log に記録を行うのは、指定されたエラー・タイプが *diaglevel* データベース・マネージャー構成パラメーターの値以下である場合だけです。ただし、管理用通知ログに書き込まれるすべての情報は、*diaglevel* データベース・マネージャー構成パラメーターがゼロに設定されない限り、 db2diag.log に複写されます。

関連資料:

• 453 ページの『SQLCA』

db2ArchiveLog - アクティブ・ログのアーカイブ

リカバリー可能データベースのアクティブ・ログ・ファイルをクローズし、切り捨てます。ユーザー出口が使用可能な場合、アーカイブ要求を発行します。

許可:

以下のいずれかが必要です。

- *sysadm*
- *sysctrl*
- *sysmaint*
- *dbadm*

必要な接続:

この API を呼び出せば、指定したデータベースへの接続が自動的に確立されます。指定したデータベースへの接続がすでに存在している場合、API はエラーを戻します。

API 組み込みファイル:

db2ApiDf.h

C API 構文:

```
/* File: db2ApiDf.h */
/* API: db2ArchiveLog */
/* ... */
SQL_API_RC SQL_API_FN
db2ArchiveLog (
  db2Uint32 version,
  void *pDB2ArchiveLogStruct,
  struct sqlca *pSqlca);

typedef struct
{
   char                *piDatabaseAlias;
   char                *piUserName;
   char                *piPassword;
   db2Uint16           iAllNodeFlag;
   db2Uint16           iNumNodes;
   SQL_PDB_NODE_TYPE   *piNodeList;
   db2Uint32           iOptions;
} db2ArchiveLogStruct
/* ... */
```

汎用 API 構文:

```
/* File: db2ApiDf.h */
/* API: db2gArchiveLog */
/* ... */
SQL_API_RC SQL_API_FN
db2gArchiveLog (
  db2Uint32 version,
  void *pDB2ArchiveLogStruct,
  struct sqlca *pSqlca);

typedef struct
{
   db2Uint32           iAliasLen;
   db2Uint32           iUserNameLen;
   db2Uint32           iPasswordLen;
```

```
   char                *piDatabaseAlias;
   char                *piUserName;
   char                *piPassword;
   db2Uint16           iAllNodeFlag;
   db2Uint16           iNumNodes;
   SQL_PDB_NODE_TYPE   *piNodeList;
   db2Uint32           iOptions;
} db2ArchiveLogStruct
/* ... */
```

API パラメーター:

version
入力。 2 番目のパラメーター *pDB2ArchiveLogStruct* として渡される変数のバージョンおよびリリース・レベルを指定します。

pDB2ArchiveLogStruct
入力。 *db2ArchiveLogStruct* 構造を指すポインターです。

pSqlca
出力。 *sqlca* 構造へのポインター。

iAliasLen
入力。データベースの別名の長さを示す 4 バイトの符号なし整数 (バイト単位)。

iUserNameLen
入力。ユーザー名の長さを示す 4 バイトの符号なし整数 (バイト単位) です。ユーザー名が使用されていない場合は、ゼロに設定してください。

iPasswordLen
入力。パスワードの長さを示す 4 バイトの符号なし整数 (バイト単位) です。パスワードが使用されていない場合は、ゼロに設定してください。

piDatabaseAlias
入力。アクティブ・ログをアーカイブする対象のデータベースのデータベース別名 (システム・データベース・ディレクトリーにカタログされている) を含むストリングです。

piUserName
入力。接続の試行時に使用されるユーザー名を含むストリングを指定します。

piPassword
入力。接続の試行時に使用されるパスワードを含むストリングです。

iAllNodeFlag
パーティション・データベース環境のみ。入力。操作を `db2nodes.cfg` ファイルでリストされているすべてのノードに適用するかどうかを示すフラグです。有効な値は以下のとおりです。

DB2ARCHIVELOG_NODE_LIST
piNodeList で渡されたノード・リスト内でノードに適用されます。

DB2ARCHIVELOG_ALL_NODES
すべてのノードに適用します。 *piNodeList* は NULL にする必要があります。これはデフォルト値です。

DB2ARCHIVELOG_ALL_EXCEPT
piNodeList で渡されたノード・リスト内で指定されたノードを除き、すべてのノードに適用されます。

iNumNodes
パーティション・データベース環境のみ。入力。 *piNodeList* 配列内のノードの数を指定します。

piNodeList
パーティション・データベース環境のみ。入力。アーカイブ・ログ操作を適用する対象のノード番号の配列を指すポインターです。

iOptions
入力。将来の利用のために予約されています。

関連資料:

- 「コマンド・リファレンス」の『ARCHIVE LOG コマンド』

db2AutoConfig - 自動構成

アプリケーション・プログラムが、コントロール・センターで構成アドバイザーにアクセスできるようにします。このアドバイザーに関する詳細は、コントロール・センター内のオンライン・ヘルプ機能によって提供されます。

許可:

sysadm

必要な接続:

データベース

API 組み込みファイル:

db2AuCfg.h

C API 構文:

```
/* File: db2AuCfg.h */
/* API: db2AutoConfig */
/* ... */
SQL_API_RC_SQL_API_FN
db2AutoConfig(
  db2Uint32   db2VersionNumber,
  void *pAutoConfigInterface,
  struct sqlca *pSqlca);

typedef struct {
    db2int32  iProductID;
    char  iProductVersion[DB2_SG_PROD_VERSION_SIZE];
    char  iDbAlias[SQL_ALIAS_SZ];
    db2int32  iApply;
    db2AutoConfigInput  iParams;
    db2AutoConfigOutput  oResult;
} db2AutoConfigInterface;

typedef struct {
   db2int32  token;
```

```
   db2int32  value;
} db2AutoConfigElement;

typedef struct {
   db2Uint32  numElements;
   db2AutoConfigElement *pElements;
} db2AutoConfigArray;

typedef db2AutoConfigArray  db2AutoConfigInput;
typedef db2AutoConfigArray  db2AutoConfigDiags;

typedef struct {
   db2Uint32  numElements;
   struct sqlfupd *pConfigs;
   void *pDataArea;
} db2ConfigValues;

typedef struct {
   char *pName;
   db2int32  value;
} db2AutoConfigNameElement;

typedef struct {
   db2Uint32  numElements;
   db2AutoConfigElement *pElements;
} db2AutoConfigNameArray;

typedef db2AutoConfigNameArray db2BpValues;

typedef struct {
   db2ConfigValues  oOldDbValues;
   db2ConfigValues  oOldDbmValues;
   db2ConfigValues  oNewDbValues;
   db2ConfigValues  oNewDbmValues
   db2AutoConfigDiags  oDiagnostics;
   db2BpValues  oOldBpValues;
   db2BpValues  oNewBpValues;
} db2AutoConfigOutput;
/* ... */
```

API パラメーター:

db2VersionNumber
入力。 2 番目のパラメーター *pAutoConfigInterface* として渡される構造のバージョンとリリースのレベルを指定します。

pAutoConfigInterface
入力。 *db2AutoConfigInterface* 構造を指すポインター。

pSqlca
出力。*sqlca* 構造へのポインター。

iProductID
入力。ユニークな製品 ID を指定します。有効な製品 ID の値については、API 組み込みファイル `db2AuCfg.h` を参照してください。

iProductVersion
入力。製品のバージョンを指定する 16 バイトのストリングです。

iDbAlias
入力。データベース別名を指定するストリングです。

iApply
入力。構成を自動的に更新します。有効な値については、API 組み込みファイル db2AuCfg.h を参照してください。

iParams
入力。 アドバイザーにパラメーターを渡します。

oResult
出力。 アドバイザーからの結果がすべて含まれます。

token 入力パラメーターと出力診断の両方に関する構成値を指定します。

value トークンによって指定されたデータを保持します。

numElements
配列エレメントの数を示します。

pElements
エレメント配列を指すポインター。

db2AutoConfigDiags
診断と問題判別を行うためのトークンと値を返します。トークンは問題を識別し、値は適切な勧告 (ある場合) を示します。トークンと値のリストについては、 API 組み込みファイル db2AuCfg.h を参照してください。

pConfigs
SQLFUPD 構造を指すポインター。

pDataArea
構成の値が含まれるデータ域を指すポインター。

pName
出力。出力バッファー・プールの名前。

value 名前で指定されたバッファー・プールのサイズ (ページ単位) を保持します。

oOldDbValues
出力。 *iApply* の値がデータベース構成またはすべての構成を更新するように設定されている場合、この値は、アドバイザー が使用される前のデータベースの構成値を表します。そうでなければ、これは現行値です。

oOldDbmValues
出力。 *iApply* の値がすべての構成を更新するように設定されている場合、この値は アドバイザーが使用される前のデータベース・マネージャーの構成値を表します。そうでなければ、これは現行値です。

oNewDbValues
出力。 *iApply* の値がデータベース構成またはすべての構成を更新するよう設定されている場合、この値は現行データベースの構成値を表します。そうでなければ、これはアドバイザーに対する推奨値です。

oNewDbmValues
出力。 *iApply* の値がすべての構成を更新するよう設定されている場合、この値は現行のデータベース・マネージャーの構成値を表します。そうでなければ、これはアドバイザーに対する推奨値です。

oDiagnostics
出力。 アドバイザーからの診断が含まれます。

oOldBpValues
出力。 *iApply* の値がデータベース構成またはすべての構成を更新するように設定されている場合、この値は、アドバイザーが使用される前のバッファー・プールのサイズ (ページ単位) を表します。そうでなければ、これは現行値です。

oNewBpValues
出力。 *iApply* の値がデータベース構成またはすべての構成を更新するよう設定されている場合、この値は現行のバッファー・プールのサイズ (ページ単位) を表します。そうでなければ、これはアドバイザーに対する推奨値です。

使用上の注意:

db2AutoConfig によって割り振られたメモリーを解放するには、db2AutoConfigFreeMemory を呼び出します。

関連資料:

- 453 ページの『SQLCA』
- 482 ページの『SQLFUPD』
- 27 ページの『db2AutoConfigFreeMemory - 自動構成メモリーの解放』
- 39 ページの『db2CfgSet - 構成パラメーターの設定』

関連サンプル:

- 『dbcfg.sqc -- Configure database and database manager configuration parameters (C)』
- 『dbcfg.sqC -- Configure database and database manager configuration parameters (C++)』

db2AutoConfigFreeMemory - 自動構成メモリーの解放

db2AutoConfig によって割り振られたメモリーを解放します。

許可:

sysadm

必要な接続:

データベース

API 組み込みファイル:

db2AuCfg.h

C API 構文:

```
/* File: db2AuCfg.h */
/* API: db2AutoConfigFreeMemory */
/* ... */
```

```
SQL_API_RC_SQL_API_FN
db2AutoConfigFreeMemory(
    db2Uint32  db2VersionNumber,
    void *pAutoConfigInterface,
    struct sqlca *pSqlca);
/* ... */
```

API パラメーター:

db2VersionNumber
入力。 2 番目のパラメーター *pAutoConfigInterface* として渡される構造のバージョンとリリースのレベルを指定します。

pAutoConfigInterface
入力。 *db2AutoConfigInterface* 構造を指すポインター。

pSqlca
出力。*sqlca* 構造へのポインター。

関連資料:

- 453 ページの『SQLCA』
- 24 ページの『db2AutoConfig - 自動構成』

関連サンプル:

- 『dbcfg.sqc -- Configure database and database manager configuration parameters (C)』
- 『dbcfg.sqC -- Configure database and database manager configuration parameters (C++)』

db2Backup - データベースのバックアップ

データベースまたは表スペースのバックアップ・コピーを作成します。

有効範囲:

この API は、それが実行されるデータベース・パーティションにのみ影響を与えます。

許可:

以下のいずれかが必要です。

- *sysadm*
- *sysctrl*
- *sysmaint*

必要な接続:

データベース。この API を呼び出せば、指定したデータベースへの接続が自動的に確立されます。

接続はバックアップの完了時に終了します。

API 組み込みファイル:

db2ApiDf.h

C API 構文:

```
/* File: db2ApiDf.h */
/* API: db2Backup */
/* ... */
SQL_API_RC SQL_API_FN
db2Backup (
  db2Uint32     versionNumber,
  void         *pDB2BackupStruct,
  struct sqlca *pSqlca);

typedef SQL_STRUCTURE db2BackupStruct
{
  char                        *piDBAlias;
  char                        oApplicationId[SQLU_APPLID_LEN+1];
  char                        oTimestamp[SQLU_TIME_STAMP_LEN+1];
  struct db2TablespaceStruct  *piTablespaceList;
  struct db2MediaListStruct   *piMediaList;
  char                        *piUsername;
  char                        *piPassword;
  void                        *piVendorOptions;
  db2Uint32                   iVendorOptionsSize;
  db2Uint32                   oBackupSize;
  db2Uint32                   iCallerAction;
  db2Uint32                   iBufferSize;
  db2Uint32                   iNumBuffers;
  db2Uint32                   iParallelism;
  db2Uint32                   iOptions;
  db2Uint32                   iUtilImpactPriority;
  char                        *piComprLibrary;
  void                        *piComprOptions;
  db2Uint32                   iComprOptionsSize;
} db2BackupStruct;

typedef SQL_STRUCTURE db2TablespaceStruct
{
  char                        **tablespaces;
  db2Uint32                   numTablespaces;
} db2TablespaceStruct;

typedef SQL_STRUCTURE db2MediaListStruct
{
  char                        **locations;
  db2Uint32                   numLocations;
  char                        locationType;
} db2MediaListStruct;
/* ... */
```

汎用 API 構文:

```
/* File: db2ApiDf.h */
/* API: db2Backup */
/* ... */
SQL_API_RC SQL_API_FN
db2gBackup (
  db2Uint32     versionNumber,
  void         *pDB2gBackupStruct,
  struct sqlca *pSqlca);

typedef SQL_STRUCTURE db2gBackupStruct
{
  char                        *piDBAlias;
  db2Uint32                   iDBAliasLen;
  char                        *poApplicationId;
  db2Uint32                   iApplicationIdLen;
```

```
  char                        *poTimestamp;
  db2Uint32                   iTimestampLen;
  struct db2gTablespaceStruct *piTablespaceList;
  struct db2gMediaListStruct  *piMediaList;
  char                        *piUsername;
  db2Uint32                   iUsernameLen;
  char                        *piPassword;
  db2Uint32                   iPasswordLen;
  void                        *piVendorOptions;
  db2Uint32                   iVendorOptionsSize;
  db2Uint32                   oBackupSize;
  db2Uint32                   iCallerAction;
  db2Uint32                   iBufferSize;
  db2Uint32                   iNumBuffers;
  db2Uint32                   iParallelism;
  db2Uint32                   iOptions;
  db2Uint32                   iUtilImpactPriority;
  char                        *piComprLibrary;
  db2Uint32                   iComprLibraryLen;
  void                        *piComprOptions;
  db2Uint32                   iComprOptionsSize;
} db2gBackupStruct;

typedef SQL_STRUCTURE db2gTablespaceStruct
{
  struct db2Char              *tablespaces;
  db2Uint32                   numTablespaces;
} db2gTablespaceStruct;

typedef SQL_STRUCTURE db2gMediaListStruct
{
  struct db2Char              *locations;
  db2Uint32                   numLocations;
  char                        locationType;
} db2gMediaListStruct;

typedef SQL_STRUCTURE db2Char
{
   char            *pioData;
   db2Uint32       iLength;
   db2Uint32       oLength;
} db2Char;
/* ... */
```

API パラメーター:

versionNumber
入力。 2 番目のパラメーター *pDB2BackupStruct* として渡される構造のバージョンとリリースのレベルを指定します。

pDB2BackupStruct
入力。 *db2BackupStruct* 構造を指すポインター。

pSqlca
出力。*sqlca* 構造へのポインター。

piDBAlias
入力。バックアップをとるデータベースのデータベース別名 (システム・データベース・ディレクトリーにカタログされている) を含むストリングを指定します。

iDBAliasLen
入力。データベースの別名の長さを示す 4 バイトの符号なし整数 (バイト単位)。

oApplicationId
出力。 API によって、アプリケーションにサービスを提供しているエージェントを識別するストリングが戻されます。データベース・モニターを使用するバックアップ操作の進行状況に関する情報を取得することもできます。

poApplicationId
出力。長さ SQLU_APPLID_LEN+1 (sqlutil.h で定義) のバッファーを提供します。 API によって、アプリケーションにサービスを提供しているエージェントを識別するストリングが戻されます。データベース・モニターを使用するバックアップ操作の進行状況に関する情報を取得することもできます。

iApplicationIdLen
入力。 *poApplicationId* バッファーの長さを示す 4 バイトの符号なし整数 (バイト単位)。 SQLU_APPLID_LEN+1 (sqlutil.h に定義) と等しくなければなりません。

oTimestamp
出力。 API によって、バックアップ・イメージのタイム・スタンプが戻されます。

poTimestamp
出力。長さ SQLU_TIME_STAMP_LEN+1 (sqlutil.h で定義) のバッファーを提供します。 API によって、バックアップ・イメージのタイム・スタンプが戻されます。

iTimestampLen
入力。 *poTimestamp* バッファーの長さを示す 4 バイトの符号なし整数 (バイト単位)。 SQLU_TIME_STAMP_LEN+1 (sqlutil.h で定義) と等しくする必要があります。

piTablespaceList
入力。バックアップをとる表スペースのリストです。表スペース・レベルのバックアップの場合にのみ必要です。データベース・レベルのバックアップの場合は NULL に設定しなければなりません。 DB2TablespaceStruct 構造を参照してください。

piMediaList
入力。この構造を使用することにより、呼び出し側はバックアップ操作の宛先を指定することができます。提供される情報は、*locationType* パラメーターの値によって異なります。 *locationType* に有効な値 (sqlutil.h で定義) は、以下のとおりです。

SQLU_LOCAL_MEDIA
ローカル装置 (テープ、ディスクまたはディスケットの組み合わせ)。

SQLU_TSM_MEDIA
TSM。ロケーション・ポインターが NULL に設定されている場合、 DB2 で提供される TSM 共用ライブラリーが使用されます。別のバージョンの TSM 共用ライブラリーが必要な場合には、SQLU_OTHER_MEDIA を使用し、共用ライブラリー名を入力してください。

SQLU_OTHER_MEDIA
ベンダー製品。ロケーション・フィールド内の共用ライブラリー名を提供します。

SQLU_USER_EXIT
ユーザー出口。追加の入力は必要ありません (サーバーが OS/2 上にある場合のみ使用可能です)。

詳しくは、*db2MediaListStruct* 構造を参照してください。

piUsername
入力。接続の試行時に使用されるユーザー名を含むストリングを指定します。 NULL にすることもできます。

iUsernameLen
入力。ユーザー名の長さを示す 4 バイトの符号なし整数 (バイト単位) です。ユーザー名が提供されていない場合は、ゼロに設定してください。

piPassword
入力。ユーザー名とともに使用されるパスワードを含むストリングを指定します。 NULL にすることもできます。

iPasswordLen
入力。パスワードの長さを示す 4 バイトの符号なし整数 (バイト単位) です。パスワードが提供されていない場合は、ゼロに設定してください。

piVendorOptions
入力。情報をアプリケーションからベンダー関数へ渡すのに使用されます。このデータ構造はフラットでなければなりません。つまり、間接のレベルはサポートされません。このデータについては、バイト反転が行われず、また、コード・ページがチェックされないことに注意してください。

iVendorOptionsSize
入力。 *piVendorOptions* フィールドの長さです。65535 バイト以下でなければなりません。

oBackupSize
出力。バックアップ・イメージのサイズ (MB バイト単位) を示します。

iCallerAction
入力。実行するアクションを指定します。有効な値 (`db2ApiDf.h` で定義) は、以下のとおりです。

DB2BACKUP_BACKUP
バックアップを開始します。

DB2BACKUP_NOINTERRUPT
バックアップを開始します。バックアップを自動実行するよう指定します。通常ユーザーの介入を要求するシナリオは、呼び出し側への最初の戻りなしに試行されるか、エラーを生成します。この呼び出し側アクションは、たとえば、バックアップに必要なメディアがすべて装てんされていることが明らかで、ユーティリティーによるプロンプトが必要とされない場合に使用してください。

DB2BACKUP_CONTINUE
ユーザーがユーティリティーによって要求された何らかのアクション (たとえば、新しいテープの装てん) を実行した後で、バックアップを継続します。

DB2BACKUP_TERMINATE
ユーザーがユーティリティーによって要求された何らかのアクションの実行に失敗した場合、バックアップを終了します。

DB2BACKUP_DEVICE_TERMINATE
バックアップに使用される装置のリストから特定の装置を除外します。特定のメディアがいっぱいになると、バックアップは呼び出し側に警告を戻します (一方、残りの装置を使用して処理を継続します)。その場合、この呼び出し側アクションを指定してバックアップを再び呼び出すことによって、警告が生成される原因となった装置を使用装置のリストから除外してください。

DB2BACKUP_PARM_CHK
バックアップを実行することなく、パラメーターの妥当性を検査します。このオプションは、呼び出しが戻った後でデータベース接続を終了しません。この呼び出しが正常に戻った後、ユーザーが SQLUB_CONTINUE で呼び出しを発行し、処置を進めることが期待されます。

DB2BACKUP_PARM_CHK_ONLY
バックアップを実行することなく、パラメーターの妥当性を検査します。この呼び出しが戻る前に、この呼び出しによって確立したデータベース接続は終了し、後続する呼び出しは必要なくなります。

iBufferSize
入力。バッファー・サイズを 4 KB の割り振り単位 (ページ) でバックアップします。最小値は 8 単位です。

iNumBuffers
入力。使用するバックアップ・バッファーの数を指定します。最小値は 2 です。最大値はメモリーによって制限されます。

iParallelism
入力。並列処理の度合い (バッファー・マニピュレーターの数) を指定します。最小値は 1 です。最大値は 1024 です。

iOptions
入力。バックアップ・プロパティーのビットマップ。オプションは組み合わされて、ビット単位 OR 演算子を使用して *iOptions* の値を生成します。有効な値 (db2ApiDf.h で定義) は、以下のとおりです。

DB2BACKUP_OFFLINE
オフラインで、データベースへの排他的接続が確立されます。

DB2BACKUP_ONLINE
オンラインで、バックアップ操作の実行中に他のアプリケーションがデータベースにアクセスできるようになります。

注: ユーザーが SMS LOB データに対するロックを保持している場合は、オンライン・バックアップ操作は停止しているようにみえる場合があります。

DB2BACKUP_DB
データベースの全バックアップ。

DB2BACKUP_TABLESPACE
表スペース・レベルのバックアップ。表スペース・レベルのバックアップの場合は、 *piTablespaceList* パラメーターに表スペースのリストを提供してください。

DB2BACKUP_INCREMENTAL
累積 (増分) バックアップ・イメージを指定します。増分バックアップ・イメージとは、最新の成功した全バックアップ操作以降に変更された、すべてのデータベース・データのコピーです。

DB2BACKUP_DELTA
非累積 (差分) バックアップ・イメージを指定します。差分バックアップ・イメージとは、最新の成功した任意のタイプのバックアップ操作以降に変更された、すべてのデータベース・データのコピーです。

DB2BACKUP_COMPRESS
バックアップを圧縮することを指定します。

DB2BACKUP_INCLUDE_COMPR_LIB
バックアップの圧縮に使用するライブラリーがバックアップ・イメージに含まれることを指定します。

DB2BACKUP_EXCLUDE_COMPR_LIB
バックアップの圧縮に使用するライブラリーがバックアップ・イメージに含まれないことを指定します。

DB2BACKUP_INCLUDE_LOGS
ログ・ファイルのうち、特定の整合ポイント・イン・タイムまでこのイメージをリストアおよびロールフォワードするために必要な範囲もバックアップ・イメージに含めることを指定します。オフライン・バックアップまたは複数パーティション・バックアップの場合、このオプションは無効です。

DB2BACKUP_EXCLUDE_LOGS
バックアップ・イメージにログ・ファイルをまったく含めないことを指定します。

注: オフライン・バックアップ操作の実行の場合、このオプションが指定されていてもいなくても、ログは除外されます。

iUtilImpactPriority
バックアップ時に使用される優先順位の値を指定します。優先順位の値は 0 から 100 までの範囲の任意の数であり、 0 が非スロットル、100 が優先順位最高を意味します。

piComprLibrary
入力。バックアップ・イメージの圧縮を実行するために使用する外部ライブ

ラリーの名前を示します。この名前は、サーバー上の 1 個のファイルを参照する完全修飾パスでなければなりません。値が NULL ポインターであるか、空ストリングへのポインターである場合、 DB2 は、圧縮のためにデフォルトのライブラリーを使用します。指定されたライブラリーが見つからない場合、バックアップは失敗します。

piComprLibraryLen
入力。 piComprLibrary で指定したライブラリー名の長さを示す 4 バイトの符号なし整数 (バイト単位) です。ライブラリー名が提供されていない場合は、ゼロに設定してください。

piComprOptions
入力。バイナリー・データのうち、圧縮ライブラリーの初期設定ルーチンに渡すブロックを記述します。 DB2 はこのストリングをクライアントからサーバーに直接渡すため、バイト反転やコード・ページ変換の問題がある場合は圧縮ライブラリーで処理する必要があります。データ・ブロックの最初の文字が '@' なら、データの残りの部分は、サーバー上に存在するファイルの名前を指定するものとして解釈されます。その場合 DB2 は、piComprOptions および iComprOptionsSize の内容をそのファイルの内容およびサイズで置き換え、そのようにして得られる新しい値を初期設定ルーチンに渡します。

iComprOptionsSize
入力。 piComprOptions として渡されるデータ・ブロックのサイズを表す 4 バイトの符号なし整数。 piComprOptions が NULL ポインターである場合に限り、iComprOptionsSize はゼロになります。

tablespaces
バックアップを取る表スペースのリストを指すポインター。 C の場合、リストは NULL で終了するストリングです。一般的には、*db2Char* 構造のリストです。

numTablespaces
tablespaces パラメーター内の項目数。

locations
メディア・ロケーションのリストを指すポインター。 C の場合、リストは NULL で終了するストリングです。一般的には、db2Char 構造のリストです。

numLocations
locations パラメーター内の項目数。

locationType
メディア・タイプを示す文字。有効な値 (`sqlutil.h` で定義) は、以下のとおりです。

SQLU_LOCAL_MEDIA
ローカル装置 (テープ、ディスク、ディスケット、または名前付きパイプ)

SQLU_TSM_MEDIA
Tivoli Storage Manager。

SQLU_OTHER_MEDIA
ベンダー・ライブラリー。

SQLU_USER_EXIT
ユーザー出口 (サーバーが OS/2 上にある場合のみ使用可能です)。

pioData
文字データ・バッファーを指すポインター。

iLength
入力。 *pioData* バッファーのサイズ。

oLength
出力。将来の利用のために予約されています。

関連資料:

- 390 ページの『sqlemgdb - データベースの移行』
- 257 ページの『db2Rollforward - データベースのロールフォワード』
- 453 ページの『SQLCA』
- 244 ページの『db2Restore - データベースのリストア』

関連サンプル:

- 『dbrecov.sqc -- How to recover a database (C)』
- 『dbrecov.sqC -- How to recover a database (C++)』

db2CfgGet - 構成パラメーターの入手

特定のデータベース構成ファイル、またはデータベース・マネージャー構成ファイルにある個々の項目の値を戻します。

有効範囲:

特定のデータベース構成ファイルに関する情報は、それが実行されるデータベース・パーティションに対してのみ戻されます。

許可:

なし

必要な接続:

特定のデータベース構成ファイルの構成パラメーターの現行オンライン値を入手するには、データベースへの接続が必要です。データベース・マネージャーの構成パラメーターの現行オンライン値を入手するには、インスタンスへのアタッチが必要です。それ以外の場合は、データベースまたはインスタンスへの接続は必要ありません。

API 組み込みファイル:

db2ApiDf.h

C API 構文:

```
/* File: db2ApiDf.h */
/* API: db2CfgGet */
/* ... */
SQL_API_RC SQL_API_FN
db2CfgGet (
  db2Uint32 versionNumber,
  void *pParmStruct,
  struct sqlca *pSqlca);

typedef SQL_STRUCTURE db2Cfg
{
   db2Uint32                            numItems;
   struct db2CfgParam                   *paramArray;
   db2Uint32                            flags;
   char                                 *dbname;
} db2Cfg;

typedef SQL_STRUCTURE db2CfgParam
{
   db2Uint32                            token;
   char                                 *ptrvalue;
   db2Uint32                            flags;
} db2CfgParam;
/* ... */
```

汎用 API 構文:

```
/* File: db2ApiDf.h */
/* API db2gCfgGet */
/* ... */
SQL_API_RC SQL_API_FN
db2gCfgGet (
  db2Uint32 versionNumber,
  void *pParmStruct,
  struct sqlca *pSqlca);

typedef SQL_STRUCTURE db2gCfg
{
   db2Uint32                            numItems;
   struct db2gCfgParam                  *paramArray;
   db2Uint32                            flags;
   db2Uint32                            dbname_len;
   char                                 *dbname;
} db2gCfg;

typedef SQL_STRUCTURE db2gCfgParam
{
   db2Uint32                            token;
   db2Uint32                            ptrvalue_len;
   char                                 *ptrvalue;
   db2Uint32                            flags;
} db2gCfgParam;
/* ... */
```

API パラメーター:

versionNumber
入力。 2 番目のパラメーター *pParmStruct* として渡される構造のバージョンとリリースのレベルを指定します。

pParmStruct
入力。 *db2Cfg* 構造を指すポインター。

pSqlca
出力。*sqlca* 構造へのポインター。

numItems
入力。 *paramArray* 配列内の構成パラメーターの数。この値を db2CfgMaxParam に設定し、*paramArray* 内のエレメントの最大数を指定します。

paramArray
入力。 *db2CfgParam* 構造を指すポインター。

flags (db2Cfg 構造)
入力。実行するアクションのタイプを指定します。有効な値 (db2ApiDf.h で定義) は、以下のとおりです。

db2CfgDatabase
データベース構成ファイルの値を戻すために指定します。

db2CfgDatabaseManager
データベース・マネージャー構成ファイルの値を戻すために指定します。

db2CfgImmediate
メモリーに保管された構成パラメーターの現行値を戻します。

db2CfgDelayed
ディスク上の構成パラメーターの値を入手します。次回のデータベースまたはインスタンス接続までは、これらの値はメモリー内の現行値にはなりません。

db2CfgGetDefaults
構成パラメーターのデフォルト値を戻します。

dbname_len
入力。 *dbname* の長さ (バイト単位)。

dbname
入力。データベース名。

token 入力。構成パラメーター ID。

db2CfgParam トークン・エレメントの有効な項目とデータ・タイプは、構成パラメーターのサマリーにリストされています。

ptrvalue_len
入力。 *ptrvalue* の長さ (バイト単位)。

ptrvalue
出力。構成パラメーターの値。

flags (db2CfgParam 構造)
入力。要求内の各パラメーターに関する特定の情報を提供します。有効な値 (db2ApiDf.h で定義) は、以下のとおりです。

db2CfgParamAutomatic
検索されるパラメーターの値が *automatic* かどうかを示します。特定の構成パラメーターが *automatic* に設定されているかどうかを判別するには、フラグによって戻される値および db2ApiDf.h で定義されている *db2CfgParamAutomatic* キーワードに対して、ブール AND 演算を実行します。

関連概念:

- 「*管理ガイド: パフォーマンス*」の『構成パラメーターの調整』

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』

関連資料:

- 453 ページの『SQLCA』
- 「*管理ガイド: パフォーマンス*」の『構成パラメーターのサマリー』
- 39 ページの『db2CfgSet - 構成パラメーターの設定』

関連サンプル:

- 『dbinfo.c -- Set and get information at the database level (C)』
- 『dbrecov.sqc -- How to recover a database (C)』
- 『inauth.sqc -- How to display authorities at instance level (C)』
- 『ininfo.c -- Set and get information at the instance level (C)』
- 『tscreate.sqc -- How to create and drop buffer pools and table spaces (C)』
- 『dbinfo.C -- Set and get information at the database level (C++)』
- 『dbrecov.sqC -- How to recover a database (C++)』
- 『inauth.sqC -- How to display authorities at instance level (C++)』
- 『ininfo.C -- Set and get information at the instance level (C++)』
- 『tscreate.sqC -- How to create and drop buffer pools and table spaces (C++)』

db2CfgSet - 構成パラメーターの設定

特定のデータベース構成ファイル、またはデータベース・マネージャー構成ファイルにある個々の項目を変更します。データベース構成ファイルは、データベースが作成されたすべてのノードに存在します。

有効範囲:

データベース構成ファイルの変更は、それが実行されるノードに影響を与えます。

許可:

データベース構成ファイルの変更の場合は、以下のいずれかです。

- *sysadm*
- *sysctrl*
- *sysmaint*

データベース・マネージャー構成ファイルの変更の場合は、以下のものです。

- *sysadm*

必要な接続:

特定のデータベースの構成パラメーターをオンラインで変更するには、データベースに接続する必要があります。データベース・マネージャーの構成パラメーターを

オンラインで変更するには、インスタンスにアタッチする必要があります。それ以外の場合は、データベースまたはインスタンスへの接続は必要ありません。

API 組み込みファイル:

db2ApiDf.h

C API 構文:

```
/* File: db2ApiDf.h */
/* API: db2CfgSet */
/* ... */
SQL_API_RC SQL_API_FN
db2CfgSet (
  db2Uint32 versionNumber,
  void *pParmStruct,
  struct sqlca *pSqlca);

typedef SQL_STRUCTURE db2Cfg
{
   db2Uint32                            numItems;
   struct db2CfgParam                   *paramArray;
   db2Uint32                            flags;
   char                                 *dbname;
} db2Cfg;

typedef SQL_STRUCTURE db2CfgParam
{
   db2Uint32                            token;
   char                                 *ptrvalue;
   db2Uint32                            flags;
} db2CfgParam;
/* ... */
```

汎用 API 構文:

```
/* File: db2ApiDf.h */
/* API db2gCfgGet */
/* ... */
SQL_API_RC SQL_API_FN
db2gCfgSet (
  db2Uint32 versionNumber,
  void *pParmStruct,
  struct sqlca *pSqlca);

typedef SQL_STRUCTURE db2gCfg
{
   db2Uint32                            numItems;
   struct db2gCfgParam                  *paramArray;
   db2Uint32                            flags;
   db2Uint32                            dbname_len;
   char                                 *dbname;
} db2gCfg;

typedef SQL_STRUCTURE db2gCfgParam
{
   db2Uint32                            token;
   db2Uint32                            ptrvalue_len;
   char                                 *ptrvalue;
   db2Uint32                            flags;
} db2gCfgParam;
/* ... */
```

API パラメーター:

versionNumber
入力。 2 番目のパラメーター *pParmStruct* として渡される構造のバージョンとリリースのレベルを指定します。

pParmStruct
入力。 *db2Cfg* 構造を指すポインター。

pSqlca
出力。*sqlca* 構造へのポインター。

numItems
入力。 *paramArray* 配列内の構成パラメーターの数。

paramArray
入力。 *db2CfgParam* 構造を指すポインター。

flags (db2Cfg 構造)
入力。実行するアクションのタイプを指定します。有効な値 (db2ApiDf.h で定義) は、以下のとおりです。

db2CfgDatabase
データベース構成ファイルの値を戻すために指定します。

db2CfgDatabaseManager
データベース・マネージャー構成ファイルの値を戻すために指定します。

db2CfgImmediate
メモリー内の構成パラメーターの現行値を設定します。

db2CfgDelayed
ディスクに構成パラメーターの値を設定します。次回のデータベースまたはインスタンス接続までは、これらの値はメモリー内の現行値にはなりません。

db2CfgReset
構成パラメーターをデフォルト値にリセットします。

dbname_len
入力。 *dbname* の長さ (バイト単位)。

dbname
入力。データベース名。

token 入力。構成パラメーター ID。

db2CfgParam トークン ・エレメントの有効な項目とデータ・タイプは、構成パラメーターのサマリー にリストされています。

ptrvalue_len
入力。 *ptrvalue* の長さ (バイト単位)。

ptrvalue
入力。構成パラメーターの値。

flags (db2CfgParam 構造)
入力。要求内の各パラメーターに対して取られるアクションのタイプを指定します。デフォルトでは、このフィールドはゼロに設定する必要があります。有効な値 (db2ApiDf.h で定義) は、以下のとおりです。

db2CfgParamAutomatic
構成パラメーター値を *automatic* に設定します。 DB2 はこのパラメーターをエスカレーションして、現行のリソース要件を反映させます。自動動作をサポートするパラメーターだけが、*automatic* に設定できます。

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』

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』

関連資料:

- 453 ページの『SQLCA』
- 「*管理ガイド: パフォーマンス*」の『構成パラメーターのサマリー』
- 36 ページの『db2CfgGet - 構成パラメーターの入手』

関連サンプル:

- 『dbinfo.c -- Set and get information at the database level (C)』
- 『dbrecov.sqc -- How to recover a database (C)』
- 『ininfo.c -- Set and get information at the instance level (C)』
- 『dbinfo.C -- Set and get information at the database level (C++)』
- 『dbrecov.sqC -- How to recover a database (C++)』
- 『ininfo.C -- Set and get information at the instance level (C++)』

db2ConvMonStream - モニター・ストリームの変換

単一の論理データ・エレメントに使用する新規の自己記述形式 (SQLM_ELM_DB2 など) を、対応するバージョン 6 以前の外部モニター構造 (sqlm_db2 など) に変換します。 API 呼び出しをアップグレードしてバージョン 5 以降のストリームを使用する場合は、新規のストリーム形式を使用してモニター・データで検索を行う必要があります (たとえば、 SQLM_ELM_DB2 を検索する必要があります)。その後、ストリームのこの部分が変換 API に渡され、関連するバージョン 6 以前のデータが取得されます。

許可:

なし

必要な接続:

なし

API 組み込みファイル:

db2ApiDf.h

C API 構文:

```
/* File: db2ApiDf.h */
/* API: db2ConvMonStream */
/* ... */
db2ConvMonStream (
  unsigned char version,
  db2ConvMonStreamData *data,
  struct sqlca *pSqlca);

typedef struct
{
  void *poTarget;
  sqlm_header_info *piSource;
  db2Uint32 iTargetType;
  db2Uint32 iTargetSize;
  db2Uint32 iSourceType
} db2ConvMonStreamData;
/* ... */
```

API パラメーター:

version
入力。 2 番目のパラメーター *data* として渡される構造のバージョンとリリースのレベルを指定します。

data 入力。 *db2ConvMonStreamData* 構造を指すポインター。

pSqlca
出力。*sqlca* 構造へのポインター。

poTarget
出力。ターゲット・モニター出力構造 (sqlm_db2 など) を指すポインター。以下に、出力タイプと対応する入力タイプのリストを示します。

piSource
入力。変換中の論理データ・エレメント (SQLM_ELM_DB2 など) を指すポインター。以下に、出力タイプと対応する入力タイプのリストを示します。

iTargetType
入力。実行中の変換のタイプを示します。インスタンス SQLM_DB2_SS については、 sqlmon.h で v5 タイプの値を指定してください。

iTargetSize
入力。通常このパラメーターは、*poTarget* が指す構造のサイズに設定できます。ただし、通常は構造の終わりからのオフセット値によって参照されるエレメント (*sqlm_stmt* のステートメント・テキストなど) の場合は、sqlm_stmt 統計サイズ・エレメント、および抽出する最大サイズのステートメントが入る十分な大きさ (つまり、 SQL_MAX_STMT_SIZ と sizeof(sqlm_stmt) の合計) のバッファーを指定してください。

iSourceType
入力。ソース・ストリームのタイプを示します。有効な値は、SQLM_STREAM_SNAPSHOT (スナップショット・ストリーム)、または SQLM_STREAM_EVMON (イベント・モニター・ストリーム) です。

使用上の注意:

以下に、サポートされている変換可能なデータ・エレメントのリストを記載します。

表 4. サポートされている変換可能なデータ・エレメント : スナップショット変数

スナップショット変数の データ・ストリーム・タイプ	構造体
SQLM_ELM_APPL	sqlm_appl
SQLM_ELM_APPL_INFO	sqlm_applinfo
SQLM_ELM_DB2	sqlm_db2
SQLM_ELM_FCM	sqlm_fcm
SQLM_ELM_FCM_NODE	sqlm_fcm_node
SQLM_ELM_DBASE	sqlm_dbase
SQLM_ELM_TABLE_LIST	sqlm_table_header
SQLM_ELM_TABLE	sqlm_table
SQLM_ELM_DB_LOCK_LIST	sqlm_dbase_lock
SQLM_ELM_APPL_LOCK_LIST	sqlm_appl_lock
SQLM_ELM_LOCK	sqlm_lock
SQLM_ELM_STMT	sqlm_stmt
SQLM_ELM_SUBSECTION	sqlm_subsection
SQLM_ELM_TABLESPACE_LIST	sqlm_tablespace_header
SQLM_ELM_TABLESPACE	sqlm_tablespace
SQLM_ELM_ROLLFORWARD	sqlm_rollfwd_info
SQLM_ELM_BUFFERPOOL	sqlm_bufferpool
SQLM_ELM_LOCK_WAIT	sqlm_lockwait
SQLM_ELM_DCS_APPL	sqlm_dcs_appl、 sqlm_dcs_applid_info、 sqlm_dcs_appl_snap_stats、 sqlm_xid、sqlm_tpmon
SQLM_ELM_DCS_DBASE	sqlm_dcs_dbase
SQLM_ELM_DCS_APPL_INFO	sqlm_dcs_applid_info
SQLM_ELM_DCS_STMT	sqlm_dcs_stmt
SQLM_ELM_COLLECTED	sqlm_collected

表 5. サポートされている変換可能なデータ・エレメント : イベント・モニター変数

イベント・モニター変数の データ・ストリーム・タイプ	構造体
SQLM_ELM_EVENT_DB	sqlm_db_event
SQLM_ELM_EVENT_CONN	sqlm_conn_event
SQLM_ELM_EVENT_TABLE	sqlm_table_event
SQLM_ELM_EVENT_STMT	sqlm_stmt_event
SQLM_ELM_EVENT_XACT	sqlm_xaction_event
SQLM_ELM_EVENT_DEADLOCK	sqlm_deadlock_event
SQLM_ELM_EVENT_DLCONN	sqlm_dlconn_event
SQLM_ELM_EVENT_TABLESPACE	sqlm_tablespace_event
SQLM_ELM_EVENT_DBHEADER	sqlm_dbheader_event
SQLM_ELM_EVENT_START	sqlm_evmon_start_event
SQLM_ELM_EVENT_CONNHEADER	sqlm_connheader_event

表 5. サポートされている変換可能なデータ・エレメント : イベント・モニター変数 (続き)

イベント・モニター変数の データ・ストリーム・タイプ	構造体
SQLM_ELM_EVENT_OVERFLOW	sqlm_overflow_event
SQLM_ELM_EVENT_BUFFERPOOL	sqlm_bufferpool_event
SQLM_ELM_EVENT_SUBSECTION	sqlm_subsection_event
SQLM_ELM_EVENT_LOG_HEADER	sqlm_event_log_header

sqlm_rollfwd_ts_info 構造は変換されません。この構造には、ストリームから直接アクセスできる表スペース名だけが入っています。 *sqlm_agent* 構造も変換されません。この構造には、ストリームから直接アクセス可能なエージェントの *pid* が入っています。

関連資料:

• 453 ページの『SQLCA』

db2DatabasePing - データベースの PING

クライアントとデータベース・サーバーと間の基礎接続に要するネットワーク応答時間をテストします。この API は、ホスト・データベース・サーバーに DB2 Connect を介して直接、またはゲートウェイを介して接続する場合に、アプリケーションによって使用できます。

許可:

なし

必要な接続:

データベース

API 組み込みファイル:

db2ApiDf.h

C API 構文:

```
/* File: db2ApiDf.h */
/* API: db2DatabasePing */
/* ... */
SQL_API_RC SQL_API_FN
db2DatabasePing (
  db2Uint32 versionNumber,
  void *pParmStruct,
  struct sqlca *pSqlca);

typedef SQL_STRUCTURE db2DatabasePingStruct
{
char          iDbAlias[SQL_ALIAS_SZ + 1];
db2int32      RequestPacketSz;
db2int32      ResponsePacketSz;
```

```
db2Uint16     iNumIterations;
db2Uint32    *poElapsedTime;
}
/* ... */
```

汎用 API 構文:

```
/* File: db2ApiDf.h */
/* API: db2gDatabasePing */
/* ... */
SQL_API_RC SQL_API_FN
db2gDatabasePing (
  db2Uint32 versionNumber,
  void *pParmStruct,
  struct sqlca *pSqlca);

typedef SQL_STRUCTURE db2gDatabasePingStruct
{
db2Uint16     iDbAliasLength;
char          iDbAlias[SQL_ALIAS_SZ];
db2int32      RequestPacketSz;
db2int32      ResponsePacketSz;
db2Uint16     iNumIterations;
db2Uint32    *poElapsedTime;
}
/* ... */
```

API パラメーター:

versionNumber
入力。アプリケーションが使用する DB2 Universal Database または DB2 Connect 製品のバージョンおよびリリースを指定します。

pParmStruct
入力。 *db2DatabasePingStruct* 構造を指すポインター。

pSqlca
出力。*sqlca* 構造へのポインター。

iDbAliasLength
入力。データベース別名の長さ。将来の利用のために予約されています。

iDbAlias
入力。データベース別名。将来の利用のために予約されています。

RequestPacketSz
入力。サーバーに送信されるパケットのサイズ (バイト)。サイズは 0 から 32767 の範囲でなければなりません。このパラメーターは、DB2 UDB for Linux、UNIX、および Windows バージョン 8 以降、または DB2 UDB for z/OS バージョン 8 以降を実行しているサーバーでのみ有効です。

ResponsePacketSz
入力。クライアントに戻されるパケットのサイズ (バイト)。サイズは 0 から 32767 の範囲でなければなりません。このパラメーターは、DB2 UDB for Linux、UNIX、および Windows バージョン 8 以降、または DB2 UDB for z/OS バージョン 8 以降を実行しているサーバーでのみ有効です。

iNumIterations
入力。テスト要求の反復回数。この値は 1 から 32767 の範囲でなければなりません。

poElapsedTime

出力。エレメントの数が iNumIterations と等しい 32 ビット整数の配列を指すポインター。配列内の各エレメントは、1 つのテスト要求反復に要する経過時間 (マイクロ秒単位) が入ります。

注: アプリケーションは、この API を呼び出す前に、配列に対してメモリーを割り振る責任があります。

使用上の注意:

この関数は PING コマンドを使用して呼び出すこともできます。

関連資料:

- 453 ページの『SQLCA』
- 「コマンド・リファレンス」の『PING コマンド』

db2DatabaseQuiesce - データベースの静止

すべてのユーザーを強制的にデータベースから切り離し、すべてのアクティブ・トランザクションを即時にロールバックし、データベースを静止モードにします。この API はデータベースへの排他的アクセスを提供します。この静止期間中、データベースでシステム管理が実行されます。システム管理の完了後、db2DatabaseUnquiesce API を使用して、データベースの静止を解除できます。db2DatabaseUnquiesce API を使用することによって、シャットダウンしたり別のデータベースを始動しなくても、データベースに接続することができます。

このモードでは、*QUIESCE CONNECT* 権限を持つグループまたはユーザーと、および *sysadm*、*sysmaint*、または *sysctrl* のみがデータベースおよびそのオブジェクトにアクセスできます。

許可:

以下のいずれかです。

- *sysadm*
- *dbadm*

必要な接続:

データベース

API 組み込みファイル:

db2ApiDf.h

C API 構文:

```
/* File: db2ApiDf.h */
/* API: db2DatabaseQuiesce */
/* ... */
SQL_API_RC SQL_API_FN
db2DatabaseQuiesce (
  db2Uint32 versionNumber,
  void *pParmStruct,
  struct sqlca *pSqlca);
```

```
typedef SQL_STRUCTURE db2DbQuiesceStruct
{
  char              *piDatabaseName;
  db2Uint32         iImmediate;
  db2Uint32         iForce;
  db2Uint32         iTimeout;
} db2DbQuiesceStruct;
/* ... */
```

汎用 API 構文:

```
/* File: db2ApiDf.h */
/* API: db2gDatabaseQuiesce */
/* ... */
SQL_API_RC SQL_API_FN
db2gDatabaseQuiesce (
  db2Uint32 versionNumber,
  void *pParmStruct,
  struct sqlca *pSqlca);

typedef SQL_STRUCTURE db2gDbQuiesceStruct
{
  db2Uint32         iDatabaseNameLen;
  char              *piDatabaseName;
  db2Uint32         iImmediate;
  db2Uint32         iForce;
  db2Uint32         iTimeout;
} db2gDbQuiesceStruct;
/* ... */
```

API パラメーター:

versionNumber
入力。 2 番目のパラメーター *pParmStruct* として渡される構造のバージョンとリリースのレベルを指定します。

pParmStruct
入力。 *db2DbQuiesceStruct* 構造を指すポインター。

pSqlca
出力。 *sqlca* 構造を指すポインター。

iDatabaseNameLen
入力。 *piDatabaseName* の長さ (バイト単位) を指定します。

piDatabaseName
入力。データベース名。

iImmediate
入力。将来の利用のために予約されています。

iForce 入力。将来の利用のために予約されています。

iTimeout
入力。アプリケーションが現在の作業単位をコミットするのを待機する時間 (分) を指定します。 *iTimeout* を指定しない場合、単一パーティション・データベース環境でのデフォルト値は 10 分です。パーティション・データベース環境では、データベース・マネージャー構成パラメータ *start_stop_timeout* によって指定された値が使用されます。

関連資料:

• 453 ページの『SQLCA』

• 49 ページの『db2DatabaseUnquiesce - データベースの静止解除』

db2DatabaseUnquiesce - データベースの静止解除

保守または他の理由で静止されていたデータベースへのアクセスをリストアします。シャットダウンおよびデータベースの再始動を行わずに、ユーザー・アクセスがリストアされます。

許可:

以下のいずれかです。

- *sysadm*
- *dbadm*

必要な接続:

データベース

API 組み込みファイル:

db2ApiDf.h

C API 構文:

```
/* File: db2ApiDf.h */
/* API: db2DatabaseUnquiesce */
/* ... */
SQL_API_RC SQL_API_FN
db2DatabaseUnquiesce (
  db2Uint32 versionNumber,
  void *pParmStruct,
  struct sqlca *pSqlca);

typedef SQL_STRUCTURE db2DbUnquiesceStruct
{
             char          *piDatabaseName;
} db2DbUnquiesceStruct;
/* ... */
```

汎用 API 構文:

```
/* File: db2ApiDf.h */
/* API: db2gDatabaseunquiesce */
/* ... */
SQL_API_RC SQL_API_FN
db2gDatabaseUnquiesce (
  db2Uint32 versionNumber,
  void *pParmStruct,
  struct sqlca *pSqlca);

typedef SQL_STRUCTURE db2gDbUnquiesceStruct
{
             db2Uint32     iDatabaseNameLen;
             char          *piDatabaseName;
} db2gDbUnquiesceStruct;
/* ... */
```

API パラメーター:

versionNumber
入力。 2 番目のパラメーター *pParmStruct* として渡される構造のバージョンとリリースのレベルを指定します。

pParmStruct
入力。 *db2DbUnquiesceStruct* 構造を指すポインター。

pSqlca
出力。 *sqlca* 構造を指すポインター。

iDatabaseNameLen
入力。 *piDatabaseName* の長さ (バイト単位) を指定します。

piDatabaseName
入力。データベース名。

関連資料:

- 453 ページの『SQLCA』
- 47 ページの『db2DatabaseQuiesce - データベースの静止』

db2DatabaseRestart - データベースの再始動

異常終了し、矛盾した状態のままであるデータベースを再始動します。この API の正常終了時に、アプリケーションは、ユーザーが CONNECT 特権を持っていれば、データベースに接続されたままになります。

有効範囲:

この API は、それが実行されるデータベース・パーティション・サーバーにのみ影響を与えます。

許可:

なし

必要な接続:

この API によってデータベース接続が確立されます。

API 組み込みファイル:

db2ApiDf.h

C API 構文:

```
/* File: db2ApiDf.h */
/* API: db2DatabaseRestart */
/* ... */
SQL_API_RC SQL_API_FN
db2DatabaseRestart (
  db2Uint32 versionNumber;
  void *pParamStruct;
  struct sqlca *pSqlca);

typedef struct
{
  char *piDatabaseName;
  char *piUserId;
  char *piPassword;
  char *piTablespaceNames;
```

```
  int  *iOption;

} db2RestartDbStruct;
/* ... */
```

汎用 API 構文:

```
/* File: db2ApiDf.h */
/* API: db2DatabaseRestart */
/* ... */
SQL_API_RC SQL_API_FN
  db2DatabaseRestart (
    db2Uint32 versionNumber;
    void *pParamStruct;
    struct sqlca *pSqlca);

typedef struct
{
  char *piDatabaseName;
  char *piUserId;
  char *piPassword;
  char *piTablespaceNames;
  int  *iOption;

} db2RestartDbStruct;
/* ... */
```

API パラメーター:

versionNumber
入力。 2 番目のパラメーター *pParamStruct* として渡される構造のバージョンとリリースのレベルを指定します。

pParamStruct
入力。 *db2RestartDbStruct* 構造を指すポインター。

pSqlca
出力。*sqlca* 構造へのポインター。

piDatabaseName
入力。再始動するデータベースの別名を含むストリングを指すポインター。

piUserId
入力。アプリケーションのユーザー名を含むストリングを指すポインター。 NULL にすることもできます。

piPassword
入力。指定したユーザー名 (ある場合) のパスワードを含むストリングを指すポインター。 NULL にすることもできます。

piTablespaceNames
入力。再始動操作時にドロップする表スペース名のリストを含むストリングを指すポインター。 NULL にすることもできます。

iOption
入力。有効な値は以下のとおりです。

DB2_DB_SUSPEND_NONE
通常のクラッシュ・リカバリーを実行します。

DB2_DB_RESUME_WRITE
I/O 書き込みが中断したデータベースのクラッシュ・リカバリーを実行する必要があります。

REXX API 構文:

```
RESTART DATABASE database_alias [USER username USING password]
```

REXX API パラメーター:

database_alias
再始動するデータベースの別名です。

username
データベースの再始動に使用されるユーザー名。

password
ユーザー名の認証に使用されるパスワード。

使用上の注意:

この API は、データベースへの接続を試みた際に、データベースの再始動が必要であることを示すエラー・メッセージが戻された場合に呼び出してください。このアクションは、このデータベースを用いた前のセッションが異常終了した (たとえば、電源障害により) 場合にのみ起こります。

この API の完了時に、ユーザーに CONNECT 特権があれば、データベースへの共用接続は保たれます。未確定トランザクションが存在する場合には、SQL 警告を受け取ることになります。この場合、データベースはまだ使用可能ですが、未確定トランザクションが、データベースへの最終接続をドロップするまでに解決されない場合には、この API への別の呼び出しを完了してから、再度データベースを使用しなければなりません。

循環ロギングの場合、入出力エラー、アンマウントされたファイル・システムなど、表スペースに問題があると、データベース再始動操作は失敗します。そのような表スペースが失われても問題がない場合は、それらの表スペースの名前を明示的に指定できます。このようにすると、表スペースがドロップ・ペンディング状態になり、再始動操作を正常に完了できます。

関連資料:

• 453 ページの『SQLCA』

関連サンプル:

- 『dbconn.sqc -- How to connect to and disconnect from a database (C)』
- 『dbconn.sqC -- How to connect to and disconnect from a database (C++)』

db2DbDirCloseScan - データベース・ディレクトリー・スキャンのクローズ

db2DbDirOpenScan によって割り振られたリソースを解放します。

許可:

なし

必要な接続:

なし

API 組み込みファイル:

db2ApiDf.h

C API 構文:

```
/* File: db2ApiDf.h */
/* API: db2DbDirCloseScan */
/* ... */
SQL_API_RC SQL_API_FN
db2DbDirCloseScan (
  db2Uint32 versionNumber,
  void *pParmStruct,
  struct sqlca *pSqlca);

typedef struct
{
 db2Uint16 iHandle;
} db2DbDirCloseScanStruct;
/* ... */
```

汎用 API 構文:

```
/* File: db2ApiDf.h */
/* API: db2gDbDirCloseScan */
/* ... */
SQL_API_RC SQL_API_FN
db2gDbDirCloseScan (
  db2Uint32 versionNumber,
  void *pParmStruct,
  struct sqlca *pSqlca);

typedef struct
{
 db2Uint16 iHandle;
} db2gDbDirCloseScanStruct;
/* ... */
```

API パラメーター:

versionNumber
入力。 2 番目のパラメーター *pParmStruct* として渡される構造のバージョンとリリースのレベルを指定します。

pParmStruct
入力。 *db2DbDirCloseScanStruct* 構造を指すポインター。

pSqlca
出力。*sqlca* 構造へのポインター。

iHandle
入力。関連する db2DbDirOpenScan API から戻される ID。

関連資料:

- 54 ページの『db2DbDirGetNextEntry - データベース・ディレクトリーの次項目の入手』
- 58 ページの『db2DbDirOpenScan - データベース・ディレクトリー・スキャンのオープン』
- 453 ページの『SQLCA』

関連サンプル:

- 『ininfo.c -- Set and get information at the instance level (C)』
- 『ininfo.C -- Set and get information at the instance level (C++)』

db2DbDirGetNextEntry - データベース・ディレクトリーの次項目の入手

db2DbDirOpenScan によって戻されたシステム・データベース・ディレクトリーまたはローカル・データベース・ディレクトリーのコピーの次項目を戻します。この API への以降の呼び出しは、追加の項目を戻します。

許可:

なし

必要な接続:

なし

API 組み込みファイル:

db2ApiDf.h

C API 構文:

```
/* File: db2ApiDf.h */
/* API: db2DbDirGetNextEntry */
/* ... */
SQL_API_RC SQL_API_FN
db2DbDirGetNextEntry (
  db2Uint32 versionNumber,
  void *pParmStruct,
  struct sqlca *pSqlca);

typedef struct
{
 db2Uint16 iHandle;
 struct db2DbDirInfo *poDbDirEntry;
} db2DbDirNextEntryStruct;

SQL_STRUCTURE db2DbDirInfo
{
   _SQLOLDCHAR               alias[SQL_ALIAS_SZ];
   _SQLOLDCHAR               dbname[SQL_DBNAME_SZ];
```

```
   _SQLOLDCHAR              drive[SQL_DRIVE_SZ];
   _SQLOLDCHAR              intname[SQL_INAME_SZ];
   _SQLOLDCHAR              nodename[SQL_NNAME_SZ];
   _SQLOLDCHAR              dbtype[SQL_DBTYP_SZ];
   _SQLOLDCHAR              comment[SQL_CMT_SZ];
   short                    com_codepage;
   _SQLOLDCHAR              type;
   unsigned short           authentication;
   char                     glbdbname[SQL_DIR_NAME_SZ];
   _SQLOLDCHAR              dceprincipal[SQL_DCEPRIN_SZ];
   short                    cat_nodenum;
   short                    nodenum;
   _SQLOLDCHAR              althostname[SQL_HOSTNAME_SZ];
   _SQLOLDCHAR              altportnumber[SQL_SERVICE_NAME_SZ];
};

/* ... */
```

汎用 API 構文:

```
/* File: db2ApiDf.h */
/* API: db2gDbDirGetNextEntry */
/* ... */
SQL_API_RC SQL_API_FN
db2gDbDirGetNextEntry (
  db2Uint32 versionNumber,
  void *pParmStruct,
  struct sqlca *pSqlca);

typedef struct
{
 db2Uint16 iHandle;
 struct db2DbDirInfo *poDbDirEntry;
} db2gDbDirNextEntryStruct;
};

/* ... */
```

API パラメーター:

versionNumber
入力。 2 番目のパラメーター *pParmStruct* として渡される構造のバージョンとリリースのレベルを指定します。

pParmStruct
入力。 *db2DbDirGetNextEntryStruct* 構造を指すポインター。

pSqlca
出力。*sqlca* 構造へのポインター。

iHandle
入力。関連する db2DbDirOpenScan API から戻される ID。

poDbDirEntry
出力。 db2DbDirInfo 構造を指すポインター。ディレクトリー・データのスペースが API によって割り振られます。また、そのスペースを指すポインターは呼び出し側に戻されます。

alias 代替データベース名。

dbname
データベースの名前。

drive データベースが存在する、ローカル・データベース・ディレクトリーのパス

名。このフィールドは、システム・データベース・ディレクトリーがスキャン用にオープンしている場合のみ戻されます。

注: Windows NT では、このパラメーターは CHAR(12) です。

intname
データベース・サブディレクトリーを識別するトークン。このフィールドは、ローカル・データベース・ディレクトリーがスキャン用にオープンしている場合のみ戻されます。

nodename
データベースが位置するノードの名前。このフィールドは、カタログ・データベースがリモート・データベースである場合のみ戻されます。

dbtype
データベース・マネージャーのリリース情報。

comment
データベースに関するコメント。

com_codepage
コメントのコード・ページ。使用されません。

type 項目タイプ。有効な値は以下のとおりです。

SQL_INDIRECT
(DB2INSTANCE 環境変数の値によって定義された) 現行のインスタンスにより作成されたデータベース。

SQL_REMOTE
別のインスタンスに存在するデータベース。

SQL_HOME
このボリュームに存在するデータベース (常に、ローカル・データベース・ディレクトリー内の HOME)。

SQL_DCE
DCE ディレクトリーに存在するデータベース。

authentication
認証タイプ。有効な値は以下のとおりです。

SQL_AUTHENTICATION_SERVER
ユーザー名とパスワードの認証は、サーバーで行われます。

SQL_AUTHENTICATION_CLIENT
ユーザー名とパスワードの認証は、クライアントで行われます。

SQL_AUTHENTICATION_DCS
DB2 Connect 用に使用します。

SQL_AUTHENTICATION_DCE
認証は、DCE セキュリティー・サービスを用いて行われます。

SQL_AUTHENTICATION_KERBEROS
認証は、kerberos セキュリティー・メカニズムを用いて行われます。

SQL_AUTHENTICATION_NOT_SPECIFIED
DB2 では、認証をデータベース・ディレクトリーに保持する必要がもはやありません。下位レベル (DB2 V2 またはそれ以下) のサーバー以外に接続するときは、この値を指定してください。

SQL_AUTHENTICATION_SVR_ENCRYPT
認証が宛先データベースを含むノードで行われることと、認証のパスワードが暗号化されることを指定します。

SQL_AUTHENTICATION_DATAENC
宛先データベースを含むノードで認証が行われること、および接続でデータ暗号化を使用する必要があることを指定します。

SQL_AUTHENTICATION_GSSPLUGIN
外部 GSS API ベースのプラグイン・セキュリティー機構を使って認証が行われることを指定します。

glbdbname
項目のタイプが SQL_DCE である場合、グローバル (DCE) ディレクトリーにあるターゲット・データベースのグローバル名。

dceprincipal
認証のタイプが DCE または KERBEROS である場合、プリンシパル名。

cat_nodenum
カタログ・ノード番号。

nodenum
ノード番号

althostname
フェイルオーバー時にデータベースの再接続先となる代替サーバーのホスト名または IP アドレス。

altportnumber
フェイルオーバー時にデータベースの再接続先となる代替サーバーのポート番号。

使用上の注意:

ディレクトリー項目情報バッファーのすべてのフィールドは、右方にブランクで埋め込まれます。

これ以降の db2DbDirGetNextEntry は、現行の項目に続く項目を入手します。

db2DbDirGetNextEntry が呼び出されるとき、スキャンする項目がもはや存在しないならば、 *sqlca* の *sqlcode* 値は 1014 に設定されます。

db2DbDirOpenScan API によって戻されたカウント値を使用して db2DbDirGetNextEntry を呼び出すことにより、スキャンの数と項目のカウントが等しくなるまで、ディレクトリー全体を一度ずつスキャンできます。

関連資料:

- 53 ページの『db2DbDirCloseScan - データベース・ディレクトリー・スキャンのクローズ』

- 58 ページの『db2DbDirOpenScan - データベース・ディレクトリー・スキャンのオープン』
- 453 ページの『SQLCA』

関連サンプル:

- 『ininfo.c -- Set and get information at the instance level (C)』
- 『ininfo.C -- Set and get information at the instance level (C++)』

db2DbDirOpenScan - データベース・ディレクトリー・スキャンのオープン

システム・データベース・ディレクトリーまたはローカル・データベース・ディレクトリーのコピーをメモリーに保管するとともに、項目の数を戻します。このコピーは、ディレクトリーがオープンするときの、ディレクトリーのスナップショットを表します。後になってディレクトリー自体に変更が加えられることがあっても、このコピーが更新されることはありません。

データベース・ディレクトリーの中でデータベース項目に関する情報を調べるには、 db2DbDirGetNextEntry を使用してます。スキャンをクローズするには、db2DbDirCloseScan を使用します。このことを行うと、ディレクトリーのコピーがメモリーから除去されます。

許可:

なし

必要な接続:

なし

API 組み込みファイル:

db2ApiDf.h

C API 構文:

```
/* File: db2ApiDf.h */
/* API: db2DbDirOpenScan */
/* ... */
SQL_API_RC SQL_API_FN
db2DbDirOpenScan (
  db2Uint32 versionNumber,
  void *pParmStruct,
    struct sqlca *pSqlca);

typedef struct
{
 char      *piPath;
 db2Uint16  oHandle;
 db2Uint16  oNumEntries;
} db2DbDirOpenScanStruct;
/* ... */
```

汎用 API 構文:

```
/* File: db2ApiDf.h */
/* API: db2gDbDirOpenScan */
/* ... */
SQL_API_RC SQL_API_FN
db2gDbDirOpenScan (
  db2Uint32 versionNumber,
  void *pParmStruct,
    struct sqlca *pSqlca);

typedef struct
{
 db2Uint32 *iPath_len;
 char      *piPath;
 db2Uint16  oHandle;
 db2Uint16  oNumEntries;
} db2gDbDirOpenScanStruct;
/* ... */
```

API パラメーター:

versionNumber
入力。 2 番目のパラメーター *pParmStruct* として渡される構造のバージョンとリリースのレベルを指定します。

pParmStruct
入力。 *db2DbDirOpenScanStruct* 構造を指すポインター。

pSqlca
出力。*sqlca* 構造へのポインター。

iPath_len
入力。 piPath の長さ (バイト単位)。

piPath
入力。ローカル・データベース・ディレクトリーが存在しているパスの名前を指定します。指定されたパスが NULL ポインターである場合、システム・データベース・ディレクトリーが使用されます。

oHandle
出力。戻された ID の 2 バイト域。このデータベース項目をスキャンするには、この ID を db2DbDirGetNextEntry に渡す必要があります。リソースを解放するには、この ID を db2DbDirCloseScan に渡す必要があります。

oNumEntries
出力。ディレクトリー項目の数が戻される 2 バイト域。

使用上の注意:

この API が割り振った記憶域は、db2DbDirCloseScan によって解放されます。

同一のディレクトリーに対して、複数の db2DbDirOpenScan API を発行できます。ただし、同じ結果になるとは限りません。次に走査をオープンするまでの間に、ディレクトリーが変更されている場合もあります。

プロセスごとに最大 8 つのデータベース・ディレクトリー・スキャンをオープンすることができます。

関連資料:

- 53 ページの『db2DbDirCloseScan - データベース・ディレクトリー・スキャンのクローズ』
- 54 ページの『db2DbDirGetNextEntry - データベース・ディレクトリーの次項目の入手』
- 453 ページの『SQLCA』

関連サンプル:

- 『ininfo.c -- Set and get information at the instance level (C)』
- 『ininfo.C -- Set and get information at the instance level (C++)』

db2DropContact - 連絡先のドロップ

連絡先のリストから、連絡先を消去します。連絡先は、通知メッセージが送信されるユーザーです。

許可:

なし

必要な接続:

なし

API 組み込みファイル:

db2ApiDf.h

C API 構文:

```
/* File: db2ApiDf.h */
/* API: db2DropContact */
/* ... */
SQL_API_RC SQL_API_FN
db2DropContact (
  db2Uint32 versionNumber,
  void *pParmStruct,
  struct sqlca *pSqlca);

typedef SQL_STRUCTURE db2DropContactData
{
   char *piUserid;
   char *piPassword;
   char *piName;
} db2DropContactData;
/* ... */
```

API パラメーター:

versionNumber
入力。 2 番目のパラメーター *pParmStruct* として渡される構造のバージョンとリリースのレベルを指定します。

pParmStruct
入力。 *db2DropContactData* 構造を指すポインター。

pSqlca
出力。*sqlca* 構造へのポインター。

piUserid;
入力。ユーザー名。

piPassword
入力。 *piUserid* 用のパスワード。

piName
入力。ドロップされる連絡先の名前。

関連資料:

- 453 ページの『SQLCA』
- 「*管理ガイド: パフォーマンス*」の『contact_host - 「連絡先リストのロケーション」構成パラメーター』
- 17 ページの『db2AddContact - 連絡先の追加』
- 83 ページの『db2GetContacts - 連絡先の入手』
- 288 ページの『db2UpdateContact - 連絡先の更新』

db2DropContactGroup - 連絡先グループのドロップ

連絡先のリストから、連絡先グループを除去します。連絡先グループには、通知メッセージが送信されるユーザーのリストが入っています。

許可:

なし

必要な接続:

なし

API 組み込みファイル:

db2ApiDf.h

C API 構文:

```
/* File: db2ApiDf.h */
/* API: db2DropContactGroup */
/* ... */
SQL_API_RC SQL_API_FN
db2DropContactGroup (
  db2Uint32 versionNumber,
  void *pParmStruct,
  struct sqlca *pSqlca);

typedef SQL_STRUCTURE db2DropContactData
{
   char *piUserid;
   char *piPassword;
   char *piName;
} db2DropContactData;
/* .. */
```

API パラメーター:

versionNumber
入力。 2 番目のパラメーター *pParmStruct* として渡される構造のバージョンとリリースのレベルを指定します。

pParmStruct
入力。 *db2DropContactData* 構造を指すポインター。

pSqlca
出力。*sqlca* 構造へのポインター。

piUserid
入力。ユーザー名。

piPassword
入力。 *piUserid* 用のパスワード。

piName
入力。ドロップされる連絡先グループの名前。

関連資料:

- 453 ページの『SQLCA』
- 「*管理ガイド: パフォーマンス*」の『contact_host - 「連絡先リストのロケーション」構成パラメーター』
- 18 ページの『db2AddContactGroup - 連絡先グループの追加』
- 80 ページの『db2GetContactGroup - 連絡先グループの入手』
- 81 ページの『db2GetContactGroups - 連絡先グループの入手』
- 289 ページの『db2UpdateContactGroup - 連絡先グループの更新』

db2Export - エクスポート

データベースから、いくつかある外部ファイル形式のいずれかにデータをエクスポートします。ユーザーは、SQL SELECT ステートメントによって、またはタイプ表の階層情報によってエクスポートするデータを指定します。

許可:

以下のいずれかです。

- *sysadm*
- *dbadm*

または、関係するそれぞれの表またはビューに対する CONTROL または SELECT 特権

必要な接続:

データベース。暗黙的な接続が可能である場合には、デフォルトのデータベースへの接続が確立されます。

API 組み込みファイル:

db2ApiDf.h

C API 構文:

```
/* File: db2ApiDf.h */
/* API: db2Export */
/* ... */

SQL_API_RC SQL_API_FN
  db2Export (
    db2Uint32 versionNumber,
  void * pParmStruct,
  struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2ExportStruct
{
   char                                 *piDataFileName;
   struct sqlu_media_list               *piLobPathList;
   struct sqlu_media_list               *piLobFileList;
   struct sqldcol                       *piDataDescriptor;
   struct sqllob                        *piActionString;
   char                                 *piFileType;
   struct sqlchar                       *piFileTypeMod;
   char                                 *piMsgFileName;
   db2int16                             iCallerAction;
   struct db2ExportOut                  *poExportInfoOut;
} db2ExportStruct;

typedef SQL_STRUCTURE db2ExportOut
{
   db2Uint64                            oRowsExported;
} db2ExportOut;
/* ... */
```

汎用 API 構文:

```
/* File: db2ApiDf.h */
/* API: db2gExport */
/* ... */

SQL_API_RC SQL_API_FN
  db2gExport (
    db2Uint32 versionNumber,
  void * pParmStruct,
  struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2gExportStruct
{
   char                                 *piDataFileName;
   struct sqlu_media_list               *piLobPathList;
   struct sqlu_media_list               *piLobFileList;
   struct sqldcol                       *piDataDescriptor;
   struct sqllob                        *piActionString;
   char                                 *piFileType;
   struct sqlchar                       *piFileTypeMod;
   char                                 *piMsgFileName;
   db2int16                             iCallerAction;
   struct db2ExportOut                  *poExportInfoOut;
   db2Uint16                            iDataFileNameLen;
   db2Uint16                            iFileTypeLen;
   db2Uint16                            iMsgFileNameLen;
} db2gExportStruct;
/* ... */
```

API パラメーター:

versionNumber
入力。 2 番目のパラメーター *pParmStruct* として渡される構造のバージョンとリリースのレベルを指定します。

pParmStruct
入力。 *db2ExportStruct* 構造を指すポインター。

pSqlca
出力。*sqlca* 構造へのポインター。

iDataFileNameLen
入力。データ・ファイル名の長さを示す 2 バイトの符号なし整数 (バイト単位) です。

iFileTypeLen
入力。ファイル・タイプの長さを示す 2 バイトの符号なし整数 (バイト単位) です。

iMsgFileNameLen
入力。メッセージ・ファイル名の長さを示す 2 バイトの符号なし整数 (バイト単位) です。

piDataFileName
入力。データがエクスポートされるパスおよび外部ファイル名を含むストリングを指定します。

piLobPathList
入力。 *media_type* SQLU_LOCAL_MEDIA を使用する *sqlu_media_list*、および LOB ファイルが保管されるクライアント上のパスをリストする *sqlu_media_entry* 構造。

このリスト内の最初のパス上でファイル・スペースが使い尽くされると、API は 2 番目のパスを使用し、以下同様に続きます。

piLobFileList
入力。 *media_type* SQLU_CLIENT_LOCATION を使用する *sqlu_media_list*、およびベース・ファイル名を含む *sqlu_location_entry* 構造です。

このリスト内の最初の名前を使用している名前スペースが使い尽くされると、 API は 2 番目の名前を使用し、以下同様に続きます。

エクスポート操作中に LOB ファイルが作成されるときには、(*pLobFilePath* からの) 現行パスにこのリストからの現行基本名を追加し、その後に 3 桁のシーケンス番号を追加した形のファイル名が構成されます。たとえば、現行の LOB パスが /u/foo/lob/path ディレクトリーで、現行の LOB ファイル名が bar の場合、作成される LOB ファイルの名前は、 /u/foo/lob/path/bar.001、/u/foo/lob/path/bar.002 (以下同様) となります。

piDataDescriptor
入力。出力ファイルの列名を指定する *sqldcol* 構造を指すポインター。*dcolmeth* フィールドの値によって、このパラメーターに提供される残りの情報をエクスポート・ユーティリティーがどのように解釈するかが判別されます。このパラメーターに有効な値 (sqlutil で定義) は、以下のとおりです。

SQL_METH_N
名前。出力ファイルで使用する列名を指定します。

SQL_METH_D
デフォルト。表の既存の列の名前が、出力ファイルで使用されます。この場合、列数および列指定配列は、どちらも無視されます。列名は、*pActionString* で指定された SELECT ステートメントの出力から派生します。

piActionString
入力。有効な動的 SQL SELECT ステートメントを含む *sqllob* 構造を指すポインター。この構造には、4 バイトの長さフィールドと、 SELECT ステートメントを構成する文字が順に含まれます。 SELECT ステートメントは、データベースからデータを取り出し、外部ファイルに書き込むことを指定します。

(*piDataDescriptor* からの) 外部ファイルの列と、 SELECT ステートメントからのデータベース列とは、それぞれのリストまたは構造における位置に従って対応付けられます。データベースから選択されたデータの最初の列は、外部ファイルの最初の列に置かれ、その列名は外部列配列の最初のエレメントから取られます。

piFileType
入力。外部ファイル内のデータの形式を示すストリングを指定します。サポートされている外部ファイルの形式 (`sqlutil` で定義) は、以下のとおりです。

SQL_DEL
区切り文字付き ASCII。これは dBase プログラム、BASIC プログラム、 IBM パーソナル・デシジョン・シリーズ・プログラム、およびその他の多数のデータベース・マネージャー/ファイル・マネージャーとの交換のための形式です。

SQL_WSF
ワークシート形式。 Lotus Symphony および 1-2-3 プログラムとの交換のための形式です。

SQL_IXF
IXF (統合交換フォーマット、PC バージョン)。表からデータをエクスポートする場合の推奨方式です。このファイル形式にエクスポートされたデータは、後で同じ表または別のデータベース・マネージャー表にインポートまたはロードできます。

piFileTypeMod
入力。 2 バイトの長さフィールドと、 1 つまたは複数の処理オプションを指定する文字の配列を含む *sqldcol* 構造を指すポインターです。このポインターが NULL であるか、このポインターが指す構造に 1 文字も入っていない場合、このアクションはデフォルトの指定が選択されたものとして解釈されます。

サポートされるすべてのファイル・タイプに、すべてのオプションを使用できるわけではありません。エクスポート用のファイル・タイプ修飾子を参照してください。

piMsgFileName
入力。このユーティリティーが戻すエラー、警告、および情報メッセージの宛先を含むストリングを指定します。オペレーティング・システム・ファイルまたは標準装置のパスおよび名前を指定できます。ファイルがすでに存在する場合は上書きされます。存在していない場合は、新たに作成されます。

iCallerAction
入力。呼び出し側が要求するアクションを示します。有効な値 (sqlutil で定義) は、以下のとおりです。

SQLU_INITIAL
最初の呼び出し。この値は、API への最初の呼び出しの際には必ず使用してください。

最初の呼び出しまたは後続の呼び出しのいずれかが戻され、要求されたエクスポート操作が完了する前に呼び出し側のアプリケーションが何らかのアクションを行うことが必要な場合、呼び出し側のアクションを以下のどちらかに設定しなければなりません。

SQLU_CONTINUE
処理の継続。この値を使用できるのは、最初の呼び出しが戻されたときにユーティリティーがユーザー入力 (たとえば、テープの終わり条件への応答) を要求した後で、 API への後続呼び出しを出す場合だけです。この値は、ユーティリティーが要求したユーザー・アクションが完了したら、ユーティリティーが最初の要求の処理を続行するよう指定するものです。

SQLU_TERMINATE
処理の終了。この値を使用できるのは、最初の呼び出しが戻されたときにユーティリティーがユーザー入力 (たとえば、テープの終わり条件への応答) を要求した後で、 API への後続呼び出しを出す場合だけです。この値は、ユーティリティーが要求したユーザー・アクションが実行されなかった場合、ユーティリティーが最初の要求の処理を中断するよう指定するものです。

poExportInfoOut
db2ExportOut 構造を指すポインター。

oRowsExported
出力。ターゲット・ファイルにエクスポートされたレコードの数を戻します。

REXX API 構文:

```
EXPORT :stmt TO datafile OF filetype
[MODIFIED BY :filetmod] [USING :dcoldata]
MESSAGES msgfile [ROWS EXPORTED :number]

CONTINUE EXPORT

STOP EXPORT
```

REXX API パラメーター:

stmt 有効な動的 SQL SELECT ステートメントを含む REXX ホスト変数。このステートメントにより、データベースから取り出すデータが指定されます。

datafile
データのエクスポート先となるファイルの名前。

filetype
エクスポート・ファイルのデータの形式。サポートされているファイル形式は、以下のとおりです。

DEL 区切り文字付き ASCII

WSF ワークシート形式

IXF 統合交換フォーマットの PC バージョン

filetmod
追加の処理オプションを含むホスト変数。

dcoldata
エクスポート・ファイルで使用する列名を含むコンパウンド REXX ホスト変数。以下の項目において、XXX はホスト変数の名前を表しています。

XXX.0 列数 (残りの変数内のエレメントの数)

XXX.1 最初の列名。

XXX.2 2 番目の列名。

XXX.3 以降、3 番目、4 番目 ... と続きます。

このパラメーターが NULL の場合、または *dcoldata* に値が指定されていない場合、ユーティリティーはデータベース表からの列名を使用します。

msgfile
エラーおよび警告メッセージが送られるファイル、パス、または装置の名前。

number
エクスポートされた行の数が入れられるホスト変数。

使用上の注意:

エクスポート操作を開始する前に、すべての表操作を完了し、すべてのロックを解除するようにしてください。このことは、 WITH HOLD でオープンしているカーソルをすべてクローズした後に COMMIT を発行するか、または ROLLBACK を発行することにより、行うことができます。

SELECT ステートメントでは表の別名を使用できます。

メッセージ・ファイルに置かれたメッセージには、メッセージ検索サービスから戻される情報が含まれています。各メッセージは新しい行から始まります。

DEL 形式ファイルへエクスポートするために 254 を超える長さの文字列を選択すると常に、エクスポート・ユーティリティーによって警告メッセージが出されます。

外部列名配列 *piDataDescriptor* の列数 (*dcolnum*) が SELECT ステートメントによって生成される列数と同じでない場合には、警告メッセージが出されます。この場

合、外部ファイルに書き込まれる列数はそれらのうち小さい方の数になります。出力ファイルを生成するために、余分のデータベース列または外部列名が使用されることはありません。

db2uexpm.bnd モジュールまたは配布された他の .bnd ファイルを手動でバインドする場合には、バインド・プログラムの**フォーマット**・オプションを使用しないでください。

PC/IXF インポートは、データベース間でデータを移動する場合に使用します。行区切り文字を含む文字データが区切り文字付き ASCII (DEL) ファイルにエクスポートされ、テキスト転送プログラムによって処理される場合、行区切り文字を含むフィールドは長さが変わることがあります。

DB2 Connect は、DB2 for z/OS and OS/390、DB2 for VM and VSE、および DB2 for iSeries などの DRDA サーバーから表をエクスポートするために使用できます。PC/IXF エクスポートだけがサポートされています。

エクスポート・ユーティリティーは、 AIX システムから呼び出されたときには複数部から成る PC/IXF ファイルを作成しません。

表の索引定義が PC/IXF ファイルに組み込まれるのは、単一のデータベース表の内容が、 SELECT * FROM tablename で始まる *pActionString* を指定して PC/IXF ファイルにエクスポートされ、 *piDataDescriptor* パラメーターにデフォルト名が指定されているときです。索引はビューについては保管されません。 *piActionString* の SELECT 文節が結合を含む場合も同様です。 *piActionString* の WHERE 文節、GROUP BY 文節、または HAVING 文節は索引の保管を妨げません。どの場合も、型付き表からのエクスポート時に、階層全体をエクスポートする必要があります。

提供された SELECT ステートメントが SELECT * FROM tablename の形式である場合には、エクスポート・ユーティリティーにより表の NOT NULL WITH DEFAULT 属性が IXF ファイルに保管されます。

型付き表をエクスポートする場合、副選択ステートメントを表すことができるのは、ターゲット表の名前と WHERE 文節だけです。階層をエクスポートする場合、全選択と *select-statement* は指定できません。

IXF 以外のファイル形式の場合は、階層の全探索の方法、およびエクスポートする副表とが DB2 に知らされるよう、全探索順序リストを指定することをお勧めします。このリストが指定されていないと、階層のすべての表がエクスポートされ、OUTER 順序がデフォルトの順序になります。 OUTER 関数によって指定されるデフォルトの順序を使うこともできます。

注: インポート操作時には、同じ全探索順序を使用してください。ロード・ユーティリティーでは、階層または副階層のロードはサポートされていません。

DB2 Data Links Manager についての考慮事項:

エクスポート時に、整合のとれた表のコピーと、 DATALINK 列によって参照される対応するファイルが確実にコピーされるようにするには、以下のようにします。

1. QUIESCE TABLESPACES FOR TABLE tablename SHARE コマンドを発行する。

これにより、EXPORT の実行時に更新トランザクションが進行しなくなります。

2. EXPORT コマンドを発行する。
3. それぞれの データ・リンク・サーバーで、**dlfm_export** ユーティリティーを実行する。 **dlfm_export** ユーティリティーへの入力は制御ファイル名です。これは、エクスポート・ユーティリティーによって生成されます。このようにすると、制御ファイル内でリストされるファイルの tar (または同等の) アーカイブが生成されます。 **dlfm_export** は、アーカイブされるファイルの ACL 情報をキャプチャーしません。
4. QUIESCE TABLESPACES FOR TABLE tablename RESET コマンドを発行する。

   この操作により表が更新可能になります。

EXPORT は、SQL アプリケーションとして実行されます。 SELECT ステートメント条件を満たす行と列が、データベースから抽出されます。 DATALINK 列の場合、 SELECT ステートメントではスカラー関数を指定しないようにしてください。

EXPORT が正常に実行されると、以下のファイルが生成されます。

- EXPORT コマンドで指定されたエクスポート・データ・ファイル。このファイルの DATALINK 列の値は、 IMPORT および LOAD ユーティリティーが使用するのと同じ形式です。 DATALINK 列の値が SQL NULL 値の場合、処理は他のデータ型の場合と同様になります。
- 制御ファイル *server_name* は、各 データ・リンク・サーバーに対して生成されます。 Windows NT オペレーティング・システムでは、単一の制御ファイル `ctrlfile.lst` がすべての データ・リンク・サーバーによって使用されます。これらの制御ファイルは、 <data-file path>/dlfm/YYYYMMDD/HHMMSS ディレクトリーに置かれます (Windows NT オペレーティング・システムでは、`ctrlfile.lst` は <data-file path>¥dlfm¥YYYYMMDD¥HHMMSS ディレクトリーに置かれます)。 YYYYMMDD は日付 (年月日) を表し、HHMMSS は時刻 (時分秒) を表します。

データ・リンク・サーバーからファイルをエクスポートするため、 **dlfm_export** ユーティリティーが提供されています。このユーティリティーが生成するアーカイブ・ファイルを使用して、ターゲットの データ・リンク・サーバーにファイルをリストアすることができます。

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『エクスポートを使用した DB2 Data Links Manager データの移動 - 概念』

関連資料:

- 453 ページの『SQLCA』
- 454 ページの『SQLCHAR』
- 456 ページの『SQLDCOL』
- 495 ページの『SQLU-MEDIA-LIST』
- 70 ページの『エクスポートのファイル・タイプ修飾子』
- 205 ページの『データを移動する際の区切り文字の制約事項』

関連サンプル:

- 『expsamp.sqb -- Export and import tables with table data to a DRDA database (IBM COBOL)』
- 『impexp.sqb -- Export and import tables with table data (IBM COBOL)』
- 『tload.sqb -- How to export and load table data (IBM COBOL)』
- 『tbmove.sqc -- How to move table data (C)』
- 『tbmove.sqC -- How to move table data (C++)』

エクスポートのファイル・タイプ修飾子

表 6. エクスポート用の有効なファイル・タイプ修飾子: すべてのファイル・フォーマット

修飾子	説明
lobsinfile	*lob-path* には、LOB データを含むファイルへのパスを指定します。 各パスには少なくとも 1 つのファイルが含まれており、そのファイルには、データ・ファイル内の LOB ロケーション指定子 (LLS) によって指される少なくとも 1 つの LOB が入っています。 LLS は、LOB ファイル・パスに保管されるファイル内の LOB のロケーションのストリング表現です。 LLS の形式は、*filename.ext.nnn.mmm/* です。 *filename.ext* は LOB を含むファイルの名前、 *nnn* はファイル内の LOB のオフセット (バイト単位)、 *mmm* は LOB の長さ (バイト単位) を表します。たとえば、ストリング `db2exp.001.123.456/` がデータ・ファイルに保存される場合、 LOB はファイル db2exp.001 のオフセット 123 に位置し、456 バイト長です。 EXPORT の使用時に ″lobsinfile″ 修飾子を指定した場合、LOB データは LOBS TO 文節に指定されたロケーションに置かれます。指定しない場合、 LOB データは現行作業ディレクトリーに送られます。 LOBS TO 文節は、LOB ファイルが保管されるディレクトリーに、1 つまたは複数のパスを指定します。LOB パスごとに少なくとも 1 つのファイルが存在し、各ファイルには少なくとも 1 つの LOB が含まれます。 LOB が NULL であることを示すには、サイズを -1 と入力します。サイズが 0 と指定されている場合、LOB は長さ 0 と見なされます。長さが -1 の NULL LOB の場合、オフセットとファイル名は無視されます。たとえば、NULL LOB の LLS は db2exp.001.7.-1/ です。

表 7. エクスポート用の有効なファイル・タイプ修飾子: DEL (区切り文字付き ASCII) ファイル・フォーマット

修飾子	説明

chardel*x*	*x* は単一文字のストリング区切り文字です。デフォルトは二重引用符 (") です。指定した文字は、文字ストリングを囲むために、二重引用符の代わりに使用されます。 [2] 文字ストリング区切り文字として明示的に二重引用符を指定したい場合、次のように指定します。
	
	`modified by chardel""`
	
	単一引用符 (') も、以下のように、文字ストリングの区切り文字として指定できます。
	
	`modified by chardel''`

表 7. エクスポート用の有効なファイル・タイプ修飾子: DEL (区切り文字付き ASCII) ファイル・フォーマット (続き)

修飾子	説明
codepage=*x*	*x* は ASCII 文字ストリングです。この値は、出力データ・セット内のデータのコード・ページとして解釈されます。エクスポート操作時に、文字データをアプリケーションのコード・ページからこのコード・ページに変換します。 純 DBCS (GRAPHIC)、混合 DBCS、および EUC の場合、区切り文字の範囲は x00からx3F に制限されます。 **注:** `codepage` 修飾子は、 `lobsinfile` 修飾子とともに使用することはできません。

coldel*x*	*x* は単一文字の列区切り文字です。デフォルトはコンマ (,) です。指定した文字は、列の終わりを表すために、コンマの代わりに使用されます。 [2]
	
	以下の例では、`coldel;` が指定されており、エクスポート・ユーティリティーは検出するすべてのセミコロン (;) を列の区切り文字として解釈します。
	`db2 "export to temp of del modified by coldel;`
	`select * from staff where dept = 20"`

datesiso	日付形式。すべての日付データ値を ISO 形式 (*"YYYY -MM -DD "*) でエクスポートします。 [3]
decplusblank	正符号文字。正の 10 進値の先頭に正符号 (+) ではなく、ブランク・スペースが置かれます。デフォルトのアクションでは、正の 10 進数の前に正符号 (+) が付けられます。
decpt*x*	*x* は、小数点としてピリオドと置換される単一文字です。デフォルト値はピリオド (.) です。指定した文字は、小数点文字としてピリオドの代わりに使用されます。 [2]
dldel*x*	*x* は単一文字の DATALINK 区切り文字です。デフォルト値はセミコロン (;) です。指定した文字は、 DATALINK 値のフィールド間区切り文字としてセミコロンの代わりに使用されます。 DATALINK 値には 2 つ以上の副値を指定できるため、この区切り文字が必要です。 [2] **注:** *x* は、行、列、または文字ストリングの区切り文字とは異なる文字にしてください。
nochardel	列データは、区切り文字で囲まれません。データを DB2 を使用してインポートまたはロードするつもりの場合は、このオプションを指定しないでください。これは、区切り文字を持たないベンダー・データ・ファイルをサポートするために提供されています。不適切に使用すると、データが損失または破壊される場合があります。 このオプションを `chardelx` または `nodoubledel` と一緒に指定することはできません。これらのオプションは、相互に排他的です。
nodoubledel	二重になっている区切り文字 [2] の認識を抑止します。

表 7. エクスポート用の有効なファイル・タイプ修飾子: DEL (区切り文字付き ASCII) ファイル・フォーマット (続き)

修飾子	説明

striplzeros	エクスポートされるすべての 10 進列から先行ゼロを除去します。
	
	以下の例を考慮してください。
	
	`db2 create table decimalTable ( c1 decimal( 31, 2 ) )`
	`db2 insert into decimalTable values ( 1.1 )`
	
	`db2 export to data of del select * from decimalTable`
	
	`db2 export to data of del modified by STRIPLZEROS`
	`select * from decimalTable`
	
	最初のエクスポート操作では、エクスポート・ファイル・データの内容が +0000000000000000000000000000001.10 になります。 2 番目の操作は、striplzeros 修飾子を除いて最初の操作と同じですが、エクスポート・ファイル・データの内容が +1.10 になります。

表 7. エクスポート用の有効なファイル・タイプ修飾子: DEL (区切り文字付き ASCII) ファイル・フォーマット (続き)

修飾子	説明
timestampformat="x"	x はソース・ファイルのタイム・スタンプの形式です。 [4] 有効なタイム・スタンプ・エレメントは以下のとおりです。 YYYY - 年 (0000 から 9999 の範囲の 4 桁の数字) M - 月 (1 から 12 の範囲の 1 桁または 2 桁の数) MM - 月 (01 から 12 の 2 桁の数。 M および MMM とは相互に排他的) MMM - 月 (月を表す 3 文字の大文字小文字を区別しない略語; M および MM とは相互に排他的) D - 日 (1 から 31 の範囲の 1 桁または 2 桁の数) DD - 日 (1 から 31 の範囲の 1 桁または 2 桁の数; D と相互排他的) DDD - 元日から数えた日数 (001 から 366 の 3 桁; 他の日または月エレメントと相互に排他) H - 時 (12 時間制の場合は 0 から 12、 24 時間制では 0 から 24 の 1 桁または 2 桁。) HH - 時 (12 時間制の場合は 0 から 12、 24 時間制では 0 から 24 の 1 桁または 2 桁: H と相互に排他) M - 分 (0 から 59 の 1 桁または 2 桁) MM - 分 (0 から 59 の範囲の 2 桁の数。 M (分) とは相互排他的) S - 秒 (0 から 59 の 1 桁または 2 桁) SS - 秒 (0 から 59 の範囲の 2 桁の数。 S とは相互排他的) SSSSS - 夜の 12 時から数えた秒数 (00000 から 86399 の 5 桁; 他の時刻エレメントとは相互排他的) UUUUUU - マイクロ秒 (000000 から 999999 の 6 桁; 他のすべてのマイクロ秒エレメントとは相互排他的) UUUUU - マイクロ秒 (00000 から 99999 の 5 桁、 000000 から 999990 の範囲にマップする; 他のすべてのマイクロ秒エレメントとは相互排他的) UUUU - マイクロ秒 (0000 から 9999 の 4 桁、 000000 から 999900 の範囲にマップする; 他のすべてのマイクロ秒エレメントとは相互排他的) UUU - マイクロ秒 (000 から 999 の 3 桁、 000000 から 999000 の範囲にマップする; 他のすべてのマイクロ秒エレメントとは相互排他的) UU - マイクロ秒 (00 から 99 の 2 桁、 000000 から 990000 の範囲にマップする; 他のすべてのマイクロ秒エレメントとは相互排他的) U - マイクロ秒 (0 から 9 の 2 桁、 000000 から 900000 の範囲にマップする; 他のすべてのマイクロ秒エレメントとは相互排他的) TT - 正午の標識 (AM または PM) タイム・スタンプ形式の例を以下に示します。 "YYYY/MM/DD HH:MM:SS.UUUUUU" MMM エレメントは、以下の値を生成します。「Jan」、「Feb」、「Mar」、「Apr」、「May」、「Jun」、「Jul」、「Aug」、「Sep」、「Oct」、「Nov」、および「 Dec」。「Jan」は 1 月と等しく、「Dec」は 12 月と等しいです。 以下の例は、「schedule」と呼ばれる表から、ユーザー定義のタイム・スタンプ形式を含むデータをエクスポートする方法を示しています。 db2 export to delfile2 of del modified by timestampformat="yyyy.mm.dd hh:mm tt" select * from schedule

表 8. エクスポート用の有効なファイル・タイプ修飾子: WSF ファイル・フォーマット

修飾子	説明
1	Lotus 1-2-3 リリース 1、または Lotus 1-2-3 リリース 1a との互換がある WSF ファイルを作成します。 [5] この値がデフォルトです。
2	Lotus Symphony リリース 1.0 と互換性のある WSF ファイルを作成します。 [5]
3	Lotus 1-2-3 バージョン 2、または Lotus Symphony リリース 1.1 との互換がある WSF ファイルを作成します。[5]
4	DBCS 文字を含む WSF ファイルを作成します。

注:

1. サポートされていないファイル・タイプを MODIFIED BY オプションで使用しようとしても、エクスポート・ユーティリティーは警告を出しません。サポートされていないファイル・タイプを使おうとすると、エクスポート操作は失敗し、エラー・コードが戻されます。
2. データ移動のための区切り文字の制約事項には、区切り文字の指定変更として使用できる文字に適用される制限事項がリストされています。
3. 通常、エクスポート・ユーティリティーでの記述形式は次のとおりです。
   - *YYYYMMDD* 形式の日付データ
   - *"YYYY-MM-DD"* 形式の char(date) データ
   - *"HH.MM.SS"* 形式の時間データ
   - *"YYYY-MM-DD-HH.MM.SS.uuuuuu"* 形式のタイム・スタンプ・データ

   エクスポート操作のために SELECT ステートメントで指定される日時列に含まれるデータも、これらの形式になります。
4. タイム・スタンプ形式の場合、月の記述子と分の記述子のどちらも文字 M を使用するため、あいまいさを避けるよう注意する必要があります。月のフィールドは、他の日付フィールドと隣接していなければなりません。分のフィールドは、他の時刻フィールドに隣接していなければなりません。以下のタイム・スタンプ形式は、あいまいな形式の例です。

   ```
   "M" (月と分のどちらかがはっきりしない)
   "M:M" (どちらが何を表しているのか分からない)
   "M:YYYY:M" (どちらも月として解釈される)
   "S:M:YYYY" (時刻値と日付値の両方に隣接している)
   ```

   あいまいな場合、ユーティリティーはエラー・メッセージを報告し、操作は失敗します。

   以下に、明確なタイム・スタンプ形式を示します。

   ```
   "M:YYYY" (月)
   "S:M" (分)
   "M:YYYY:S:M" (月....分)
   "M:H:YYYY:M:D" (分....月)
   ```
5. *filetype-mod* パラメーター・ストリングの中で、ロータス 1-2-3 の場合は L、Symphony の場合は S を指定することにより、これらのファイルを特定の製品を指すようにできます。 1 つの値または製品指定子だけを指定できます。

関連資料:

- 62 ページの『db2Export - エクスポート』

- 「コマンド・リファレンス」の『EXPORT コマンド』
- 205 ページの『データを移動する際の区切り文字の制約事項』

db2GetAlertCfg - アラート構成の入手

ヘルス・インディケーターのアラート構成設定を戻します。

許可:

なし

必要な接続:

インスタンス。インスタンス・アタッチが存在しない場合は、デフォルトのインスタンス・アタッチが作成されます。

API 組み込みファイル:

db2ApiDf.h

C API 構文:

```
/* File: db2ApiDf.h */
/* API:  db2GetAlertCfg */
/* ... */
SQL_API_RC SQL_API_FN
db2GetAlertCfg(
  db2Uint32 versionNumber,
  void *pParmStruct,
  struct sqlca *pSqlca );

typedef SQL_STRUCTURE db2GetAlertCfgData
{
   db2Uint32             iObjType;
   char                 *piObjName;
   db2Uint32             iDefault;
   char                 *piDbname;
   db2Uint32             ioNumIndicators;
   db2GetAlertCfgInd    *pioIndicators;
} db2GetAlertCfgData;

typedef SQL_STRUCTURE db2GetAlertCfgInd
{
   db2Uint32               ioIndicatorID;
   db2int32                oAlarm;
   db2int32                oWarning;
   db2Uint32               oSensitivity;
   char                   *poFormula;
   db2Uint32               oActionEnabled;
   db2Uint32               oCheckThresholds;
   db2Uint32               oNumTaskActions;
   db2AlertTaskAction     *poTaskActions;
   db2Uint32               oNumScriptActions;
   db2AlertScriptAction   *poScriptActions;
   db2Uint32               oDefault;
} db2GetAlertCfgInd;

typedef SQL_STRUCTURE db2AlertTaskAction
{
   char                 *pTaskname;
   db2Uint32             condition;
```

```
   char                *pUserId;
   char                *pPassword;
   char                *pHostName;
} db2AlertTaskAction;

typedef SQL_STRUCTURE db2AlertScriptAction
 {
   db2Uint32            scriptType;
   db2Uint32            condition;
   char                *pPathname;
   char                *pWorkingDir;
   char                *pCmdLineParms;
   char                 stmtTermChar;
   char                *pUserID;
   char                *pPassword;
   char                *pHostname;
} db2AlertScriptAction;
/* ... */
```

API パラメーター:

versionNumber
入力。 2 番目のパラメーター *pParmStruct* として渡される構造のバージョンとリリースのレベルを指定します。

pParmStruct
入力。 *db2GetAlertCfgData* 構造を指すポインター。

pSqlca
出力。*sqlca* 構造へのポインター。

iObjType
入力。構成が要求されるオブジェクトのタイプを指定します。有効な値は以下のとおりです。

- DB2ALERTCFG_OBJTYPE_DBM
- DB2ALERTCFG_OBJTYPE_DATABASES
- DB2ALERTCFG_OBJTYPE_TABLESPACES
- DB2ALERTCFG_OBJTYPE_TS_CONTAINERS
- DB2ALERTCFG_OBJTYPE_DATABASE
- DB2ALERTCFG_OBJTYPE_TABLESPACE
- DB2ALERTCFG_OBJTYPE_TS_CONTAINER

piObjName
入力。オブジェクト・タイプ *iObjType* が、DB2ALERTCFG_OBJTYPE_TABLESPACE、または DB2ALERTCFG_OBJTYPE_TS_CONTAINER に設定されている場合、表スペースまたは表スペース・コンテナーの名前。

iDefault
入力。デフォルトのインストール構成値が検索されることを示します。

piDbname
入力。オブジェクト・タイプ *iObjType* が DB2ALERTCFG_OBJTYPE_TS_CONTAINER、

DB2ALERTCFG_OBJTYPE_TABLESPACE、および DB2ALERTCFG_OBJTYPE_DATABASE に設定される場合、構成が要求されるデータベースの別名。

ioNumIndicators
このパラメーターは入力または出力パラメーターとして使用できます。

入力。ヘルス・インディケーターのサブセットの設定を要求するときにサブミットされる、 *pioIndicators* の数を示します。

出力。 API によって戻されるヘルス・インディケーターの総数を示します。

pioIndicators
db2GetAlertCfgInd 構造を指すポインター。これが NULL に設定される場合、オブジェクトのすべてのヘルス・インディケーターが戻されます。

ioIndicatorID
ヘルス・インディケーター (`sqlmon.h` で定義)。

oAlarm
出力。ヘルス・インディケーターのアラームしきい値の設定。この設定は、しきい値に基づくヘルス・インディケーターにのみ有効です。

oWarning
出力。ヘルス・インディケーターの警告しきい値の設定。この設定は、しきい値に基づくヘルス・インディケーターにのみ有効です。

oSensitivity
出力。関連するアラームまたは警告状況が登録される前に、ヘルス・インディケーターの値がしきい値域内にとどまっていなければならない期間。

poFormula
出力。ヘルス・インディケーターの値の計算に使用される公式のストリング表記。

oActionEnabled
出力。 TRUE の場合で、しきい値が不履行された場合、 *poTaskActions* または *poScriptActions* に定義されたすべてのアラート・アクションが呼び出されます。 FALSE の場合、定義されたどのオプションも呼び出されません。

oCheckThresholds
出力。 TRUE の場合、しきい値の不履行、または状態変更が評価されます。しきい値の不履行、または状態変更が評価されない場合、アラートは発行されず、アラート・アクションは *oActionEnabled* が TRUE であるかどうかに関係なく呼び出されません。

oNumTaskActions
出力。 *pTaskAction* 配列内のタスク・アラート・アクションの数。

poTaskActions
db2AlertTaskAction 構造を指すポインター。

oNumScriptActions
出力。 *poScriptActions* 配列内のスクリプト・アクションの数。

poScriptActions
db2AlertScriptAction 構造を指すポインター。

oDefault
出力。現在の設定値がデフォルトから継承されたものかどうかを示します。現在の設定値がデフォルトから継承されたことを示すには、 TRUE に設定されます。そうでなければ、FALSE に設定されます。

pTaskname
タスク名。

condition
アクションを実行する条件。

scriptType
スクリプトのタイプを指定します。有効な値は以下のとおりです。

- DB2ALERTCFG_SCRIPTTYPE_DB2CMD
- DB2ALERTCFG_SCRIPTTYPE_OS

pPathname
スクリプトの絶対パス名。

pWorkingDir
スクリプトが実行されるディレクトリーの絶対パス名。

pCmdLineParms
呼び出し時にスクリプトに渡されるコマンド行パラメーター。
DB2ALERTCFG_SCRIPTTYPE_OS 専用のオプションです。

stmtTermChar
ステートメントを終了するスクリプトに使用される文字。
DB2ALERTCFG_SCRIPTTYPE_DB2CMD 専用のオプションです。

pUserID
スクリプトを実行するユーザー・アカウント。

pPassword
ユーザー・アカウント *pUserId* のパスワード。

pHostName
スクリプトを実行するホスト名。これは、タスクとスクリプトの両方に適用されます。

Script スクリプトが常駐し、実行されるホスト名。

Task スケジューラーが常駐するホスト名。

使用上の注意:

pioIndicators が NULL である場合、オブジェクトのすべてのヘルス・インディケーターが戻されます。このパラメーターは、構成したいヘルス・インディケーターに *ioIndicatorID* を設定した、 *db2GetAlertCfgInd* 構造の配列に設定できます。この方法で使用される場合、必ず *ioNumIndicators* を入力配列長に設定し、*db2GetAlertCfgInd* 内の他のすべてのフィールドを 0 または NULL に設定してください。

このポインターの下にあるすべてのメモリーは、エンジンによって割り振られ、db2GetAlertCfg がエラーを戻さない場合はいつでも、db2GetAlertCfgFree を解放しなければなりません。 db2GetAlertCfgFree の詳細については、db2ApiDf.h を参照してください。

関連資料:

- 453 ページの『SQLCA』
- 239 ページの『db2ResetAlertCfg - アラート構成のリセット』
- 281 ページの『db2UpdateAlertCfg - アラート構成の更新』
- 「システム・モニター *ガイドおよびリファレンス*」の『ヘルス・インディケーター』
- 79 ページの『db2GetAlertCfgFree - アラート構成メモリー入手の解放』

db2GetAlertCfgFree - アラート構成メモリー入手の解放

db2GetAlertCfg によって割り振られたメモリーを解放します。

許可:

なし

必要な接続:

なし

API 組み込みファイル:

db2ApiDf.h

C API 構文:

```
/* File: db2ApiDf.h */
/* API: db2GetAlertCfgFree */
/* ... */
SQL_API_RC_SQL_API_FN
db2GetAlertCfgFree (
   db2Uint32 versionNumber,
   void * pParmStruct,
   struct sqlca * pSqlca);
/* ... */
```

API パラメーター:

versionNumber
入力。 2 番目のパラメーター *pParmStruct* として渡される構造のバージョンとリリースのレベルを指定します。

pParmStruct
入力。 *db2GetAlertCfgData* 構造を指すポインター。

pSqlca
出力。*sqlca* 構造へのポインター。

参考文献

関連資料:

- 453 ページの『SQLCA』
- 75 ページの『db2GetAlertCfg - アラート構成の入手』
- 239 ページの『db2ResetAlertCfg - アラート構成のリセット』
- 281 ページの『db2UpdateAlertCfg - アラート構成の更新』

db2GetContactGroup - 連絡先グループの入手

単一の連絡先グループに含まれる連絡先を戻します。連絡先は、通知メッセージが送信されるユーザーです。連絡先は、システムでローカルに定義することも、グローバル・リストで定義することもできます。 DB2 Administration Server (DAS) の構成パラメーター *contact_host* の設定は、リストがローカルかグローバルかを判別します。

許可:

なし

必要な接続:

なし

API 組み込みファイル:

db2ApiDf.h

C API 構文:

```
/* File: db2ApiDf.h */
/* API: db2GetContactGroup */
/* ... */
SQL_API_RC SQL_API_FN
db2GetContactGroup (
  db2Uint32 versionNumber,
  void *pParmStruct,
  struct sqlca *pSqlca);

typedef SQL_STRUCTURE db2ContactGroupData
{
   char                       *pGroupName;
   char                       *pDescription;
   db2Uint32                  numContacts;
   struct db2ContactTypeData *pContacts;
} db2ContactGroupData;

typedef SQL_STRUCTURE db2ContactTypeData
{
   db2Uint32                  contactType;
   char                       *pName;
} db2ContactTypeData;
/* ... */
```

API パラメーター:

versionNumber
入力。 2 番目のパラメーター *pParmStruct* として渡される構造のバージョンとリリースのレベルを指定します。

pParmStruct
入力。 *db2GetContactGroupData* 構造を指すポインター。

pSqlca
出力。*sqlca* 構造へのポインター。

pGroupName
入力。検索されるグループ名。

pDescription
グループの説明。

numContacts
pContacts の数。

pContacts
db2ContactTypeData 構造を指すポインター。 *pGroupName* フィールド、*pDescription* フィールド、*pContacts* フィールド、および *pContacts.pName* フィールドは、ユーザーによってそれぞれの最大サイズで事前に割り振られる必要があります。 db2GetContactGroup を *numContacts* =0 および *pContacts* =NULL で呼び出し、 *numContacts* で戻される *pContacts* を必要な長さにします。

contactType
連絡先のタイプを指定します。有効な値は以下のとおりです。

- DB2CONTACT_SINGLE
- DB2CONTACT_GROUP

pName
連絡先グループ名、または *ioContactType* が DB2CONTACT_SINGLE に設定されている場合は連絡先の名前。

関連資料:

- 453 ページの『SQLCA』
- 「*管理ガイド: パフォーマンス*」の『contact_host - 「連絡先リストのロケーション」構成パラメーター』
- 18 ページの『db2AddContactGroup - 連絡先グループの追加』
- 61 ページの『db2DropContactGroup - 連絡先グループのドロップ』
- 81 ページの『db2GetContactGroups - 連絡先グループの入手』
- 289 ページの『db2UpdateContactGroup - 連絡先グループの更新』

db2GetContactGroups - 連絡先グループの入手

連絡先グループのリストを戻します。連絡先は、通知メッセージが送信されるユーザーです。連絡先グループは、システムでローカルに定義することも、グローバル・リストで定義することもできます。 DB2 Administration Server (DAS) の構成パラメーター *contact_host* の設定は、リストがローカルかグローバルかを判別します。

許可:

なし

必要な接続:

なし

API 組み込みファイル:

db2ApiDf.h

C API 構文:

```
/* File: db2ApiDf.h */
/* API: db2GetContactGroups */
/* ... */
SQL_API_RC SQL_API_FN
db2GetContactGroups (
  db2Uint32 versionNumber,
  void *pParmStruct,
  struct sqlca *pSqlca);

typedef SQL_STRUCTURE db2GetContactGroupsData
{
   db2Uint32                 ioNumGroups;
   struct db2ContactGroupDesc *poGroups;
} db2GetContactGroupsData;

typedef SQL_STRUCTURE db2ContactGroupDesc
{
   char                      *poName;
   char                      *poDescription;
} db2ContactGroupDesc;
/* ... */
```

API パラメーター:

versionNumber
入力。 2 番目のパラメーター *pParmStruct* として渡される構造のバージョンとリリースのレベルを指定します。

pParmStruct
入力。 *db2GetContactGroupsData* 構造を指すポインター。

pSqlca
出力。*sqlca* 構造へのポインター。

ioNumGroups
グループの数。 *oNumGroups* = 0 および *poGroups* = NULL の場合、*poGroups* に必要とされる *db2ContactGroupDesc* 構造の数が入ります。

poGroups
出力。 *db2ContactGroupDesc* 構造を指すポインター。

poName
出力。グループ名。このパラメーターは、呼び出し側によってそれぞれの最大サイズで事前に割り振られる必要があります。

poDescription
出力。グループの説明。このパラメーターは、呼び出し側によってそれぞれの最大サイズで事前に割り振られる必要があります。

関連資料:

- 453 ページの『SQLCA』
- 「*管理ガイド: パフォーマンス*」の『contact_host - 「連絡先リストのロケーション」構成パラメーター』
- 18 ページの『db2AddContactGroup - 連絡先グループの追加』
- 61 ページの『db2DropContactGroup - 連絡先グループのドロップ』
- 80 ページの『db2GetContactGroup - 連絡先グループの入手』
- 289 ページの『db2UpdateContactGroup - 連絡先グループの更新』

db2GetContacts - 連絡先の入手

連絡先のリストを戻します。連絡先は、通知メッセージが送信されるユーザーです。連絡先は、システムでローカルに定義することも、グローバル・リストで定義することもできます。 DB2 Administration Server (DAS) の構成パラメーター *contact_host* の設定は、リストがローカルかグローバルかを判別します。

許可:

なし

必要な接続:

なし

API 組み込みファイル:

db2ApiDf.h

C API 構文:

```
/* File: db2ApiDf.h */
/* API: db2GetContacts */
/* ... */
SQL_API_RC SQL_API_FN
db2GetContacts (
  db2Uint32 versionNumber,
  void *pParmStruct,
  struct sqlca *pSqlca);

typedef SQL_STRUCTURE db2GetContactsData
{
   db2Uint32                ioNumContacts;
   struct db2ContactData    *poContacts;
} db2GetContactsData;

typedef SQL_STRUCTURE db2ContactData
{
   char *pName;
   db2Uint32                type;
   char                     *pAddress;
   db2Uint32                maxPageLength;
   char                     *pDescription;
} db2ContactData;
/* ... */
```

API パラメーター:

versionNumber
入力。 2 番目のパラメーター *pParmStruct* として渡される構造のバージョンとリリースのレベルを指定します。

pParmStruct
入力。 *db2GetContactsData* 構造を指すポインター。

pSqlca
出力。*sqlca* 構造へのポインター。

ioNumContacts
poContacts の数。

poContacts
出力。 *db2ContactData* 構造を指すポインター。 *poContacts* フィールド、*pocontacts.pAddress* フィールド、*pocontacts.pDescription* フィールド、および *pocontacts.pName* フィールドは、ユーザーによってそれぞれの最大サイズで事前に割り振られる必要があります。 db2GetContacts を *numContacts* =0 および *poContacts* =NULL で呼び出し、 *numContacts* で戻される *poContacts* を必要な長さにします。

pName
連絡先名。

type 連絡先のタイプを指定します。有効な値は以下のとおりです。

- DB2CONTACT_EMAIL
- DB2CONTACT_PAGE

pAddress
type パラメーターのアドレス。

maxPageLength
type が DB2CONTACT_PAGE に設定されたときの最大メッセージ長。

pDescription
ユーザー提供の連絡先の説明。

関連資料:

- 453 ページの『SQLCA』
- 「*管理ガイド: パフォーマンス*」の『contact_host - 「連絡先リストのロケーション」構成パラメーター』
- 17 ページの『db2AddContact - 連絡先の追加』
- 60 ページの『db2DropContact - 連絡先のドロップ』
- 288 ページの『db2UpdateContact - 連絡先の更新』

db2GetHealthNotificationList - ヘルス通知リストの入手

インスタンスのヘルスについて通知される連絡先および連絡先グループの一方または両方のリストを戻します。連絡先リストは、非常時ヘルス状況がインスタンスまたはそのデータベース・オブジェクトに示されたときに通知される、個人の電子メール・アドレスまたはページャー・インターネット・アドレスで構成されます。

許可:

なし

必要な接続:

インスタンス。インスタンス・アタッチが存在しない場合は、デフォルトのインスタンス・アタッチが作成されます。

API 組み込みファイル:

db2ApiDf.h

C API 構文:

```
/* File: db2ApiDf.h */
/* API: db2GetHealthNotificationList */
/* ... */
SQL_API_RC SQL_API_FN
db2GetHealthNotificationList (
  db2Uint32 versionNumber,
  void *pParmStruct,
  struct sqlca *pSqlca);

typedef SQL_STRUCTURE db2GetHealthNotificationListData
{
   db2Uint32                 ioNumContacts;
   struct db2ContactTypeData *poContacts;
} db2GetHealthNotificationListData;

typedef SQL_STRUCTURE db2ContactTypeData
{
   db2Uint32                 contactType;
   char *pName;
} db2ContactTypeData;
/* ... */
```

API パラメーター:

versionNumber
入力。 2 番目のパラメーター *pParmStruct* として渡される構造のバージョンとリリースのレベルを指定します。

pParmStruct
入力。 *db2GetHealthNotificationListData* 構造を指すポインター。

pSqlca
出力。*sqlca* 構造へのポインター。

ioNumContacts
連絡先の数。 API が NULL *poContact* で呼び出された場合、*ioNumContacts* は、呼び出しを正常に実行するようユーザーが割り振る連絡先の数に設定されます。

poContacts
出力。 *db2ContactTypeData* 構造を指すポインター。

contactType
連絡先のタイプを指定します。有効な値は以下のとおりです。

- DB2CONTACT_SINGLE

- DB2CONTACT_GROUP

pName

連絡先グループ名、または *contactType* が DB2CONTACT_SINGLE に設定されている場合は連絡先の名前。この値を、サイズ DB2CONTACT_MAX_SZ の事前割り振りバッファーに設定します。

関連資料:

- 453 ページの『SQLCA』
- 291 ページの『db2UpdateHealthNotificationList - ヘルス通知リストの更新』

db2GetRecommendations - アラート状態のヘルス・インディケーターに関する推奨の入手

特定のオブジェクトに関してアラート状態にあるヘルス・インディケーターを解決するための推奨のセットを検索します。推奨は XML 文書として戻されます。

許可:

なし

必要な接続:

インスタンス。インスタンス・アタッチが存在しない場合は、デフォルトのインスタンス・アタッチが作成されます。

API 組み込みファイル:

db2ApiDf.h

C API 構文:

```
/* File: db2ApiDf.h */
/* API: db2GetRecommendations */
/* ... */
SQL_API_RC SQL_API_FN
  db2GetRecommendations(
    db2Uint32 versionNumber,
    void *pParmStruct,
    struct sqlca *pSqlca ) ;

typedef struct db2GetRecommendationsData {
  db2Uint32 iSchemaVersion;
  db2Uint32 iNodeNumber;
  db2Uint32 iIndicatorID ;
  db2Uint32 iObjType ;
  char      *piObjName ;
  char      *piDbname ;
  char      *poRecommendation ;
} db2GetRecommendationsData ;

/* ... */
```

API パラメーター:

versionNumber
入力。 2 番目のパラメーター *pParmStruct* として渡される構造のバージョンとリリースのレベルを指定します。

pParmStruct
入力。 *db2GetRecommendationsData* 構造を指すポインター。

pSqlca
出力。*sqlca* 構造へのポインター。

iSchemaVersion
入力。 XML 文書を表すために使用されるスキーマのバージョン ID。勧告文書には、そのスキーマ・バージョンで定義されたエレメントまたは属性だけが含まれます。このパラメーターを、以下のいずれかの定数に設定してください。

DB2HEALTH_RECSCHEMA_VERSION8_2

iNodeNumber
入力。ヘルス・インディケーター (HI) がアラート状態に入ったパーティション番号を示します。すべてのパーティションを対象とする特定の HI の特定のオブジェクトに関する推奨を検索するには、定数 SQLM_ALL_NODES を使用します。複数の異なるパーティションに関する同じ HI 推奨がいくつか存在する場合、これらの推奨は 1 つの推奨セットにグループ化されます。この場合、異なるパーティションに関する複数の HI からなるグループが問題となり、一連の推奨はこれらすべての HI に適用されます。現在のパーティションに関する推奨を検索するには、 SQLM_CURRENT_NODE を使用します。独立型インスタンスの場合には、 SQLM_CURRENT_NODE を使用する必要があります。

iIndicatorID
入力。推奨を検索する対象である、アラート状態に入ったヘルス・インディケーター。値は `sqlmon.h` で外部化されます。

iObjType
入力。 (iIndicatorID によって識別される) ヘルス・インディケーターがアラート状態に入ったオブジェクトのタイプを指定します。可能な値は以下のとおりです。

DB2HEALTH_OBJTYPE_DBM

DB2HEALTH_OBJTYPE_DATABASE

DB2HEALTH_OBJTYPE_TABLESPACE

DB2HEALTH_OBJTYPE_TS_CONTAINER

piObjName
入力。オブジェクト・タイプ *iObjType* が DB2HEALTH_OBJTYPE_TABLESPACE または DB2HEALTH_OBJTYPE_TS_CONTAINER に設定されている場合、表スペースまたは表スペース・コンテナーの名前。必要でない場合、NULL を指定します。表スペース・コンテナーの場合、オブジェクト名は <tablespace name>.<container name> と指定されます。

piDbname
入力。オブジェクト・タイプ *iObjType* が

DB2HEALTH_OBJTYPE_TS_CONTAINER、
DB2HEALTH_OBJTYPE_TABLESPACE、および
DB2HEALTH_OBJTYPE_DATABASE の場合、 HI がアラート状態に入った対象のデータベースの別名。それ以外の場合、NULL を指定します。

poRecommendation
出力。推奨テキストが入っているメモリー内のバッファー・アドレスに設定される文字ポインター。このテキストは、sqllib/misc/DB2RecommendationSchema.xsd で指定されるスキーマに従って XML 文書にフォーマットされます。 XML 文書は UTF-8 でエンコードされ、文書内のテキストは呼び出し側のロケールになります。 DB2_HEALTH ノードの xml:lang 属性は、適切なクライアント言語に設定されます。この API は信頼されたソースとして扱われる必要があり、 XML 文書を妥当性検査してはなりません。 XML は出力データを構造化する手段として使用されます。このポインターの下にあるすべてのメモリーはエンジンによって割り振られます。 db2GetRecommendations がエラーを戻さない場合には、db2GetRecommendationsFree 呼び出しによって必ずこれを解放しなければなりません。

使用上の注意:

- この API を呼び出すことにより、特定の DB2 オブジェクトに関するヘルス・アラートを解決するための推奨のセットを検索します。識別されたオブジェクトに関する入力ヘルス・インディケーターがアラート状態でない場合、エラーが戻されます。
- 推奨は XML 文書として戻され、そこには、アラートを解決するために実行する操作およびスクリプトについての情報が含まれます。この API によって戻されるすべてのスクリプトを実行する場所は、ヘルス・インディケーターがアラート状態に入ったインスタンス上でなければなりません。戻される推奨 XML 文書の構造および内容については、 sqllib/misc/DB2RecommendationSchema.xsd のスキーマを参照してください。
- db2GetRecommendations がエラーを戻さない場合、エンジンによって割り振られ、この関数によって戻されるすべてのメモリー (推奨文書) を db2GetRecommendationsFree 呼び出しによって必ず解放しなければなりません。

関連資料:

- 88 ページの『db2GetRecommendationsFree - db2GetRecommendations メモリーの解放』

db2GetRecommendationsFree - db2GetRecommendations メモリーの解放

db2GetRecommendations API によって割り振られたメモリーを解放します。

許可:

なし

必要な接続:

| なし

| **API 組み込みファイル:**

| *db2ApiDf.h*

| **C API 構文:**

```
| /* File: db2ApiDf.h */
| /* API: db2GetRecommendationsFree */
| /* ... */
| SQL_API_RC SQL_API_FN
|   db2GetRecommendationsFree(
|     db2Uint32 versionNumber,
|     void *pParmStruct,
|     struct sqlca *pSqlca ) ;
| /* ... */
```

| **API パラメーター:**

versionNumber
入力。 2 番目のパラメーター *pParmStruct* として渡される構造のバージョンとリリースのレベルを指定します。

pParmStruct
入力。 *db2GetRecommendationsData* 構造を指すポインター。

pSqlca
出力。*sqlca* 構造へのポインター。

| **関連資料:**

- 86 ページの『db2GetRecommendations - アラート状態のヘルス・インディケーターに関する推奨の入手』

db2GetSnapshot - スナップショットの入手

データベース・マネージャー・モニター情報を収集し、それをユーザーが割り振ったデータ・バッファーに戻します。戻される情報は、 API が呼び出された時点でのデータベース・マネージャーの操作状況のスナップショット です。

有効範囲:

この API はインスタンス上のデータベース・パーティション・サーバー、またはインスタンス上のすべてのデータベース・パーティションに関する情報を戻すことができます。

許可:

以下のいずれかです。

- *sysadm*
- *sysctrl*
- *sysmaint*
- *sysmon*

必要な接続:

インスタンス。インスタンス・アタッチが存在しない場合は、デフォルトのインスタンス・アタッチが作成されます。

リモート・インスタンス (または別のローカル・インスタンス) からスナップショットを獲得するには、まず最初にそのインスタンスとアタッチすることが必要です。

API 組み込みファイル:

db2ApiDf.h

C API 構文:

```
/* File: db2ApiDf.h */
/* API: db2GetSnapshot */
/* ... */
SQL_API_RC SQL_API_FN
db2GetSnapshot (
  db2Uint32 versionNumber,
  void *pParmStruct,
  struct sqlca *pSqlca);

typedef SQL_STRUCTURE db2GetSnapshotData
{
  struct sqlma                *piSqlmaData;
  struct sqlm_collected       *poCollectedData;
  void                        *poBuffer;
  db2Uint32                   iVersion;
  db2Uint32                   iBufferSize;
  db2Uint32                   iStoreResult;
  db2int32                    iNodeNumber;
  db2Uint32                   *poOutputFormat;
  db2Uint32                   iSnapshotClass;
} db2GetSnapshotData;
/* ... */
```

汎用 API 構文:

```
/* File: db2ApiDf.h */
/* API: db2gGetSnapshot */
/* ... */
SQL_API_RC SQL_API_FN
db2gGetSnapshot (
  db2Uint32 versionNumber,
  void *pParmStruct,
  struct sqlca *pSqlca);

typedef SQL_STRUCTURE db2gGetSnapshotData
{
  struct sqlma                *piSqlmaData;
  struct sqlm_collected       *poCollectedData;
  void                        *poBuffer;
  db2Uint32                   iVersion;
  db2Uint32                   iBufferSize;
  db2Uint32                   iStoreResult;
  db2int32                    iNodeNumber;
  db2Uint32                   *poOutputFormat;
  db2Uint32                   iSnapshotClass;
} db2gGetSnapshotData;
/* ... */
```

API パラメーター:

versionNumber

| 入力。 2 番目のパラメーター pParmStruct として渡される構造のバージョ

ンとリリースのレベルを指定します。上記のような構造を使用するには、db2Version810 を指定します。この構造の別のバージョンを使用したい場合には、 include ディレクトリー内の db2ApiDf.h ヘッダー・ファイルを調べて、サポートされるバージョンの詳細リストを確認してください。指定しバージョン番号に対応する *db2GetSnapshotData* 構造を必ず使用してください。

pParmStruct
入力/出力。 *db2GetSnapshotData* 構造を指すポインター。

pSqlca
出力。*sqlca* 構造へのポインター。

piSqlmaData
入力。ユーザー割り当ての *sqlma* (モニター域) 構造を指すポインター。この構造には、収集したいデータのタイプ (複数可) を指定します。

poCollectedData
出力。データベース・モニターが、サマリー統計とバッファー域に戻された各タイプのデータ構造の数を渡す *sqlm_collected* 構造を指すポインター。

注: この構造を使用するのは、バージョン 6 より前のデータ・ストリームの場合だけです。しかし、バックレベルのリモート・サーバーへのスナップショット呼び出しが実行される場合は、結果を処理するため、この構造を渡す必要があります。このため、常にこのパラメーターを渡すようお勧めします。

poBuffer
出力。スナップショット情報が戻される、ユーザー定義のデータ域を指すポインター。

iVersion
入力。収集するデータベース・モニター・データのバージョン ID です。データベース・モニターは、要求されたバージョンについて使用可能なデータのみを戻します。このパラメーターは、以下のいずれかのシンボリック定数に設定してください。

- SQLM_DBMON_VERSION1
- SQLM_DBMON_VERSION2
- SQLM_DBMON_VERSION5
- SQLM_DBMON_VERSION5_2
- SQLM_DBMON_VERSION6
- SQLM_DBMON_VERSION7
- SQLM_DBMON_VERSION8

注: バージョンとして SQLM_DBMON_VERSION1 が指定された場合、 API はリモートでは実行できません。

iBufferSize
入力。データ・バッファーの長さ。このバッファーのサイズを見積もるには、db2GetSnapshotSize を使用してください。バッファーの大きさが十分で

ない場合には、割り当てられたバッファーに収まるだけの情報とともに、警告が戻されます。バッファーのサイズを変更し、API を再び呼び出すことが必要になる可能性があります。

iStoreResult
入力。スナップショットの結果を DB2 サーバーに格納し、 SQL を介して表示するかどうかに応じて、 TRUE または FALSE に設定されます。このパラメーターが TRUE に設定されるのは、データベース接続を介してスナップショットを取る場合と、 *sqlma* のスナップショット・タイプの 1 つが SQLMA_DYNAMIC_SQL である場合だけです。

iNodeNumber
入力。要求の送信先となるノードを示します。この値に基づき、要求は現在のノード、すべてのノード、またはユーザーが指定したノードに対して処理されます。有効な値は以下のとおりです。

- SQLM_CURRENT_NODE
- SQLM_ALL_NODES。 iVersion が SQLM_DBMON_VERSION7 または SQLM_DBMON_VERSION8 に設定されている場合に限り、使用できます。
- *ノード値*

注: 独立型インスタンスの場合には、 SQLM_CURRENT_NODE を使用する必要があります。

poOutputFormat
サーバーから返されるストリームの形式。次のいずれかです。

- SQLM_STREAM_STATIC_FORMAT
- SQLM_STREAM_DYNAMIC_FORMAT

iSnapshotClass
入力。スナップショットのクラス識別コード。有効な値 (sqlmon.h で定義) は以下のとおりです。

- SQLM_CLASS_DEFAULT (標準スナップショット用)
- SQLM_CLASS_HEALTH (ヘルス・スナップショット用)
- SQLM_CLASS_HEALTH_WITH_DETAIL (追加の詳細情報を含むヘルス・スナップショット用)

使用上の注意:

別のインスタンスに存在するデータベースの別名が指定されている場合には、エラー・メッセージが戻されます。

すべての情報を収集するヘルス・スナップショットを検索するには、 SQLMA データ構造の AGENT_ID フィールドを使用します。

関連資料:

- 211 ページの『db2MonitorSwitches - モニター・スイッチの入手/更新』
- 93 ページの『db2GetSnapshotSize - db2GetSnapshot 出力バッファーに必要なサイズの見積もり』
- 241 ページの『db2ResetMonitor - モニターのリセット』

- 453 ページの『SQLCA』
- 488 ページの『SQLM-COLLECTED』
- 491 ページの『SQLMA』
- 42 ページの『db2ConvMonStream - モニター・ストリームの変換』

関連サンプル:

- 『utilsnap.c -- Utilities for the snapshot monitor samples (C)』
- 『utilsnap.C -- Utilities for the snapshot monitor samples (C++)』

db2GetSnapshotSize - db2GetSnapshot 出力バッファーに必要なサイズの見積もり

db2GetSnapshot に必要なバッファー・サイズを見積もります。

有効範囲:

この API はインスタンス上のデータベース・パーティション・サーバー、またはインスタンス上のすべてのデータベース・パーティションに影響を与えます。

許可:

以下のいずれかです。

- *sysadm*
- *sysctrl*
- *sysmaint*
- *sysmon*

必要な接続:

インスタンス。インスタンス・アタッチが存在しない場合は、デフォルトのインスタンス・アタッチが作成されます。

リモート・インスタンス (または別のローカル・インスタンス) から情報を取得するには、まず最初にそのインスタンスにアタッチすることが必要です。アタッチが存在しない場合は、 DB2INSTANCE 環境変数によって指定されているノードへの暗黙的なインスタンス・アタッチが確立されます。

API 組み込みファイル:

db2ApiDf.h

C API 構文:

```
/* File: db2ApiDf.h */
/* API: db2GetSnapshotSize */
/* ... */

SQL_API_RC SQL_API_FN
db2GetSnapshotSize (
  db2Uint32 versionNumber,
  void *pParmStruct,
  struct sqlca *pSqlca);
```

```
typedef SQL_STRUCTURE db2GetSnapshotSizeData
{
  struct sqlma                *piSqlmaData;
  sqluint32                   *poBufferSize;
  db2Uint32                   iVersion;
  db2int32                    iNodeNumber;
  db2Uint32                   iSnapshotClass;
} db2GetSnapshotSizeData;
/* ...*/
```

汎用 API 構文:

```
/* File: db2ApiDf.h */
/* API: db2gGetSnapshotSize */
/* ... */
SQL_API_RC SQL_API_FN
db2gGetSnapshotSize (
  db2Uint32 versionNumber,
  void *pParmStruct,
  struct sqlca *pSqlca);

typedef SQL_STRUCTURE db2gGetSnapshotSizeData
{
  struct sqlma                *piSqlmaData;
  sqluint32                   *poBufferSize;
  db2Uint32                   iVersion;
  db2int32                    iNodeNumber;
  db2Uint32                   iSnapshotClass;
} db2gGetSnapshotSizeData;
/* ... */
```

API パラメーター:

version
入力。 2 番目のパラメーター pParmStruct として渡される構造のバージョンとリリースのレベルを指定します。上記のような構造を使用するには、db2Version810 を指定します。この構造の別のバージョンを使用したい場合には、 include ディレクトリー内の db2ApiDf.h ヘッダー・ファイルを調べて、サポートされるバージョンの詳細リストを確認してください。指定しバージョン番号に対応する *db2GetSnapshotSizeStruct* 構造を必ず使用してください。

pParmStruct
入力。 *db2GetSnapshotSizeStruct* 構造を指すポインター。

pSqlca
出力。*sqlca* 構造へのポインター。

piSqlmaData
入力。ユーザー割り当ての *sqlma* (モニター域) 構造を指すポインター。この構造は収集されるスナップショット・データのタイプを指定し、db2GetSnapshot への入力として再利用できます。

poBufferSize
出力。 GET SNAPSHOT (スナップショットの入手) API に必要とされる、戻された見積バッファー・サイズを指すポインター。

iVersion
入力。収集するデータベース・モニター・データのバージョン ID です。デ

ータベース・モニターは、要求されたバージョンについて使用可能なデータのみを戻します。このパラメーターは、以下のいずれかのシンボリック定数に設定してください。

- SQLM_DBMON_VERSION1
- SQLM_DBMON_VERSION2
- SQLM_DBMON_VERSION5
- SQLM_DBMON_VERSION5_2
- SQLM_DBMON_VERSION6
- SQLM_DBMON_VERSION7
- SQLM_DBMON_VERSION8

注: バージョンとして SQLM_DBMON_VERSION1 が指定された場合、 API はリモートでは実行できません。

iNodeNumber

入力。要求の送信先となるデータベース・パーティション・サーバーを示します。この値に基づき、要求は現行のデータベース・パーティション・サーバー、すべてのデータベース・パーティション・サーバー、またはユーザーが指定したデータベース・パーティション・サーバーに対して処理されます。有効な値は以下のとおりです。

- SQLM_CURRENT_NODE
- SQLM_ALL_NODES。 iVersion が SQLM_DBMON_VERSION7 または SQLM_DBMON_VERSION8 に設定されている場合に限り、使用できます。
- *ノード値*

独立型インスタンスの場合には、 SQLM_CURRENT_NODE を使用する必要があります。

iSnapshotClass

入力。スナップショットのクラス識別コード。有効な値 (sqlmon.h で定義) は以下のとおりです。

- SQLM_CLASS_DEFAULT (標準スナップショット用)
- SQLM_CLASS_HEALTH (ヘルス・スナップショット用)
- SQLM_CLASS_HEALTH_WITH_DETAIL (追加の詳細情報を含むヘルス・スナップショット用)

使用上の注意:

この関数は、かなりの量のオーバーヘッドをもたらします。また、1 回の db2GetSnapshot 呼び出しごとにメモリーを動的に割り振り、解放することにもコストがかかります。 db2GetSnapshot を繰り返し呼び出す場合 (たとえば、ある期間にわたってデータを抽出するとき) は、db2GetSnapshotSize を呼び出すよりも、一定サイズのバッファーを割り振る方が望ましいこともあります。

データベース・システム・モニターは、アクティブなデータベースまたはアプリケーションがないことを検出すると、ゼロのバッファー・サイズを戻す場合があります (たとえば、アクティブでないデータベースに関連するロック情報が要求された

場合)。 db2GetSnapshot を呼び出す前に、この API によって戻された見積バッファー・サイズがゼロ以外の値であることを確認してください。バッファー・スペースが足りなくて出力を保持できないため、 db2GetSnapshot によってエラーが戻された場合には、この API を再び呼び出して新しいサイズ要件を判別してください。

関連資料:

- 89 ページの『db2GetSnapshot - スナップショットの入手』
- 211 ページの『db2MonitorSwitches - モニター・スイッチの入手/更新』
- 241 ページの『db2ResetMonitor - モニターのリセット』
- 453 ページの『SQLCA』
- 491 ページの『SQLMA』
- 「システム・モニター *ガイドおよびリファレンス*」の『スナップショット・モニターの論理データ・グループおよびモニター・エレメント』
- 「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・モニターの論理データ・グループおよびモニター・エレメント』

db2GetSyncSession - サテライト同期セッションの入手

サテライトの現行の同期セッション ID を取得します。

許可:

なし

必要な接続:

なし

API 組み込みファイル:

db2ApiDf.h

C API 構文:

```
/* File: db2ApiDf.h */
/* API: db2GetSyncSession */
/* ... */
SQL_API_RC SQL_API_FN
  db2GetSyncSession (
    db2Uint32 versionNumber,
    void *pParmStruct,
    struct sqlca *pSqlca);

typedef struct
{
  char *poSyncSessionID;
} db2GetSyncSessionStruct;
/* ... */
```

API パラメーター:

versionNumber
入力。 2 番目のパラメーター *pParmStruct* として渡される構造のバージョンとリリースのレベルを指定します。

pParmStruct
入力。 *db2GetSyncSessionStruct* 構造を指すポインター。

pSqlca
出力。*sqlca* 構造へのポインター。

poSyncSessionID
出力。サテライトが現在使用している同期セッションの ID を指定します。

関連資料:

• 453 ページの『SQLCA』

db2HADRStart - HADR の開始

データベースで HADR 操作を開始します。

許可:

以下のいずれかが必要です。

- *sysadm*
- *sysctrl*
- *sysmaint*

必要な接続:

インスタンス。データベース接続が存在しない場合、この API でデータベース接続が確立され、 API 完了時にそのデータベース接続がクローズされます。

API 組み込みファイル:

db2ApiDf.h

C API 構文:

```
/* File: db2ApiDf.h */
/* API: db2HADRStart */
/* ... */
SQL_API_RC SQL_API_FN
  db2HADRStart (
       db2Uint32 versionNumber,
       void * pParmStruct,
       struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2HADRStartStruct
{
  char                        *piDbAlias;
  char                        *piUserName;
  char                        *piPassword;
  db2Uint32                   iDbRole;
  db2Uint16                   iByForce;
} db2HADRStartStruct;
```

汎用 API 構文:

```
/* File: db2ApiDf.h */
/* API: db2gHADRStart */
/* ... */
SQL_API_RC SQL_API_FN
  db2gHADRStart (
       db2Uint32 versionNumber,
```

```
        void * pParmStruct,
        struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2gHADRStartStruct
{
  char                        *piDbAlias;
  db2Uint32                   iAliasLen;
  char                        *piUserName;
  db2Uint32                   iUserNameLen;
  char                        *piPassword;
  db2Uint32                   iPasswordLen;
  db2Uint32                   iDbRole;
  db2Uint16                   iByForce;
} db2gHADRStartStruct;
```

API パラメーター:

versionNumber
入力。 2 番目のパラメーター *pParmStruct* として渡される構造のバージョンとリリースのレベルを指定します。

pParmStruct
入力。 *db2HADRStartStruct* 構造を指すポインター。

pSqlca
出力。*sqlca* 構造へのポインター。

piDbAlias
入力。データベース別名を指すポインター。

iAliasLen
入力。データベース別名の長さ (バイト単位) を指定します。

piUserName
入力。コマンドを実行するときのユーザー名へのポインター。

iUserNameLen
入力。ユーザー名の長さ (バイト単位) を指定します。

piPassword
入力。パスワードを含むストリングを指すポインター。

iPasswordLen
入力。パスワードの長さ (バイト単位) を指定します。

iDbRole
入力。データベースを HADR 1 次データベースとして開始するか、スタンバイ・データベースとして開始するか指定します。有効な値は以下のとおりです。

DB2HADR_DB_ROLE_PRIMARY

DB2HADR_DB_ROLE_STANDBY

iByForce
入力。 *iDbRole* が DB2HADR_DB_ROLE_STANDBY に設定される場合、この引き数は無視されます。有効な値は以下のとおりです。

DB2HADR_NO_FORCE
スタンバイ・データベースが指定した時間制限内に 1 次データベースへ接続する場合にのみ、 HADR が 1 次データベースで開始されることを指定します。

DB2HADR_FORCE
スタンバイ・データベースが 1 次データベースへ接続することを待機せずに、 HADR を強制的に開始することを指定します。

関連資料:

- 99 ページの『db2HADRStop - HADR の停止』
- 100 ページの『db2HADRTakeover - 1 次データベースとしてのテークオーバー』
- 「コマンド・リファレンス」の『START HADR コマンド』

db2HADRStop - HADR の停止

データベースで HADR 操作を停止します。

許可:

以下のいずれかが必要です。

- *sysadm*
- *sysctrl*
- *sysmaint*

必要な接続:

インスタンス。データベース接続が存在しない場合、この API でデータベース接続が確立され、 API 完了時にそのデータベース接続がクローズされます。

API 組み込みファイル:

db2ApiDf.h

C API 構文:

```
/* File: db2ApiDf.h */
/* API: db2HADRStop */
/* ... */
SQL_API_RC SQL_API_FN
  db2HADRStop (
       db2Uint32 versionNumber,
       void * pParmStruct,
       struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2HADRStopStruct
{
  char                        *piDbAlias;
  char                        *piUserName;
  char                        *piPassword;
} db2HADRStopStruct;
```

汎用 API 構文:

```
/* File: db2ApiDf.h */
/* API: db2gHADRStop */
/* ... */
SQL_API_RC SQL_API_FN
  db2gHADRStop (
       db2Uint32 versionNumber,
       void * pParmStruct,
       struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2gHADRStopStruct
```

```
{
  char                      *piDbAlias;
  db2Uint32                 iAliasLen;
  char                      *piUserName;
  db2Uint32                 iUserNameLen;
  char                      *piPassword;
  db2Uint32                 iPasswordLen;
} db2gHADRStopStruct;
```

API パラメーター:

versionNumber
入力。 2 番目のパラメーター *pParmStruct* として渡される構造のバージョンとリリースのレベルを指定します。

pParmStruct
入力。 *db2HADRStopStruct* 構造を指すポインター。

pSqlca
出力。*sqlca* 構造へのポインター。

piDbAlias
入力。データベース別名を指すポインター。

iAliasLen
入力。データベース別名の長さ (バイト単位) を指定します。

piUserName
入力。コマンドを実行するときのユーザー名へのポインター。

iUserNameLen
入力。ユーザー名の長さ (バイト単位) を指定します。

piPassword
入力。パスワードを含むストリングを指すポインター。

iPasswordLen
入力。パスワードの長さ (バイト単位) を指定します。

関連資料:

- 97 ページの『db2HADRStart - HADR の開始』
- 100 ページの『db2HADRTakeover - 1 次データベースとしてのテークオーバー』
- 「コマンド・リファレンス」の『STOP HADR コマンド』

db2HADRTakeover - 1 次データベースとしてのテークオーバー

スタンバイ・データベースが 1 次データベースとしてテークオーバーすることを指示します。

許可:

以下のいずれかが必要です。

- *sysadm*
- *sysctrl*
- *sysmaint*

必要な接続:

インスタンス。データベース接続が存在しない場合、この API でデータベース接続が確立され、 API 完了時にそのデータベース接続がクローズされます。

API 組み込みファイル:

db2ApiDf.h

C API 構文:

```
/* File: db2ApiDf.h */
/* API: db2HADRTakeover */
/* ... */
SQL_API_RC SQL_API_FN
  db2HADRTakeover (
       db2Uint32 versionNumber,
       void * pParmStruct,
       struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2HADRTakeoverStruct
{
  char                        *piDbAlias;
  char                        *piUserName;
  char                        *piPassword;
  db2Uint16                   iByForce;
} db2HADRTakeoverStruct;
```

汎用 API 構文:

```
/* File: db2ApiDf.h */
/* API: db2gHADRTakeover */
/* ... */
SQL_API_RC SQL_API_FN
  db2gHADRTakeover (
       db2Uint32 versionNumber,
       void * pParmStruct,
       struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2gHADRTakeoverStruct
{
  char                        *piDbAlias;
  db2Uint32                   iAliasLen;
  char                        *piUserName;
  db2Uint32                   iUserNameLen;
  char                        *piPassword;
  db2Uint32                   iPasswordLen;
  db2Uint16                   iByForce;
} db2gHADRTakeoverStruct;
```

API パラメーター:

versionNumber
入力。 2 番目のパラメーター *pParmStruct* として渡される構造のバージョンとリリースのレベルを指定します。

pParmStruct
入力。 *db2HADRStartStruct* 構造を指すポインター。

pSqlca
出力。*sqlca* 構造へのポインター。

piDbAlias
入力。データベース別名を指すポインター。

iAliasLen
入力。データベース別名の長さ (バイト単位) を指定します。

piUserName
入力。コマンドを実行するときのユーザー名へのポインター。

iUserNameLen
入力。ユーザー名の長さ (バイト単位) を指定します。

piPassword
入力。パスワードを含むストリングを指すポインター。

iPasswordLen
入力。パスワードの長さ (バイト単位) を指定します。

iByForce
入力。有効な値は以下のとおりです。

DB2HADR_NO_FORCE
2 つのシステムが、通信の確立された対等状態である場合にのみ、テークオーバーが発生することを指定します。これにより、HADR 1 次データベースと HADR スタンバイ・データベースとの間で役割の逆転が生じます。

DB2HADR_FORCE
スタンバイ・データベースが、元の 1 次データベースがシャットダウンされたことの確認を待機せずに、 1 次データベースとしてテークオーバーすることを指定します。

関連資料:

- 97 ページの『db2HADRStart - HADR の開始』
- 99 ページの『db2HADRStop - HADR の停止』
- 「コマンド・リファレンス」の『TAKEOVER HADR コマンド』

db2HistoryCloseScan - 履歴ファイルのスキャンのクローズ

履歴ファイルのスキャンを終了し、スキャンに必要だった DB2 リソースを解放します。この API は、db2HistoryOpenScan の正常呼び出しの後でなければ使用できません。

許可:

なし

必要な接続:

インスタンス。この API を呼び出す前に、sqleatin を呼び出す必要はありません。

API 組み込みファイル:

db2ApiDf.h

C API 構文:

```
/* File: db2ApiDf.h */
/* API: db2HistoryCloseScan */
/* ... */
SQL_API_RC  SQL_API_FN
db2HistoryCloseScan (
  db2Uint32 version,
  void *piHandle,
  struct sqlca *pSqlca);
/* ... */
```

汎用 API 構文:

```
/* File: db2ApiDf.h */
/* API: db2GenHistoryCloseScan */
/* ... */
SQL_API_RC  SQL_API_FN
db2GenHistoryCloseScan (
  db2Uint32 version,
   void *piHandle,
  struct sqlca *pSqlca);
/* ... */
```

API パラメーター:

version
入力。 2 番目のパラメーター *piHandle* のバージョンとリリースのレベルを指定します。

piHandle
入力。 db2HistoryOpenScan によって戻された、スキャン・アクセス用のハンドルを指すポインターを指定します。

pSqlca
出力。*sqlca* 構造へのポインター。

REXX API 構文:

```
CLOSE RECOVERY HISTORY FILE :scanid
```

REXX API パラメーター:

scanid
OPEN RECOVERY HISTORY FILE SCAN から戻されたスキャン ID を含むホスト変数。

使用上の注意:

履歴ファイル API の使用の詳細については、『db2HistoryOpenScan』を参照してください。

関連資料:

- 214 ページの『db2Prune - 履歴ファイルの整理』
- 111 ページの『db2HistoryUpdate - 履歴ファイルの更新』
- 106 ページの『db2HistoryOpenScan - 履歴ファイルのスキャンのオープン』
- 104 ページの『db2HistoryGetEntry - 履歴ファイルの次項目の入手』
- 453 ページの『SQLCA』

関連サンプル:

- 『dbrecov.sqc -- How to recover a database (C)』
- 『dbrecov.sqC -- How to recover a database (C++)』

db2HistoryGetEntry - 履歴ファイルの次項目の入手

履歴ファイルの次項目を入手します。この API は、db2HistoryOpenScan の正常呼び出しの後でなければ使用できません。

許可:

なし

必要な接続:

インスタンス。この API を呼び出す前に、sqleatin を呼び出す必要はありません。

API 組み込みファイル:

db2ApiDf.h

C API 構文:

```
/* File: db2ApiDf.h */
/* API: db2HistoryGetEntry */
/* ... */
SQL_API_RC SQL_API_FN
db2HistoryGetEntry (
  db2Uint32 version,
  void *pDB2HistoryGetEntryStruct,
  struct sqlca *pSqlca);

typedef struct
{
  db2Uint16 iHandle,
  db2Uint16 iCallerAction,
  struct db2HistData *pioHistData
} db2HistoryGetEntryStruct;
/* ... */
```

汎用 API 構文:

```
/* File: db2ApiDf.h */
/* API: db2GenHistoryGetEntry */
/* ... */
SQL_API_RC SQL_API_FN
db2GenHistoryGetEntry (
  db2Uint32 version,
  void *pDB2GenHistoryGetEntryStruct,
  struct sqlca *pSqlca);

typedef struct
{
  db2Uint16 iHandle,
  db2Uint16 iCallerAction,
  struct db2HistData *pioHistData
} db2GenHistoryGetEntryStruct;
/* ... */
```

API パラメーター:

version

入力。 2 番目のパラメーター *pDB2HistoryGetEntryStruct* として渡される構造のバージョンとリリースのレベルを指定します。

pDB2HistoryGetEntryStruct

入力。 *db2HistoryGetEntryStruct* 構造を指すポインター。

pSqlca

出力。*sqlca* 構造へのポインター。

iHandle

入力。 db2HistoryOpenScan によって戻された、スキャン・アクセス用のハンドルが含まれます。

iCallerAction

入力。実行するアクションのタイプを指定します。有効な値 (db2ApiDf で定義) は、以下のとおりです。

DB2HISTORY_GET_ENTRY

次項目を入手しますが、コマンド・データはありません。

DB2HISTORY_GET_DDL

直前のフェッチからコマンド・データだけを入手します。

DB2HISTORY_GET_ALL

すべてのデータを含め、次項目を入手します。

pioHistData

入力。 *db2HistData* 構造を指すポインター。

REXX API 構文:

```
GET RECOVERY HISTORY FILE ENTRY :scanid [USING :value]
```

REXX API パラメーター:

scanid

OPEN RECOVERY HISTORY FILE SCAN から戻されたスキャン ID を含むホスト変数。

value 履歴ファイルの項目情報が戻されるコンパウンド REXX ホスト変数。以下の項目において、XXX はホスト変数名を表しています。

XXX.0	変数内の第 1 レベル・エレメントの数 (常に 15)
XXX.1	表スペース・エレメントの数
XXX.2	使用された表スペース・エレメントの数
XXX.3	OPERATION (実行された操作のタイプ)
XXX.4	OBJECT (操作の細分性)
XXX.5	OBJECT_PART (タイム・スタンプおよびシーケンス番号)
XXX.6	OPTYPE (操作の修飾子)
XXX.7	DEVICE_TYPE (使用された装置のタイプ)
XXX.8	FIRST_LOG (最初のログ ID)
XXX.9	LAST_LOG (現行のログ ID)

XXX.10	BACKUP_ID (バックアップ用の ID)
XXX.11	SCHEMA (表名の修飾子)
XXX.12	TABLE_NAME (ロードされた表の名前)
XXX.13.0	NUM_OF_TABLESPACES (バックアップまたはリストアに関係した表スペースの数)
XXX.13.1	最初にバックアップまたはリストアされた表スペースの名前
XXX.13.2	2 番目にバックアップまたはリストアされた表スペースの名前
XXX.13.3	以下同じ
XXX.14	LOCATION (バックアップまたはコピーが保管されている場所)
XXX.15	COMMENT (項目を記述するテキスト)

使用上の注意:

戻されるレコードは、db2HistoryOpenScan への呼び出しで指定した値を使用して選択されます。

履歴ファイル API の使用の詳細については、『db2HistoryOpenScan』を参照してください。

関連資料:

- 214 ページの『db2Prune - 履歴ファイルの整理』
- 111 ページの『db2HistoryUpdate - 履歴ファイルの更新』
- 106 ページの『db2HistoryOpenScan - 履歴ファイルのスキャンのオープン』
- 102 ページの『db2HistoryCloseScan - 履歴ファイルのスキャンのクローズ』
- 453 ページの『SQLCA』
- 439 ページの『db2HistData』

関連サンプル:

- 『dbrecov.sqc -- How to recover a database (C)』
- 『dbrecov.sqC -- How to recover a database (C++)』

db2HistoryOpenScan - 履歴ファイルのスキャンのオープン

履歴ファイルのスキャンを開始します。

許可:

なし

必要な接続:

インスタンス。データベースがリモートとしてカタログされている場合には、この API を呼び出す前に sqleatin を呼び出してください。

API 組み込みファイル:

db2ApiDf.h

C API 構文:

```
/* File: db2ApiDf.h */
/* API: db2HistoryOpenScan */
/* ... */
SQL_API_RC SQL_API_FN
db2HistoryOpenScan (
  db2Uint32 version,
  void *pDB2HistoryOpenStruct,
  struct sqlca *pSqlca);

typedef struct
{
  char *piDatabaseAlias,
  char *piTimestamp,
  char *piObjectName,
  db2Uint32 oNumRows,
  db2Uint32 oMaxTbspaces,
  db2Uint16 iCallerAction,
  db2Uint16 oHandle
} db2HistoryOpenStruct;
/* ... */
```

汎用 API 構文:

```
/* File: db2ApiDf.h */
/* API: db2GenHistoryOpenScan */
/* ... */
SQL_API_RC SQL_API_FN
db2GenHistoryOpenScan (
  db2Uint32 version,
  void *pDB2GenHistoryOpenStruct,
  struct sqlca *pSqlca);

typedef struct
{
  char *piDatabaseAlias,
  char *piTimestamp,
  char *piObjectName,
  db2Uint32 oNumRows,
  db2Uint32 oMaxTbspaces,
  db2Uint16 iCallerAction,
  db2Uint16 oHandle
} db2GenHistoryOpenStruct;
/* ... */
```

API パラメーター:

version
入力。 2 番目のパラメーター *pDB2HistoryOpenStruct* として渡される構造のバージョンとリリースのレベルを指定します。

pDB2HistoryOpenStruct
入力。 *db2HistoryOpenStruct* 構造を指すポインター。

pSqlca
出力。*sqlca* 構造へのポインター。

piDatabaseAlias
入力。データベース別名を含むストリングを指すポインター。

piTimestamp
入力。レコードの選択に使用されるタイム・スタンプを指定するストリングを指すポインター。この値と同じタイム・スタンプまたはこの値より大きいタイム・スタンプを持つレコードが選択されます。このパラメーターを NULL に設定するか、ゼロを指すようにすれば、タイム・スタンプを用いての項目のフィルターを実行しないようすることができます。

piObjectName
入力。レコードの選択に使用されるオブジェクト名を指定するストリングを指すポインター。オブジェクトとして表または表スペースを使用できます。オブジェクトが表の場合、表の完全修飾名を指定する必要があります。このパラメーターを NULL に設定するか、ゼロを指すようにすれば、オブジェクト名を用いての項目のフィルターを実行しないようにすることができます。

oNumRows
出力。 API からの戻り時に、このパラメーターには、一致した履歴ファイルの項目の数が入ります。

oMaxTbspaces
出力。任意の履歴項目で保管された表スペース名の最大数。

iCallerAction
入力。実行するアクションのタイプを指定します。有効な値 (db2ApiDf で定義) は、以下のとおりです。

DB2HISTORY_LIST_HISTORY
現在履歴ファイルの中に記録されているイベントのすべてのリストを表示します。

DB2HISTORY_LIST_BACKUP
バックアップ操作およびリストア操作をリストします。

DB2HISTORY_LIST_ROLLFORWARD
ロールフォワード操作をリストします。

DB2HISTORY_LIST_DROPPED_TABLE
ドロップした表レコードをリストします。項目と関連付けられた DDL フィールドは戻されません。項目の DDL 情報を検索するには、この項目が取り出された直後に、呼び出しアクション DB2HISTORY_GET_DDL を指定して db2HistoryGetEntry を呼び出す必要があります。

DB2HISTORY_LIST_LOAD
ロード操作をリストします。

DB2HISTORY_LIST_CRT_TABLESPACE
表スペースの作成およびドロップ操作をリストします。

DB2HISTORY_LIST_REN_TABLESPACE
表スペースの名前変更操作をリストします。

DB2HISTORY_LIST_ALT_TABLESPACE
表スペースの変更操作をリストします。項目と関連付けられた DDL フィールドは戻されません。項目の DDL 情報を検索するには、こ

の項目が取り出された直後に、呼び出しアクション DB2HISTORY_GET_DDL を指定して db2HistoryGetEntry を呼び出す必要があります。

DB2HISTORY_LIST_REORG
REORGANIZE TABLE 操作をリストします。この値は、現在サポートされていません。

oHandle
出力。 API からの戻り時に、このパラメーターには、スキャン・アクセス用のハンドルが入れられます。このハンドルは、その後 db2HistoryGetEntry および db2HistoryCloseScan で使用されます。

REXX API 構文:

```
OPEN [BACKUP] RECOVERY HISTORY FILE FOR database_alias
[OBJECT objname] [TIMESTAMP :timestamp]
USING :value
```

REXX API パラメーター:

database_alias
履歴ファイルがリストされる、データベースの別名です。

objname
レコードの選択に使用されるオブジェクト名を指定します。オブジェクトとして表または表スペースを使用できます。オブジェクトが表の場合、表の完全修飾名を指定する必要があります。このパラメーターを NULL に設定すれば、 *objname* を使用しての項目のフィルターを実行しないようにすることができます。

timestamp
レコードの選択に使用されるタイム・スタンプを指定します。この値と同じタイム・スタンプまたはこの値より大きいタイム・スタンプを持つレコードが選択されます。このパラメーターを NULL に設定すれば、 *timestamp* を使用しての項目のフィルターを実行しないようにすることができます。

value 履歴ファイル情報が戻されるコンパウンド REXX ホスト変数です。以下の項目において、XXX はホスト変数名を表しています。

XXX.0 変数内のエレメント数 (常に 2)。

XXX.1 将来のスキャン・アクセスに使用される ID (ハンドル)。

XXX.2 一致した履歴ファイル項目の数

使用上の注意:

タイム・スタンプ、オブジェクト名、および呼び出し側アクションの組み合わせを使用して、レコードをフィルターにかけることもできます。指定したすべてのフィルターを通過するレコードだけが戻されます。

オブジェクト名のフィルター操作の結果は、指定した値によって異なります。

- 表を指定した場合、ロード操作に関するレコードだけが戻されます (これが履歴ファイル内の表に関する唯一の情報であるため)。

- 表スペースを指定した場合、その表スペースに関するバックアップ、リストア操作、およびロード操作に関するレコードが戻されます。

注: 表のレコードを戻すには、その表を *schema.tablename* として指定しなければなりません。 *tablename* を指定した場合は、表スペースのレコードしか戻されません。

1 つのプロセスで、最大 8 つの履歴ファイル・スキャンが許可されています。

履歴ファイル中のすべての項目をリストする場合、通常のアプリケーションであれば、以下のステップを実行します。

1. db2HistoryOpenScan を呼び出す *oNumRows* が戻されます。
2. *db2HistData* 構造に、 *n* 個の *oTablespace* フィールド用のスペースを割り振る。 *n* は任意の数値です。
3. *db2HistData* 構造の *iDB2NumTablespace* フィールドを *n* に設定する。
4. ループの中で、以下を実行してください。
   - db2HistoryGetEntry を呼び出して履歴ファイルから取り出しを行います。
   - db2HistoryGetEntry によって、 SQL_RC_OK という SQLCODE が戻されたら、 *db2HistData* 構造の *sqld* フィールドを使用して、戻された表スペース項目の数を判別します。
   - db2HistoryGetEntry によって、 SQLUH_SQLUHINFO_VARS_WARNING という SQLCODE が戻された場合は、 DB2 が戻そうとしている表スペースのために十分なスペースが割り振られていません。この場合は、スペースをいったん解放し、 *db2HistData* 構造に *oDB2UsedTablespace* の表スペース項目にとって十分なスペースを割り振り直し、 *iDB2NumTablespace* を *oDB2UsedTablespace* に設定してください。
   - db2HistoryGetEntry によって SQLE_RC_NOMORE という SQLCODE が戻された場合は、すべての履歴ファイル項目が検索されています。
   - 他の SQLCODE は、特定の問題が生じたことを示します。その指示に従ってください。
5. すべての情報の取り出しが終了したら、db2HistoryCloseScan を呼び出して、db2HistoryOpenScan の呼び出しに伴って割り振られたリソースを解放します。

db2HistData 構造の、 *n* 個の *oTablespace* フィールド用のスペースに必要とされるメモリーの量を判別しやすくするため、 (sqlutil で定義された) マクロ SQLUHINFOSIZE (*n*) が用意されています。

関連資料:

- 214 ページの『db2Prune - 履歴ファイルの整理』
- 111 ページの『db2HistoryUpdate - 履歴ファイルの更新』
- 104 ページの『db2HistoryGetEntry - 履歴ファイルの次項目の入手』
- 102 ページの『db2HistoryCloseScan - 履歴ファイルのスキャンのクローズ』
- 453 ページの『SQLCA』

関連サンプル:

- 『dbrecov.sqc -- How to recover a database (C)』

- 『dbrecov.sqC -- How to recover a database (C++)』

db2HistoryUpdate - 履歴ファイルの更新

履歴ファイル項目にあるロケーション、装置タイプ、またはコメントを更新します。

許可:

以下のいずれかが必要です。

- *sysadm*
- *sysctrl*
- *sysmaint*
- *dbadm*

必要な接続:

データベース。デフォルトのデータベース以外のデータベースの履歴ファイル内にある項目を更新する場合は、この API を呼び出す前に、そのデータベースへの接続を確立しておく必要があります。

API 組み込みファイル:

db2ApiDf.h

C API 構文:

```
/* File: db2ApiDf.h */
/* API: db2HistoryUpdate */
/* ... */
SQL_API_RC SQL_API_FN
db2HistoryUpdate (
  db2Uint32 version,
  void *pDB2HistoryUpdateStruct,
  struct sqlca *pSqlca);

typedef SQL_STRUCTURE db2HistoryUpdateStruct
{
   char          *piNewLocation;
   char          *piNewDeviceType;
   char          *piNewComment;
   char          *piNewStatus;
   db2HistoryEID iEID;
} db2HistoryUpdateStruct;

/* Structure db2HistoryEID */
typedef SQL_STRUCTURE db2HistoryEID
{
   SQL_PDB_NODE_TYPE ioNode;
   db2Uint32         ioHID;
} db2HistoryEID;

/* ... */
```

汎用 API 構文:

```
/* File: db2ApiDf.h */
/* API: db2gHistoryUpdate */
/* ... */
SQL_API_RC SQL_API_FN
db2GenHistoryUpdate (
```

```
  db2Uint32 version,
  void *pDB2GenHistoryUpdateStruct,
  struct sqlca *pSqlca);

typedef SQL_STRUCTURE db2gHistoryUpdateStruct
{
   char                                *piNewLocation;
   char                                *piNewDeviceType;
   char                                *piNewComment;
   char                                *piNewStatus;
   db2Uint32                           iNewLocationLen;
   db2Uint32                           iNewDeviceLen;
   db2Uint32                           iNewCommentLen;
   db2Uint32                           iNewStatusLen;
   db2HistoryEID                       iEID;
} db2gHistoryUpdateStruct;
/* ... */
```

API パラメーター:

version
入力。 2 番目のパラメーター *pDB2HistoryUpdateStruct* として渡される構造のバージョンとリリースのレベルを指定します。

pDB2HistoryUpdateStruct
入力。 *db2HistoryUpdateStruct* 構造を指すポインター。

pSqlca
出力。*sqlca* 構造へのポインター。

piNewLocation
入力。バックアップ、リストア、またはロード・コピー・イメージ用の新規ロケーションを指定するストリングを指すポインター。このパラメーターを NULL に設定するか、ゼロを指すようにすれば、値は変更されません。

piNewDeviceType
入力。バックアップ、リストア、またはロード・コピー・イメージを格納するための新規装置タイプを指定するストリングを指すポインター。このパラメーターを NULL に設定するか、ゼロを指すようにすれば、値は変更されません。

piNewComment
入力。項目について説明する新規のコメントを指定するストリングを指すポインター。このパラメーターを NULL に設定するか、ゼロを指すようにすれば、コメントは変更されません。

piNewStatus
入力。項目の新規の状況タイプを指定するストリングを指すポインター。このパラメーターを NULL に設定するか、ゼロを指すようにすれば、状況は変更されません。

iNewLocationLen
入力。 piNewLocationLen フィールドの長さ。

iNewDeviceLen
入力。 piNewDeviceLen フィールドの長さ。

iNewCommentLen
入力。 piNewCommentLen フィールドの長さ。

iNewStatusLen
入力。 piNewStatusLen フィールドの長さ。

iEID 入力。履歴ファイルの特定の項目を更新するときに使用できる、ユニークな ID です。

ioNode
このパラメーターは、入力パラメーターまたは出力パラメーターのどちらにでも使用できます。

ノード番号を示します。

ioHID このパラメーターは、入力パラメーターまたは出力パラメーターのどちらにでも使用できます。

ローカル履歴ファイルの項目 ID を示します。

REXX API 構文:

```
UPDATE RECOVERY HISTORY USING :value
```

REXX API パラメーター:

value 履歴ファイル項目の新規ロケーションに関する情報を含む、コンパウンド REXX ホスト変数です。以下の項目において、XXX はホスト変数名を表しています。

XXX.0 変数内のエレメント数 (必ず 1 から 4)

XXX.1 OBJECT_PART (タイム・スタンプとシーケンス番号 001 から 999)

XXX.2 バックアップまたはコピー・イメージの新規ロケーション (このパラメーターはオプションです)

XXX.3 バックアップまたはコピー・イメージの保管に使用される新規装置 (このパラメーターはオプションです)

XXX.4
新規コメント (このパラメーターはオプションです)

使用上の注意:

この API は更新関数であり、変更前の情報はすべて新しい情報に置き換えられ、再作成することができなくなります。これらの変更は記録されません。

データベース履歴ファイルの主な用途は情報を記録することですが、履歴に含まれるデータは、自動リストア操作で直接に使用されます。 AUTOMATIC オプションを指定したリストアにおいては、リストア・ユーティリティーによりバックアップ・イメージとそのロケーションの履歴が参照および使用されることにより、自動リストア要求が処理されます。自動リストア機能を使用する場合に、バックアップ・イメージが作成されて以来に再配置されているなら、現在のロケーションを反映するよう、データベース履歴レコードを更新することをお勧めします。データベース履歴の中のバックアップ・イメージのロケーションが更新されない場合、自動リストア処理においてはバックアップ・イメージを見つけることができなくなりますが、手動リストア・コマンドは正常に使用できます。

関連資料:

- 257 ページの『db2Rollforward - データベースのロールフォワード』
- 214 ページの『db2Prune - 履歴ファイルの整理』
- 106 ページの『db2HistoryOpenScan - 履歴ファイルのスキャンのオープン』
- 104 ページの『db2HistoryGetEntry - 履歴ファイルの次項目の入手』
- 102 ページの『db2HistoryCloseScan - 履歴ファイルのスキャンのクローズ』
- 453 ページの『SQLCA』
- 「コマンド・リファレンス」の『UPDATE HISTORY FILE コマンド』
- 28 ページの『db2Backup - データベースのバックアップ』

関連サンプル:

- 『dbrecov.sqc -- How to recover a database (C)』
- 『dbrecov.sqC -- How to recover a database (C++)』

db2Import - インポート

サポートされているファイル形式を用いて、外部ファイルから表、階層、またはビューにデータを挿入します。 Load はより高速にこれを行うことができますが、ロード・ユーティリティーは、階層レベルでのデータのロードをサポートしません。

許可:

- INSERT オプションを使用して IMPORT する場合、以下のいずれかが必要です。
  - *sysadm*
  - *dbadm*
  - 関係するそれぞれの表またはビューに対する CONTROL 特権
  - 関係するそれぞれの表またはビューに対する INSERT および SELECT 特権
- INSERT_UPDATE オプションを使用して既存の表に IMPORT するには、以下のいずれかが必要です。
  - *sysadm*
  - *dbadm*
  - 表またはビューに対する CONTROL 特権
  - 関係するそれぞれの表またはビューに対する INSERT、 SELECT、 UPDATE、および DELETE 特権
- REPLACE または REPLACE_CREATE オプションを使用して既存の表に IMPORT するには、以下のいずれかが必要です。
  - *sysadm*
  - *dbadm*
  - 表またはビューに対する CONTROL 特権
  - 表またはビューに対する INSERT、SELECT、および DELETE 特権
- CREATE または REPLACE_CREATE オプションを使用して新規の表に IMPORT するには、以下のいずれかが必要です。
  - *sysadm*
  - *dbadm*

  - データベースに対する CREATETAB 権限および表スペースに対する USE 特権に加えて、以下のいずれか。
    - データベースに対する IMPLICIT_SCHEMA 権限 (表の暗黙的または明示的スキーマ名が存在しない場合)
    - スキーマに対する CREATIN 特権 (表のスキーマ名が既存のスキーマを指す場合)
- CREATE または REPLACE_CREATE オプションを使って、存在しない表または階層に IMPORT するには、以下のいずれかが必要です。
  - *sysadm*
  - *dbadm*
  - データベースに対する CREATETAB 権限と、次のいずれか
    - データベースに対する IMPLICIT_SCHEMA 権限 (表のスキーマ名が存在しない場合)
    - スキーマに対する CREATEIN 特権 (表のスキーマが存在する場合)
    - 階層全体に対して REPLACE_CREATE オプションが使用されている場合は、階層内のすべての副表に対する CONTROL 特権
- REPLACE オプションを使用して既存の階層に IMPORT するには、以下のどれかが必要です。
  - *sysadm*
  - *dbadm*
  - 階層内のすべての副表に対する CONTROL 特権

必要な接続:

データベース。暗黙的な接続が可能である場合には、デフォルトのデータベースへの接続が確立されます。

API 組み込みファイル:

db2ApiDf.h

C API 構文:

```
/* db2Import - API */
SQL_API_RC SQL_API_FN
  db2Import (
    db2Uint32 versionNumber,
    void * pParmStruct,
    struct sqlca * pSqlca);

/* db2Import parameter structure */
typedef SQL_STRUCTURE db2ImportStruct
{
   char                                *piDataFileName;
   struct sqlu_media_list              *piLobPathList;
   struct sqldcol                      *piDataDescriptor;
   struct sqlchar                      *piActionString;
   char                                *piFileType;
   struct sqlchar                      *piFileTypeMod;
   char                                *piMsgFileName;
   db2int16                            iCallerAction;
   struct db2ImportIn                  *piImportInfoIn;
   struct db2ImportOut                 *poImportInfoOut;
   db2int32                            *piNullIndicators;
} db2ImportStruct;
```

```
| /* Import input structure */
| typedef SQL_STRUCTURE db2ImportIn
| {
|    db2Uint64                          iRowcount;
|    db2Uint64                          iRestartcount;
|    db2Uint64                          iSkipcount;
|    db2int32                           *piCommitcount;
|    db2Uint32                          iWarningcount;
|    db2Uint16                          iNoTimeout;
|    db2Uint16                          iAccessLevel;
| } db2ImportIn;
|
| /* Import output structure */
| typedef SQL_STRUCTURE db2ImportOut
| {
|    db2Uint64                          oRowsRead;
|    db2Uint64                          oRowsSkipped;
|    db2Uint64                          oRowsInserted;
|    db2Uint64                          oRowsUpdated;
|    db2Uint64                          oRowsRejected;
|    db2Uint64                          oRowsCommitted;
| } db2ImportOut;
```

汎用 API 構文:

```
| /* db2gImport - Generic API */
| SQL_API_RC SQL_API_FN
|   db2gImport (
|    db2Uint32 versionNumber,
|    void * pParmStruct,
|    struct sqlca * pSqlca);
|
| /* db2gImport parameter structure */
| typedef SQL_STRUCTURE db2gImportStruct
| {
|    char                               *piDataFileName;
|    struct sqlu_media_list             *piLobPathList;
|    struct sqldcol                     *piDataDescriptor;
|    struct sqlchar                     *piActionString;
|    char                               *piFileType;
|    struct sqlchar                     *piFileTypeMod;
|    char                               *piMsgFileName;
|    db2int16                           iCallerAction;
|    struct db2gImportIn                *piImportInfoIn;
|    struct dbg2ImportOut               *poImportInfoOut;
|    db2int32                           *piNullIndicators;
|    db2Uint16                          iDataFileNameLen;
|    db2Uint16                          iFileTypeLen;
|    db2Uint16                          iMsgFileNameLen;
| } db2gImportStruct;
|
| /* Generic Import input structure */
| typedef SQL_STRUCTURE db2gImportIn
| {
|    db2Uint64                          iRowcount;
|    db2Uint64                          iRestartcount;
|    db2Uint64                          iSkipcount;
|    db2int32                           *piCommitcount;
|    db2Uint32                          iWarningcount;
|    db2Uint16                          iNoTimeout;
|    db2Uint16                          iAccessLevel;
| } db2gImportIn;
|
| /* Generic Import output structure */
| typedef SQL_STRUCTURE db2gImportOut
| {
|    db2Uint64                          oRowsRead;
```

```
   db2Uint64                       oRowsSkipped;
   db2Uint64                       oRowsInserted;
   db2Uint64                       oRowsUpdated;
   db2Uint64                       oRowsRejected;
   db2Uint64                       oRowsCommitted;
} db2gImportOut;
```

API パラメーター:

versionNumber
入力。 2 番目のパラメーター *pParmStruct* として渡される構造のバージョンとリリースのレベルを指定します。

pParmStruct
入力/出力。 *db2ImportStruct* 構造を指すポインター。

pSqlca
出力。*sqlca* 構造へのポインター。

piDataFileName
入力。データがインポートされるパスおよび外部入力ファイル名を含むストリングを指定します。

piLobPathList
入力。 *media_type* SQLU_LOCAL_MEDIA を使用する *sqlu_media_list*、および LOB ファイルがあるクライアント上のパスをリストする *sqlu_media_entry* 構造を示します。

piDataDescriptor
入力。外部ファイルからインポートするよう選択された列に関する情報を含む *sqldcol* 構造を指すポインターです。 *dcolmeth* フィールドの値によって、このパラメーターに提供される残りの情報をインポート・ユーティリティーがどのように解釈するかが判別されます。このパラメーターに有効な値は以下のとおりです。

SQL_METH_N
名前。外部入力ファイルの列は、列名によって選択されます。

SQL_METH_P
位置。外部入力ファイルの列は、列の位置によって選択されます。

SQL_METH_L
ロケーション。外部入力ファイルの列は、列のロケーションによって選択されます。以下のいずれかの条件のために無効であるロケーションの対を指定したインポート呼び出しは、データベース・マネージャーによってリジェクトされます。

- 開始または終了ロケーションが有効な範囲 (1 から 符号付き 2 バイト整数の最大値) に入っていない場合。
- 終了ロケーションが開始ロケーションよりも小さい場合。
- ロケーションの対により定義された入力列幅に、ターゲット列のタイプおよび長さとの互換性がない場合。

開始ロケーションと終了ロケーションの対がゼロに等しい場合は、NULL 可能列が NULL で埋め込まれることを示します。

SQL_METH_D
デフォルト。 *piDataDescriptor* が NULL の場合、または SQL_METH_D に設定されている場合には、外部入力ファイルの列のデフォルト選択が実行されます。この場合、列数および列指定配列は、どちらも無視されます。 DEL、IXF、または WSF ファイルの場合、外部入力ファイルにある最初の *n* 個の列のデータは、そのままの順序で取り出されます。 *n* は、データがインポートされるデータベース列の数です。

piActionString
入力。 2 バイトの長さフィールドと、データをインポートする列を識別する文字の配列を含む *sqlchar* 構造を指すポインターです。

文字配列の形式は、以下のようになります。

```
{INSERT | INSERT_UPDATE | REPLACE | CREATE | REPLACE_CREATE}
INTO {tname[(tcolumn-list)] |
[{ALL TABLES | (tname[(tcolumn-list)][, tname[(tcolumn-list)]])}]
[IN] HIERARCHY {STARTING tname | (tname[, tname])}
[UNDER sub-table-name | AS ROOT TABLE]}
[DATALINK SPECIFICATION datalink-spec]
```

INSERT
既存の表データを変更することなく、インポートされたデータを表に追加します。

INSERT_UPDATE
主キー値が表にない場合はインポートした行を追加し、主キー値がある場合はそれらの行を更新に使用します。このオプションは、ターゲット表に 主キーがあり、指定された (または暗黙指定された) インポートされるターゲット列のリストに、主キーのすべての列が組み込まれているときのみ有効です。このオプションは、ビューには適用されません。

REPLACE
表オブジェクトを切り捨てることによって表から既存データをすべて削除し、インポートされたデータを挿入します。表定義および索引定義は変更されません。 (indexixf が *FileTypeMod* に入っていて、 *FileType* が SQL_IXF である場合、索引は削除および置換されます。) 表がまだ定義されていない場合には、エラーが戻されます。

重要: 既存のデータを削除した後にエラーが発生した場合、そのデータは失われてしまいます。

CREATE
指定された表が定義されていない場合に、指定された PC/IXF ファイルにある情報を使用して表定義と行の内容が作成されます。データベース・マネージャーによりファイルが事前にエクスポートされている場合、索引も作成されます。指定された表がすでに存在している場合、エラーが戻されます。このオプションは、PC/IXF ファイル形式にのみ有効です。

REPLACE_CREATE
指定された表が定義されている場合に、 PC/IXF ファイルにある

PC/IXF 行情報を使用して表の内容が置き換えられます。表がまだ定義されていない場合は、指定された PC/IXF ファイルにある情報を使用して表定義と行の内容が作成されます。 DB2 により PC/IXF ファイルが事前にエクスポートされている場合は、索引も作成されます。このオプションは、PC/IXF ファイル形式にのみ有効です。

重要: 既存のデータを削除した後にエラーが発生した場合、そのデータは失われてしまいます。

tname データが挿入される表、型付き表、ビュー、またはオブジェクト・ビューの名前。下位のサーバーの場合を除き、REPLACE、INSERT_UPDATE、または INSERT には、修飾または非修飾の名前を使わなければならないようなときでも、別名を指定することができます。ビューの場合、読み取り専用ビューにすることはできません。

tcolumn-list
データが挿入される先の表またはビュー内にある列名のリスト。列名は、コンマで区切らなければなりません。列名が指定されない場合、 CREATE TABLE または ALTER TABLE ステートメントで定義された列名が使用されます。型付き表に指定されている列のリストがない場合、それぞれの副表のすべての列にデータが挿入されます。

sub-table-name
CREATE オプションで 1 つまたは複数の副表を作成する際に、親表を指定します。

ALL TABLES
階層専用の暗黙キーワード。階層をインポートする際、デフォルトでは*全探索順序リスト* で指定されているすべての表がインポートされます。

HIERARCHY
階層データをインポートするよう指定します。

STARTING
階層専用のキーワード。指定された副表の名前から開始して、デフォルト順序を使用するよう指定します。

UNDER
階層および CREATE 専用のキーワード。新しい階層、副階層、または副表を、指定された副表の下に作成するよう指定します。

AS ROOT TABLE
階層および CREATE 専用のキーワード。新しい階層、副階層、または副表を、独立型の階層として作成するよう指定します。

DATALINK SPECIFICATION *datalink-spec*
DB2 Data Links Manager に関連するパラメーターを指定します。これらのパラメーターは、IMPORT コマンドと同じ構文を使って指定できます。

tname および *tcolumn-list* は、 SQL INSERT ステートメントの *tablename* および *colname* リストに対応し、同一の制限の下にあります。

tcolumn-list 内の列と、外部列 (指定または暗黙指定された) とは、リストまたは構造における位置に従って対応付けられます (*sqldcol* 構造で指定された最初の列からのデータは、 *tcolumn-list* の最初のエレメントに対応する表またはビュー・フィールドに挿入されます)。

異なる数の列が指定された場合、実際に処理される列の数は 2 つのうちの小さい方です。このことにより、エラーが発生する (一部の NULL 不可の表フィールドに入れるべき値がないため) か、または情報メッセージが表示される (一部の外部ファイル列が無視されるため) 可能性があります。

piFileType

入力。外部ファイル内のデータの形式を示すストリングを指定します。サポートされている外部ファイル形式は、以下のとおりです。

SQL_ASC
区切り文字なし ASCII。

SQL_DEL
区切り文字付き ASCII。これは dBase プログラム、BASIC プログラム、 IBM パーソナル・デシジョン・シリーズ・プログラム、およびその他の多数のデータベース・マネージャー/ファイル・マネージャーとの交換のための形式です。

SQL_IXF
IXF (統合交換フォーマットの PC バージョン)。表からデータをエクスポートする場合の推奨方式で、同じ表または別のデータベース・マネージャー表にそれを再インポートすることが可能です。

SQL_WSF
ワークシート形式。 Lotus Symphony および 1-2-3 プログラムとの交換のための形式です。

piFileTypeMod

入力。 2 バイトの長さフィールドと、 1 つまたは複数の処理オプションを指定する文字の配列を含む構造を指すポインターです。このポインターが NULL であるか、このポインターが指す構造に 1 文字も入っていない場合、このアクションはデフォルトの指定が選択されたものとして解釈されます。

サポートされるすべてのファイル・タイプに、すべてのオプションを使用できるわけではありません。インポート用のファイル・タイプ修飾子 を参照してください。

piMsgFileName

入力。このユーティリティーが戻すエラー、警告、および情報メッセージの宛先を含むストリングを指定します。オペレーティング・システム・ファイルまたは標準装置のパスおよび名前を指定できます。ファイルがすでに存在する場合は、そのファイルが付加されます。存在していない場合は、新たに作成されます。

iCallerAction
入力。呼び出し側が要求するアクションを示します。有効な値は以下のとおりです。

SQLU_INITIAL
最初の呼び出し。この値は、API への最初の呼び出しの際には必ず使用してください。

最初の呼び出しまたは後続の呼び出しのいずれかが戻され、要求されたインポート操作が完了する前に呼び出し側のアプリケーションが何らかのアクションを行うことが必要な場合、呼び出し側のアクションを以下のどちらかに設定しなければなりません。

SQLU_CONTINUE
処理の継続。この値を使用できるのは、最初の呼び出しが戻されたときにユーティリティーがユーザー入力 (たとえば、テープの終わり条件への応答) を要求した後で、 API への後続呼び出しを出す場合だけです。この値は、ユーティリティーが要求したユーザー・アクションが完了したら、ユーティリティーが最初の要求の処理を続行するよう指定するものです。

SQLU_TERMINATE
処理の終了。この値を使用できるのは、最初の呼び出しが戻されたときにユーティリティーがユーザー入力 (たとえば、テープの終わり条件への応答) を要求した後で、 API への後続呼び出しを出す場合だけです。この値は、ユーティリティーが要求したユーザー・アクションが実行されなかった場合、ユーティリティーが最初の要求の処理を中断するよう指定するものです。

piImportInfoIn
入力。 *db2ImportIn* 構造を指すポインター。

poImportInfoOut
出力。 *db2ImportOut* 構造を指すポインター。

piNullIndicators
入力。 ASC ファイルの場合にのみ使用します。列データが NULL 可能であるかどうかを示す整数の配列です。この配列内のエレメント数は、入力ファイル内の列数と一致していなければなりません。この配列のエレメントとデータ・ファイルからインポートされる列との間には、 1 対 1 の順序付けられた対応関係があります。このため、エレメントの数は、*piDataDescriptor* パラメーターの *dcolnum* フィールドと同じでなければなりません。配列の各エレメントには、NULL 標識フィールドとして使用される、データ・ファイル内の列を識別する数値、または表の列が NULL 可能ではないことを示すゼロが含まれます。エレメントがゼロでない場合は、データ・ファイル内の識別された列に、 Y と N のどちらかが含まれているはずです。 Y は表列のデータが NULL であることを示し、 N は表列のデータが NULL ではないことを示します。

iRowcount
入力。ロードされる物理レコードの数。これを使用すると、ファイル内の最初の *iRowcount* 個の行だけをロードすることができます。 *iRowcount* が 0 の場合、インポートではファイルのすべての行が処理されます。

iSkipcount
入力。レコードを挿入または更新する前にスキップするレコードの数。 *iRestartcount* と同じ機能です。

piCommitcount
入力。データベースにコミットする前にインポートするレコードの数。 *piCommitcount* 個のレコードがインポートされるたびに、コミットが実行されます。 NULL 値は、デフォルトのコミット・カウント値 (オフライン・インポートの場合はゼロ、オンライン・インポートの場合は AUTOMATIC) を指定します。 Commitcount AUTOMATIC は、DB2IMPORT_COMMIT_AUTO の値を渡すことによって指定されます。

iWarningcount
入力。 *iWarningcount* 個の警告後に、インポート操作を停止します。このパラメーターは、警告は予期されないが、正しいファイルと表が使用されていることを確認するのが望ましい場合に設定してください。インポート・ファイルまたはターゲット表が不適切に指定されると、インポート対象の各行ごとにインポート・ユーティリティーによって警告が生成され、このためにインポートが失敗する可能性があります。 *iWarningcount* が 0 の場合、またはこのオプションが指定されていない場合、警告が何度出されてもインポート操作は続行します。

iNoTimeout
入力。インポート・ユーティリティーがロックの待機中にタイムアウトしないことを指定します。このオプションは、 *locktimeout* データベース構成パラメーターに置き換わります。他のアプリケーションは影響を受けません。有効な値は以下のとおりです。

DB2IMPORT_LOCKTIMEOUT
locktimeout 構成パラメーターの値を優先することを示します。

DB2IMPORT_NO_LOCKTIMEOUT
タイムアウトしないことを示します。

iAccessLevel
入力。アクセス・レベルを指定します。有効な値は以下のとおりです。

SQLU_ALLOW_NO_ACCESS
インポート・ユーティリティーが表を排他ロックすることを指定します。

SQLU_ALLOW_WRITE_ACCESS
インポート中も、表内のデータに対する読み取りまたは書き込みアクセスが可能であることを指定します。

oRowsRead
出力。インポート中にファイルから読み取られたレコードの数。

oRowsSkipped
出力。挿入または更新を開始する前にスキップしたレコードの数。

oRowsInserted
出力。ターゲット表に挿入された行の数。

oRowsUpdated
出力。インポートされたレコード (主キーの値がすでに表内に存在するレコード) からの情報によって更新された、ターゲット表内の行数。

oRowsRejected
出力。インポートできなかったレコードの数。

oRowsCommitted
出力。正常にインポートされ、データベースにコミット済みのレコード数。

使用上の注意:

インポート操作を開始する前に、すべての表操作が完了し、すべてのロックが解除されていることを確認してください。このことは、 WITH HOLD でオープンしているカーソルをすべてクローズした後に COMMIT を発行するか、または ROLLBACK を発行することにより、行うことができます。

インポート・ユーティリティーは、 SQL INSERT ステートメントを使用してターゲット表に行を追加します。このユーティリティーは、入力ファイル中の各行のデータにつき 1 つずつ INSERT ステートメントを発行します。 INSERT ステートメントが失敗した場合、以下の 2 通りの結果のいずれかになります。

- 後続の INSERT ステートメントが成功すると予測される場合には、警告メッセージがメッセージ・ファイルに書き込まれ、処理が継続されます。
- 後続の INSERT ステートメントが失敗すると予測され、データベースが損傷する可能性がある場合には、エラー・メッセージが書き込まれ、処理が停止されます。

このユーティリティーは、 REPLACE または REPLACE_CREATE 操作時に以前の行が削除された後、自動 COMMIT を実行します。したがって、表オブジェクトが切り捨てられた後、システムに障害が起こったり、アプリケーションがデータベース・マネージャーに割り込んだりすると、元のデータのすべてが失われてしまいます。これらのオプションを使用する前に、元のデータがもはや必要ないことを確認してください。

CREATE、REPLACE、または REPLACE_CREATE 操作時にログが満杯になると、このユーティリティーは挿入されたレコードに対して自動 COMMIT を実行します。自動 COMMIT の後に、システムに障害が起こるか、またはアプリケーションがデータベース・マネージャーに割り込むと、部分的にデータの挿入された表はデータベース内に残ります。 REPLACE または REPLACE_CREATE オプションを使用してインポート操作全体を再実行するか、 *iRestartcount* パラメーターを正常にインポートされた行の数に設定して INSERT を使用してください。

デフォルトでは、 INSERT または INSERT_UPDATE オプションについては自動 COMMIT は実行されません。ただし、**piCommitcount* パラメーターがゼロでない場合は実行されます。ログが満杯であれば、ROLLBACK が実行されます。

インポート・ユーティリティーが COMMIT を実行するたびに、 2 つのメッセージがメッセージ・ファイルに書き込まれます。一方は、コミットされるレコードの数を示し、もう一方は、 COMMIT の成功後に書き込まれます。障害の後にインポート操作を再開するときには、スキップするレコードの数 (最後の正常なコミットから判別される) を指定してください。

インポート・ユーティリティーでは、多少の非互換性問題がある入力データは受け入れられます (たとえば、文字データは埋め込みまたは切り捨てを用いてインポートできます。数値データは異なる数値データ・タイプを用いてインポートできます)。しかし、大きな非互換性問題のあるデータは受け入れられません。

オブジェクト表に何らかの従属 (それ自体への従属は除く) がある場合は、そのオブジェクト表を REPLACE または REPLACE_CREATE することはできません。また、オブジェクト・ビューの基本表に何らかの従属 (それ自体への従属を含む) がある場合は、そのオブジェクト・ビューを REPLACE または REPLACE_CREATE することはできません。そのような表またはビューを置換するには、以下のとおりに行ってください。

1. その表が親となっているすべての外部キーをドロップします。
2. インポート・ユーティリティーを実行します。
3. 表を変更して、外部キーを再作成します。

外部キーの再作成中にエラーが発生する場合、参照保全を保守するためにデータを変更してください。

PC/IXF ファイルから表を作成するときは、参照制約と外部キー定義は保存されません。 (以前に SELECT * を使用してエクスポートされたデータの場合、主キー定義は*保持されます*。)

リモート・データベースへのインポートでは、サーバーに、入力データ・ファイルのコピー、出力メッセージ・ファイル、およびデータベースのサイズ拡大を見込んだ十分なディスク・スペースが必要とされます。

インポート操作がリモート・データベースに対して実行され、出力メッセージ・ファイルが非常に長くなった (60 KB を超過) 場合、クライアントのユーザーに戻されるメッセージ・ファイルがインポート操作中に欠落する可能性があります。メッセージ情報の最初の 30 KB と最後の 30 KB は、常に保存されます。

PC/IXF ファイルのリモート・データベースへのインポートは、 PC/IXF ファイルがディスケットにあるときよりも、ハード・ディスクにあるときの方がより速く行うことができます。 *piDataDescriptor* でデフォルト以外の値を使用したり、*piActionString* で明示的な表列のリストを指定したりすると、リモート・データベースへのインポート速度は遅くなります。

データベース表または階層が存在していないと、ASC、DEL、または WSF ファイル形式のデータをインポートできません。ただし、表が存在していない場合でも、IMPORT CREATE または IMPORT REPLACE_CREATE は、 PC/IXF ファイルからデータをインポートする際に表を作成します。型付き表の場合、IMPORT CREATE はタイプ階層と表階層も作成することができます。

PC/IXF インポートは、データベース間でデータ (階層データも含む) を移動する場合に使用します。行区切り文字を含む文字データが区切り文字付き ASCII (DEL) ファイルにエクスポートされ、テキスト転送プログラムによって処理される場合、行区切り文字を含むフィールドは長さが変わることがあります。

ASC および DEL ファイルのデータは、インポートを実行するクライアント・アプリケーションのコード・ページであると仮定されます。異なるコード・ページのデ

ータをインポートする場合は、異なるコード・ページを使用することのできる PC/IXF ファイルをお勧めします。 PC/IXF ファイルとインポート・ユーティリティーが同じコード・ページである場合は、通常のアプリケーションの場合のように処理が行われます。それぞれのコード・ページが異なっており、FORCEIN オプションが指定されている場合、インポート・ユーティリティーは、PC/IXF ファイルのデータのコード・ページと、インポートを実行中のアプリケーションのコード・ページが同じであると見なします。この処理は、それら 2 つのコード・ページ用の変換テーブルが存在する場合であっても行われます。それぞれのコード・ページが異なっており、FORCEIN オプションが指定されておらず、変換テーブルが存在する場合、PC/IXF ファイルのすべてのデータは、そのファイルのコード・ページからアプリケーションのコード・ページに変換されます。それぞれのコード・ページが異なっており、FORCEIN オプションが指定されておらず、変換テーブルが存在しない場合、インポート操作は失敗します。これが適用されるのは、DB2 for AIX クライアント上の PC/IXF ファイルの場合だけです。

8KB ページ上の表オブジェクトの量が 1012 列の制限に近い場合、 PC/IXF データ・ファイルをインポートすると、 SQL ステートメントの最大サイズを超過するため、DB2 はエラーを戻します。この状態が発生する可能性があるのは、列が CHAR、 VARCHAR、または CLOB タイプの場合だけです。 DEL または ASC ファイルのインポートには、この制限事項は適用されません。

DB2 Connect は、DB2 for OS/390、DB2 for VM and VSE、および DB2 for OS/400 などの DRDA サーバーにデータをインポートするために使用できます。サポートされているのは、PC/IXF インポート (INSERT オプション) だけです。 restartcnt パラメーターもサポートされていますが、 commitcnt パラメーターはサポートされていません。

型付き表で CREATE オプションを使用するときは、 PC/IXF ファイルで定義されているすべての副表を作成してください。副表の定義は変更できません。型付き表で CREATE 以外のオプションを使用するときは、全探索順序リストによって全探索順序を指定できます。このため、全探索順序リストはエクスポート操作時に使用したものと一致する必要があります。 PC/IXF ファイル形式の場合は、ターゲット副表の名前を指定して、ファイルに格納されている全探索順序を使用するだけです。

インポート・ユーティリティーは、以前 PC/IXF ファイルにエクスポートされた表をリカバリーする場合に使用できます。その表は、エクスポート時の状態に戻ります。

システム表、宣言された一時表、またはサマリー表にデータをインポートすることはできません。

インポート・ユーティリティーを使用して、ビューを作成することはできません。

Windows NT オペレーティング・システムの場合は、以下のとおりです。

- 論理分割された PC/IXF ファイルのインポートはサポートされていません。
- 不正な形式の PC/IXF または WSF ファイルのインポートは、サポートされていません。

DB2 Data Links Manager についての考慮事項

DB2 インポート・ユーティリティーを実行する前に、以下のようにしてください。

1. 参照されているファイルを適切なデータ・リンク・サーバーにコピーする。 **dlfm_import** ユーティリティーを使用すれば、 **dlfm_export** ユーティリティーで生成されたアーカイブからファイルを抽出することができます。
2. 必要な接頭名を DB2 Data Links Manager に登録する。さらに、データベースの登録など、他の管理作業も必要になることがあります。
3. 必要な場合は、(DATALINK 列の) URL のデータ・リンク・サーバー情報を、SQL 表のエクスポート済みデータから更新する。 (元の構成のデータ・リンク・サーバーのターゲット位置が同じである場合、データ・リンク・サーバー名を更新する必要はありません。)
4. Data Links サーバーを、 DB2 Data Links Manager 構成ファイルのターゲット構成で定義する。

インポート・ユーティリティーをターゲット・データベースに対して実行する場合は、 DATALINK 列データが参照するファイルは、適切なデータ・リンク・サーバー上でリンクします。

挿入操作の際、DATALINK 列処理では、ターゲット・データベースでの列指定に従って、適切なデータ・リンク・サーバーのファイルがリンクされます。

関連資料:

- 453 ページの『SQLCA』
- 456 ページの『SQLDCOL』
- 495 ページの『SQLU-MEDIA-LIST』
- 127 ページの『インポートのファイル・タイプ修飾子』
- 205 ページの『データを移動する際の区切り文字の制約事項』

関連サンプル:

- 『dtformat.sqc -- Load and import data format extensions (C)』
- 『tbmove.sqc -- How to move table data (C)』
- 『expsamp.sqb -- Export and import tables with table data to a DRDA database (IBM COBOL)』
- 『impexp.sqb -- Export and import tables with table data (IBM COBOL)』
- 『tbmove.sqC -- How to move table data (C++)』

インポートのファイル・タイプ修飾子

表 9. インポート用の有効なファイル・タイプ修飾子: すべてのファイル・フォーマット

修飾子	説明
compound=*x*	*x* は 1 から 100 の数字です。データの挿入に非アトミック・コンパウンド SQL を使用し、 1 回につき *x* ステートメントずつ挿入が試みられます。 この修飾子を指定した場合、トランザクション・ログに十分な大きさがないと、インポート操作は失敗します。トランザクション・ログには、COMMITCOUNT で指定された行数、またはデータ・ファイル内の行数 (COMMITCOUNT が指定されていない場合) が入るだけの大きさが必要です。したがって、トランザクション・ログのオーバーフローを避けるために、COMMITCOUNT オプションを指定することをお勧めします。 この修飾子は、INSERT_UPDATE モード、階層表、および修飾子 usedefaults、identitymissing、identityignore、generatedmissing、generatedignore とは互換性がありません。
generatedignore	この修飾子はインポート・ユーティリティーに対して、すべての生成列のデータがデータ・ファイル内にあるものの、それらのデータは無視するべきものであることを通知します。その結果、生成列のすべての値はユーティリティーが生成します。この修飾子は、generatedmissing 修飾子とともに使用することはできません。
generatedmissing	この修飾子が指定されている場合、ユーティリティーは、生成列のデータが入力データ・ファイルに入っていない (NULL も入っていない) ものと見なし、行ごとに値を生成します。この修飾子は、generatedignore 修飾子とともに使用することはできません。
identityignore	この修飾子はインポート・ユーティリティーに対して、 ID 列のデータがデータ・ファイル内にあるものの、それらのデータは無視するべきものであることを通知します。その結果、すべての識別値はユーティリティーが生成します。 GENERATED ALWAYS ID 列と GENERATED BY DEFAULT ID 列のどちらの場合も動作は同じになります。つまり、GENERATED ALWAYS 列の場合には、リジェクトされる行はありません。この修飾子は、identitymissing 修飾子とともに使用することはできません。
identitymissing	この修飾子が指定されている場合、ユーティリティーは、 ID 列のデータが入力データ・ファイルに入っていない (NULL も入っていない) ものと見なし、行ごとに値を生成します。 GENERATED ALWAYS ID 列と GENERATED BY DEFAULT ID 列のどちらの場合も動作は同じになります。この修飾子は、identityignore 修飾子とともに使用することはできません。

表 9. インポート用の有効なファイル・タイプ修飾子: すべてのファイル・フォーマット (続き)

修飾子	説明
lobsinfile	*lob-path* には、LOB データを含むファイルへのパスを指定します。 各パスには少なくとも 1 つのファイルが含まれており、そのファイルには、データ・ファイル内の LOB ロケーション指定子 (LLS) によって指される少なくとも 1 つの LOB が入っています。 LLS は、LOB ファイル・パスに保管されるファイル内の LOB のロケーションのストリング表現です。 LLS の形式は、*filename.ext.nnn.mmm/* です。ここで、*filename.ext* は LOB を含むファイルの名前、*nnn* はファイル内の LOB のオフセット (バイト単位)、*mmm* は LOB の長さ (バイト単位) です。たとえば、ストリング `db2exp.001.123.456/` がデータ・ファイルに保存される場合、 LOB はファイル db2exp.001 のオフセット 123 に位置し、456 バイト長です。 LOBS FROM 文節は、″lobsinfile″ 修飾子が使用されているときの、 LOB ファイルの場所を指定します。 LOBS FROM 文節には、`lobsinfile` 修飾子のコンテキスト以外の意味はありません。 LOBS FROM 文節は、データのインポート中に、IMPORT ユーティリティーに LOB ファイルを検索するためのパスのリストを送ります。 LOB が NULL であることを示すには、サイズを -1 と入力します。サイズが 0 と指定されている場合、LOB は長さ 0 と見なされます。長さが -1 の NULL LOB の場合、オフセットとファイル名は無視されます。たとえば、NULL LOB の LLS は db2exp.001.7.-1/ です。
no_type_id	単一の副表にインポートするときにのみ有効です。通常の使用法としては、正規の表からデータをエクスポートしてから、(この修飾子を使用して) インポート操作を呼び出し、データを単一の副表に変換します。
nodefaults	ターゲット表の列のソース列が明示的に指定されておらず、表の列が NULL 不可の場合、デフォルト値はロードされません。このオプションが指定されていない場合、ターゲット表の 1 つの列のソース列が明示的に指定されていないと、以下のいずれかの処理が行われます。 • 列のデフォルト値を指定できる場合は、デフォルト値がロードされます。 • 列が NULL 可能で、その列にデフォルト値が指定できない場合は、NULL がロードされます。 • 列が NULL 可能ではなく、その列にデフォルト値も指定できない場合は、エラーが戻され、ユーティリティーは処理を停止します。
norowwarnings	リジェクトされた行についてのすべての警告を抑制します。
usedefaults	ターゲット表の列に対応するソース列が指定されているが、 1 つまたは複数の行インスタンスのデータが入っていない場合、デフォルト値がロードされます。欠落データの例は、以下のとおりです。 • DEL ファイルの場合、列に「,,」が指定されています。 • ASC ファイルの場合: 列の NULL 標識が「yes」に設定されています。 • DEL/ASC/WSF ファイルの場合: 十分な数の列がない行、または元の指定に対して長さが十分でない行。 このオプションが指定されていない場合、ソース列に行インスタンスのデータがないと、以下のいずれかの処理が行われます。 • その列が NULL 可能な場合は、NULL がロードされます。 • その列が NULL 可能でない場合、ユーティリティーはその行をリジェクトします。

表 10. インポート用の有効なファイル・タイプ修飾子: ASCII ファイル・フォーマット (ASC/DEL)

修飾子	説明
codepage=*x*	*x* は ASCII 文字ストリングです。この値は、出力データ・セット内のデータのコード・ページとして解釈されます。インポート操作時に、文字データをアプリケーションのコード・ページからこのコード・ページに変換します。 以下の規則が適用されます。 • 純 DBCS (GRAPHIC)、混合 DBCS、および EUC では、区切り文字は x00 から x3F の範囲に制限されます。 • nullindchar には、標準の ASCII セットに含まれる (コード・ポイント x20 から x7F の範囲の) 記号を指定する必要があります。これは、ASCII 記号およびコード・ポイントを示します。 **注:** 1. codepage 修飾子は、 lobsinfile 修飾子とともに使用することはできません。 2. コード・ページが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・ページに変換されているときにデータの拡張が発生する場合は、データは切り捨てられ、データの消失が発生する可能性があります。
dateformat="*x*"	*x* はソース・ファイル内の日付の形式です。 [2] 有効な日付エレメントは以下のとおりです。 YYYY - 年 (0000 から 9999 の範囲の 4 桁の数字) M - 月 (1 から 12 の範囲の 1 桁または 2 桁の数) MM - 月 (1 から 12 の範囲の 2 桁の数。 M と相互排他的) D - 日 (1 から 31 の範囲の 1 桁または 2 桁の数) DD - 日 (1 から 31 の範囲の 1 桁または 2 桁の数; D と相互排他的) DDD - 元日から数えた日数 (001 から 366 の 3 桁 他の日または月エレメントと相互に排他) 指定されていないエレメントには、デフォルト値 1 が割り当てられます。日付形式の例を以下に示します。 "D-M-YYYY" "MM.DD.YYYY" "YYYYDDD"
implieddecimal	暗黙指定されている小数点の位置が列定義によって決定され、値の終わりにあるとは見なされなくなります。たとえば、値 12345 は、 12345.00 ではなく、 123.45 として DECIMAL(8,2) 列にロードされます。
noeofchar	任意指定のファイル終わり文字 x'1A' が、ファイルの終わりとして認識されません。通常の文字の場合のように処理が続行されます。
timeformat="*x*"	*x* はソース・ファイル内の時刻の形式です。 [2] 有効な時刻エレメントは以下のとおりです。 H - 時 (12 時間制の場合は 0 から 12、 24 時間制の場合は 0 から 24 の範囲の 1 桁または 2 桁の数) HH - 時 (12 時間制の場合は 0 から 12、 24 時間制の場合は 0 から 24 の範囲の 2 桁の数。 H と相互に排他) M - 分 (0 から 59 の範囲の 1 桁または 2 桁の数) MM - 分 (0 から 59 の範囲の 2 桁の数。 M と相互排他的) S - 秒 (0 から 59 の範囲の 1 桁または 2 桁の数) SS - 秒 (0 から 59 の範囲の 2 桁の数。 S とは相互排他的) SSSSS - 夜の 12 時から数えた秒数 (00000 から 86399 の 5 桁; 他の時刻エレメントとは相互排他的) TT - 正午の標識 (AM または PM) 指定されていないエレメントには、デフォルト値 0 が割り当てられます。時刻形式の例を以下に示します。 "HH:MM:SS" "HH.MM TT" "SSSSS"

表 10. インポート用の有効なファイル・タイプ修飾子: ASCII ファイル・フォーマット (ASC/DEL) (続き)

修飾子	説明

timestampformat="x"	*x* はソース・ファイル内のタイム・スタンプの形式です。 [2] 有効なタイム・スタンプ・エレメントは以下のとおりです。
	
	YYYY - 年 (0000 から 9999 の範囲の 4 桁の数字)
	M - 月 (1 から 12 の範囲の 1 桁または 2 桁の数)
	MM - 月 (01 から 12 の 2 桁の数。
	M および MMM とは相互に排他的)
	MMM - 月 (月を表す 3 文字の大文字小文字を区別しない略語;
	M および MM とは相互に排他的)
	D - 日 (1 から 31 の範囲の 1 桁または 2 桁の数)
	DD - 日 (1 から 31 の範囲の 1 桁または 2 桁の数; D と相互排他的)
	DDD - 元日から数えた日数 (001 から 366 の 3 桁;
	他の日または月エレメントと相互に排他)
	H - 時 (12 時間制の場合は 0 から 12、
	24 時間制では 0 から 24 の 1 桁または 2 桁。)
	HH - 時 (12 時間制の場合は 0 から 12、
	24 時間制では 0 から 24 の 1 桁または 2 桁:
	H と相互に排他)
	M - 分 (0 から 59 の 1 桁または 2 桁)
	MM - 分 (0 から 59 の範囲の 2 桁の数。
	M (分) とは相互排他的)
	S - 秒 (0 から 59 の 1 桁または 2 桁)
	SS - 秒 (0 から 59 の範囲の 2 桁の数。
	S とは相互排他的)
	SSSSS - 夜の 12 時から数えた秒数
	(00000 から 86399 の 5 桁;
	他の時刻エレメントとは相互排他的)
	UUUUUU - マイクロ秒 (000000 から 999999 の 6 桁;
	他のすべてのマイクロ秒エレメントとは相互排他的)
	UUUUU - マイクロ秒 (00000 から 99999 の 5 桁、
	000000 から 999990 の範囲にマップする;
	他のすべてのマイクロ秒エレメントとは相互排他的)
	UUUU - マイクロ秒 (0000 から 9999 の 4 桁、
	000000 から 999900 の範囲にマップする;
	他のすべてのマイクロ秒エレメントとは相互排他的)
	UUU - マイクロ秒 (000 から 999 の 3 桁、
	000000 から 999000 の範囲にマップする;
	他のすべてのマイクロ秒エレメントとは相互排他的)
	UU - マイクロ秒 (00 から 99 の 2 桁、
	000000 から 990000 の範囲にマップする;
	他のすべてのマイクロ秒エレメントとは相互排他的)
	U - マイクロ秒 (0 から 9 の 2 桁、
	000000 から 900000 の範囲にマップする;
	他のすべてのマイクロ秒エレメントとは相互排他的)
	TT - 正午の標識 (AM または PM)
	
	YYYY、M、MM、D、DD、または DDD エレメントに値が指定されていない場合、デフォルト値の 1 が割り当てられます。値が指定されていない MMM エレメントには、デフォルト値の「Jan」が割り当てられます。それ以外のエレメントに値が指定されていない場合、デフォルト値の 0 が割り当てられます。以下に、タイム・スタンプ形式の例を示します。
	
	`"YYYY/MM/DD HH:MM:SS.UUUUUU"`
	
	MMM エレメントの有効な値は、
	「jan」、「feb」、「mar」、「apr」、「may」、「jun」、「jul」、「aug」、「sep」、「oct」、「nov」、および「dec」です。これらの値は大文字小文字の区別をします。
	
	以下の例は、ユーザー定義の日付および時刻の形式を含んでいるデータを、 schedule という表にインポートする方法を示しています。
	
	`db2 import from delfile2 of del`
	`modified by timestampformat="yyyy.mm.dd hh:mm tt"`
	`insert into schedule`

表 10. インポート用の有効なファイル・タイプ修飾子: ASCII ファイル・フォーマット (ASC/DEL) (続き)

修飾子	説明
usegraphiccodepage	usegraphiccodepage が指定された場合、GRAPHIC または 2 バイト文字ラージ・オブジェクト (DBCLOB) データ・フィールドにインポートされるデータは、GRAPHIC コード・ページであると見なされます。データの残りは、文字コード・ページであると見なされます。 GRAPHIC コード・ページは、文字コード・ページと関連付けられます。 IMPORT は、codepage 修飾子 (指定されている場合)、または codepage 修飾子が指定されていない場合はアプリケーションのコード・ページを介して、文字コード・ページを決定します。 この修飾子は、リカバリーされている表に GRAPHIC データが含まれている場合にのみ、表リカバリーのドロップによって生成された区切りデータ・ファイルとともに使用される必要があります。 **制約事項** usegraphiccodepage 修飾子は、 EXPORT ユーティリティーによって作成された DEL または ASC ファイルで指定することはできません。これらのファイルは、1 コード・ページのみでエンコードされたデータを含むためです。 usegraphiccodepage 修飾子はまた、ファイル内の 2 バイト文字ラージ・オブジェクト (DBCLOB) には無視されます。

表 11. インポート用の有効なファイル・タイプ修飾子: ASC (区切り文字なし ASCII) ファイル・フォーマット

修飾子	説明
nochecklengths	nochecklengths を指定した場合は、ソース・データの中にターゲット表列のサイズを超える列定義が含まれている場合であっても、各行のインポートが試みられます。このような行が正常にインポートされるのは、コード・ページ変換でソース・データが縮小する場合です。たとえば、ソースにある 4 バイトの EUC データがターゲットで 2 バイトの DBCS データに縮小すれば、 必要スペースは半分になります。このオプションが特に役立つのは、列の定義は不一致であるがソース・データが常に適合することが分かっている場合です。
nullindchar=*x*	*x* は単一文字です。 NULL 値を示す文字を *x* に変更します。 *x* のデフォルト値は Y です。[3] 文字が 1 つの英字である場合を除いて、この修飾子は EBCDIC データ・ファイルで大文字小文字を区別します。たとえば、NULL 標識文字が文字 N として指定されている場合、n も NULL 標識として認識されます。
reclen=*x*	*x* は 32 767 以下の整数です。各行ごとに *x* 個の文字が読み取られ、行の終わりを示すのに改行文字は使用されません。

striptblanks	データを可変長フィールドにロードする際に、後書きブランク・スペースを切り捨てます。このオプションを指定しない場合、ブランク・スペースはそのまま保持されます。
	
	以下の例の場合、インポート・ユーティリティーは、striptblanks によって後書きブランク・スペースを切り捨てます。
	
	`db2 import from myfile.asc of asc`
	`  modified by striptblanks`
	`  method l (1 10, 12 15) messages msgs.txt`
	`  insert into staff`
	
	このオプションは、striptnulls と一緒には指定できません。これらのオプションは、相互に排他的です。
	注: このオプションは以前の t オプションを置き換えるもので、バックレベルとの互換性のみサポートされています。

表 11. インポート用の有効なファイル・タイプ修飾子: ASC (区切り文字なし ASCII) ファイル・フォーマット (続き)

修飾子	説明
striptnulls	データを可変長フィールドにロードする際に、後書き NULL (0x00 文字) を切り捨てます。このオプションを指定しない場合、NULL はそのまま保持されます。 このオプションは、striptblanks と一緒には指定できません。これらのオプションは、相互に排他的です。 **注:** このオプションは以前の padwithzero オプションを置き換えるもので、バックレベルとの互換性のみサポートされています。

表 12. インポート用の有効なファイル・タイプ修飾子: DEL (区切り文字付き ASCII) ファイル・フォーマット

修飾子	説明

chardel*x*	*x* は単一文字のストリング区切り文字です。デフォルトは二重引用符 (") です。指定した文字は、文字ストリングを囲むために、二重引用符の代わりに使用されます。[34] 文字ストリング区切り文字として明示的に二重引用符を指定したい場合、次のように指定します。
	
	`modified by chardel""`
	
	単一引用符 (') も、以下のように、文字ストリングの区切り文字として指定できます。以下の例では、chardel'' が指定されており、インポート・ユーティリティーは検出するすべての単一引用符 (') を文字ストリングの区切り文字として解釈します。
	
	`db2 "import from myfile.del of del`
	`modified by chardel''`
	`method p (1, 4) insert into staff (id, years)"`

coldel*x*	*x* は単一文字の列区切り文字です。デフォルトはコンマ (,) です。指定した文字は、列の終わりを表すために、コンマの代わりに使用されます。[34]
	
	以下の例では、coldel; が指定されており、インポート・ユーティリティーは検出するすべてのセミコロン (;) を列の区切り文字として解釈します。
	
	`db2 import from myfile.del of del`
	`modified by coldel;`
	`messages msgs.txt insert into staff`

datesiso	日付形式。すべての日付データ値を ISO 形式でインポートします。
decplusblank	正符号文字。正の 10 進値の先頭に正符号 (+) ではなく、ブランク・スペースが置かれます。デフォルトのアクションでは、正の 10 進数の前に正符号 (+) が付けられます。

decpt*x*	*x* は、小数点としてピリオドと置換される単一文字です。デフォルト値はピリオド (.) です。指定した文字は、小数点文字としてピリオドの代わりに使用されます。[34]
	
	以下の例では、decpt; が指定されており、インポート・ユーティリティーは検出するすべてのセミコロン (;) を小数点として解釈します。
	
	`db2 "import from myfile.del of del`
	`modified by chardel'`
	`decpt; messages msgs.txt insert into staff"`

表 12. インポート用の有効なファイル・タイプ修飾子: DEL (区切り文字付き ASCII) ファイル・フォーマット (続き)

修飾子	説明

delprioritychar	区切り文字の現在のデフォルト優先順位は、(1) レコード区切り文字、(2) 区切り文字、(3) 列区切り文字です。この修飾子は、区切り文字の優先順位を区切り文字、レコード区切り文字、列区切り文字と逆順にすることにより、以前の優先順位に依存する既存のアプリケーションを保護します。構文は次のとおりです。
	`db2 import ... modified by delprioritychar ...`
	たとえば、以下のような DEL データ・ファイルがあるとします。
	`"Smith, Joshua",4000,34.98<row delimiter>`
	`"Vincent,<row delimiter>, is a manager", ...`
	`... 4005,44.37<row delimiter>`
	`delprioritychar` 修飾子が指定されている場合、このデータ・ファイルには 2 行しかありません。 2 番目の <row delimiter> は 2 番目の行の最初のデータ列の一部と解釈されますが、 1 番目と 3 番目の <row delimiter> は実レコードの区切り文字と解釈されます。この修飾子が指定されていない 場合、このデータ・ファイルでは 3 行になり、各行は <row delimiter> によって区切られます。

dldel*x*	*x* は単一文字の DATALINK 区切り文字です。デフォルト値はセミコロン (;) です。指定した文字は、 DATALINK 値のフィールド間区切り文字としてセミコロンの代わりに使用されます。 DATALINK 値には 2 つ以上の副値を指定できるので、この区切り文字が必要です。[34] **注:** *x* は、行、列、または文字ストリングの区切り文字とは異なる文字にしてください。
keepblanks	タイプが CHAR、VARCHAR、LONG VARCHAR、または CLOB の各フィールドの前後のブランクを保持します。このオプションが指定されない場合、区切り文字の外側にある前後のブランクはすべて除去され、表のすべてのブランク・フィールドに NULL が挿入されます。
nochardel	インポート・ユーティリティーは、列区切り文字の間にあるすべてのバイトを列のデータの一部であると見なします。区切り文字は、列データの一部として構文解析されます。データが DB2 を使用してエクスポートされている場合は、このオプションを指定しないでください (エクスポート時に nochardel が指定されない限り)。これは、区切り文字を持たないベンダー・データ・ファイルをサポートするために提供されています。不適切に使用すると、データが損失または破壊される場合があります。 このオプションを `chardelx`、 `delprioritychar` または `nodoubledel` と一緒に指定することはできません。これらのオプションは、相互に排他的です。
nodoubledel	二重になっている区切り文字の認識を抑止します。

表 13. インポート用の有効なファイル・タイプ修飾子: IXF ファイル・フォーマット

修飾子	説明
forcein	コード・ページが不一致でもデータを受け入れ、コード・ページ間の変換を抑止するようにユーティリティーに指示します。 固定長ターゲット・フィールドに、データが入るだけの十分な大きさがあるかどうかチェックされます。 `nochecklengths` が指定されていると、チェックは実行されず、各行のインポートが試行されます。

表 13. インポート用の有効なファイル・タイプ修飾子: IXF ファイル・フォーマット (続き)

修飾子	説明
indexixf	既存の表に現在定義されているすべての索引をドロップし、 PC/IXF ファイルの索引定義から新しい索引を作成するよう、ユーティリティーに指示します。このオプションを使用できるのは、表の内容が置き換えられる場合だけです。ビューでは使用できません。また、*insert-column* が指定されている場合にも使用できません。
indexschema=*schema*	指定したスキーマ を索引作成時の索引名に使用します。スキーマ が指定されていない (しかし、キーワード indexschema が*指定されている*) 場合は、接続ユーザー ID を使用します。このキーワードを指定しない場合、IXF ファイルのスキーマが使用されます。
nochecklengths	nochecklengths を指定した場合は、ソース・データの中にターゲット表列のサイズを超える列定義が含まれている場合であっても、各行のインポートが試みられます。このような行が正常にインポートされるのは、コード・ページ変換でソース・データが縮小する場合です。たとえば、ソースにある 4 バイトの EUC データがターゲットで 2 バイトの DBCS データに縮小すれば、 必要スペースは半分になります。このオプションが特に役立つのは、列の定義は不一致であるがソース・データが常に適合することが分かっている場合です。

注:

1. サポートされていないファイル・タイプを MODIFIED BY オプションで使用しようとしても、インポート・ユーティリティーは警告を出しません。この場合、インポート操作が失敗し、エラー・コードが戻されます。
2. 日付形式ストリングは必ず二重引用符で囲まなければなりません。フィールド区切り文字には、 a から z、A から Z、および 0 から 9 を含めることはできません。フィールド区切り文字は、 DEL ファイル形式の区切り文字またはフィールド区切り文字と同じにすることはできません。エレメントの開始および終了位置が明らかな場合、フィールド区切り文字は任意指定です。開始および終了位置が明らかでないのは、項目の長さが一定でない D、H、M、または S などのエレメントが使用されている場合です (修飾の仕方によって異なります)。

   タイム・スタンプ形式の場合、月の記述子と分の記述子のどちらも文字 M を使用するため、あいまいさを避けるよう注意する必要があります。月のフィールドは、他の日付フィールドと隣接していなければなりません。分のフィールドは、他の時刻フィールドに隣接していなければなりません。以下のタイム・スタンプ形式は、あいまいな形式の例です。

   ```
   "M" (月と分のどちらかがはっきりしない)
   "M:M" (どちらが何を表しているのか分からない)
   "M:YYYY:M" (どちらも月として解釈される)
   "S:M:YYYY" (時刻値と日付値の両方に隣接している)
   ```

   あいまいな場合、ユーティリティーはエラー・メッセージを報告し、操作は失敗します。

   以下に、明確なタイム・スタンプ形式を示します。

   ```
   "M:YYYY" (月)
   "S:M" (分)
   "M:YYYY:S:M" (月....分)
   "M:H:YYYY:M:D" (分....月)
   ```

二重引用符や円記号などの文字の前には、エスケープ文字 (たとえば、¥) を付けなければなりません。

3. この文字は、ソース・データのコード・ページで指定してください。

   文字コード・ポイント (文字記号ではない) は、 xJJ または 0xJJ という構文で指定することができます (JJ はコード・ポイントの 16 進表記)。たとえば、列区切りとして # 文字を指定するには、以下のいずれかを使用します。

   ```
   ... modified by coldel# ...
   ... modified by coldel0x23 ...
   ... modified by coldelX23 ...
   ```

4. データ移動のための区切り文字の制約事項には、区切り文字の指定変更として使用できる文字に適用される制限事項がリストされています。

表 14. codepage および usegraphiccodepage 使用時の IMPORT 動作

codepage=N	usegraphiccodepage	IMPORT 動作
なし	なし	ファイル内のすべてのデータは、アプリケーション・コード・ページであると見なされます。
あり	なし	ファイル内のすべてのデータは、コード・ページ N であると見なされます。 **警告:** N が単一バイト・コード・ページの場合、GRAPHIC データをデータベースにインポートすると、壊れます。
なし	あり	ファイル内の文字データは、アプリケーション・コード・ページであると見なされます。 GRAPHIC データは、アプリケーション GRAPHIC データのコード・ページであると見なされます。 アプリケーション・コード・ページが単一バイトの場合は、すべてのデータはアプリケーション・コード・ページであると見なされます。 **警告:** アプリケーション・コード・ページが単一バイトの場合、 GRAPHIC データは、データベースにたとえ GRAPHIC 列が含まれていても、データベースにインポートされると壊れます。
あり	あり	文字データは、コード・ページ N であると見なされます。 GRAPHIC データは、N の GRAPHIC コード・ページであると見なされます。 N が単一バイトまたは 2 バイト・コード・ページの場合は、すべてのデータは、コード・ページ N であると見なされます。 **警告:** N が単一バイト・コード・ページの場合、GRAPHIC データをデータベースにインポートすると、壊れます。

関連資料:

- 114 ページの『db2Import - インポート』
- 「コマンド・リファレンス」の『IMPORT コマンド』

- 205 ページの『データを移動する際の区切り文字の制約事項』

db2Inspect - データベースの検査

データベースの構造上の保全性を検査し、ページの整合性についてデータベースのページをチェックします。

有効範囲:

単一パーティション・データベースでは、有効範囲は単一パーティションのみです。パーティション・データベース環境では、これは db2nodes.cfg に定義されたすべての論理パーティションの集合です。

許可:

以下のいずれかです。

- *sysadm*
- *sysctrl*
- *sysmaint*
- *dbadm*
- 表に対する CONTROL 特権

必要な接続:

データベース

API 組み込みファイル:

db2ApiDf.h

C API 構文:

```
/* File: db2ApiDf.h */
/* API: db2Inspect */
/* ... */
SQL_API_RC SQL_API_FN
  db2Inspect (
  db2Uint32 versionNumber,
  void *pParmStruct,
  struct sqlca *pSqlca);

typedef SQL_STRUCTURE db2InspectStruct
{
  char                        *piTablespaceName;
  char                        *piTableName;
  char                        *piSchemaName;
  char                        *piResultsName;
  char                        *piDataFileName;
  SQL_PDB_NODE_TYPE           *piNodeList;
  db2Uint32                   iAction;
  db2int32                    iTablespaceID;
  db2int32                    iObjectID;
  db2Uint32                   iBeginCheckOption;
  db2int32                    iLimitErrorReported;
  db2Uint16                   iObjectErrorState;
  db2Uint16                   iKeepResultfile;
  db2Uint16                   iAllNodeFlag;
  db2Uint16                   iNumNodes;
  db2Uint16                   iLevelObjectData;
```

```
  db2Uint16                   iLevelObjectIndex;
  db2Uint16                   iLevelObjectLong;
  db2Uint16                   iLevelObjectLOB;
  db2Uint16                   iLevelObjectBlkMap;
  db2Uint16                   iLevelExtentMap;
} db2InspectStruct;
/* ... */
```

汎用 API 構文:

```
/* File: db2ApiDf.h */
/* API: db2gInspect */
/* ... */
SQL_API_RC SQL_API_FN
  db2gInspect (
  db2Uint32 versionNumber,
  void *pParmStruct,
  struct sqlca *pSqlca);

typedef SQL_STRUCTURE db2gInspectStruct
{
  char                        *piTablespaceName;
  char                        *piTableName;
  char                        *piSchemaName;
  char                        *piResultsName;
  char                        *piDataFileName;
  SQL_PDB_NODE_TYPE           *piNodeList;
  db2Uint32                   iResultsNameLength;
  db2Uint32                   iDataFileNameLength;
  db2Uint32                   iTablespaceNameLength;
  db2Uint32                   iTableNameLength;
  db2Uint32                   iSchemaNameLength;
  db2Uint32                   iAction;
  db2int32                    iTablespaceID;
  db2int32                    iObjectID;
  db2Uint32                   iBeginCheckOption;
  db2int32                    iLimitErrorReported;
  db2Uint16                   iObjectErrorState;
  db2Uint16                   iKeepResultfile;
  db2Uint16                   iAllNodeFlag;
  db2Uint16                   iNumNodes;
  db2Uint16                   iLevelObjectData;
  db2Uint16                   iLevelObjectIndex;
  db2Uint16                   iLevelObjectLong;
  db2Uint16                   iLevelObjectLOB;
  db2Uint16                   iLevelObjectBlkMap;
  db2Uint16                   iLevelExtentMap;
} db2gInspectStruct;
/* ... */
```

API パラメーター:

versionNumber

入力。 2 番目のパラメーター *pParmStruct* として渡される構造のバージョンとリリースのレベルを指定します。

pParmStruct

入力。 *db2InspectStruct* 構造を指すポインター。

pSqlca

出力。 *sqlca* 構造を指すポインター。

piTablespaceName
入力。表スペース名を含むストリング。表スペースは、表スペースでの操作で識別される必要があります。ポインターが NULL の場合、表スペース ID の値が入力として使用されます。

piTableName
入力。表名を含むストリング。表は、表または表オブジェクトでの操作で識別される必要があります。ポインターが NULL の場合、表スペース ID および表オブジェクト ID の値が入力として使用されます。

piSchemaName
入力。スキーマ名を含むストリング。

piResultsName
入力。結果出力ファイルの名前を含むストリング。この入力は提供する必要があります。ファイルは診断データ・ディレクトリー・パスに書き込まれます。

piDataFileName
入力。将来の使用のために予約済み。 NULL に設定する必要があります。

piNodeList
入力。操作を実行するパーティション番号の配列を指すポインター。

iResultsNameLength
入力。結果ファイル名のストリングの長さ。

iDataFileNameLength
入力。データ出力ファイル名のストリングの長さ。

iTablespaceNameLength
入力。表スペース名のストリングの長さ。

iTableNameLength
入力。表名のストリングの長さ。

iSchemaNameLength
入力。スキーマ名のストリングの長さ。

iAction
入力。検査アクションを指定します。有効な値は以下のとおりです。

DB2INSPECT_ACT_CHECK_DB
データベース全体を検査します。

DB2INSPECT_ACT_CHECK_TABSPACE
表スペースを検査します。

DB2INSPECT_ACT_CHECK_TABLE
表を検査します。

iTablespaceID
入力。表スペース ID を指定します。表スペースを識別しなければならない場合、表スペース名へのポインターが NULL であれば、表スペース ID の値が入力として使用されます。

iObjectID
入力。オブジェクト ID を指定します。表を識別しなければならない場合、表名へのポインターが NULL であれば、オブジェクト ID の値が入力として使用されます。

iBeginCheckOption
入力。データベースのチェック、または操作の開始地点を示す表スペース操作をチェックするオプション。通常の開始地点から開始するには、ゼロに設定する必要があります。値は以下のとおりです。

DB2INSPECT_BEGIN_TSPID
この値をデータベースのチェックに使用し、表スペース ID フィールドによって指定された表スペースから開始します。表スペース ID が設定されている必要があります。

DB2INSPECT_BEGIN_TSPID_OBJID
この値をデータベースのチェックに使用し、表スペース ID およびオブジェクト ID フィールドによって指定された表から開始します。このオプションを使用するには、表スペース ID およびオブジェクト ID が設定されている必要があります。

DB2INSPECT_BEGIN_OBJID
この値を表スペースのチェックに使用し、オブジェクト ID フィールドによって指定された表から開始します。オブジェクト ID が設定されている必要があります。

iLimitErrorReported
入力。オブジェクトのエラーとなったページ数の報告限度を指定します。限度値として使用する数、または以下のいずれかの値を指定します。

DB2INSPECT_LIMIT_ERROR_DEFAULT
この値を使用して、報告されるエラーとなったページの最大数が、オブジェクトのエクステント・サイズであることを指定します。

DB2INSPECT_LIMIT_ERROR_ALL
この値を使用して、エラーとなったすべてのページを報告します。

iObjectErrorState
入力。エラー状態のオブジェクトをスキャンするかどうかを指定します。有効な値は以下のとおりです。

DB2INSPECT_ERROR_STATE_NORMAL
正常な状態にあるオブジェクトのみ処理します。

DB2INSPECT_ERROR_STATE_ALL
エラー状態のオブジェクトを含め、すべてのオブジェクトを処理します。

iKeepResultfile
入力。結果ファイルの保存を指定します。有効な値は以下のとおりです。

DB2INSPECT_RESFILE_CLEANUP
エラーが報告された場合、結果出力ファイルが保存されます。そうでない場合、結果ファイルは操作終了時に除去されます。

DB2INSPECT_RESFILE_KEEP_ALWAYS
結果出力ファイルが保存されます。

iAllNodeFlag
入力。操作が、db2nodes.cfg に定義されているすべてのノードに適用されるかどうかを示します。有効な値は以下のとおりです。

DB2_NODE_LIST
pNodeList で渡されたノード・リスト内のすべてのノードに適用されます。

DB2_ALL_NODES
すべてのノードに適用されます。 *pNodeList* は NULL でなければなりません。これがデフォルト値です。

DB2_ALL_EXCEPT
pNodeList で渡されたノード・リスト内のノードを除く、すべてのノードに適用されます。

iNumNodes
入力。 *pNodeList* 配列内のノードの数を指定します。

iLevelObjectData
入力。データ・オブジェクトの処理レベルを指定します。有効な値は以下のとおりです。

DB2INSPECT_LEVEL_NORMAL
通常レベルです。

DB2INSPECT_LEVEL_LOW
低いレベルです。

DB2INSPECT_LEVEL_NONE
レベルはありません。

iLevelObjectIndex
入力。索引オブジェクトの処理レベルを指定します。有効な値は以下のとおりです。

DB2INSPECT_LEVEL_NORMAL
通常レベルです。

DB2INSPECT_LEVEL_LOW
低いレベルです。

DB2INSPECT_LEVEL_NONE
レベルはありません。

iLevelObjectLong
入力。ロング・オブジェクトの処理レベルを指定します。有効な値は以下のとおりです。

DB2INSPECT_LEVEL_NORMAL
通常レベルです。

DB2INSPECT_LEVEL_LOW
低いレベルです。

DB2INSPECT_LEVEL_NONE
レベルはありません。

iLevelObjectLOB
入力。 LOB オブジェクトの処理レベルを指定します。有効な値は以下のとおりです。

DB2INSPECT_LEVEL_NORMAL
通常レベルです。

DB2INSPECT_LEVEL_LOW
低いレベルです。

DB2INSPECT_LEVEL_NONE
レベルはありません。

iLevelObjectBlkMap
入力。ブロック・マップ・オブジェクトの処理レベルを指定します。有効な値は以下のとおりです。

DB2INSPECT_LEVEL_NORMAL
通常レベルです。

DB2INSPECT_LEVEL_LOW
低いレベルです。

DB2INSPECT_LEVEL_NONE
レベルはありません。

iLevelExtentMap
入力。エクステント・マップの処理レベルを指定します。有効な値は以下のとおりです。

DB2INSPECT_LEVEL_NORMAL
通常レベルです。

DB2INSPECT_LEVEL_LOW
低いレベルです。

DB2INSPECT_LEVEL_NONE
レベルはありません。

使用上の注意:

オンライン検査処理では、分離レベルを非コミット読み取りに指定してデータベース・オブジェクトにアクセスします。コミット処理は、検査処理時に行われます。検査操作を開始する前に、COMMIT または ROLLBACK を発行して作業単位を終了することをお勧めします。

検査チェック処理では、不定形式の検査データ結果を結果ファイルに書き込みます。ファイルは診断データ・ディレクトリー・パスに書き込まれます。チェック処理でエラーが検出されない場合、結果出力ファイルは検査操作の終了時に消去されます。チェック処理でエラーが検出された場合、結果出力ファイルは検査操作の終了時に消去されません。検査の詳細について調べるには、検査結果出力ファイルを `db2inspf` ユーティリティーでフォーマットしてください。

パーティション・データベース環境では、結果出力ファイルの拡張部分はデータベース・パーティション番号に対応します。ファイルは、データベース・マネージャーの診断データ・ディレクトリー・パスに置かれます。

ユニークの結果出力ファイル名を指定する必要があります。結果出力ファイルがすでに存在する場合は、操作は処理されません。

表スペースの処理では、その表スペースにあるオブジェクトだけを処理します。

関連資料:

• 453 ページの『SQLCA』

db2InstanceQuiesce - インスタンスの静止

すべてのユーザーを強制的にインスタンスから切り離し、すべてのアクティブ・トランザクションを即時にロールバックし、データベースを静止モードにします。この API はインスタンスへの排他的アクセスを提供します。この静止期間中、インスタンスに対してシステム管理が実行されます。システム管理の完了後、db2DatabaseUnquiesce API を使用して、データベースを静止解除できます。この API を使用することによって、シャットダウンしたり別のデータベースを始動しなくても、データベースに接続することができます。

このモードでは、*QUIESCE CONNECT* 権限を持つグループまたはユーザーと、および *sysadm*、*sysmaint*、または *sysctrl* のみがデータベースおよびそのオブジェクトにアクセスできます。

許可:

以下のいずれかです。

- *sysadm*
- *sysctrl*

必要な接続:

なし

API 組み込みファイル:

db2ApiDf.h

C API 構文:

```
/* File: db2ApiDf.h */
/* API: db2InstanceQuiesce */
/* ... */
SQL_API_RC SQL_API_FN
db2InstanceQuiesce (
  db2Uint32 versionNumber,
  void *pParmStruct,
  struct sqlca *pSqlca);

typedef SQL_STRUCTURE db2InsQuiesceStruct
{
             char          *piInstanceName;
             char          *piUserId;
             char          *piGroupId;
             db2Uint32     iImmediate;
```

```
            db2Uint32     iForce;
            db2Uint32     iTimeout;
} db2InsQuiesceStruct;
/* ... */
```

汎用 API 構文:

```
/* File: db2ApiDf.h */
/* API: db2gInstanceQuiesce */
/* ... */
SQL_API_RC SQL_API_FN
db2gInstanceQuiesce (
  db2Uint32 versionNumber,
  void *pParmStruct,
  struct sqlca *pSqlca);

typedef SQL_STRUCTURE db2gInsQuiesceStruct
{
            db2Uint32     iInstanceNameLen;
            char          *piInstanceName;
            db2Uint32     iUserIdLen;
            char          *piUserId;
            db2Uint32     iGroupIdLen;
            char          *piGroupId;
            db2Uint32     iImmediate;
            db2Uint32     iForce;
            db2Uint32     iTimeout;
} db2gInsQuiesceStruct;
/* ... */
```

API パラメーター:

versionNumber
入力。 2 番目のパラメーター *pParmStruct* として渡される構造のバージョンとリリースのレベルを指定します。

pParmStruct
入力。 *db2InsQuiesceStruct* 構造を指すポインター。

pSqlca
出力。 *sqlca* 構造を指すポインター。

iInstanceNameLen
入力。 *piInstanceName* の長さ (バイト単位) を指定します。

piInstanceName
入力。インスタンス名。

iUserIdLen
入力。 *piUserID* の長さ (バイト単位) を指定します。

piUserId
入力。インスタンスが静止している間、インスタンスへのアクセスを許可されるユーザーの名前。

iGroupIdLen
入力。 *piGroupId* の長さ (バイト単位) を指定します。

piGroupId
入力。インスタンスが静止している間、インスタンスへのアクセスを許可されるグループの名前。

iImmediate
入力。将来の利用のために予約されています。

iForce 入力。将来の利用のために予約されています。

iTimeout
入力。アプリケーションが現在の作業単位をコミットするのを待機する時間 (分) を指定します。 *iTimeout* を指定しない場合、単一パーティション・データベース環境でのデフォルト値は 10 分です。パーティション・データベース環境では、データベース・マネージャー構成パラメータ *start_stop_timeout* によって指定された値が使用されます。

関連資料:

- 453 ページの『SQLCA』
- 153 ページの『db2InstanceUnquiesce - インスタンスの静止解除』

db2InstanceStart - インスタンスの開始

ローカルまたはリモート・インスタンスを開始します。

有効範囲:

単一パーティション・データベース環境では、有効範囲は単一データベース・パーティションのみになります。複数パーティション・データベース環境では、ノード構成ファイル `db2nodes.cfg` に定義されたデータベース・パーティション・サーバーすべてを集めたものが有効範囲です。

許可:

以下のいずれかです。

- *sysadm*
- *sysctrl*
- *sysmaint*

必要な接続:

なし

API 組み込みファイル:

db2ApiDf.h

C API 構文:

```
/* File: db2ApiDf.h */
/* API: db2InstanceStart */
/* ... */
SQL_API_RC SQL_API_FN
db2InstanceStart (
  db2Uint32 versionNumber,
  void *pParmStruct,
  struct sqlca *pSqlca);

typedef SQL_STRUCTURE db2InstanceStartStruct
{
             db2int8       iIsRemote;
             char          *piRemoteInstName;
```

```
            db2DasCommData *piCommData;
            db2StartOptionsStruct *piStartOpts;
} db2InstanceStartStruct;

typedef SQL_STRUCTURE db2DasCommData
{
            db2int8       iCommParam;
            char          *piNodeOrHostName;
            char          *piUserId;
            char          *piUserPw;
} db2DasCommData;

typedef SQL_STRUCTURE db2StartOptionsStruct
{
            db2Uint32     iIsProfile;
            char          *piProfile;
            db2Uint32     iIsNodeNum;
            db2NodeType   iNodeNum;
            db2Uint32     iOption;
            db2Uint32     iIsHostName;
            char          *piHostName;
            db2Uint32     iIsPort;
            db2PortType   iPort;
            db2Uint32     iIsNetName;
            char          *piNetName;
            db2Uint32     iTblspaceType;
            db2NodeType   iTblspaceNode;
            db2Uint32     iIsComputer;
            char          *piComputer;
            char          *piUserName;
            char          *piPassword;
            db2QuiesceStartStruct  iQuiesceOpts;
} db2StartOptionsStruct;

typedef SQL_STRUCTURE db2QuiesceStartStruct
{
            db2int8       iIsQRequested;
            char          *piQUsrName;
            char          *piQGrpName;
            db2int8       iIsQUsrGrpDef;
} db2QuiesceStartStruct;
/* ... */
```

汎用 API 構文:

```
/* File: db2ApiDf.h */
/* API: db2gInstanceStart */
SQL_API_RC SQL_API_FN
  db2gInstanceStart (
            db2Uint32 versionNumber,
            void *pParmStruct,
            struct sqlca *pSqlca);

typedef SQL_STRUCTURE db2gInstanceStStruct
{
            db2int8       iIsRemote;
            db2Uint32     iRemoteInstLen;
            char          *piRemoteInstName;
            db2gDasCommData *piCommData;
            db2gStartOptionsStruct *piStartOpts;
} db2gInstanceStStruct;

typedef SQL&STRUCTURE db2gDasCommData
{
            db2int8       iCommParam;
            db2Uint32     iNodeOrHostNameLen;
            char          *piNodeOrHostName;
```

```
            db2Uint32     iUserIdLen;
            char          *piUserId;
            db2Uint32     iUserPwLen;
            char          *piUserPw;
} db2gDasCommData;

typedef SQL_STRUCTURE db2gStartOptionsStruct
{
            db2Uint32     iIsProfile;
            char          *piProfile;
            db2Uint32     iIsNodeNum;
            db2NodeType   iNodeNum;
            db2Uint32     iOption;
            db2Uint32     iIsHostName;
            char          *piHostName;
            db2Uint32     iIsPort;
            db2PortType   iPort;
            db2Uint32     iIsNetName;
            char          *piNetName;
            db2Uint32     iTblspaceType;
            db2NodeType   iTblspaceNode;
            db2Uint32     iIsComputer;
            char          *piComputer;
            char          *piUserName;
            char          *piPassword;
            db2gQuiesceStartStruct  iQuiesceOpts;
} db2gStartOptionsStruct;

typedef SQL_STRUCTURE db2gQuiesceStartStruct
{
            db2int8       iIsQRequested;
            db2Uint32     iQUsrNameLen;
            char          *piQUsrName;
            db2Uint32     iQGrpNameLen;
            char          *piQGrpName;
            db2int8       iIsQUsrGrpDef;
} db2gQuiesceStartStruct;
/* ... */
```

API パラメーター:

versionNumber
入力。 2 番目のパラメーター *pParmStruct* として渡される構造のバージョンとリリースのレベルを指定します。

pParmStruct
入力。 *db2InstanceStartStruct* 構造を指すポインター。

pSqlca
出力。 *sqlca* 構造を指すポインター。

iIsRemote
入力。標識を TRUE または FALSE に設定します。リモート開始の場合、このパラメーターは TRUE に設定されます。

iRemoteInstLen
入力。 *piRemoteInstName* の長さ (バイト単位) を指定します。

piRemoteInstName
入力。リモート・インスタンスの名前。

piCommData
入力。 *db2DasCommData* 構造を指すポインター。

piStartOpts
入力。 *db2StartOptionsStruct* 構造を指すポインター。

iCommParam
入力。標識を TRUE または FALSE に設定します。リモート開始の場合、このパラメーターは TRUE に設定されます。

iNodeOrHostNameLen
入力。 *piNodeOrHostName* の長さ (バイト単位) を指定します。

piNodeOrHostName
入力。データベース・パーティションまたはホスト名。

iUserIdLen
入力。 *piUserId* の長さ (バイト単位) を指定します。

piUserId
入力。ユーザー名。

iUserPwLen
入力。 *piUserPw* の長さ (バイト単位) を指定します。

piUserPw
入力。ユーザー・パスワード。

iIsProfile
入力。プロファイルが指定されるかどうかを示します。このフィールドで、プロファイルが指定されないことが示されると、ファイル db2profile が使用されます。

piProfile
入力。 DB2 環境を定義するために各ノードで実行されるプロファイル・ファイルの名前 (MPP のみ)。このファイルは、ノードが開始される前に実行されます。デフォルト値は db2profile です。

iIsNodeNum
入力。ノード番号が指定されるかどうかを示します。指定される場合、開始コマンドは指定されたノードにのみ影響を与えます。

iNodeNum
入力。データベース・パーティション番号。

iOption
入力。アクションを指定します。 *OPTION* に有効な値 (sqlenv.h で定義) は、以下のとおりです。

SQLE_NONE
通常の db2start 操作を発行します。

SQLE_ADDNODE
ADD NODE コマンドを発行します。

SQLE_RESTART
RESTART DATABASE コマンドを発行します。

SQLE_STANDALONE
STANDALONE モードでノードを開始します。

iIsHostName
入力。ホスト名が指定されるかどうかを示します。

piHostName
入力。システム名。

iIsPort
入力。ポート番号が指定されるかどうかを示します。

iPort 入力。ポート番号。

iIsNetName
入力。ネット名が指定されるかどうかを示します。

piNetName
入力。ネットワーク名。

iTbIspaceType
入力。追加されるノードのために使用される SYSTEM TEMPORARY 表スペース定義のタイプを指定します。有効な値は以下のとおりです。

SQLE_TABLESPACES_NONE
システム TEMPORARY 表スペースを作成しません。

SQLE_TABLESPACES_LIKE_NODE
システム TEMPORARY 表スペース用のコンテナーは、指定されたノード用のものと同じでなければなりません。

SQLE_TABLESPACES_LIKE_CATALOG
システム TEMPORARY 表スペース用のコンテナーは、各データベースのカタログ・ノード用のものと同じでなければなりません。

iTbIspaceNode
入力。 SYSTEM TEMPORARY 表スペース定義を取得するノード番号を指定します。このノード番号は、`db2nodes.cfg` ファイルに存在しなければならず、 *tblspace_type* フィールドが `SQLE_TABLESPACES_LIKE_NODE` に設定される場合にのみ使用されます。

iIsComputer
入力。コンピューター名が指定されるかどうかを示します。 Windows オペレーティング・システムでのみ有効です。

piComputer
入力。コンピューター名。 Windows オペレーティング・システムでのみ有効です。

piUserName
入力。ログオン・アカウント・ユーザー名。 Windows オペレーティング・システムでのみ有効です。

piPassword
入力。ログオン・アカウントのユーザー名に対応するパスワード。

iQuiesceOpts
入力。 *db2QuiesceStartStruct* 構造を指すポインター。

iIsQRequested
入力。標識を TRUE または FALSE に設定します。静止が必要な場合、このパラメーターは TRUE に設定する必要があります。

iQUsrNameLen
入力。 *piQusrName* の長さ (バイト単位) を指定します。

piQUsrName
入力。静止ユーザー名。

iQGrpNameLen
入力。 *piQGrpName* の長さ (バイト単位) を指定します。

piQGrpName
入力。静止グループ名。

iIsQUsrGrpDef
入力。標識を TRUE または FALSE に設定します。静止ユーザーまたは静止グループが定義されている場合、このパラメーターは TRUE に設定する必要があります。

関連資料:

- 453 ページの『SQLCA』
- 149 ページの『db2InstanceStop - インスタンスの停止』

関連サンプル:

- 『instart.c -- Stop and start the current local instance (C)』
- 『instart.C -- Stop and start the current local instance (C++)』

db2InstanceStop - インスタンスの停止

ローカルまたはリモート DB2 インスタンスを停止します。

有効範囲:

単一パーティション・データベース環境では、有効範囲は単一データベース・パーティションのみになります。複数パーティション・データベース環境では、ノード構成ファイル db2nodes.cfg に定義されたデータベース・パーティション・サーバーすべてを集めたものが有効範囲です。

許可:

以下のいずれかです。

- *sysadm*
- *sysctrl*
- *sysmaint*

必要な接続:

なし

API 組み込みファイル:

db2ApiDf.h

sqlenv.h

C API 構文:

```
/* File: db2ApiDf.h */
/* API: db2InstanceStop */
/* ... */
SQL_API_RC SQL_API_FN
db2InstanceStop (
  db2Uint32 versionNumber,
  void *pParmStruct,
  struct sqlca *pSqlca);

typedef SQL_STRUCTURE db2InstanceStopStruct
{
             db2int8       iIsRemote;
             char          *piRemoteInstName;
             db2DasCommData *piCommData;
             db2StopOptionsStruct *piStopOpts;
} db2InstanceStopStruct;

typedef SQL_STRUCTURE db2DasCommData
{
             db2int8       iCommParam;
             char          *piNodeOrHostName;
             char          *piUserId;
             char          *piUserPw;
} db2DasCommData;

typedef SQL_STRUCTURE db2StopOptionsStruct
{
             db2Uint32     iIsProfile;
             char          *piProfile;
             db2Uint32     iIsNodeNum;
             db2NodeType   iNodeNum;
             db2Uint32     iStopOption;
             db2Uint32     iCallerac;
} db2StopOptionsStruct;
/* ... */
```

汎用 API 構文:

```
/* File: db2ApiDf.h */
/* API: db2gInstanceStop */
/* ... */
SQL_API_RC SQL_API_FN
db2gInstanceStop (
  db2Uint32 versionNumber,
  void *pParmStruct,
  struct sqlca *pSqlca);

typedef SQL_STRUCTURE db2gInstanceStopStruct
{
             db2int8       iIsRemote;
             db2Uint32     iRemoteInstLen;
             char          *piRemoteInstName;
             db2gDasCommData *piCommData;
             db2StopOptionsStruct *piStopOpts;
} db2gInstanceStopStruct;

typedef SQL_STRUCTURE db2gDasCommData
{
             db2int8       iCommParam;
             db2Uint32     iNodeOrHostNameLen;
             char          *piNodeOrHostName;
             db2Uint32     iUserIdLen;
             char          *piUserId;
```

```
            db2Uint32     iUserPwLen;
            char          *piUserPw;
} db2gDasCommData;

typedef SQL_STRUCTURE db2StopOptionsStruct
{
            db2Uint32     iIsProfile;
            char          *piProfile;
            db2Uint32     iIsNodeNum;
            db2NodeType   iNodeNum;
            db2Uint32     iStopOption;
            db2Uint32     iCallerac;
} db2StopOptionsStruct;
/* ... */
```

API パラメーター:

versionNumber
入力。 2 番目のパラメーター *pParmStruct* として渡される構造のバージョンとリリースのレベルを指定します。

pParmStruct
入力。 *db2InstanceStopStruct* 構造を指すポインター。

pSqlca
出力。 *sqlca* 構造を指すポインター。

iIsRemote
入力。標識を TRUE または FALSE に設定します。リモート開始の場合、このパラメーターは TRUE に設定されます。

iRemoteInstLen
入力。 *piRemoteInstName* の長さ (バイト単位) を指定します。

piRemoteInstName
入力。リモート・インスタンスの名前。

piCommData
入力。 *db2DasCommData* 構造を指すポインター。

piStopOpts
入力。 *db2StopOptionsStruct* 構造を指すポインター。

iCommParam
入力。標識を TRUE または FALSE に設定します。リモート停止の場合、このパラメーターは TRUE に設定されます。

iNodeOrHostNameLen
入力。 *piNodeOrHostName* の長さ (バイト単位) を指定します。

piNodeOrHostName
入力。データベース・パーティションまたはホスト名。

iUserIdLen
入力。 *piUserId* の長さ (バイト単位) を指定します。

piUserId
入力。ユーザー名。

iUserPwLen
入力。 *piUserPw* の長さ (バイト単位) を指定します。

piUserPw
入力。ユーザー・パスワード。

iIsRemote
入力。標識を TRUE または FALSE に設定します。リモート停止の場合、このパラメーターは TRUE に設定されます。

iRemoteInstLen
入力。 *piRemoteInstName* の長さ (バイト単位) を指定します。

piRemoteInstName
入力。リモート・インスタンス名。

iIsProfile
入力。プロファイルが指定されるかどうかを示します。指定可能な値は TRUE および FALSE です。このフィールドで、プロファイルが指定されないことが示されると、ファイル db2profile が使用されます。

piProfile
入力。開始されるノード用の DB2 環境を定義するために始動時に実行されたプロファイル・ファイルの名前 (MPP のみ)。 db2InstanceStart API のプロファイルが指定された場合には、ここで同じプロファイルを指定しなければなりません。

iIsNodeNum
入力。ノード番号が指定されるかどうかを示します。指定可能な値は TRUE および FALSE です。指定される場合、停止コマンドは指定されたノードにのみ影響を与えます。

iNodeNum
入力。データベース・パーティション番号。

iStopOption
入力。オプション。有効な値は以下のとおりです。

SQLE_NONE
通常の db2stop 操作を発行します。

SQLE_FORCE
FORCE APPLICATION (ALL) コマンドを発行します。

SQLE_DROP
db2nodes.cfg ファイルからノードをドロップします。

iCallerac
入力。このフィールドは、OPTION フィールドの値が SQLE_DROP である場合にのみ有効です。有効な値は以下のとおりです。

SQLE_DROP
最初の呼び出し。これはデフォルト値です。

SQLE_CONTINUE
後続の呼び出し。プロンプトが出された後に処理を継続します。

SQLE_TERMINATE
後続の呼び出し。プロンプトが出された後に処理を終了します。

関連資料:

- 453 ページの『SQLCA』
- 144 ページの『db2InstanceStart - インスタンスの開始』

関連サンプル:

- 『instart.c -- Stop and start the current local instance (C)』
- 『instart.C -- Stop and start the current local instance (C++)』

db2InstanceUnquiesce - インスタンスの静止解除

インスタンス内のすべてのデータベースを静止解除します。

許可:

以下のいずれかです。

- *sysadm*
- *sysctrl*

必要な接続:

なし

API 組み込みファイル:

db2ApiDf.h

C API 構文:

```
/* File: db2ApiDf.h */
/* API: db2InstanceUnquiesce */
/* ... */
SQL_API_RC SQL_API_FN
db2InstanceUnquiesce (
  db2Uint32 versionNumber,
  void *pParmStruct,
  struct sqlca *pSqlca);

typedef SQL_STRUCTURE db2InsUnquiesceStruct
{
             char          *piInstanceName
} db2InsUnquiesceStruct;
/* ... */
```

汎用 API 構文:

```
/* File: db2ApiDf.h */
/* API: db2gInstanceUnquiesce */
/* ... */
SQL_API_RC SQL_API_FN
db2gInstanceUnquiesce (
  db2Uint32 versionNumber,
  void *pParmStruct,
  struct sqlca *pSqlca);

typedef SQL_STRUCTURE db2gInsUnquiesceStruct
{
             db2Uint32     iInstanceNameLen;
             char          *piInstanceName;
} db2gInsUnquiesceStruct;
/* ... */
```

API パラメーター:

versionNumber
入力。 2 番目のパラメーター *pParmStruct* として渡される構造のバージョンとリリースのレベルを指定します。

pParmStruct
入力。 *db2InsUnquiesceStruct* 構造を指すポインター。

pSqlca
出力。 *sqlca* 構造を指すポインター。

iInstanceNameLen
入力。 *piInstanceName* の長さ (バイト単位) を指定します。

piInstanceName
入力。インスタンス名。

関連資料:

- 453 ページの『SQLCA』
- 142 ページの『db2InstanceQuiesce - インスタンスの静止』

db2LdapCatalogDatabase - データベース LDAP 項目のカタログ

LDAP (Lightweight Directory Access Protocol) のデータベース項目をカタログします。

許可:

なし

必要な接続:

なし

API 組み込みファイル:

db2ApiDf.h

C API 構文:

```
/* File: db2ApiDf.h */
/* API: db2LdapCatalogDatabase */
/* ... */
SQL_API_RC SQL_API_FN
  db2LdapCatalogDatabase(
    sqlint32 versionNumber,
    void *pParamStruct,
    struct sqlca *pSqlca);

typedef struct
{
 char *piAlias;
 char *piDatabaseName;
 char *piComment
 char *piNodeName;
 char *piGWNodeName;
 char *piParameters;
 char *piARLibrary;
 unsigned short iAuthentication;
```

```
 char *piDCEPrincipalName;
 char *piBindDN;
 char *piPassword;
} db2LdapCatalogDatabaseStruct;
/* ... */
```

API パラメーター:

versionNumber
入力。 2 番目のパラメーター *pParamStruct* として渡される構造のバージョンとリリースのレベルを指定します。

pParamStruct
入力。 *db2LdapCatalogDatabaseStruct* 構造を指すポインター。

pSqlca
出力。*sqlca* 構造へのポインター。

piAlias
入力。カタログしているデータベースの代替名として使用する別名を指定します。別名を指定しない場合、データベース・マネージャーはデータベース名を別名として使用します。

piDatabaseName
入力。カタログするデータベースの名前を指定します。このパラメーターは、必須です。

piComment
入力。 DB2 サーバーについて記述します。ネットワーク・ディレクトリーに登録されているサーバーについての記述を補足する、任意のコメントを入力できます。最大長は 30 文字です。復帰文字や改行文字は許可されません。

piNodeName
入力。データベースが存在しているデータベース・サーバーのノード名を指定します。データベースがリモート・データベース・サーバーに存在している場合は、このパラメーターが必要です。

piGWNodename
入力。 DB2 Connect ゲートウェイ・サーバーのノード名を指定します。データベース・サーバーのノード・タイプが DCS (ホスト・データベース・サーバー用に予約済み) で、クライアントに DB2 Connect がインストールされていない場合、クライアントは DB2 Connect ゲートウェイ・サーバーに接続します。

piParameters
入力。アプリケーション・リクエスター (AR) に渡されるパラメーター・ストリングを指定します。認証 DCE は、サポートされていません。

piARLibrary
入力。アプリケーション・リクエスター (AR) ライブラリーの名前を指定します。

iAuthentication
入力。認証タイプを指定すると、パフォーマンスが向上する場合があります。

piDCEPrincipalName
入力。ターゲット・サーバーの完全修飾 DCE プリンシパル名を指定します。

piBindDN
入力。ユーザーの LDAP 識別名 (DN) を指定します。 LDAP ユーザー DN には、 LDAP ディレクトリーでオブジェクトを作成して更新するための十分な権限が必要です。ユーザーの LDAP DN が指定されていない場合は、現在のログオン・ユーザーの認証情報が使用されます。

piPassword
入力。アカウント・パスワードを示します。

使用上の注意:

以下の場合は、データベースを手動で LDAP に登録またはカタログする必要があります。

- データベース・サーバーで LDAP がサポートされていない。この場合、LDAP をサポートするクライアントが、それぞれのクライアント・マシンでローカルにデータベースをカタログしなくても、データベースにアクセスできるようにするには、管理者がそれぞれのデータベースを手作業で LDAP に登録する必要があります。
- アプリケーションで、別の名前を使用して、データベースに接続する。この場合、管理者は別の別名を使用してデータベースをカタログする必要があります。
- CREATE DATABASE IN LDAP 時に、データベース名がすでに LDAP に存在している。データベースは依然としてローカル・マシン上で作成されています (ローカル・アプリケーションからアクセス可能) が、 LDAP にある既存の項目には、新規データベースの内容は反映されません。この場合、管理者は以下のようにすることができます。
  - LDAP の既存のデータベース項目を削除し、LDAP に新規のデータベースを手作業で登録する。
  - 別の別名を使用して、LDAP に新規のデータベースを登録する。

関連資料:

- 453 ページの『SQLCA』

db2LdapCatalogNode - ノード LDAP 項目のカタログ

LDAP (Lightweight Directory Access Protocol) にあるノード項目の代替名、またはデータベース・サーバーに接続する別のプロトコル・タイプを指定します。

許可:

なし

必要な接続:

なし

API 組み込みファイル:

db2ApiDf.h

C API 構文:

```
/* File: db2ApiDf.h */
/* API: db2LdapCatalogNode */
/* ... */
SQL_API_RC SQL_API_FN
  db2LdapCatalogNode(
    sqlint32 versionNumber,
    void *pParamStruct,
    struct sqlca *pSqlca);

typedef struct
{
  char *piAlias;
  char *piNodeName;
  char *piBindDN;
  char *piPassword;
} db2LdapCatalogNodeStruct;
/* ... */
```

API パラメーター:

versionNumber
入力。 2 番目のパラメーター *pParamStruct* として渡される構造のバージョンとリリースのレベルを指定します。

pParamStruct
入力。 *db2LdapCatalogNodeStruct* 構造を指すポインター。

pSqlca
出力。*sqlca* 構造へのポインター。

piAlias
入力。ノード項目の代替名として使用する新規の別名を指定します。

piNodeName
入力。 LDAP にある DB2 サーバーを表すノード名を指定します。

piBindDN
入力。ユーザーの LDAP 識別名 (DN) を指定します。 LDAP ユーザー DN には、 LDAP ディレクトリーでオブジェクトを作成して更新するための十分な権限が必要です。ユーザーの LDAP DN が指定されていない場合は、現在のログオン・ユーザーの認証情報が使用されます。

piPassword
入力。アカウント・パスワードを示します。

関連資料:

• 453 ページの『SQLCA』

db2LdapDeregister - LDAP 登録解除サーバー

LDAP (Lightweight Directory Access Protocol) から DB2 サーバーの登録を解除します。

許可:

なし

必要な接続:

なし

API 組み込みファイル:

db2ApiDf.h

C API 構文:

```
/* File: db2ApiDf.h */
/* API: db2LdapDeregister */
/* ... */
SQL_API_RC SQL_API_FN
  db2LdapDeregister (
    sqlint32 versionNumber,
    void *pParamStruct,
    struct sqlca *pSqlca);

typedef struct
{
  char *piNodeName;
  char *piBindDN;
  char *piPassword;
} db2LdapDeregisterStruct;
/* ... */
```

API パラメーター:

versionNumber
入力。 2 番目のパラメーター *pParamStruct* として渡される構造のバージョンとリリースのレベルを指定します。

pParamStruct
入力。 *db2LdapDeregisterStruct* 構造を指すポインター。

pSqlca
出力。*sqlca* 構造へのポインター。

piNodeName
入力。 LDAP にある DB2 サーバーを表す短縮名を指定します。

piBindDN
入力。ユーザーの LDAP 識別名 (DN) を指定します。 LDAP ユーザー DN には、 LDAP ディレクトリーからオブジェクトを削除するための十分な権限が必要です。ユーザーの LDAP DN が指定されていない場合は、現在のログオン・ユーザーの認証情報が使用されます。

piPassword
入力。アカウント・パスワードを示します。

関連資料:

- 453 ページの『SQLCA』

db2LdapRegister - LDAP 登録サーバー

LDAP (Lightweight Directory Access Protocol) に DB2 サーバーを登録します。

許可:

なし

必要な接続:

なし

API 組み込みファイル:

db2ApiDf.h

C API 構文:

```
/* File: db2ApiDf.h */
/* API: db2LdapRegister */
/* ... */
SQL_API_RC SQL_API_FN
  db2LdapRegister (
    sqlint32 versionNumber,
    void *pParamStruct,
    struct sqlca *pSqlca);

typedef struct
{
  char *piNodeName;
  char *piComputer;
  char *piInstance;
  unsigned short iNodeType;
  db2LdapProtocolInfo iProtocol;
  char *piComment;
  char *piBindDN;
  char *piPassword;
} db2LdapRegisterStruct;

typedef struct
{
  char iType;
  char *piHostName;
  char *piServiceName;
  char *piNetbiosName;
  char *piNetworkId;
  char *piPartnerLU;
  char *piTPName;
  char *piMode;
  unsigned short iSecurityType;
  char *piLanAdapterAddress;
  char *piChangePasswordLU;
  char *piIpxAddress;
} db2LdapProtocolInfo;
/* ... */
```

API パラメーター:

versionNumber
入力。 2 番目のパラメーター *pParamStruct* として渡される構造のバージョンとリリースのレベルを指定します。

pParamStruct
入力。 *db2LdapRegisterStruct* 構造を指すポインター。

pSqlca
出力。*sqlca* 構造へのポインター。

piNodeName
入力。 LDAP にある DB2 サーバーを表す短縮名 (8 文字未満) を指定します。

piComputer
入力。 DB2 サーバーが存在しているコンピューター・システムの名前を指定します。コンピューター名の値は、 LDAP にサーバー・マシンを追加するときの値と同じでなければなりません。 Windows NT の場合、この値は NT コンピューター名になります。 UNIX ベースのシステムの場合は、 TCP/IP ホスト名です。 OS/2 の場合は、 **DB2SYSTEM** レジストリー変数に指定されている値になります。ローカル・コンピューターに DB2 サーバーを登録する場合は、NULL を指定してください。

piInstance
入力。 DB2 サーバーのインスタンス名を指定します。リモート・サーバーを登録するコンピューター名が指定されている場合は、インスタンス名を指定してください。現在のインスタンスを登録する場合は、 NULL を指定します (**DB2SYSTEM** 環境変数に定義されているとおり)。

iNodeType
入力。データベース・サーバーのノード・タイプを指定します。有効な値は以下のとおりです。

```
SQLF_NT_SERVER
SQLF_NT_MPP
SQLF_NT_DCS
```

iProtocol
入力。 *db2LdapProtocolInfo* 構造内でプロトコル情報を指定します。

piComment
入力。 DB2 サーバーについて記述します。ネットワーク・ディレクトリーに登録されているサーバーについての記述を補足する、任意のコメントを入力できます。最大長は 30 文字です。復帰文字や改行文字は許可されません。

piBindDN
入力。ユーザーの LDAP 識別名 (DN) を指定します。 LDAP ユーザー DN には、 LDAP ディレクトリーでオブジェクトを作成して更新するための十分な権限が必要です。ユーザーの LDAP DN が指定されていない場合は、現在のログオン・ユーザーの認証情報が使用されます。

piPassword
入力。アカウント・パスワードを示します。

iType
入力。このサーバーがサポートするプロトコル・タイプを指定します。サー

バーが 2 つ以上のプロトコルをサポートする場合は、複数の登録 (それぞれノード名とプロトコル・タイプが異なる) を行う必要があります。有効な値は以下のとおりです。

```
SQL_PROTOCOL_APPN   - APPC/APPN のサポート
SQL_PROTOCOL_NETB   - NetBIOS のサポート
SQL_PROTOCOL_TCPIP  - TCP/IP のサポート
SQL_PROTOCOL_SOCKS  - ソケット・セキュリティー付きの TCP/IP
SQL_PROTOCOL_IPXSPX - IPX/SPX のサポート
SQL_PROTOCOL_NPIPE  - Windows NT 名前付きパイプのサポート
```

piHostName

入力。 TCP/IP ホスト名または IP アドレスを指定します。

piServiceName

入力。 TCP/IP サービス名またはポート番号を指定します。

piNetbiosName

入力。 NetBIOS ワークステーション名を指定します。 NetBIOS 名は、NetBIOS をサポートする場合に指定します。

piNetworkID

入力。ネットワーク ID を指定します。ネットワーク ID は、APPC/APPN をサポートする場合に指定します。

piPartnerLU

入力。 DB2 サーバー・マシンのパートナー LU 名を指定します。パートナー LU は、APPC/APPN をサポートする場合に指定します。

piTPName

入力。トランザクション・プログラム名を指定します。トランザクション・プログラム名は、APPC/APPN をサポートする場合に指定します。

piMode

入力。モード名を指定します。モードは、APPC/APPN をサポートする場合に指定します。

iSecurityType

入力。 APPC セキュリティー・レベルを指定します。有効な値は以下のとおりです。

```
SQL_CPIC_SECURITY_NONE (default)
SQL_CPIC_SECURITY_SAME
SQL_CPIC_SECURITY_PROGRAM
```

piLanAdapterAddress

入力。ネットワーク・アダプター・アドレスを指定します。このパラメーターが必要なのは、APPC をサポートする場合だけです。 APPN の場合は、このパラメーターを NULL に設定できます。

piChangePasswordLU

入力。ホスト・データベース・サーバーのパスワードを変更する際に使用するパートナー LU の名前を指定します。

piIpxAddress

入力。完全な IPX アドレスを指定します。 IPX アドレスは、IPX/SPX をサポートする場合に指定します。

使用上の注意:

ユニークなノード名を指定する場合は、そのつどサーバーがサポートするプロトコルごとに DB2 サーバーを登録してください。

DB2 サーバーをローカルに登録するときにプロトコル構成パラメーターが指定されていると、データベース・マネージャー構成ファイルに指定されている値がオーバーライドされます。

LDAP に登録可能なのは、リモート DB2 サーバーだけです。リモート・サーバーのプロトコル通信の他に、コンピューター名とインスタンス名も指定する必要があります。

ホスト・データベース・サーバーを登録する場合は、 *iNodeType* パラメーターに SQLF_NT_DCS 値を指定する必要があります。

関連資料:

• 453 ページの『SQLCA』

---

db2LdapUncatalogDatabase - データベース LDAP 項目のアンカタログ

LDAP (Lightweight Directory Access Protocol) からデータベース項目を除去します。

許可:

なし

必要な接続:

なし

API 組み込みファイル:

db2ApiDf.h

C API 構文:

```
/* File: db2ApiDf.h */
/* API: db2LdapUncatalogDatabase */
/* ... */
SQL_API_RC SQL_API_FN
  db2LdapUncatalogDatabase(
    sqlint32 versionNumber,
    void *pParamStruct,
    struct sqlca *pSqlca);

typedef struct
{
  char *piAlias[SQL_ALIAS_SZ];
  char *piBindDN;
  char *piPassword;
} db2LdapUncatalogDatabaseStruct;
/* ... */
```

API パラメーター:

versionNumber
入力。 2 番目のパラメーター *pParamStruct* として渡される構造のバージョンとリリースのレベルを指定します。

pParamStruct
入力。 *db2LdapUncatalogDatabaseStruct* 構造を指すポインター。

pSqlca
出力。*sqlca* 構造へのポインター。

piAlias
入力。データベース項目の別名を指定します。このパラメーターは、必須です。

piBindDN
入力。ユーザーの LDAP 識別名 (DN) を指定します。 LDAP ユーザー DN には、 LDAP ディレクトリーからオブジェクトを削除するための十分な権限が必要です。ユーザーの LDAP DN が指定されていない場合は、現在のログオン・ユーザーの認証情報が使用されます。

piPassword
入力。アカウント・パスワードを示します。

関連資料:

• 453 ページの『SQLCA』

db2LdapUncatalogNode - ノード LDAP 項目のアンカタログ

LDAP (Lightweight Directory Access Protocol) からノード項目を除去します。

許可:

なし

必要な接続:

なし

API 組み込みファイル:

db2ApiDf.h

C API 構文:

```
/* File: db2ApiDf.h */
/* API: db2LdapUncatalogNode */
/* ... */
SQL_API_RC SQL_API_FN
  db2LdapUncatalogNode(
    sqlint32 versionNumber,
    void *pParamStruct,
    struct sqlca *pSqlca);

typedef struct
{
  char *piAlias;
  char *piBindDN;
  char *piPassword;
} db2LdapUncatalogNodeStruct;
/* ... */
```

API パラメーター:

versionNumber
入力。 2 番目のパラメーター *pParmStruct* として渡される構造のバージョンとリリースのレベルを指定します。

pParamStruct
入力。 *db2LdapUncatalogNodeStruct* 構造を指すポインター。

pSqlca
出力。*sqlca* 構造へのポインター。

piAlias
入力。 LDAP からアンカタログするノードの別名を指定します。

piBindDN
入力。ユーザーの LDAP 識別名 (DN) を指定します。 LDAP ユーザー DN には、 LDAP ディレクトリーからオブジェクトを削除するための十分な権限が必要です。ユーザーの LDAP DN が指定されていない場合は、現在のログオン・ユーザーの認証情報が使用されます。

piPassword
入力。アカウント・パスワードを示します。

関連資料:

• 453 ページの『SQLCA』

db2LdapUpdate - LDAP 更新サーバー

LDAP (Lightweight Directory Access Protocol) にある DB2 サーバーの通信プロトコル情報を更新します。

許可:

なし

必要な接続:

なし

API 組み込みファイル:

db2ApiDf.h

C API 構文:

```
/* File: db2ApiDf.h */
/* API: db2LdapUpdate */
/* ... */
SQL_API_RC SQL_API_FN
  db2LdapUpdate (
    sqlint32 versionNumber,
    void *pParamStruct,
    struct sqlca *pSqlca);

typedef struct
{
  char *piNodeName;
  char *piComment;
```

```
  unsigned short iNodeType;
  db2LdapProtocolInfo iProtocol;
  char *piBindDN;
  char *piPassword;
} db2LdapUpdateStruct;

typedef struct
{
  char iType;
  char *piHostName;
  char *piServiceName;
  char *piNetbiosName;
  char *piNetworkId;
  char *piPartnerLU;
  char *piTPName;
  char *piMode;
  unsigned short iSecurityType;
  char *piLanAdapterAddress;
  char *piChangePasswordLU;
  char *piIpxAddress;
} db2LdapProtocolInfo;
/* ... */
```

API パラメーター:

versionNumber
入力。 2 番目のパラメーター *pParamStruct* として渡される構造のバージョンとリリースのレベルを指定します。

pParamStruct
入力。 *db2LdapUpdateStruct* 構造を指すポインター。

pSqlca
出力。*sqlca* 構造へのポインター。

piNodeName
入力。 LDAP にある DB2 サーバーを表すノード名を指定します。

piComment
入力。 DB2 サーバーの新しい説明を指定します。最大長は 30 文字です。復帰文字や改行文字は許可されません。

iNodeType
入力。新規のノード・タイプを指定します。有効な値は以下のとおりです。

```
SQLF_NT_SERVER
SQLF_NT_MPP
SQLF_NT_DCS
SQL_PARM_UNCHANGE
```

iProtocol
入力。 *db2LdapProtocolInfo* 構造内で更新済みのプロトコル情報を指定します。

piBindDN
入力。ユーザーの LDAP 識別名 (DN) を指定します。 LDAP ユーザー DN には、 LDAP ディレクトリーでオブジェクトを作成して更新するための十分な権限が必要です。ユーザーの LDAP DN が指定されていない場合は、現在のログオン・ユーザーの認証情報が使用されます。

piPassword
入力。アカウント・パスワードを示します。

iType

入力。このサーバーがサポートするプロトコル・タイプを指定します。有効な値は以下のとおりです。

```
SQL_PROTOCOL_APPN   - APPC/APPN のサポート
SQL_PROTOCOL_NETB   - NetBIOS のサポート
SQL_PROTOCOL_TCPIP  - TCP/IP のサポート
SQL_PROTOCOL_SOCKS  - ソケット・セキュリティー付きの TCP/IP
SQL_PROTOCOL_IPXSPX - IPX/SPX のサポート
SQL_PROTOCOL_NPIPE  - Windows NT 名前付きパイプのサポート
```

piHostName

入力。新規の TCP/IP ホスト名または IP アドレスを指定します。

piServiceName

入力。新規の TCP/IP サービス名またはポート番号を指定します。

piNetbiosName

入力。新規の NetBIOS ワークステーション名を指定します。

piNetworkID

入力。新規のネットワーク ID を指定します。

piPartnerLU

入力。 DB2 サーバー・マシンの新しいパートナー LU 名を指定します。

piTPName

入力。新規のトランザクション・プログラム名を指定します。

piMode

入力。新規のモード名を指定します。

iSecurityType

入力。新規のセキュリティー・レベルを指定します。有効な値は以下のとおりです。

```
SQL_CPIC_SECURITY_NONE
SQL_CPIC_SECURITY_SAME
SQL_CPIC_SECURITY_PROGRAM
SQL_PARM_UNCHANGE
```

piLanAdapterAddress

入力。新規のネットワーク・アダプター・アドレスを指定します。

piChangePasswordLU

入力。ホスト・データベース・サーバーのパスワードを変更する際に使用するパートナー LU の新しい名前を指定します。

piIpxAddress

入力。新規の IPX アドレスを指定します。

関連資料:

• 453 ページの『SQLCA』

db2LdapUpdateAlternateServerForDB - データベースの代替サーバーの LDAP 更新

データベースに関連した代替サーバーを Lightweight Directory Access Protocol (LDAP) 内で更新します。

許可:

LDAP サーバーへの読み取り/書き込みアクセス。

必要な接続:

なし

API 組み込みファイル:

db2ApiDf.h

C API 構文:

```
/* File: db2ApiDf.h */
/* API: db2LdapUpdateAlternateServerForDB */
/* ... */
SQL_API_RC SQL_API_FN
  db2LdapUpdateAlternateServerForDB (
  db2Uint32 versionNumber,
  void * pParmStruct,
  struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2LdapUpdateAltServerStruct
{
  char                          *piDbAlias;
  char                          *piNode;
  char                          *piGWNode;
  char                          *piBindDN;
  char                          *piPassword;
} db2LdapUpdateAltServerStruct;
/* ... */
```

API パラメーター:

versionNumber
入力。 2 番目のパラメーター *pParmStruct* として渡される構造のバージョンとリリースのレベルを指定します。

pParmStruct
入力。 *db2LdapUpdateAltServerStruct* 構造を指すポインター。

pSqlca
出力。*sqlca* 構造へのポインター。

piDbAlias
入力。更新されるデータベースの別名を含むストリング。

piNode
入力。代替ノード名を含むストリング。このノード名は、LDAP に存在していなければなりません。

piGWNode
入力。代替ゲートウェイ・ノード名を含むストリング。このノード名は、LDAP に存在していなければなりません。これは、ゲートウェイを介してホストに接続するためにランタイム・クライアントによって使用されます。

piBindDN
入力。ユーザーの LDAP 識別名 (DN) を指定します。ユーザーの LDAP DN には、 LDAP ディレクトリーでオブジェクトを作成および更新するための十分な権限が必要です。ユーザーの LDAP DN が指定されていない場合は、現在のユーザーの認証情報が使用されます。

piPassword
入力。アカウント・パスワードを示します。

関連資料:

- 453 ページの『SQLCA』
- 154 ページの『db2LdapCatalogDatabase - データベース LDAP 項目のカタログ』
- 162 ページの『db2LdapUncatalogDatabase - データベース LDAP 項目のアンカタログ』
- 286 ページの『db2UpdateAlternateServerForDB - データベースの代替サーバーの更新』

db2Load - ロード

データを DB2 表にロードします。サーバー上にあるデータは、ファイル、カーソル、テープ、または名前付きパイプの形式とすることができます。リモートで接続しているクライアント上にあるデータは、完全修飾ファイル、カーソル、または名前付きパイプの形式とすることができます。ロード・ユーティリティーは、階層レベルでのデータのロードをサポートしていません。

許可:

以下のいずれかです。

- *sysadm*
- *dbadm*
- データベースのロード権限と以下のもの
  - 表の INSERT 特権 (ロード・ユーティリティーが INSERT モード、TERMINATE モード、または RESTART モードで呼び出される場合)。TERMINATE モードは直前のロード挿入操作を終了するためのもので、RESTART モードは直前のロード挿入操作を再開するためのものです。
  - 表の INSERT および DELETE 特権 (ロード・ユーティリティーが REPLACE モード、 TERMINATE モード、または RESTART モードで呼び出される場合)。 TERMINATE モードは直前のロード置換操作を終了するためのもので、RESTART モードは直前のロード置換操作を再開するためのものです。
  - 例外表の INSERT 特権 (例外表をロード操作の一部として使用する場合)。

注: 一般的に、すべてのロード処理、およびすべての DB2 サーバー処理は、インスタンス所有者に所有されています。これらのすべての処理では、インスタンス

所有者の ID を使用して、必要なファイルにアクセスします。そのため、インスタンス所有者は、誰がコマンドを呼び出すかに関係なく、入力ファイルへの読み取りアクセスを持っている必要があります。

必要な接続:

データベース。暗黙的な接続が可能である場合には、デフォルトのデータベースへの接続が確立されます。

インスタンス。明示的なアタッチは必要ありません。データベースへの接続が確立されている場合には、ローカル・インスタンスへの暗黙的な接続が試みられます。

API 組み込みファイル:

db2ApiDf.h

C API 構文:

```
/* File: db2ApiDf.h */
/* API: Load */
/* ... */
SQL_API_RC SQL_API_FN
  db2Load (
  db2Uint32 versionNumber,
  void * pParmStruct,
  struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2LoadStruct
{
   struct sqlu_media_list *piSourceList;
   struct sqlu_media_list *piLobPathList;
   struct sqldcol *piDataDescriptor;
   struct sqlchar *piActionString;
   char *piFileType;
   struct sqlchar *piFileTypeMod;
   char *piLocalMsgFileName;
   char *piTempFilesPath;
   struct sqlu_media_list *piVendorSortWorkPaths;
   struct sqlu_media_list *piCopyTargetList;
   db2int32 *piNullIndicators;
   struct db2LoadIn *piLoadInfoIn;
   struct db2LoadOut *poLoadInfoOut;
   struct db2PartLoadIn *piPartLoadInfoIn;
   struct db2PartLoadOut *poPartLoadInfoOut;
   db2int16 iCallerAction;
} db2LoadStruct;

typedef SQL_STRUCTURE db2LoadIn
{
   db2Uint64                          iRowcount;
   db2Uint64                          iRestartcount;
   char                               *piUseTablespace;
   db2Uint32                          iSavecount;
   db2Uint32                          iDataBufferSize;
   db2Uint32                          iSortBufferSize;
   db2Uint32                          iWarningcount;
   db2Uint16                          iHoldQuiesce;
   db2Uint16                          iCpuParallelism;
   db2Uint16                          iDiskParallelism;
   db2Uint16                          iNonrecoverable;
   db2Uint16                          iIndexingMode;
   db2Uint16                          iAccessLevel;
   db2Uint16                          iLockWithForce;
```

```
   db2Uint16                          iCheckPending;
   char                               iRestartphase;
   char                               iStatsOpt;
} db2LoadIn;

typedef SQL_STRUCTURE db2LoadOut
{
   db2Uint64                          oRowsRead;
   db2Uint64                          oRowsSkipped;
   db2Uint64                          oRowsLoaded;
   db2Uint64                          oRowsRejected;
   db2Uint64                          oRowsDeleted;
   db2Uint64                          oRowsCommitted;
} db2LoadOut;

typedef SQL_STRUCTURE db2PartLoadIn
{
   char                               *piHostname;
   char                               *piFileTransferCmd;
   char                               *piPartFileLocation;
   struct db2LoadNodeList             *piOutputNodes;
   struct db2LoadNodeList             *piPartitioningNodes;
   db2Uint16                          *piMode;
   db2Uint16                          *piMaxNumPartAgents;
   db2Uint16                          *piIsolatePartErrs;
   db2Uint16                          *piStatusInterval;
   struct db2LoadPortRange            *piPortRange;
   db2Uint16                          *piCheckTruncation;
   char                               *piMapFileInput;
   char                               *piMapFileOutput;
   db2Uint16                          *piTrace;
   db2Uint16                          *piNewline;
   char                               *piDistfile;
   db2Uint16                          *piOmitHeader;
   SQL_PDB_NODE_TYPE                  *piRunStatDBPartNum;
} db2PartLoadIn;

typedef SQL_STRUCTURE db2LoadNodeList
{
   SQL_PDB_NODE_TYPE                  *piNodeList;
   db2Uint16                          iNumNodes;
} db2LoadNodeList;

typedef SQL_STRUCTURE db2LoadPortRange
{
   db2Uint16                          iPortMin;
   db2Uint16                          iPortMax;
} db2LoadPortRange;

typedef SQL_STRUCTURE db2PartLoadOut
{
   db2Uint64                          oRowsRdPartAgents;
   db2Uint64                          oRowsRejPartAgents;
   db2Uint64                          oRowsPartitioned;
   struct db2LoadAgentInfo            *poAgentInfoList;
   db2Uint32                          iMaxAgentInfoEntries;
   db2Uint32                          oNumAgentInfoEntries;
} db2PartLoadOut;

typedef SQL_STRUCTURE db2LoadAgentInfo
{
   db2int32                           oSqlcode;
   db2Uint32                          oTableState;
   SQL_PDB_NODE_TYPE                  oNodeNum;
   db2Uint16                          oAgentType;
} db2LoadAgentInfo;
/* ... */
```

汎用 API 構文:

```
/* File: db2ApiDf.h */
/* API: Load */
/* ... */
SQL_API_RC SQL_API_FN
db2gLoad (
  db2Uint32 versionNumber,
  void * pParmStruct,
  struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2gLoadStruct
{
   struct sqlu_media_list *piSourceList;
   struct sqlu_media_list *piLobPathList;
   struct sqldcol *piDataDescriptor;
   struct sqlchar *piActionString;
   char *piFileType;
   struct sqlchar *piFileTypeMod;
   char *piLocalMsgFileName;
   char *piTempFilesPath;
   struct sqlu_media_list *piVendorSortWorkPaths;
   struct sqlu_media_list *piCopyTargetList;
   db2int32 *piNullIndicators;
   struct db2gLoadIn *piLoadInfoIn;
   struct db2LoadOut *poLoadInfoOut;
   struct db2gPartLoadIn *piPartLoadInfoIn;
   struct db2PartLoadOut *poPartLoadInfoOut;
   db2int16 iCallerAction;
   db2Uint16 iFileTypeLen;
   db2Uint16 iLocalMsgFileLen;
   db2Uint16 iTempFilesPathLen;
} db2gLoadStruct;

typedef SQL_STRUCTURE db2gLoadIn
{
   db2Uint64                          iRowcount;
   db2Uint64                          iRestartcount;
   char                               *piUseTablespace;
   db2Uint32                          iSavecount;
   db2Uint32                          iDataBufferSize;
   db2Uint32                          iSortBufferSize;
   db2Uint32                          iWarningcount;
   db2Uint16                          iHoldQuiesce;
   db2Uint16                          iCpuParallelism;
   db2Uint16                          iDiskParallelism;
   db2Uint16                          iNonrecoverable;
   db2Uint16                          iIndexingMode;
   db2Uint16                          iAccessLevel;
   db2Uint16                          iLockWithForce;
   db2Uint16                          iCheckPending;
   char                               iRestartphase;
   char                               iStatsOpt;
   db2Uint16                          iUseTablespaceLen;
} db2gLoadIn;

typedef SQL_STRUCTURE db2LoadOut
{
   db2Uint64                          oRowsRead;
   db2Uint64                          oRowsSkipped;
   db2Uint64                          oRowsLoaded;
   db2Uint64                          oRowsRejected;
   db2Uint64                          oRowsDeleted;
   db2Uint64                          oRowsCommitted;
} db2LoadOut;

typedef SQL_STRUCTURE db2gPartLoadIn
```

```
{
   char                                *piHostname;
   char                                *piFileTransferCmd;
   char                                *piPartFileLocation;
   struct db2LoadNodeList              *piOutputNodes;
   struct db2LoadNodeList              *piPartitioningNodes;
   db2Uint16                           *piMode;
   db2Uint16                           *piMaxNumPartAgents;
   db2Uint16                           *piIsolatePartErrs;
   db2Uint16                           *piStatusInterval;
   struct db2LoadPortRange             *piPortRange;
   db2Uint16                           *piCheckTruncation;
   char                                *piMapFileInput;
   char                                *piMapFileOutput;
   db2Uint16                           *piTrace;
   db2Uint16                           *piNewline;
   char                                *piDistfile;
   db2Uint16                           *piOmitHeader;
   SQL_PDB_NODE_TYPE                   *piRunStatDBPartNum;
   db2Uint16                           iHostnameLen;
   db2Uint16                           iFileTransferLen;
   db2Uint16                           iPartFileLocLen;
   db2Uint16                           iMapFileInputLen;
   db2Uint16                           iMapFileOutputLen;
   db2Uint16                           iDistfileLen;
} db2gPartLoadIn;

typedef SQL_STRUCTURE db2LoadNodeList
{
   SQL_PDB_NODE_TYPE                   *piNodeList;
   db2Uint16                           iNumNodes;
} db2LoadNodeList;

typedef SQL_STRUCTURE db2LoadPortRange
{
   db2Uint16                           iPortMin;
   db2Uint16                           iPortMax;
} db2LoadPortRange;

typedef SQL_STRUCTURE db2PartLoadOut
{
   db2Uint64                           oRowsRdPartAgents;
   db2Uint64                           oRowsRejPartAgents;
   db2Uint64                           oRowsPartitioned;
   struct db2LoadAgentInfo             *poAgentInfoList;
   db2Uint32                           iMaxAgentInfoEntries;
   db2Uint32                           oNumAgentInfoEntries;
} db2PartLoadOut;

typedef SQL_STRUCTURE db2LoadAgentInfo
{
   db2int32                            oSqlcode;
   db2Uint32                           oTableState;
   SQL_PDB_NODE_TYPE                   oNodeNum;
   db2Uint16                           oAgentType;
} db2LoadAgentInfo;
/* ... */
```

API パラメーター:

versionNumber

入力。 2 番目のパラメーター *pParmStruct* として渡される構造のバージョンとリリースのレベルを指定します。

pParmStruct

入力。 *db2LoadStruct* 構造を指すポインター。

pSqlca

出力。*sqlca* 構造へのポインター。

piSourceList

入力。ソース・ファイル、装置、ベンダー、パイプ、または SQL ステートメントを提供するのに使用される、 *sqlu_media_list* 構造を指すポインター。

この構造に提供される情報は、*media_type* フィールドの値によって異なります。有効な値 (sqlutil で定義) は、以下のとおりです。

SQLU_SQL_STMT

media_type フィールドがこの値に設定されている場合、呼び出し側は、ターゲット・フィールドの *pStatement* フィールドで SQL 照会を提供します。 *pStatement* フィールドは、*sqlu_statement_entry* のタイプです。セッション・フィールドは値を 1 に設定していなければなりません。これは、ロード・ユーティリティーはロードごとに 1 つの SQL 照会だけを受け取るからです。

SQLU_SERVER_LOCATION

media_type フィールドがこの値に設定されている場合、呼び出し側から *sqlu_location_entry* 構造によって情報が提供されます。 *sessions* フィールドは、提供される *sqlu_location_entry* 構造の数を示します。これは、ファイル、装置、および Named PIPE に使用されます。

SQLU_CLIENT_LOCATION

media_type フィールドがこの値に設定されている場合、呼び出し側から *sqlu_location_entry* 構造によって情報が提供されます。 sessions フィールドは、提供される *sqlu_location_entry* 構造の数を示します。これは、完全修飾ファイル、および Named PIPE に使用されます。この *media_type* が有効なのは、リモートで接続されているクライアントを使用して API を呼び出している場合だけであることに注意してください。

SQLU_TSM_MEDIA

media_type フィールドがこの値に設定されている場合、 *sqlu_vendor* 構造が使用されます。*filename* には、ロードされるデータにユニークな ID が入ります。 *sessions* の値がいくつであっても、 *sqlu_vendor* 項目の数は 1 つだけにする必要があります。 *sessions* フィールドは、開始される TSM セッションの数を示します。ロード・ユーティリティーは、異なるシーケンス番号を持つセッションを開始しますが、ロードされるデータは、1 つの *sqlu_vendor* 項目にあるものと同じです。

SQLU_OTHER_MEDIA

media_type フィールドがこの値に設定されている場合、 *sqlu_vendor* 構造が使用されます。*shr_lib* には共有ライブラリー名、 *filename* にはロードされるデータにユニークな ID が入ります。 *sessions* の値がいくつであっても、 *sqlu_vendor* 項目の数は 1 つだけにする必要があります。 *sessions* フィールドは、開始されるその他のベンダー・セッションの数を示します。ロード・ユーティリティーは、異

なるシーケンス番号を持つセッションを開始しますが、ロードされるデータは、1 つの *sqlu_vendor* 項目にあるものと同じです。

piLobPathList

入力。 *sqlu_media_list* 構造を指すポインター。ファイル・タイプが IXF、ASC、および DEL の場合は、ロードされる個々の LOB ファイルのロケーションを識別する、完全修飾パスまたは装置のリスト。ファイル名は、IXF、ASC、または DEL ファイルで検索され、提供されたパスに追加されます。

この構造に提供される情報は、*media_type* フィールドの値によって異なります。有効な値 (`sqlutil` で定義) は、以下のとおりです。

SQLU_LOCAL_MEDIA

この値に設定されている場合、呼び出し側から *sqlu_media_entry* 構造によって情報が提供されます。 *sessions* フィールドは、提供される *sqlu_media_entry* 構造の数を示します。

SQLU_TSM_MEDIA

この値に設定されている場合、*sqlu_vendor* 構造が使用されます。 *filename* には、ロードされるデータにユニークな ID が入ります。 *sessions* の値がいくつであっても、 *sqlu_vendor* 項目の数は 1 つだけにする必要があります。 *sessions* フィールドは、開始される TSM セッションの数を示します。ロード・ユーティリティーは、異なるシーケンス番号を持つセッションを開始しますが、ロードされるデータは、1 つの *sqlu_vendor* 項目にあるものと同じです。

SQLU_OTHER_MEDIA

この値に設定されている場合、*sqlu_vendor* 構造が使用されます。 *shr_lib* には共有ライブラリー名、*filename* にはロードされるデータにユニークな ID が入ります。 *sessions* の値がいくつであっても、 *sqlu_vendor* 項目の数は 1 つだけにする必要があります。 *sessions* フィールドは、開始されるその他のベンダー・セッションの数を示します。ロード・ユーティリティーは、異なるシーケンス番号を持つセッションを開始しますが、ロードされるデータは、1 つの *sqlu_vendor* 項目にあるものと同じです。

piDataDescriptor

入力。外部ファイルからロードするよう選択された列に関する情報を含む *sqldcol* 構造を指すポインター。

pFileType パラメーターが `SQL_ASC` に設定されている場合、この構造の *dcolmeth* フィールドは、`SQL_METH_L` または `SQL_METH_D` に設定し、ファイル名は、開始と終了の対および NULL 標識の位置を含む POSITIONSFILE *pFileTypeMod* 修飾子とともに指定する必要があります。ユーザーは、ロードされる列ごとに開始ロケーションと終了ロケーションを指定します。

ファイル・タイプ `SQL_DEL` の場合、 *dcolmeth* は `SQL_METH_P` または `SQL_METH_D` のどちらかにすることができます。 `SQL_METH_P` の場合、ソース列の位置を提供する必要があります。 `SQL_METH_D` の場合は、ファイル内の最初の列が表の最初の列にロードされ、以下同様に続きます。

ファイル・タイプが `SQL_IXF` の場合、*dcolmeth* は `SQL_METH_P`、`SQL_METH_D`、または `SQL_METH_N` のいずれかにすることができます。この場

合は、SQL_METH_N が *sqldcol* 構造でファイル列名が提供されるべきであることを示す点を除き、 DEL ファイルに関する規則が適用されます。

piActionString

入力。 *sqlchar* 構造を指すポインターと、それに続いて表に影響するアクションを指定する文字の配列。

文字配列の形式は、以下のようになります。

```
"INSERT|REPLACE|RESTART|TERMINATE
INTO tbname [(column_list)]
[DATALINK SPECIFICATION datalink-spec]
[FOR EXCEPTION e_tbname]"
```

INSERT

既存の表データを変更することなく、ロードされたデータを表に追加します。

REPLACE

表から既存データをすべて削除し、ロードされたデータを挿入します。表定義および索引定義は変更されません。

RESTART

以前に割り込みを受けたロード操作を再開します。ロード操作は、ロード、作成、または削除フェーズの最後の整合点から自動的に続行されます。

TERMINATE

以前に割り込みを受けたロード操作を終了し、ロード操作が開始された時点まで操作をロールバックします。途中に整合点があっても通過します。その操作に関係する表スペースの状態は通常に戻され、すべての表オブジェクトの整合性が保たれます (索引オブジェクトが無効とマークされる場合がありますが、そのような場合には、次回のアクセス時に索引の再作成が自動的に行われます)。表の存在する表スペースがロード・ペンディング状態でなければ、このオプションは表スペースの状態に影響しません。

ロード終了オプションでは、表スペースのバックアップ・ペンディング状態は解除されません。

tbname データのロード先の表の名前。システム表または宣言された一時表を指定することはできません。別名、完全修飾、または非修飾の表名を指定できます。修飾された表名は、*schema.tablename* の形式になります。非修飾の表名を指定すると、その表は CURRENT SCHEMA で修飾されます。

(*column_list* **)**

データの挿入先の表の列名のリスト。列名は、コンマで区切らなければなりません。名前にスペースまたは小文字が含まれている場合には、それを引用符で囲まなければなりません。

DATALINK SPECIFICATION *datalink-spec*

DB2 データ・リンクに関連するパラメーターを指定します。これらのパラメーターは、 LOAD コマンドと同じ構文を使って指定できます。

FOR EXCEPTION *e_tbname*
エラーが発生した行のコピー先となる例外表を指定します。ユニーク索引または主キー索引に違反した行がすべてコピーされます。DATALINK 例外も例外表にキャプチャーされます。

piFileType
入力。入力データ・ソースの形式を示すストリング。サポートされている外部の形式 (sqlutil で定義) は、以下のとおりです。

SQL_ASC
区切り文字なし ASCII。

SQL_DEL
区切り文字付き ASCII。これは dBase プログラム、BASIC プログラム、 IBM Personal Decision Series プログラム、およびその他の多数のデータベース・マネージャー/ファイル・マネージャーとの交換のための形式です。

SQL_IXF
IXF (統合交換フォーマットの PC バージョン)。表からデータをエクスポートする場合の推奨方式で、同じ表または別のデータベース・マネージャー表にそれをロードすることが可能です。

SQL_CURSOR
SQL 照会。 *piSourceList* パラメーターによって渡された *sqlu_media_list* 構造のタイプは SQLU_SQL_STMT で、実際の SQL 照会を参照し、それに対して宣言されているカーソルは参照しません。

piFileTypeMod
入力。 *sqlchar* 構造を指すポインターと、それに続いて 1 つ以上の処理オプションを指定する文字の配列。このポインターが NULL であるか、このポインターが指す構造に 1 文字も入っていない場合、このアクションはデフォルトの指定が選択されたものとして解釈されます。

サポートされるすべてのファイル・タイプに、すべてのオプションを使用できるわけではありません。 LOAD のファイル・タイプ修飾子を参照してください。

piLocalMsgFileName
入力。出力メッセージの書き込み先となるローカル・ファイルの名前を含むストリング。

piTempFilesPath
入力。一時ファイル用のサーバー上で使用されるパス名を含むストリング。一時ファイルは、メッセージや整合点を格納したり、フェーズ情報を削除したりするために作成されます。

piVendorSortWorkPaths
入力。ベンダー・ソート作業ディレクトリーを指定する *sqlu_media_list* 構造を指すポインター。

piCopyTargetList
入力。 *sqlu_media_list* 構造へのポインター。これは、 (コピー・イメージを

作成する予定の場合) コピー・イメージの書き込み先となるターゲット・パス、装置、または共有ライブラリーのリストを提供するときに使用します。

この構造に入力する値は、*media_type* フィールドの値によって異なります。このフィールドに有効な値 (sqlutil で定義) は、以下のとおりです。

SQLU_LOCAL_MEDIA
コピーをローカル・メディアに書き込む予定の場合、*media_type* をこの値に設定し、ターゲットに関する情報を *sqlu_media_entry* 構造に提供してください。 *sessions* フィールドは、提供される *sqlu_media_entry* 構造の数を示します。

SQLU_TSM_MEDIA
コピーを TSM に書き込む予定の場合、この値を使用してください。それ以外の情報は特に必要ありません。

SQLU_OTHER_MEDIA
ベンダー製品を使用する予定の場合、この値を使用し、 *sqlu_vendor* 構造を介して追加の情報を提供してください。この構造の *shr_lib* フィールドをベンダー製品の共用ライブラリー名に設定してください。 *sessions* の値に関係なく、1 つの *sqlu_vendor* 項目だけを提供してください。 *sessions* フィールドは、提供される *sqlu_media_entry* 構造の数を示します。ロード・ユーティリティーは、異なるシーケンス番号を持つセッションを開始しますが、ロードされるデータは、1 つの *sqlu_vendor* 項目で提供されているものと同じです。

piNullIndicators
入力。 ASC ファイルの場合にのみ使用します。列データが NULL 可能であるかどうかを示す整数の配列です。この配列のエレメントと、データ・ファイルからロードされる列との間には、 1 対 1 の順序付けられた対応関係があります。要するに、エレメントの数は、*pDataDescriptor* パラメーターの *dcolnum* フィールドと同じでなければなりません。配列の各エレメントには、NULL 標識フィールドとして使用される、データ・ファイル内のロケーションを識別する数値、または表列が NULL 可能ではないことを示すゼロが含まれます。エレメントがゼロでない場合には、データ・ファイル内の識別されたロケーションに Y または N が入っていなければなりません。 Y は表列のデータが NULL であることを示し、 N は表列のデータが NULL ではないことを示します。

piLoadInfoIn
入力。 *db2LoadIn* 構造を指すポインター。

poLoadInfoOut
入力。 *db2LoadOut* 構造を指すポインター。

piPartLoadInfoIn
入力。 *db2PartLoadIn* 構造を指すポインター。

poPartLoadInfoOut
出力。 *db2PartLoadOut* 構造を指すポインター。

iCallerAction

入力。呼び出し側が要求するアクションを示します。有効な値 (sqlutil で定義) は、以下のとおりです。

SQLU_INITIAL
最初の呼び出し。この値 (または SQLU_NOINTERRUPT) は、API への最初の呼び出しの際には必ず使用してください。

SQLU_NOINTERRUPT
最初の呼び出し。処理を中断しません。この値 (または SQLU_INITIAL) は、API への最初の呼び出しの際には必ず使用してください。

最初の呼び出しまたは後続の呼び出しのいずれかが戻され、要求されたロード操作が完了する前に呼び出し側のアプリケーションが何らかのアクションを行うことが必要な場合、呼び出し側のアクションを以下のどちらかに設定する必要があります。

SQLU_CONTINUE
処理の継続。この値を使用できるのは、最初の呼び出しが戻されたときにユーティリティーがユーザー入力 (たとえば、テープの終わり条件への応答) を要求した後で、 API への後続呼び出しを出す場合だけです。この値は、ユーティリティーが要求したユーザー・アクションが完了したら、ユーティリティーが最初の要求の処理を続行するよう指定するものです。

SQLU_TERMINATE
処理の終了。ロード中の表スペースを LOAD_PENDING 状態にしたまま、ロード・ユーティリティーを早期に終了させます。このオプションは、これ以上データの処理が行われない場合に指定します。

SQLU_ABORT
処理の終了。ロード中の表スペースを LOAD_PENDING 状態にしたまま、ロード・ユーティリティーを早期に終了させます。このオプションは、これ以上データの処理が行われない場合に指定します。

SQLU_RESTART
処理の再開。

SQLU_DEVICE_TERMINATE
単一の装置の終了。このオプションは、ユーティリティーが装置からの読み取りを停止しても、データの処理をさらに続ける場合に指定します。

iFileTypeLen

入力。 *iFileType* の長さ (バイト単位) を指定します。

iLocalMsgFileLen

入力。 *iLocalMsgFileName* の長さ (バイト単位) を指定します。

iTempFilesPathLen

入力。 *iTempFilesPath* の長さ (バイト単位) を指定します。

iRowcount
入力。ロードされる物理レコードの数。これを使用すると、ファイル内の最初の *rowcnt* 個の行だけをロードすることができます。

iRestartcount
入力。将来の使用のために予約済み。

piUseTablespace
入力。索引が再作成されている場合、索引のシャドー・コピーが表スペース *iUseTablespaceName* 内に作成され、ロード終了時に元の表スペースにコピーされます。システム TEMPORARY 表スペースのみ、このオプションを使用できます。指定されない場合、シャドー索引が、索引オブジェクトと同じ表スペース内に作成されます。

シャドー・コピーが索引オブジェクトと同じ表スペース内に作成される場合、古い索引オブジェクトを介したシャドー索引オブジェクトのコピーは瞬時に終了します。シャドー・コピーが索引オブジェクトとは異なる表スペースにある場合、物理コピーが実行されます。これにはかなりの入出力および時間を要します。コピーは、表がロード終了時にオフラインであるときに行われます。

iAccessLevel が SQLU_ALLOW_NO_ACCESS である場合、このフィールドは無視されます。

ユーザーが INDEXING MODE REBUILD または INDEXING MODE AUTOSELECT を指定しない場合、このオプションは無視されます。このオプションは INDEXING MODE AUTOSELECT が選択され、ロードが索引を徐々に更新することを選択した場合にも無視されます。

iSavecount
整合点を確立する前にロードするレコードの数。この値はページ・カウントに変換され、エクステント・サイズのインターバルに切り上げられます。それぞれの整合点でメッセージが発行されるため、 *db2LoadQuery - 照会のロード* を用いてロード操作をモニターする場合には、このオプションを選択する必要があります。 *savecnt* の値を大きく設定しておかないと、それぞれの整合点で実行される活動の同期がとられるときにパフォーマンスに影響が及びます。

デフォルト値は `0` であり、これは、必要のない限り整合点が確立されないことを意味します。

iDataBufferSize
ユーティリティー内でデータ転送用のバッファー・スペースとして使用される 4KB ページの数 (並列処理の度合いとは無関係)。指定された値がアルゴリズムの最小値よりも小さい場合には、必要最低限のページが使用され、警告は戻されません。

このメモリーは、ユーティリティー・ヒープから直接に割り振られます (ユーティリティー・ヒープのサイズは、 *util_heap_sz* データベース構成パラメーターを用いて修正できます)。

値を指定しないと、ランタイムにユーティリティーによって適切なデフォルトが計算されます。デフォルトは、ローダーのインスタンス生成時にユーティリティー・ヒープで使用可能なフリー・スペースの割合と、表の一部の特性に基づいて決まります。

iSortBufferSize
入力。このオプションは、ロード操作時に SORTHEAP データベース構成パラメーターをオーバーライドする値を指定します。これは表を索引とともにロードする場合、および *iIndexingMode* パラメーターが SQLU_INX_DEFERRED として指定されない場合にのみ関係があります。指定される値は、SORTHEAP の値を超えることはできません。このパラメーターは、一般的な照会処理にも影響を与える SORTHEAP の値を変更せずに、 LOAD によって使用されるソート・メモリーをスロットルするために役立ちます。

iWarningcount
入力。 *warningcnt* 個の警告後に、ロード操作を停止します。このパラメーターは、警告は予期されないが、正しいファイルと表が使用されていることの確認が望ましい場合に設定してください。ロード・ファイルまたはターゲット表が不適切に指定されると、ロード対象の各行ごとにロード・ユーティリティーによって警告が生成され、このためにロードが失敗する可能性があります。 *warningcnt* が `0` であるか、またはこのオプションを指定していない場合には、ロード操作は、発行された警告の数に関係なく続行されます。

警告のしきい値を超過したためにロード操作が停止された場合には、RESTART モードでもう一度ロード操作を開始することができます。ロード操作は、最後の整合点から自動的に続行します。または、入力ファイルの先頭から REPLACE モードであらためてロード操作を開始できます。

iHoldQuiesce
入力。ユーティリティーによって、ロード後に表を排他静止状態のままにする場合は `TRUE`、それ以外の場合は `FALSE` に値が設定されるフラグ。

iCpuParallelism
入力。ユーティリティーが表オブジェクトの作成時にレコードを解析、変換、および形式化するために作成するプロセスつまりスレッドの数。このパラメーターは、パーティション内並列処理を活用するために設計されています。これは、事前にソートされたデータをロードする際に役立ちます (ソース・データのレコード順序が保持されるため)。このパラメーターの値がゼロである場合には、ロード・ユーティリティーはランタイムに適切なデフォルト値を使用します。注: このパラメーターが LOB または LONG VARCHAR フィールドを含む表について使用されると、システム CPU の数やユーザーによって指定された値に関係なく、値は 1 になります。

iDiskParallelism
入力。ユーティリティーがデータを表スペース・コンテナーに書き込むために作成するプロセスつまりスレッドの数。値を指定しないと、ユーティリティーは表スペース・コンテナーの数と表の特性に基づいて適切なデフォルトを選択します。

iNonrecoverable
入力。ロード・トランザクションがリカバリー不能としてマークされ、後続のロールフォワード・アクションによってリカバリーできない場合には、

SQLU_NON_RECOVERABLE_LOAD に設定します。ロールフォワード・ユーティリティーは、このトランザクションをスキップし、データがロードされようとしていた表を「無効」としてマークします。さらに、ユーティリティーは、その表に対する後続のすべてのトランザクションを無視します。ロールフォワードが完了したら、そのような表はドロップするしかありません。このオプションを使用すると、表スペースはロード操作後にバックアップ・ペンディング状態になりません。また、ロード操作中にロードされたデータのコピーが作成される必要もなくなります。ロード・トランザクションがリカバー可能としてマークされる場合には、
SQLU_RECOVERABLE_LOAD に設定します。

iIndexingMode
入力。索引付けモードを指定します。有効な値 (sqlutil で定義) は、以下のとおりです。

SQLU_INX_AUTOSELECT
LOAD は REBUILD と INCREMENTAL 索引モードの間で選択します。

SQLU_INX_REBUILD
表索引を再作成します。

SQLU_INX_INCREMENTAL
既存の索引を拡張します。

SQLU_INX_DEFERRED
表索引を更新しません。

iAccessLevel
入力。アクセス・レベルを指定します。有効な値は以下のとおりです。

SQLU_ALLOW_NO_ACCESS
ロードが表を排他ロックするように指定します。

SQLU_ALLOW_READ_ACCESS
表の元データ (非差分部分) が、ロードが進行中の間、リーダーに対して可視のままであるように指定します。このオプションは、ロードの付加 (たとえば、ロードの挿入など) に対してのみ有効です。ロード置換に対しては無視されます。

iLockWithForce
入力。ブール・フラグ。 TRUE に設定された場合、ロードは必要に応じて他のアプリケーションに対し、必ず即時に表ロックを得るように強制します。このオプションは、FORCE APPLICATIONS コマンド (SYSADM または SYSCTRL) と同じ権限を必要とします。

SQLU_ALLOW_NO_ACCESS ロードは、ロード操作の開始時に、アプリケーションの競合を強制終了させることができます。ロードの開始時に、このユーティリティーは、表の照会または変更を試みているアプリケーションを強制終了させることができます。

SQLU_ALLOW_READ_ACCESS ロードは、ロード操作の開始時または終了時に、アプリケーションの競合を強制終了させることができます。ロードの開始時に、このロード・ユーティリティーは、表の変更を試みているアプリ

ケーションを強制終了させることができます。ロードの終了時に、このロード・ユーティリティーは、表の照会または変更を試みているアプリケーションを強制終了させることができます。

iCheckPending
入力。表をチェック・ペンディング状態にするように指定します。 SQLU_CHECK_PENDING_CASCADE_IMMEDIATE が指定されている場合、チェック・ペンディング状態は即時にすべての従属表および下層表にカスケードされます。 SQLU_CHECK_PENDING_CASCADE_DEFERRED が指定されている場合、チェック・ペンディング状態の従属表へのカスケードは、ターゲット表の保全性違反がチェックされるまで据え置かれます。このオプションが指定されない場合、 SQLU_CHECK_PENDING_CASCADE_DEFERRED がデフォルトとなります。

iRestartphase
入力。予約済み。有効な値は、シングル・スペース文字 ' ' です。

iStatsOpt
入力。収集する統計の細分性。有効な値は以下のとおりです。

SQLU_STATS_NONE
統計は収集されません。

SQLU_STATS_USE_PROFILE
現在の表に定義されたプロファイルに基づいて、統計が収集されます。このプロファイルを作成するには、RUNSTATS コマンドを使用する必要があります。現在の表に関するプロファイルが存在しない場合、警告が戻され、統計は収集されません。

iUseTablespaceLen
入力。 *piUseTablespace* の長さ (バイト単位)。

oRowsRead
出力。ロード操作中に読み取られたレコードの数。

oRowsSkipped
出力。ロード操作が開始される前にスキップされたレコードの数。

oRowsLoaded
出力。ターゲット表にロードされた行の数。

oRowsRejected
出力。ロードできなかったレコードの数。

oRowsDeleted
出力。削除された重複行の数。

oRowsCommitted
出力。処理されたレコードの合計数。正常にロードされ、データベースにコミットされたレコードの数と、スキップまたはリジェクトされたレコードの数の合計。

piHostname
入力。 *iFileTransferCmd* パラメーターのホスト名。 NULL の場合、ホスト名のデフォルトは「nohost」です。

piFileTransferCmd
入力。ファイル転送コマンドのパラメーター。必要ない場合、NULL に設定する必要があります。このパラメーターの詳細については、「Data Movement Guide」を参照してください。

piPartFileLocation
入力。 PARTITION_ONLY、LOAD_ONLY、および LOAD_ONLY_VERIFY_PART モードでは、このパラメーターは、パーティション・ファイルのロケーションを指定するために使用できます。このロケーションは、*piOutputNodes* オプションで指定された各パーティションに存在している必要があります。

SQL_CURSOR ファイル・タイプの場合、このパラメーターは NULL にすることはできません。ロケーションはパスを参照しませんが、完全修飾されたファイル名を参照します。これは、PARTITION_ONLY モードの場合は、各出力パーティションで作成されたパーティション・ファイルの完全修飾された基本ファイル名、または LOAD_ONLY モードの場合は、各パーティションから読み取られるファイルのロケーションです。 PARTITION_ONLY モードの場合、ターゲット表に LOB 列が存在するならば、指定された基本名のファイルが複数作成されることがあります。 SQL_CURSOR 以外のファイル・タイプでは、このパラメーターの値が NULL の場合、デフォルトで現行ディレクトリーになります。

piOutputNodes
入力。ロード出力パーティションのリスト。 NULL は、ターゲット表が定義されたすべてのノードを示します。

piPartitioningNodes
入力。パーティション・ノードのリスト。 NULL はデフォルトを示します。デフォルトを決定する方法については、「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『Load コマンド』を参照してください。

piMode
入力。パーティション・データベースのロード・モードを指定します。有効な値 (db2ApiDf で定義) は、以下のとおりです。

DB2LOAD_PARTITION_AND_LOAD
データは (多くの場合は並列で) パーティション化され、それぞれ対応するデータベース・パーティションに同時にロードされます。

DB2LOAD_PARTITION_ONLY
データは (多くの場合は並列で) パーティション化され、それぞれのロード・パーティションの指定したファイルに出力が書き込まれます。 SQL_CURSOR 以外のファイル・タイプに関して、各パーティション上の出力ファイルの名前の形式は `filename.xxx` となり、ここで、`filename` は、*piSourceList* で指定された最初の入力ファイルの名前で、 `xxx` はパーティションの番号です。SQL_CURSOR ファイル・タイプの場合、各パーティション上の出力ファイルの名前は、*piPartFileLocation* パラメーターによって判別されます。各パーティション上のパーティション・ファイルの位置の指定方法については、 *piPartFileLocation* パラメーターを参照してください。

注: このモードは CLI LOAD には使用できません。

DB2LOAD_LOAD_ONLY
データがすでにパーティション化されているとします。この場合、パーティション・プロセスが省略され、データはそれぞれ対応するデータベース・パーティションに同時にロードされます。SQL_CURSOR 以外のファイル・タイプの場合、各パーティションの入力ファイル名の形式は `filename.xxx` となり、ここで、`filename` は *piSourceList* で指定された最初のファイルの名前で、`xxx` は 3 桁のパーティション番号です。 SQL_CURSOR ファイル・タイプの場合、各パーティション上の入力ファイルの名前は *piPartFileLocation* パラメーターによって判別されます。各パーティション上のパーティション・ファイルの位置の指定方法については、 *piPartFileLocation* パラメーターを参照してください。

注: このモードは、リモート・クライアント上にあるデータ・ファイルのロード時に使用したり、または CLI LOAD には使用できません。

DB2LOAD_LOAD_ONLY_VERIFY_PART
データがすでにパーティション化されているとします。しかし、データ・ファイルにはパーティション・ヘッダーがありません。パーティション化プロセスは省略され、データはそれぞれ対応するデータベース・パーティションに同時にロードされます。ロード操作時に、各行が正しいパーティション上にあるかがチェックされます。パーティション違反のある行は、ダンプ・ファイル修飾子が指定されている場合、ダンプ・ファイルに置かれます。指定されていない場合、その行は廃棄されます。パーティション違反が特定のロード・パーティションに存在する場合、1 つの警告が、パーティションのロード・メッセージ・ファイルに書き込まれます。各パーティションの入力ファイル名の形式は `filename.xxx` となり、ここで、`filename` は *piSourceList* で指定された最初のファイルの名前で、`xxx` は 3 桁のパーティション番号です。

注: このモードは、リモート・クライアント上にあるデータ・ファイルのロード時に使用したり、または CLI LOAD には使用できません。

DB2LOAD_ANALYZE
すべてのデータベース・パーティション間で均等に分散される最適なパーティション・マップが生成されます。

piMaxNumPartAgents
入力。パーティション・エージェントの最大数。 NULL 値はデフォルトを示します。デフォルトは 25 です。

piIsolatePartErrs
入力。ロード操作が、個々のパーティションで発生するエラーに対応する方法を示します。有効な値 (`db2ApiDf` で定義) は、以下のとおりです。

DB2LOAD_SETUP_ERRS_ONLY
このモードでは、セットアップ時にパーティションで生じるエラー

(たとえば、パーティションへのアクセスに関する問題や、パーティションの表スペースまたは表へのアクセスに関する問題) によって、失敗したパーティションではロード操作が停止してしまいますが、残りのパーティションでは操作が続行されます。データのロード中にパーティションで生じるエラーによって、全操作が失敗し、各パーティションの最後の整合点にロールバックされます。

DB2LOAD_LOAD_ERRS_ONLY
このモードでは、セットアップ時にパーティションで生じるエラーによって、ロード操作全体が失敗します。データのロード中にエラーが生じた場合、エラーのあるパーティションは最後の整合点にロールバックされます。ロード操作は、失敗が生じるまで、またはすべてのデータがロードされるまで、残りのパーティションで続行します。すべてのデータがロードされたパーティションでは、ロード操作後は、データは可視ではありません。他のパーティションで生じたエラーのため、トランザクションは打ち切られます。すべてのパーティション上のデータは、ロードの再開操作が実行されるまで、不可視のままです。これにより、新たにロードされたデータはパーティション上で可視になります。パーティションでは、ロード操作が完了し、エラーが発生したパーティションでのロード操作が再開されます。

注: *iAccessLevel* が SQLU_ALLOW_READ_ACCESS に設定されている場合や、コピー・ターゲットが指定されている場合、このモードは使用できません。

DB2LOAD_SETUP_AND_LOAD_ERRS
このモードでは、セットアップまたはデータのロード時に生じるパーティション・レベルのエラーによって、影響を受けたパーティション上でのみ、処理が停止します。
DB2LOAD_LOAD_ERRS_ONLY モードと同様、データ・ロード中にパーティション・エラーが生じた場合、すべてのパーティション上のデータは、ロードの再開操作が実行されるまで不可視のままです。

注: *iAccessLevel* が SQLU_ALLOW_READ_ACCESS に設定されている場合や、コピー・ターゲットが指定されている場合、このモードは使用できません。

DB2LOAD_NO_ISOLATION
ロード操作時にエラーが生じると、トランザクションは打ち切られます。

パラメーターが NULL の場合、*iAccessLevel* が SQLU_ALLOW_READ_ACCESS に設定されない限り、またはコピー・ターゲットが指定されない限り、デフォルトは DB2LOAD_LOAD_ERRS_ONLY になります。設定または指定されている場合、デフォルトは DB2LOAD_NO_ISOLATION です。

piStatusInterval
入力。進行メッセージを生成する前に、ロードするデータの MB 数を指定

します。有効な値は、1 から 4000 の範囲の整数です。 NULL が指定される場合、デフォルト値の 100 が使用されます。

piPortRange
入力。内部通信用の TCP ポート範囲。 NULL の場合、使用されるポート範囲は 6000 から 6063 です。

piCheckTruncation
入力。ロードで入出力時にレコードの切り捨てをチェックします。有効な値は TRUE および FALSE です。 NULL の場合、デフォルトは FALSE です。

piMapFileInput
入力。パーティション・マップの入力ファイル名。モードが ANALYZE ではない場合、このパラメーターは NULL に設定する必要があります。モードが ANALYZE の場合、このパラメーターは指定する必要があります。

piMapFileOutput
入力。パーティション・マップの出力ファイル名。 piMapFileInput に対する規則は、ここでも同じく適用されます。

piTrace
入力。すべてのデータ変換プロセスのダンプ、およびハッシュ値の出力を検討する必要がある場合、トレースするレコードの数を指定します。 NULL の場合、レコード数のデフォルトは 0 です。

piNewline
入力。 RECLEN ファイル・タイプ修飾子も指定されている場合、ロードで ASC データ・レコードの終端で改行文字をチェックするように強制します。指定可能な値は TRUE および FALSE です。 NULL の場合、値のデフォルトは FALSE です。

piDistfile
入力。パーティション分散ファイル名。 NULL が指定された場合、値はデフォルトの ″DISTFILE″ になります。

piOmitHeader
入力。 DB2LOAD_PARTITION_ONLY モードを使用する場合に、パーティション・マップのヘッダーをパーティション・ファイルに組み込まないことを示します。指定可能な値は TRUE および FALSE です。 NULL の場合、デフォルトは FALSE です。

piRunStatDBPartNum
統計を収集するデータベース・パーティションを指定します。デフォルト値は、出力パーティション・リスト内の最初のデータベース・パーティションです。

iHostnameLen
入力。 *piHostname* の長さ (バイト単位)。

iFileTransferLen
入力。 *piFileTransferCmd* の長さ (バイト単位)。

iPartFileLocLen
入力。 *piPartFileLocation* の長さ (バイト単位)。

iMapFileInputLen
入力。 *piMapFileInput* の長さ (バイト単位)。

iMapFileOutputLen
入力。 *piMapFileOutput* の長さ (バイト単位)。

iDistfileLen
入力。 *piDistfile* の長さ (バイト単位)。

piNodeList
入力。ノード番号の配列。

iNumNodes
入力。 *piNodeList* 配列内のノードの数。 0 がデフォルトで、これはターゲット表が定義されているすべてのノードです。

iPortMin
入力。小さいポート番号。

iPortMax
入力。大きいポート番号。

oRowsRdPartAgents
出力。すべてのパーティション・エージェントによって読み取られる行の総数。

oRowsRejPartAgents
出力。すべてのパーティション・エージェントによってリジェクトされる行の総数。

oRowsPartitioned
出力。すべてのパーティション・エージェントによってパーティション分割される行の総数。

poAgentInfoList
出力。パーティション・データベースへのロード操作時には、ロード・エージェント、パーティション・エージェント、事前パーティション・エージェント、 ファイル転送コマンド・エージェント、およびファイルへのロード・エージェントなどのロード処理が関係してくる可能性があります (これらについては、「Data Movement Guide」で説明されています)。
poAgentInfoList 出力パラメーターの目的は、呼び出し側に、ロード操作に関係した各ロード・エージェントに関する情報を戻すことです。リスト内の各項目には、以下の情報が含まれます。

- oAgentType。項目が記述するロード・エージェントの種類を示すタグ。
- oNodeNum。エージェントが実行されたパーティションの数。
- oSqlcode。エージェントの処理の結果の最終 sqlcode。
- oTableState。エージェントが実行されるパーティション上の表の最終状況 (ロード・エージェントに関係するもののみ)。

API を呼び出す前に、このリストにメモリーを割り振るのは、API の呼び出し側の責任です。呼び出し側は、*iMaxAgentInfoEntries* パラメーターにメモリーを割り振った項目の数も示す必要があります。呼び出し側が *poAgentInfoList* を NULL に設定する場合、または *iMaxAgentInfoEntries* を 0 に設定する場合、ロード・エージェントに関する情報は戻されません。

iMaxAgentInfoEntries
入力。 *poAgentInfoList* 用にユーザーが割り振ったエージェント情報の項目の最大数。一般に、このパラメーターは、ロード操作に関係したパーティション数の 3 倍の数に設定すれば十分です。

oNumAgentInfoEntries
出力。ロード操作によって生成されたエージェント情報の項目の実際の数。*iMaxAgentInfoEntries* が *oNumAgentInfoEntries* の値以上である場合に限り、この項目数は *poAgentInfoList* パラメーターでユーザーに戻されます。*iMaxAgentInfoEntries* が *oNumAgentInfoEntries* より小さい場合、*poAgentInfoList* に戻される項目数は *iMaxAgentInfoEntries* と等しくなります。

oSqlcode
出力。エージェントの処理の結果の最終 sqlcode。

oTableState
出力。この出力パラメーターの目的は、ロード操作後に、表のいかなる状態も報告しないことです。その目的は、ロード処理中に表に何が起きたかについての一般情報を呼び出し側に提供するために、発生し得る表の状況の、小さなサブセットだけを報告することです。この値は、ロード・エージェントにのみ関係があります。可能な値は次のとおりです。

DB2LOADQUERY_NORMAL
ロードがパーティションで正常に完了し、表が LOAD IN PROGRESS (または LOAD PENDING) 状態ではなくなったことを示します。この場合、制約事項をさらに処理する必要があるために、表が引き続きチェック・ペンディング状態であることがありますが、これは正常な状態なので報告はされません。

DB2LOADQUERY_UNCHANGED
エラーが原因でロード・ジョブが処理を打ち切ったが、db2Load を呼び出す前の状態がどのようなものであっても、パーティション上の表の状態はまだ変更されていないことを示します。ロードの再始動、またはそのようなパーティション上での操作の終了を実行する必要はありません。

DB2LOADQUERY_LOADPENDING
処理中にロード・ジョブが打ち切られたが、パーティション上の表は LOAD PENDING 状態のままであることを示します。これは、パーティションでのロード・ジョブを、終了または再始動する必要があることを意味しています。

oNodeNum
出力。エージェントが実行されたパーティションの数。

oAgentType
出力。エージェント・タイプ。有効な値 (`db2ApiDf` で定義) は、以下のとおりです。

DB2LOAD_LOAD_AGENT

DB2LOAD_PARTITIONING_AGENT

DB2LOAD_PRE_PARTITIONING_AGENT

DB2LOAD_FILE_TRANSFER_AGENT

DB2LOAD_LOAD_TO_FILE_AGENT

使用上の注意:

データは、入力ファイル内に並んでいる順序でロードされます。特定の順序を希望する場合には、ロードが試行される前にデータをソートしてください。

ロード・ユーティリティーは、既存の定義に基づいて索引を作成します。ユニーク・キーの重複を処理するのに、例外表が使用されます。ユーティリティーは、参照保全を強制したり、制約検査を実行したり、ロードする表に従属するサマリー表を更新したりすることはありません。参照制約またはチェック制約を含む表は、チェック・ペンディング状態になります。 REFRESH IMMEDIATE として定義されているサマリー表、およびロードする表に依存するサマリー表もまた、チェック・ペンディング状態になります。表のチェック・ペンディング状態を解除するには、SET INTEGRITY ステートメントを発行してください。ロード操作は、複製されたサマリー表では実行できません。

クラスタリング索引の場合、ロードする前に、データをクラスタリング索引でソートする必要があります。マルチディメンション・クラスターされた (MDC) 表にロードする場合は、データをソートする必要はありません。

DB2 Data Links Manager についての考慮事項

各 DATALINK 列ごとに、括弧内にそれぞれ 1 つの列を指定できます。それぞれの列の指定は、1 つ以上の DL_LINKTYPE、*接頭部*、および DL_URL_SUFFIX 指定で構成されます。*接頭部* 情報は、 DL_URL_REPLACE_PREFIX、 または DL_URL_DEFAULT_PREFIX の指定のいずれかになります。

DATALINK 列指定の数は、表で定義されている DATALINK 列の数と同じだけ指定できます。指定の順序は、挿入列リストの中での DATALINK 列の順序 (挿入列リストが INSERT INTO (insert-column, ...) で指定されている場合) か、または表定義内での順序 (insert-column が指定されていない場合) に従います。

たとえば、表に列 C1、C2、C3、C4、および C5 があり、そのうち C2 と C5 だけが DATALINK 型で、挿入列 (insert-column) リストが (C1、C5、C3、C2) である場合、 2 つの DATALINK 列を指定する必要があります。最初の列の指定は C5 用であり、2 番目の列の指定は C2 用です。挿入列リストを指定しない場合、最初の列の指定は C2 用になり、 2 番目の列の指定は C5 用になります。

複数の DATALINK 列があり、一部の列では特別な指定が必要ではない場合、列の指定には、指定の順序をはっきりと示すために、少なくとも括弧を含める必要があります。どの列にも指定が行われない場合は、空の括弧のリスト全体を除くことができます。したがって、デフォルトで十分な場合には、DATALINK を指定する必要はありません。

FILE LINK CONTROL で定義されている DATALINK 列を含む表にデータをロードする場合は、ロード・ユーティリティーを呼び出す前に、以下のステップを実行してください。 (すべての DATALINK 列が NO LINK CONTROL として定義されている場合、これらのステップは必要ありません。)

1. DATALINK 列の値によって参照される Data Links サーバーに、 DB2 Data Links Manager がインストールされていることを確認する。
2. データベースが DB2 Data Links Manager に登録されていることを確認する。
3. DATALINK 値として挿入されるすべてのファイルを、適切なデータ・リンク・サーバーにコピーする。
4. Data Links サーバー上の DB2 Data Links Manager に接頭部名 (複数可) を定義する。
5. (ロードする) DATALINK データによって参照される Data Links サーバーを、DB2 Data Links Manager 構成ファイルに登録します。

ロード・ユーティリティーの実行中に、DB2 と データ・リンク・サーバー間の接続が失敗し、ロード操作も失敗してしまう場合があります。その場合には、以下のようにしてください。

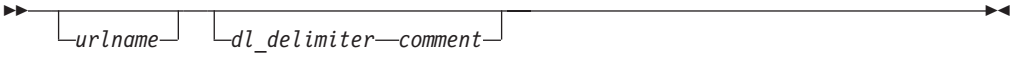
1. データ・リンク・サーバーおよび DB2 Data Links Manager を開始する。
2. ロード再開操作を起動する。

ロード操作中に失敗したリンクはデータ保全性違反と見なされ、ユニーク索引違反と同じ方法で処理されます。そのため、1 つ以上の DATALINK 列を含む表をロードする場合の特別な例外が定義されています。

入力ファイル内での DATALINK 情報の表示

LINKTYPE (現在のところ URL のみサポート) は、DATALINK 情報の一部として指定されていません。 LINKTYPE は、LOAD または IMPORT コマンドで指定され、 PC/IXF タイプの入力ファイルの場合は、適切な列記述子レコードの中で指定されます。

URL LINKTYPE の DATALINK 情報の構文は、以下のとおりです。



urlname と *comment* はいずれもオプションであることに注意してください。どちらも省略した場合、NULL 値が代入されます。

urlname

この URL 名は有効な URL 構文に適合していなければなりません。

注:

1. 現在のところ、「http」、「file」、および「unc」がスキーマ名として許可されています。
2. URL 名の接頭部 (スキーマ、ホスト、およびポート) はオプションです。接頭部がない場合は、ロード・ユーティリティーまたはインポート・ユーティリティーの DL_URL_DEFAULT_PREFIX または DL_URL_REPLACE_PREFIX 指定の接頭部が使われます。そのどちらも指定されていない場合、デフォルトの接頭部として ″file://localhost″ が使われます。したがって、ローカル・ファイルの場合、LOAD または IMPORT コマンド内で DATALINK 列を指定せずに、絶対パス名で指定したファイル名を URL 名として入力することができます。

3. 接頭部が URL 名に付けられている場合も、ロードまたはインポート操作時には、DL_URL_REPLACE_PREFIX で指定した異なる接頭部名によってオーバーライドされます。
4. (DL_URL_SUFFIX を指定した場合、それを追加した後の)「path」は、リモート・サーバーにあるリモート・ファイルの絶対パス名です。相対パス名は使用できません。 HTTP サーバーのデフォルトのパス接頭部は使用されません。

dl_delimiter

区切り付き ASCII (DEL) ファイル形式の場合、 dldel 修飾子で指定した文字、あるいは LOAD または IMPORT コマンドのデフォルトの文字。区切りなし ASCII (ASC) ファイル形式の場合、これを文字順序 ¥; (円記号とそれに続くセミコロン) に対応させる必要があります。空白文字 (ブランクやタブなど) を、このパラメーターに指定した値の前後に置くことができます。

comment

DATALINK 値のコメント部分。区切り付き ASCII (DEL) ファイル形式で指定する場合、 *comment* テキストは、文字ストリング区切り文字で囲む必要があります。文字ストリング区切り文字は、デフォルトでは二重引用符 (″) です。この文字ストリング区切り文字は、LOAD または IMPORT コマンドで MODIFIED BY *filetype-mod* を指定することによりオーバーライドできます。

コメントを指定しない場合、このコメントはデフォルトでは長さがゼロのストリングになります。

次に示すのは、区切り付き ASCII (DEL) ファイル形式での DATALINK データの例です。

- `http://www.almaden.ibm.com:80/mrep/intro.mpeg; "Intro Movie"`

  これは、以下のような部分から構成されています。

  – スキーム = http
  – サーバー = www.almaden.ibm.com
  – パス = /mrep/intro.mpeg
  – 注釈 = ″Intro Movie″

- `file://narang/u/narang; "InderPal's Home Page"`

  これは、以下のような部分から構成されています。

  – スキーム = file
  – サーバー = narang
  – パス = /u/narang
  – 注釈 = ″InderPal’s Home Page″

次に示すのは、区切りなし ASCII (ASC) ファイル形式での DATALINK データの例です。

- `http://www.almaden.ibm.com:80/mrep/intro.mpeg¥;Intro Movie`

  これは、以下のような部分から構成されています。

  - スキーム = http
  - サーバー = www.almaden.ibm.com
  - パス = /mrep/intro.mpeg
  - 注釈 = ″Intro Movie″
- `file://narang/u/narang¥; InderPal's Home Page`

  これは、以下のような部分から構成されています。
  - スキーム = file
  - サーバー = narang
  - パス = /u/narang
  - 注釈 = ″InderPal's Home Page″

以下に、DATALINK データの例を示します。列のロードまたはインポート指定が DL_URL_REPLACE_PREFIX (″http://qso″) であると想定しています。

- `http://www.almaden.ibm.com/mrep/intro.mpeg`

  これは、以下のような部分から構成されています。
  - スキーマ = http
  - サーバー = qso
  - パス = /mrep/intro.mpeg
  - 注釈 = NULL ストリング
- `/u/me/myfile.ps`

  これは、以下のような部分から構成されています。
  - スキーマ = http
  - サーバー = qso
  - パス = /u/me/myfile.ps
  - 注釈 = NULL ストリング

関連資料:

- 432 ページの『sqluvqdp - 表の表スペースの静止』
- 206 ページの『db2LoadQuery - ロードの照会』
- 456 ページの『SQLDCOL』
- 495 ページの『SQLU-MEDIA-LIST』
- 62 ページの『db2Export - エクスポート』
- 114 ページの『db2Import - インポート』
- 47 ページの『db2DatabaseQuiesce - データベースの静止』
- 142 ページの『db2InstanceQuiesce - インスタンスの静止』
- 193 ページの『ロードのファイル・タイプ修飾子』
- 205 ページの『データを移動する際の区切り文字の制約事項』

関連サンプル:

- 『dtformat.sqc -- Load and import data format extensions (C)』
- 『tbload.sqc -- How to load into a partitioned database (C)』

- 『tbmove.sqc -- How to move table data (C)』
- 『tbmove.sqC -- How to move table data (C++)』

ロードのファイル・タイプ修飾子

表 15. ロード用の有効なファイル・タイプ修飾子: すべてのファイル・フォーマット

修飾子	説明
anyorder	この修飾子は、*cpu_parallelism* パラメーターとともに使用され、ソース・データの順序を保持する必要がないことを指定します。そのため、SMP システムでは、パフォーマンスがかなり向上します。 *cpu_parallelism* の値が 1 である場合、このオプションは無視されます。整合点後のクラッシュ・リカバリーではデータを順番にロードする必要があるため、 SAVECOUNT > 0 の場合には、このオプションはサポートされません。
generatedignore	この修飾子はロード・ユーティリティーに対して、すべての生成列のデータがデータ・ファイル内にあるものの、それらのデータは無視するべきものであることを通知します。その結果、すべての生成列の値はユーティリティーが生成します。この修飾子は、 `generatedmissing` または `generatedoverride` 修飾子とともに使用することはできません。
generatedmissing	この修飾子が指定されている場合、ユーティリティーは、生成列のデータが入力データ・ファイルに入っていない (NULL も入っていない) ものと見なします。その結果、すべての生成列の値はユーティリティーが生成します。この修飾子は、 `generatedignore` または `generatedoverride` 修飾子とともに使用することはできません。

generatedoverride	この修飾子は、(こうした列のタイプの通常の規則に反して) 表内のすべての生成列で、ユーザーのデータを受け入れるようにロード・ユーティリティーに指示します。これが役立つのは、 別のデータベース・システムからデータを移行する場合や、 ROLLFORWARD DATABASE コマンドで RECOVER DROPPED TABLE オプションを使用してリカバリーしたデータから表をロードする場合です。この修飾子を使用した場合、 NULL 不可の生成列でデータまたは NULL データの入っていない行はリジェクトされます (SQL3116W)。
	注: この修飾子が使用される場合、表はチェック・ペンディング状態に置かれます。ユーザーが提供する値を検証せずに、表をチェック・ペンディング状態から解放するには、ロード操作後に以下のコマンドを発行します。
	`SET INTEGRITY FOR < table-name > GENERATED COLUMN IMMEDIATED UNCHECKED`
	表をチェック・ペンディング状態から解放し、ユーザーが提供する値の検査を強制するには、ロード操作後に以下のコマンドを発行します。
	`SET INTEGRITY FOR < table-name > IMMEDIATE CHECKED.`
	この修飾子は、`generatedmissing` または `generatedignore` 修飾子と共に使用することはできません。

identityignore	この修飾子はロード・ユーティリティーに対して、 ID 列のデータがデータ・ファイル内にあるものの、それらのデータは無視するべきものであることを通知します。その結果、すべての識別値はユーティリティーが生成します。 GENERATED ALWAYS ID 列と GENERATED BY DEFAULT ID 列のどちらの場合も動作は同じになります。つまり、GENERATED ALWAYS 列の場合には、リジェクトされる行はありません。この修飾子は、 `identitymissing` または `identityoverride` 修飾子とともに使用することはできません。

表 15. ロード用の有効なファイル・タイプ修飾子: すべてのファイル・フォーマット (続き)

修飾子	説明
identitymissing	この修飾子が指定されている場合、ユーティリティーは、 ID 列のデータが入力データ・ファイルに入っていない (NULL も入っていない) ものと見なし、行ごとに値を生成します。 GENERATED ALWAYS ID 列と GENERATED BY DEFAULT ID 列のどちらの場合も動作は同じになります。この修飾子は、 identityignore または identityoverride 修飾子とともに使用することはできません。
identityoverride	この修飾子を使用するのは、 GENERATED ALWAYS として定義されている ID 列が、ロードされる表にある場合だけです。この修飾子はユーティリティーに対し、そのような列に関して、明示的な非 NULL データを受け入れる (これらのタイプの ID 列に関する通常の規則に反する) ように指示します。これが役立つのは、別のデータベース・システムからデータを移行するときに GENERATED ALWAYS として表を定義しなければならない場合や、 ROLLFORWARD DATABASE コマンドで DROPPED TABLE RECOVERY オプションを使用してリカバリーしたデータから表をロードする場合です。この修飾子を使用した場合、 ID 列でデータまたは NULL データの入っていない行はリジェクトされます (SQL3116W)。この修飾子は、 identitymissing または identityignore 修飾子とともに使用することはできません。 **注:** このオプションが使用されていると、ロード・ユーティリティーは、表の ID 列内の値の固有性の保守または検査を行いません。
indexfreespace=*x*	*x* は 0 から 99 の整数です。その値は、各索引ページの中で索引再作成ロード時のフリー・スペースとして残しておく部分の割合を示すパーセントとして解釈されます。ロードに INDEXING MODE INCREMENTAL を指定すると、このオプションは無視されます。ページの最初の項目は、制限なしで追加されます。それより後の項目は、フリー・スペースのパーセントしきい値内である場合に追加されます。デフォルト値は、CREATE INDEX の実行時に使用した値です。 この値は、CREATE INDEX ステートメントで指定した PCTFREE 値に優先し、レジストリー変数 DB2 INDEX FREE は、索引のフリー・スペースに優先します。索引のフリー・スペース・オプションは、索引のリーフ・ページにのみ影響を与えます。

表 15. ロード用の有効なファイル・タイプ修飾子: すべてのファイル・フォーマット (続き)

修飾子	説明
lobsinfile	*lob-path* には、LOB データを含むファイルへのパスを指定します。 ASC、DEL、または IXF ロード入力ファイルには、 LOB 列に LOB データが入っているファイルの名前が含まれています。 このオプションは、CURSOR ファイル・タイプと一緒にはサポートされません。 LOBS FROM 文節は、″lobsinfile″ 修飾子が使用されているときの、 LOB ファイルの場所を指定します。 LOBS FROM 文節には、″lobsinfile″ 修飾子のコンテキスト以外の意味はありません。 LOBS FROM 文節は、データのロード中に、LOAD ユーティリティーに LOB ファイルを検索するためのパスのリストを送ります。 各パスには少なくとも 1 つのファイルが含まれており、そのファイルには、データ・ファイル内の LOB ロケーション指定子 (LLS) によって指される少なくとも 1 つの LOB が入っています。 LLS は、LOB ファイル・パスに保管されるファイル内の LOB のロケーションのストリング表現です。 LLS の形式は、*filename.ext.nnn.mmm/* です。 *filename.ext* は LOB を含むファイルの名前、 *nnn* はファイル内の LOB のオフセット (バイト単位)、 *mmm* は LOB の長さ (バイト単位) を表します。たとえば、ストリング `db2exp.001.123.456/` がデータ・ファイルに保存される場合、 LOB はファイル db2exp.001 のオフセット 123 に位置し、456 バイト長です。 LOB が NULL であることを示すには、サイズを -1 と入力します。サイズが 0 と指定されている場合、LOB は長さ 0 と見なされます。長さ -1 の NULL の LOB は、オフセットおよびファイル名が無視されます。たとえば、NULL の LOB の LLS は db2exp.001.7.-1/ です。
noheader	ヘッダー検査コードをスキップします (単一パーティション・データベースのパーティション・グループ内にある表へのロード操作にのみ適用されます)。 オートローダー・ユーティリティーは、複数パーティションのデータベース・パーティション・グループの表にデータを提供している各ファイルに、ヘッダーを書き込みます。単一パーティションのデータベース・パーティション・グループに存在する表に対してデフォルトの MPP ロード (モード PARTITION_AND_LOAD) が使用される場合、ファイルにはヘッダーが含まれないと想定されます。したがって、noheader 修飾子は必要ありません。 LOAD_ONLY モードが使用される場合、ファイルにはヘッダーが含まれると想定されます。ヘッダーが含まれないファイルを使って操作を実行したい場合にのみ、 noheader 修飾子を使用する必要があります。
norowwarnings	リジェクトされた行についてのすべての警告を抑制します。
pagefreespace=*x*	*x* は 0 から 100 の整数です。この値は各データ・ページのうち、フリー・スペースとして残される部分のパーセンテージとして解釈されます。最小行サイズのため、指定した値が無効である場合 (たとえば、最も小さい行の大きさが 3 000 バイトで、*x* の値が 50 である場合)、その行は新しいページに置かれます。値 100 を指定した場合、各行が新しいページに置かれます。 **注:** 表の PCTFREE 値は、ページごとに指定されたフリー・スペースの量を決定します。ロード操作の pagefreespace 値、または表の PCTFREE 値が設定されていない場合、ユーティリティーは、それぞれのページにできるだけ大きなスペースを割り当てます。 pagefreespace で設定した値は、その表について指定された PCTFREE 値をオーバーライドします。

表 15. ロード用の有効なファイル・タイプ修飾子: すべてのファイル・フォーマット (続き)

修飾子	説明
subtableconvert	単一副表へのロードの場合のみ有効。これを使う場合として典型的な例は、正規の表からデータをエクスポートした後、この修飾子を使ってロード操作を呼び出してそのデータを単一の副表に変換する場合です。
totalfreespace=*x*	*x* は 0 以上の整数です。この値は表内の合計ページのうち、表の終わりにフリー・スペースとして追加される部分のパーセンテージとして解釈されます。たとえば、*x* が 20 で、データのロード後に、表に 100 のデータ・ページがある場合、20 の追加の空ページが付加されます。この表のデータ・ページの合計数は、120 になります。データ・ページの総数は、表の索引ページの数には影響を与えません。このオプションは、索引オブジェクトには影響を与えません。 **注:** このオプションを指定して 2 つのロードが行われる場合、 2 番目のロードは、最初のロードによって最後に付加された余分のスペースを再利用しません。
usedefaults	ターゲット表の列に対応するソース列が指定されているが、 1 つまたは複数の行インスタンスのデータが入っていない場合、デフォルト値がロードされます。欠落データの例は、以下のとおりです。 • DEL ファイルの場合、列に「,,」が指定されています。 • DEL/ASC/WSF ファイルの場合: 十分な数の列がない行、または元の指定に対して長さが十分でない行。 このオプションが指定されていない場合、ソース列に行インスタンスのデータがないと、以下のいずれかの処理が行われます。 • その列が NULL 可能な場合は、NULL がロードされます。 • その列が NULL 可能でない場合、ユーティリティーはその行をリジェクトします。

表 16. ロード用の有効なファイル・タイプ修飾子: ASCII ファイル・フォーマット (ASC/DEL)

修飾子	説明
codepage=*x*	*x* は ASCII 文字ストリングです。この値は、入力データ・セット内のデータのコード・ページとして解釈されます。ロード操作時には、文字データ (および文字で指定された数値データ) は、このコード・ページからデータベースのコード・ページへ変換されます。 以下の規則が適用されます。 • 純 DBCS (GRAPHIC)、混合 DBCS、および EUC の場合、区切り文字は x00 から x3F の範囲に制限されます。 • EBCDIC コード・ページで指定された DEL データの場合、区切り文字をシフトインおよびシフトアウト DBCS 文字と一致させることはできません。 • nullindchar には、標準の ASCII セットに含まれる (コード・ポイント x20 から x7F の範囲の) 記号を指定する必要があります。これは、ASCII 記号およびコード・ポイントを示します。 EBCDIC データでは、コード・ポイントが異なるとしても、対応する記号を使用することができます。 このオプションは、CURSOR ファイル・タイプと一緒にはサポートされません。

表 16. ロード用の有効なファイル・タイプ修飾子: ASCII ファイル・フォーマット (ASC/DEL) (続き)

修飾子	説明
dateformat="x"	x はソース・ファイル内の日付の形式です。[1] 有効な日付エレメントは以下のとおりです。 YYYY - 年 (0000 から 9999 の 4 桁) M - 月 (1 から 12 の 1 桁または 2 桁) MM - 月 (1 から 12 の 2 桁; M と相互排他的) D - 日 (1 から 31 の範囲の 1 桁または 2 桁の数) DD - 日 (1 から 31 の範囲の 1 桁または 2 桁の数; D と相互排他的) DDD - 元日から数えた日数 (001 から 366 の 3 桁の数。他の日または月エレメントとは相互排他的) 指定されていないエレメントには、デフォルト値の 1 が割り当てられます。日付形式の例を以下に示します。 "D-M-YYYY" "MM.DD.YYYY" "YYYYDDD"
dumpfile = x	x は、リジェクトされた行を書き込む例外ファイルの (サーバー・データベース・パーティションによる) 完全修飾名です。 1 レコード当たり、最大で 32 KB のデータが書き込まれます。次の例は、ダンプ・ファイルを指定する方法を示すものです。 db2 load from data of del modified by dumpfile = /u/user/filename insert into table_name ファイルはインスタンス所有者によって作成および所有されます。デフォルトのファイル許可をオーバーライドするには、 dumpfileaccessall ファイル・タイプ修飾子を使用します。 **注:** 1. パーティション・データベース環境の場合、パスはロードを実行するデータベース・パーティションにローカルなものでなければなりません。それによって、並行して実行される複数のロード操作が同じファイルに書き込むことを防ぐことができます。 2. ファイルの内容は、非同期バッファー・モードでディスクに書き込まれます。ロード操作が失敗したり割り込まれたりした場合には、ディスクにコミットされたレコード数が正確にわからず、 LOAD RESTART 後の整合性が保証されない可能性があります。ファイルが完全であるとされるのは、 1 回のパスの中で開始して完了するロード操作の場合だけです。 3. この修飾子では、ファイル拡張子が複数あるファイル名はサポートされません。たとえば、 dumpfile = /home/svtdbm6/DUMP.FILE ロード・ユーティリティーでは、上のファイル名は受け入れられますが、 dumpfile = /home/svtdbm6/DUMP.LOAD.FILE このファイル名は受け入れられません。
dumpfileaccessall = x	ダンプ・ファイルが作成されるとき、読み取りアクセスを 'OTHERS' に付与します。 このファイル・タイプ修飾子は、以下の場合にのみ有効です。 1. dumpfile ファイル・タイプ修飾子とともに使用される 2. ユーザーがロードのターゲット表に対する SELECT 特権を持っている 3. UNIX ベースのオペレーティング・システムに常駐する DB2 サーバー・データベース・パーティション上で発行される
fastparse	ユーザー提供の列値に対して簡略化された構文チェックが実行され、パフォーマンスが向上します。このオプションの下でロードした表は、体系的に正確であることが保証され、ユーティリティーは、セグメント化違反またはトラップを防ぐための十分なデータ・チェックを実行することが保証されます。正しい形式のデータが正しくロードされます。 たとえば、ASC ファイル内の整数列のフィールド項目として値 123qwr4 が検出された場合、この値は有効な数値を表すものではないので、ロード・ユーティリティーは通常 構文エラーのフラグを付けます。 fastparse を指定した場合、構文エラーは検出されず、整数フィールドに任意の数値がロードされます。この修飾子を使用する場合は、正しい形式のデータだけを使用するように注意してください。このオプションを ASCII データとともに使用すると、パフォーマンスがかなり改善される可能性があります。 このオプションは、 CURSOR または IXF ファイル・タイプと一緒にはサポートされません。

表 16. ロード用の有効なファイル・タイプ修飾子: ASCII ファイル・フォーマット (ASC/DEL) (続き)

修飾子	説明
implieddecimal	暗黙指定されている小数点の位置が列定義によって決定され、値の終わりにあるとは見なされなくなります。たとえば、値 12345 は、 12345.00 ではなく、 123.45 として DECIMAL(8,2) 列にロードされます。 この修飾子は、 packeddecimal 修飾子と共に使用することはできません。
timeformat="*x*"	*x* はソース・ファイル内の時刻の形式です。[1] 有効な時刻エレメントは以下のとおりです。 H - 時 (12 時間制の場合は 0 から 12、 24 時間制の場合は 0 から 24 の範囲の 1 桁または 2 桁の数) HH - 時 (12 時間制の場合は 0 から 12、 24 時間制の場合は 0 から 24 の範囲の 2 桁の数。 H と相互に排他) M - 分 (0 から 59 の範囲の 1 桁または 2 桁の数) MM - 分 (0 から 59 の範囲の 2 桁の数。 M と相互排他的) S - 秒 (0 から 59 の範囲の 1 桁または 2 桁の数) SS - 秒 (0 から 59 の範囲の 2 桁の数。 S とは相互排他的) SSSSS - 夜の 12 時から数えた秒数 (00000 から 86399 の 5 桁; 他の時刻エレメントとは相互排他的) TT - 正午の標識 (AM または PM) 指定されていないエレメントには、デフォルト値 0 が割り当てられます。時刻形式の例を以下に示します。 "HH:MM:SS" "HH.MM TT" "SSSSS"

表 16. ロード用の有効なファイル・タイプ修飾子: ASCII ファイル・フォーマット (ASC/DEL) (続き)

修飾子	説明

timestampformat="x"	x はソース・ファイル内のタイム・スタンプの形式です。[1] 有効なタイム・スタンプ・エレメントは以下のとおりです。
	
	YYYY - 年 (0000 から 9999 の範囲の 4 桁の数字)
	M - 月 (1 から 12 の範囲の 1 桁または 2 桁の数)
	MM - 月 (01 から 12 の 2 桁の数。
	M および MMM とは相互に排他的)
	MMM - 月 (月を表す 3 文字の大文字小文字を区別しない略語;
	M および MM とは相互に排他的)
	D - 日 (1 から 31 の範囲の 1 桁または 2 桁の数)
	DD - 日 (1 から 31 の範囲の 1 桁または 2 桁の数; D と相互排他的)
	DDD - 元日から数えた日数 (001 から 366 の 3 桁;
	他の日または月エレメントと相互に排他)
	H - 時 (12 時間制の場合は 0 から 12、
	24 時間制では 0 から 24 の 1 桁または 2 桁。)
	HH - 時 (12 時間制の場合は 0 から 12、
	24 時間制では 0 から 24 の 1 桁または 2 桁:
	H と相互に排他)
	M - 分 (0 から 59 の 1 桁または 2 桁)
	MM - 分 (0 から 59 の範囲の 2 桁の数。
	M (分) とは相互排他的)
	S - 秒 (0 から 59 の 1 桁または 2 桁)
	SS - 秒 (0 から 59 の範囲の 2 桁の数。
	S とは相互排他的)
	SSSSS - 夜の 12 時から数えた秒数
	(00000 から 86399 の 5 桁;
	他の時刻エレメントとは相互排他的)
	UUUUUU - マイクロ秒 (000000 から 999999 の 6 桁;
	他のすべてのマイクロ秒エレメントとは相互排他的)
	UUUUU - マイクロ秒 (00000 から 99999 の 5 桁、
	000000 から 999990 の範囲にマップする;
	他のすべてのマイクロ秒エレメントとは相互排他的)
	UUUU - マイクロ秒 (0000 から 9999 の 4 桁、
	000000 から 999900 の範囲にマップする;
	他のすべてのマイクロ秒エレメントとは相互排他的)
	UUU - マイクロ秒 (000 から 999 の 3 桁、
	000000 から 999000 の範囲にマップする;
	他のすべてのマイクロ秒エレメントとは相互排他的)
	UU - マイクロ秒 (00 から 99 の 2 桁、
	000000 から 990000 の範囲にマップする;
	他のすべてのマイクロ秒エレメントとは相互排他的)
	U - マイクロ秒 (0 から 9 の 2 桁、
	000000 から 900000 の範囲にマップする;
	他のすべてのマイクロ秒エレメントとは相互排他的)
	TT - 正午の標識 (AM または PM)
	
	YYYY、M、MM、D、DD、または DDD エレメントに値が指定されていない場合、デフォルト値の 1 が割り当てられます。値が指定されていない MMM エレメントには、デフォルト値の「Jan」が割り当てられます。それ以外のエレメントに値が指定されていない場合、デフォルト値の 0 が割り当てられます。タイム・スタンプ形式の例を以下に示します。
	
	`"YYYY/MM/DD HH:MM:SS.UUUUUU"`
	
	MMM エレメントの有効な値は、
	「jan」、「feb」、「mar」、「apr」、「may」、「jun」、「jul」、「aug」、「sep」、「oct」、「nov」、および「dec」です。これらの値は大文字小文字の区別をします。
	
	以下の例では、ユーザー定義の日付および時刻の形式を含んでいるデータを、 schedule という表にインポートする方法を示しています。
	
	`db2 import from delfile2 of del`
	`modified by timestampformat="yyyy.mm.dd hh:mm tt"`
	`insert into schedule`

noeofchar	任意指定のファイル終わり文字 x'1A' が、ファイルの終わりとして認識されません。通常の文字の場合のように処理が続行されます。

表 16. ロード用の有効なファイル・タイプ修飾子: ASCII ファイル・フォーマット (ASC/DEL) (続き)

修飾子	説明
usegraphiccodepage	usegraphiccodepage が指定された場合、GRAPHIC または 2 バイト文字ラージ・オブジェクト (DBCLOB) データ・フィールドにロードされるデータは、GRAPHIC コード・ページであると見なされます。データの残りは、文字コード・ページであると見なされます。 GRAPHIC コード・ページは、文字コード・ページと関連付けられます。 LOAD は、指定されている場合の codepage 修飾子、または codepage 修飾子が指定されていない場合は、データベースのコード・ページを通じて、文字コード・ページを決定します。 この修飾子は、リカバリーされている表に GRAPHIC データが含まれている場合にのみ、表リカバリーのドロップによって生成された区切りデータ・ファイルとともに使用される必要があります。 **制約事項** usegraphiccodepage 修飾子は、 EXPORT ユーティリティーによって作成された DEL または ASC ファイルで指定することはできません。これらのファイルは、1 コード・ページのみでエンコードされたデータを含むためです。 usegraphiccodepage 修飾子はまた、ファイル内の 2 バイト文字ラージ・オブジェクト (DBCLOB) には無視されます。

表 17. ロード用の有効なファイル・タイプ修飾子: ASC ファイル・フォーマット (区切り文字なし ASCII)

修飾子	説明
binarynumerics	数値データ (DECIMAL 以外) は、文字表記ではなく、バイナリー形式でなければなりません。これにより、コストのかかる変換を避けられます。 このオプションがサポートされるのは、定位置 ASC において、 reclen オプションによって固定長レコードが指定されている場合だけです。 noeofchar オプションが前提です。 以下の規則が適用されます。 • BIGINT、INTEGER、および SMALLINT を除き、データ・タイプ間の変換は実行されません。 • データ長は、それぞれのターゲット列定義と一致している必要があります。 • FLOAT は、IEEE 浮動小数点形式でなければなりません。 • ロード・ソース・ファイルに含まれるバイナリー・データは、ロード操作を実行するプラットフォームに関係なく、ビッグ・エンディアンであると見なされます。 **注:** この修飾子の影響を受ける列のデータに、NULL を指定することはできません。この修飾子を使用すると、ブランク (通常は NULL と解釈される) は、バイナリー値であると解釈されます。
nochecklengths	nochecklengths を指定した場合、ソース・データの中にターゲット表の列サイズを超える列定義が含まれているとしても、各行のロードが試みられます。このような行が正常にロードされるのは、コード・ページ変換でソース・データが縮小する場合です。たとえば、ソースにある 4 バイトの EUC データがターゲットで 2 バイトの DBCS データに縮小すれば、必要スペースは半分になります。このオプションが特に役立つのは、列の定義は不一致であるがソース・データが常に適合することが分かっている場合です。
nullindchar=*x*	*x* は単一文字です。 NULL 値を示す文字を *x* に変更します。 *x* のデフォルト値は Y です。[2] 文字が 1 つの英字である場合を除いて、この修飾子は EBCDIC データ・ファイルで大文字小文字を区別します。たとえば、NULL 標識文字が文字 N として指定されている場合、 n も NULL 標識として認識されます。

表 17. ロード用の有効なファイル・タイプ修飾子: ASC ファイル・フォーマット (区切り文字なし ASCII) (続き)

修飾子	説明
packeddecimal	binarynumerics 修飾子は DECIMAL フィールド・タイプを含まないため、パック 10 進数データを直接ロードします。 このオプションがサポートされるのは、定位置 ASC において、 reclen オプションによって固定長レコードが指定されている場合だけです。 noeofchar オプションが前提です。 符号ニブル用にサポートされる値は以下のとおりです。 + = 0xC 0xA 0xE 0xF - = 0xD 0xB この修飾子の影響を受ける列のデータに、NULL を指定することはできません。この修飾子を使用すると、ブランク (通常は NULL と解釈される) は、バイナリー値であると解釈されます。 サーバーのプラットフォームには関係なく、ロードのソース・ファイルに含まれるバイナリー・データのバイト順はビッグ・エンディアンであることが前提となっています。つまり、この修飾子を Windows オペレーティング・システムで使用する場合も、バイト順を逆にしてはなりません。 この修飾子は、 implieddecimal 修飾子と共に使用することはできません。
reclen=*x*	*x* は 32 767 以下の整数です。各行ごとに *x* 個の文字が読み取られ、行の終わりを示すのに改行文字は使用されません。
striptblanks	データを可変長フィールドにロードする際に、後書きブランク・スペースを切り捨てます。このオプションを指定しない場合、ブランク・スペースはそのまま保持されます。 このオプションは、striptnulls と一緒には指定できません。これらのオプションは、相互に排他的です。 **注:** このオプションは以前の t オプションを置き換えるもので、バックレベルとの互換性のみサポートされています。
striptnulls	データを可変長フィールドにロードする際に、後書き NULL (0x00 文字) を切り捨てます。このオプションを指定しない場合、NULL はそのまま保持されます。 このオプションは、striptblanks と一緒には指定できません。これらのオプションは、相互に排他的です。 **注:** このオプションは以前の padwithzero オプションを置き換えるもので、バックレベルとの互換性のみサポートされています。
zoneddecimal	BINARYNUMERICS 修飾子は DECIMAL フィールド・タイプを含まないため、ゾーン 10 進数データをロードします。このオプションがサポートされるのは、定位置 ASC において、 RECLEN オプションによって固定長レコードが指定されている場合だけです。 NOEOFCHAR オプションが前提です。 ハーフバイト符号値は、以下のいずれかです。 + = 0xC 0xA 0xE 0xF - = 0xD 0xB サポートされている数値は、0x0 から 0x9 です。 サポートされているゾーン値は、0x3 および 0xF です。

表 18. ロード用の有効なファイル・タイプ修飾子: DEL ファイル・フォーマット (区切り文字付き ASCII)

修飾子	説明

chardel*x*	*x* は単一文字のストリング区切り文字です。デフォルト値は、二重引用符 (") です。指定した文字は、文字ストリングを囲むために、二重引用符の代わりに使用されます。[23] 文字ストリング区切り文字として明示的に二重引用符 (") を指定したい場合、以下のように指定します。
	`modified by chardel""`
	単一引用符 (') も、以下のように文字ストリングの区切り文字として指定できます。
	`modified by chardel''`

coldel*x*	*x* は単一文字の列区切り文字です。デフォルト値はコンマ (,) です。指定した文字は、列の終わりを表すために、コンマの代わりに使用されます。[23]
datesiso	日付形式。すべての日付データ値を ISO 形式でロードします。
decplusblank	正符号文字。正の 10 進値の先頭に正符号 (+) ではなく、ブランク・スペースが置かれます。デフォルトのアクションでは、正の 10 進数の前に正符号 (+) が付けられます。
decpt*x*	*x* は、小数点としてピリオドと置換される単一文字です。デフォルト値はピリオド (.) です。指定した文字は、小数点文字としてピリオドの代わりに使用されます。[23]

delprioritychar	区切り文字の現在のデフォルト優先順位は、(1) レコード区切り文字、(2) 区切り文字、(3) 列区切り文字です。この修飾子は、区切り文字の優先順位を区切り文字、レコード区切り文字、列区切り文字と逆順にすることにより、以前の優先順位に依存する既存のアプリケーションを保護します。構文は次のとおりです。
	`db2 load ... modified by delprioritychar ...`
	たとえば、以下のような DEL データ・ファイルがあるとします。
	`"Smith, Joshua",4000,34.98<row delimiter>`
	`"Vincent,<row delimiter>, is a manager", ...`
	`... 4005,44.37<row delimiter>`
	`delprioritychar` 修飾子が指定されている場合、このデータ・ファイルには 2 行しかありません。 2 番目の <row delimiter> は 2 番目の行の最初のデータ列の一部と解釈されますが、 1 番目と 3 番目の <row delimiter> は実レコードの区切り文字と解釈されます。この修飾子が指定されていない 場合、このデータ・ファイルでは 3 行になり、各行は <row delimiter> によって区切られます。

dldel*x*	*x* は単一文字の DATALINK 区切り文字です。デフォルト値はセミコロン (;) です。指定した文字は、 DATALINK 値のフィールド間区切り文字としてセミコロンの代わりに使用されます。 DATALINK 値には 2 つ以上の副値を指定できるため、この区切り文字が必要です。[234] **注:** *x* は、行、列、または文字ストリングの区切り文字とは異なる文字にしてください。

表 18. ロード用の有効なファイル・タイプ修飾子: DEL ファイル・フォーマット (区切り文字付き ASCII) (続き)

修飾子	説明

keepblanks	タイプが CHAR、VARCHAR、LONG VARCHAR、または CLOB の各フィールドの前後のブランクを保持します。このオプションが指定されない場合、区切り文字の外側にある前後のブランクはすべて除去され、表のすべてのブランク・フィールドに NULL が挿入されます。
	
	以下の例では、データ・ファイル内の前後のスペースをすべて保持しながら、TABLE1 という表にデータをロードする方法を示します。
	
	`db2 load from delfile3 of del`
	`modified by keepblanks`
	`insert into table1`

nochardel	ロード・ユーティリティーは、列区切り文字と列区切り文字の間にあるすべてのバイトが列データの一部であると見なします。区切り文字は、列データの一部として構文解析されます。データが DB2 を使用してエクスポートされている場合は、このオプションを指定しないでください (エクスポート時に nochardel が指定されない限り)。これは、区切り文字を持たないベンダー・データ・ファイルをサポートするために提供されています。不適切に使用すると、データが損失または破壊される場合があります。 このオプションを `chardelx`、 `delprioritychar` または `nodoubledel` と一緒に指定することはできません。これらのオプションは、相互に排他的です。
nodoubledel	二重になっている区切り文字の認識を抑止します。

表 19. ロード用の有効なファイル・タイプ修飾子: IXF ファイル・フォーマット

修飾子	説明
forcein	コード・ページが不一致でもデータを受け入れ、コード・ページ間の変換を抑止するようにユーティリティーに指示します。 固定長ターゲット・フィールドに、データが入るだけの十分な大きさがあるかどうかチェックされます。 `nochecklengths` が指定されていると、チェックは実行されず、各行のロードが試行されます。
nochecklengths	`nochecklengths` を指定した場合、ソース・データの中にターゲット表の列サイズを超える列定義が含まれているとしても、各行のロードが試みられます。このような行が正常にロードされるのは、コード・ページ変換でソース・データが縮小する場合です。たとえば、ソースにある 4 バイトの EUC データがターゲットで 2 バイトの DBCS データに縮小すれば、必要スペースは半分になります。このオプションが特に役立つのは、列の定義は不一致であるがソース・データが常に適合することが分かっている場合です。

注:

1. 日付形式ストリングは必ず二重引用符で囲まなければなりません。フィールド区切り文字には、 a から z、A から Z、および 0 から 9 を含めることはできません。フィールド区切り文字は、 DEL ファイル形式の区切り文字またはフィールド区切り文字と同じにすることはできません。エレメントの開始および終了位置が明らかな場合、フィールド区切り文字は任意指定です。開始および終了位置が明らかでないのは、項目の長さが一定でない D、H、M、または S などのエレメントが使用されている場合です (修飾の仕方によって異なります)。

タイム・スタンプ形式の場合、月の記述子と分の記述子のどちらも文字 M を使用するため、あいまいさを避けるよう注意する必要があります。月のフィールドは、他の日付フィールドと隣接していなければなりません。分のフィールドは、他の時刻フィールドに隣接していなければなりません。以下のタイム・スタンプ形式は、あいまいな形式の例です。

```
"M" (月と分のどちらかがはっきりしない)
"M:M" (どちらが何を表しているのか分からない)
"M:YYYY:M" (どちらも月として解釈される)
"S:M:YYYY" (時刻値と日付値の両方に隣接している)
```

あいまいな場合、ユーティリティーはエラー・メッセージを報告し、操作は失敗します。

以下に、明確なタイム・スタンプ形式を示します。

```
"M:YYYY" (月)
"S:M" (分)
"M:YYYY:S:M" (月....分)
"M:H:YYYY:M:D" (分....月)
```

二重引用符や円記号などの文字の前には、エスケープ文字 (たとえば、¥) を付けなければなりません。

2. この文字は、ソース・データのコード・ページで指定してください。

   文字コード・ポイント (文字記号ではない) は、 xJJ または 0xJJ という構文で指定することができます (JJ はコード・ポイントの 16 進表記)。たとえば、列区切りとして # 文字を指定するには、以下のいずれかを使用します。

   ```
   ... modified by coldel# ...
   ... modified by coldel0x23 ...
   ... modified by coldelX23 ...
   ```

3. データ移動のための区切り文字の制約事項には、区切り文字の指定変更として使用できる文字に適用される制限事項がリストされています。
4. DATALINK 区切り文字が URL 構文内で有効な文字である場合でも、ロード操作の有効範囲内ではその特別な意味はなくなります。
5. サポートされていないファイル・タイプを MODIFIED BY オプションで使用しようとしても、ロード・ユーティリティーは警告を出しません。この場合、ロード操作が失敗し、エラー・コードが戻されます。

表 20. codepage および usegraphiccodepage 使用時の LOAD 動作

codepage=N	usegraphiccodepage	LOAD 動作
なし	なし	ファイル内のすべてのデータは、アプリケーション・コード・ページではなく、データベース・コード・ページであると見なされます (CLIENT オプションが指定された場合でも)。
あり	なし	ファイル内のすべてのデータは、コード・ページ N であると見なされます。 **警告:** N が単一バイト・コード・ページの場合、GRAPHIC データをデータベースにロードすると壊れます。

表 20. *codepage* および *usegraphiccodepage* 使用時の *LOAD* 動作 (続き)

codepage=N	usegraphiccodepage	LOAD 動作
なし	あり	ファイル内の文字データは、データベース・コード・ページあると見なされます (CLIENT オプションが指定された場合でも)。 GRAPHIC データは、データベース GRAPHIC データのコード・ページであると見なされます (CLIENT オプションが指定された場合でも)。 データベース・コード・ページが単一バイトの場合は、すべてのデータはデータベース・コード・ページであると見なされます。 **警告:** 単一バイト・データベースに GRAPHIC データをロードすると、壊れます。
あり	あり	文字データは、コード・ページ N であると見なされます。 GRAPHIC データは、N の GRAPHIC コード・ページであると見なされます。 N が単一バイトまたは 2 バイト・コード・ページの場合は、すべてのデータは、コード・ページ N であると見なされます。 **警告:** N が単一バイト・コード・ページの場合、GRAPHIC データをデータベースにロードすると壊れます。

関連資料:

- 「コマンド・リファレンス」の『LOAD コマンド』
- 168 ページの『db2Load - ロード』
- 205 ページの『データを移動する際の区切り文字の制約事項』

データを移動する際の区切り文字の制約事項

区切り文字についての制約事項:

選択した区切り文字が移動されるデータの一部となっていないことを確かめるのは、ユーザーの責任です。データの一部になっている場合、予期しないエラーが発生する可能性があります。データを移動する際には、以下の制限が列、ストリング、DATALINK、および小数点区切り文字に適用されます。

- 区切り文字は相互に排他的である。
- 区切り文字として バイナリー・ゼロ、改行文字、ブランク・スペースを使用することはできない。
- デフォルトの小数点 (.) をストリング区切り文字として使用することはできない。
- 以下の文字は、ASCII ファミリー・コード・ページと EBCDIC ファミリー・コード・ページで、仕様が異なっています。
  - シフトイン (0x0F) とシフトアウト (0x0E) 文字を、 EBCDIC MBCS データ・ファイルの区切り文字として使用することはできない。

- MBCS、EUC、または DBCS コード・ページの区切り文字は、 0x40 より大きくすることはできません。ただし、EBCDIC MBCS データのデフォルトの小数点 0x4b は例外です。
- ASCII コード・ページまたは EBCDIC MBCS コード・ページにおけるデータ・ファイルのデフォルト区切り文字は、以下のとおりです。

  ```
  " (0x22、二重引用符。ストリング区切り文字)
  , (0x2c、コンマ。列区切り文字)
  ```

- EBCDIC SBCS コード・ページにおけるデータ・ファイルのデフォルト区切り文字は、以下のとおりです。

  ```
  " (0x7F、二重引用符。ストリング区切り文字)
  , (0x6B、コンマ。列区切り文字)
  ```

- ASCII データ・ファイルのデフォルト小数点は 0x2e (ピリオド) です。
- EBCDIC データ・ファイルのデフォルト小数点は 0x4B (ピリオド) です。
- サーバーのコード・ページがクライアントのコード・ページと異なっている場合は、非デフォルトの区切り文字を 16 進表示で指定するようお勧めします。たとえば、

  ```
  db2 load from ... modified by chardel0x0C coldelX1e ...
  ```

DEL ファイルでの二重になっている区切り文字の認識のサポートに関する以下の情報は、エクスポート、インポート、およびロード・ユーティリティーに適用されます。

- 区切り文字を、DEL ファイルの文字ベースのフィールド内で使用することができます。これは、タイプ CHAR、VARCHAR、LONG VARCHAR、または CLOB (lobsinfile が指定されている場合を除く) のフィールドに適用されます。区切り文字で囲まれている区切り文字の対は、データベースにインポートまたはロードされます。たとえば、

  ```
  "What a ""nice"" day!"
  ```

  これは、以下のようにインポートされます。

  ```
  What a "nice" day!
  ```

  エクスポートの場合は、逆の規則が適用されます。たとえば、

  ```
  I am 6" tall.
  ```

  これは、以下のように DEL ファイルにエクスポートされます。

  ```
  "I am 6"" tall."
  ```

- DBCS 環境では、パイプ (|) 区切り文字はサポートされません。

db2LoadQuery - ロードの照会

処理中のロード操作の状況をチェックします。

許可:

なし

必要な接続:

データベース

API 組み込みファイル:

db2ApiDf.h

C API 構文:

```
/* File: db2ApiDf.h */
/* API: db2LoadQuery */
/* ... */
SQL_API_RC SQL_API_FN
db2LoadQuery (
  db2Uint32 versionNumber,
  void *pParmStruct,
  struct sqlca *pSqlca);

typedef struct
{
  db2Uint32 iStringType;
  char *piString;
  db2Uint32 iShowLoadMessages;
  db2LoadQueryOutputStruct *poOutputStruct;
  char *piLocalMessageFile;
} db2LoadQueryStruct;

typedef struct
{
  db2Uint32 oRowsRead;
  db2Uint32 oRowsSkipped;
  db2Uint32 oRowsCommitted;
  db2Uint32 oRowsLoaded;
  db2Uint32 oRowsRejected;
  db2Uint32 oRowsDeleted;
  db2Uint32 oCurrentIndex;
  db2Uint32 oNumTotalIndexes;
  db2Uint32 oCurrentMPPNode;
  db2Uint32 oLoadRestarted;
  db2Uint32 oWhichPhase;
  db2Uint32 oWarningCount;
  db2Uint32 oTableState;
} db2LoadQueryOutputStruct;
/* ... */
```

汎用 API 構文:

```
/* File: db2ApiDf.h */
/* API: db2gLoadQuery */
/* ... */
SQL_API_RC SQL_API_FN
db2gLoadQuery (
  db2Uint32 versionNumber,
  void *pParmStruct,
  struct sqlca *pSqlca);

typedef struct
{
  db2Uint32 iStringType;
  db2Uint32 iStringLen;
  char *piString;
  db2Uint32 iShowLoadMessages;
  db2LoadQueryOutputStruct *poOutputStruct;
  db2Uint32 iLocalMessageFileLen;
  char *piLocalMessageFile
} db2gLoadQueryStruct;
```

```
typedef struct
{
  db2Uint32 oRowsRead;
  db2Uint32 oRowsSkipped;
  db2Uint32 oRowsCommitted;
  db2Uint32 oRowsLoaded;
  db2Uint32 oRowsRejected;
  db2Uint32 oRowsDeleted;
  db2Uint32 oCurrentIndex;
  db2Uint32 oNumTotalIndexes;
  db2Uint32 oCurrentMPPNode;
  db2Uint32 oLoadRestarted;
  db2Uint32 oWhichPhase;
  db2Uint32 oWarningCount;
  db2Uint32 oTableState;
} db2LoadQueryOutputStruct;
/* ... */
```

API パラメーター:

versionNumber
入力。 2 番目のパラメーター *pParmStruct* として渡される構造のバージョンとリリースのレベルを指定します。

pParmStruct
入力。 *db2LoadQueryStruct* 構造を指すポインター。

pSqlca
出力。*sqlca* 構造へのポインター。

iStringType
入力。 *piString* のタイプを指定します。有効な値 (db2ApiDf.h で定義) は、以下のとおりです。

DB2LOADQUERY_TABLENAME
db2LoadQuery API が使用する表の名前を指定します。

iStringLen
入力。 *piString* の長さ (バイト単位) を指定します。

piString
入力。 *iStringType* 値に応じて、一時ファイルのパス名または表名を指定します。

iShowLoadMessages
入力。ロード・ユーティリティーが戻すメッセージのレベルを指定します。有効な値 (db2ApiDf.h で定義) は、以下のとおりです。

DB2LOADQUERY_SHOW_ALL_MSGS
すべてのロード・メッセージを戻す。

DB2LOADQUERY_SHOW_NO_MSGS
ロード・メッセージを戻さない。

DB2LOADQUERY_SHOW_NEW_MSGS
この API を最後に呼び出した後で生成されたメッセージだけを戻す。

poOutputStruct
出力。ロード・サマリー情報が含まれる、*db2LoadQueryOutputStruct* 構造を指すポインター。サマリーが必要でない場合は、NULL に設定してください。

iLocalMessageFileLen
入力。 *piLocalMessageFile* の長さ (バイト単位) を指定します。

piLocalMessageFile
入力。出力メッセージ用に使用されるローカル・ファイル名を指定します。

oRowsRead
出力。ロード・ユーティリティーがこれまでに読み取ったレコードの数を示します。

oRowsSkipped
出力。ロード操作が開始される前にスキップされたレコードの数を示します。

oRowsCommitted
出力。これまでにターゲット表にコミットされた行数を示します。

oRowsLoaded
出力。これまでにターゲット表にロードされた行の数を示します。

oRowsRejected
出力。これまでにターゲット表からリジェクトされた行数を示します。

oRowsDeleted
出力。これまでにターゲット表から (削除フェーズで) 削除された行数を示します。

oCurrentIndex
出力。現在 (作成フェーズ時に) 作成中の索引を示します。

oCurrentMPPNode
出力。照会されるデータベース・パーティション・サーバーを示します (パーティション・データベース環境モードのみ)。

oLoadRestarted
出力。照会中のロード操作がロード再始動操作である場合に値が TRUE になるフラグを示します。

oWhichPhase
出力。照会中のロード操作の現在のフェーズを示します。有効な値 (db2ApiDf.h で定義) は、以下のとおりです。

DB2LOADQUERY_LOAD_PHASE
ロード・フェーズ。

DB2LOADQUERY_BUILD_PHASE
作成フェーズ。

DB2LOADQUERY_DELETE_PHASE
削除フェーズ。

oNumTotalIndexes
出力。 (作成フェーズで) 作成する索引の合計数を示します。

oWarningCount
出力。これまでに戻された警告の合計数を示します。

oTableState
出力。表の状態。有効な値 (db2ApiDf で定義) は、以下のとおりです。

DB2LOADQUERY_NORMAL
表の状態は表には影響を及ぼしません。

DB2LOADQUERY_CHECK_PENDING
表には制約事項があり、その制約事項を検証する必要があります。SET INTEGRITY コマンドを使用して、表を DB2LOADQUERY_CHECK_PENDING 状態から解放してください。制約事項のある表でロードが開始されると、ロード・ユーティリティーは表を DB2LOADQUERY_CHECK_PENDING 状態にします。

DB2LOADQUERY_LOAD_IN_PROGRESS
この表は現在ロードが進行中です。

DB2LOADQUERY_LOAD_PENDING
この表でロードがアクティブでしたが、ロードがコミットする前に打ち切られました。表を DB2LOADQUERY_LOAD_PENDING 状態から解放するために、ロードの終了、ロードの再開、またはロードの置換を発行してください。

DB2LOADQUERY_READ_ACCESS
表のデータは読み取りアクセス照会に使用可能です。DB2LOADQUERY_READ_ACCESS オプションを使用してロードし、表を読み取り専用状態にします。

DB2LOADQUERY_NOTAVAILABLE
表は使用不可です。表は単にドロップされたか、またはバックアップからリストアされた可能性があります。リカバリー不能なロードによるロールフォワードによって、表は利用不能状態になります。

DB2LOADQUERY_NO_LOAD_RESTART
表は部分的にロードされた状態で、ロードの再開は許可されません。さらにこの表はロード・ペンディング状態です。ロードの終了またはロードの置換を発行し、表をロード再開不能状態から解放します。ロールフォワード操作中、表は DB2LOADQUERY_NO_LOAD_RESTART 状態にすることができます。ロード操作の終了前の時点までロールフォワードした場合、または打ち切られたロード操作を介してロールフォワードするが、ロード終了操作またはロード再開操作の終了時点までロールフォワードしない場合、この状態が発生します。

DB2LOADQUERY_TYPE1_INDEXES
現在、表はタイプ 1 索引を使用しています。この索引に対して REORG ユーティリティーを使用する場合、この索引は、CONVERT オプションを使用してタイプ 2 に変換できます。

使用上の注意:

この API は、*piString* で指定された表に対するロード操作の状況を読み取り、*pLocalMsgFileName* で指定されたファイルに状況を書き込みます。

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『ロード照会コマンドを使用したパーティション・データベース・ロードのモニター』

関連資料:

- 453 ページの『SQLCA』

関連サンプル:

- 『loadqry.sqb -- Query the current status of a load (MF COBOL)』
- 『tbload.sqc -- How to load into a partitioned database (C)』
- 『tbmove.sqc -- How to move table data (C)』
- 『tbmove.sqC -- How to move table data (C++)』

db2MonitorSwitches - モニター・スイッチの入手/更新

データベース・マネージャーによって収集されるモニター・データのグループについて、スイッチを選択的にオンまたはオフに切り替えます。呼び出しを発行しているアプリケーションについては、これらのスイッチの現行状態を戻します。

有効範囲:

この API はインスタンス上のデータベース・パーティション・サーバー、またはインスタンス上のすべてのデータベース・パーティションに関する情報を戻すことができます。

許可:

以下のいずれかです。

- *sysadm*
- *sysctrl*
- *sysmaint*
- *sysmon*

必要な接続:

インスタンス。インスタンス・アタッチが存在しない場合は、デフォルトのインスタンス・アタッチが作成されます。

リモート・インスタンス (または別のローカル・インスタンス) の設定を表示するには、まず最初にそのインスタンスにアタッチすることが必要です。

API 組み込みファイル:

db2ApiDf.h

C API 構文:

```
/* File: db2ApiDf.h */
/* API: db2MonitorSwitches */
/* ... */
SQL_API_RC SQL_API_FN
db2MonitorSwitches (
  db2Uint32 versionNumber,
  void *pParmStruct,
  struct sqlca *pSqlca);

typedef struct
{
        struct sqlm_recording_group     *piGroupStates;
        void                            *poBuffer;
        db2Uint32                       iBufferSize;
        db2Uint32                       iReturnData;
        db2Uint32                       iVersion;
        db2int32                        iNodeNumber;
        db2Uint32                       *poOutputFormat;
}db2MonitorSwitchesData;
/* ... */
```

汎用 API 構文:

```
/* File: db2ApiDf.h */
/* API: db2gMonitorSwitches */
/* ... */
SQL_API_RC SQL_API_FN
db2gMonitorSwitches (
  db2Uint32 versionNumber,
  void *pParmStruct,
  struct sqlca *pSqlca);

typedef SQL_STRUCTURE db2gMonitorSwitchesData
{
  struct sqlm_recording_group *piGroupStates;
  void                        *poBuffer;
  db2Uint32                   iBufferSize;
  db2Uint32                   iReturnData;
  db2Uint32                   iVersion;
  db2int32                    iNodeNumber;
  db2Uint32                   *poOutputFormat;
} db2gMonitorSwitchesData;
/* ... */
```

API パラメーター:

versionNumber
入力。 2 番目のパラメーター pParmStruct として渡される構造のバージョンとリリースのレベルを指定します。上記のような構造を使用するには、db2Version810 を指定します。この構造の別のバージョンを使用したい場合には、 include ディレクトリー内の db2ApiDf.h ヘッダー・ファイルを調べて、サポートされるバージョンの詳細リストを確認してください。指定しバージョン番号に対応する *db2MonitorSwitchesStruct* 構造を必ず使用してください。

pParmStruct
入力。 *db2MonitorSwitchesStruct* 構造を指すポインター。

pSqlca
出力。*sqlca* 構造へのポインター。

piGroupStates
入力。スイッチのリストが含まれている *sqlm-recording-group* 構造 (sqlmon.h で定義) を指すポインター。

poBuffer
スイッチの状態データが書き込まれるバッファーを指すポインター。

iBufferSize
入力。出力バッファーのサイズを指定します。

iReturnData
入力。現在のスイッチの状態を *poBuffer* が指示するバッファーに書き込むかどうかを指定するフラグ。

iVersion
入力。収集するデータベース・モニター・データのバージョン ID。データベース・モニターは、要求されたバージョンについて使用可能なデータのみを戻します。このパラメーターは、以下のいずれかのシンボリック定数に設定してください。

- SQLM_DBMON_VERSION1
- SQLM_DBMON_VERSION2
- SQLM_DBMON_VERSION5
- SQLM_DBMON_VERSION5_2
- SQLM_DBMON_VERSION6
- SQLM_DBMON_VERSION7
- SQLM_DBMON_VERSION8

注: バージョンとして SQLM_DBMON_VERSION1 が指定された場合、 API はリモートでは実行できません。

iNodeNumber
入力。要求の送信先となるデータベース・パーティション・サーバーを示します。この値に基づき、要求は現行のデータベース・パーティション・サーバー、すべてのデータベース・パーティション・サーバー、またはユーザーが指定したデータベース・パーティション・サーバーに対して処理されます。有効な値は以下のとおりです。

- SQLM_CURRENT_NODE
- SQLM_ALL_NODES
- *ノード値*

注: 独立型インスタンスの場合には、 SQLM_CURRENT_NODE を使用する必要があります。

poOutputFormat
サーバーから返されるストリームの形式。次のいずれかです。

SQLM_STREAM_STATIC_FORMAT
スイッチの状態が、バージョン 7 より前の静的スイッチ構造で戻されることを示します。

SQLM_STREAM_DYNAMIC_FORMAT
スイッチが、db2GetSnapshot で戻されるのと同様の、自己記述形式で戻されることを示します。

使用上の注意:

データベース・マネージャー・レベルでのスイッチの状態を入手するには、*OBJ_TYPE* に SQMA_DB2 (データベース・マネージャーのスナップショットの入手) を指定して、 db2GetSnapshot を呼び出してください。

iVersion が SQLM_DBMON_VERSION8 より小さい場合、タイム・スタンプ・スイッチは使用できません。

関連資料:

- 89 ページの『db2GetSnapshot - スナップショットの入手』
- 93 ページの『db2GetSnapshotSize - db2GetSnapshot 出力バッファーに必要なサイズの見積もり』
- 241 ページの『db2ResetMonitor - モニターのリセット』
- 453 ページの『SQLCA』
- 490 ページの『SQLM-RECORDING-GROUP』

関連サンプル:

- 『utilsnap.c -- Utilities for the snapshot monitor samples (C)』
- 『utilsnap.C -- Utilities for the snapshot monitor samples (C++)』

db2Prune - 履歴ファイルの整理

履歴ファイルから項目を削除するか、アクティブ・ログ・パスからログ・ファイルを削除します。

許可:

以下のいずれかが必要です。

- *sysadm*
- *sysctrl*
- *sysmaint*
- *dbadm*

必要な接続:

データベース。デフォルト・データベース以外のデータベースの履歴ファイルから項目を削除する場合は、この API を呼び出す前に、そのデータベースへの接続を確立しておく必要があります。

API 組み込みファイル:

db2ApiDf.h

C API 構文:

```
/* File: db2ApiDf.h */
/* API: db2Prune */
/* ... */
SQL_API_RC SQL_API_FN
db2Prune (
  db2Uint32 version,
  void *pDB2PruneStruct,
  struct sqlca *pSqlca);

typedef struct
{
  char *piString,
  db2Uint32 iEID,
  db2Uint32 iCallerAction,
  db2Uint32 iOptions
} db2PruneStruct;
/* ... */
```

汎用 API 構文:

```
/* File: db2ApiDf.h */
/* API: db2GenPrune */
/* ... */
SQL_API_RC SQL_API_FN
db2GenPrune (
  db2Uint32 version,
  void *pDB2GenPruneStruct,
  struct sqlca *pSqlca);

typedef struct
{
  db2Uint32 iStringLen;
  char *piString,
  db2Uint32 iEID,
  db2Uint32 iCallerAction,
  db2Uint32 iOptions
} db2GenPruneStruct;
/* ... */
```

API パラメーター:

version
入力。 2 番目のパラメーター *pDB2PruneStruct* として渡される構造のバージョンとリリースのレベルを指定します。

pDB2PruneStruct
入力。 *db2PruneStruct* 構造を指すポインター。

pSqlca
出力。*sqlca* 構造へのポインター。

iStringLen
入力。 *piString* の長さ (バイト単位) を指定します。

piString
入力。タイム・スタンプまたはログ・シーケンス番号 (LSN) を指定するストリングを指すポインターです。タイム・スタンプまたはその一部 (最小値は *yyyy*、つまり年) が、削除対象のレコードの選択に使用されます。タイム・スタンプと等しいかまたはタイム・スタンプよりも小さい、すべての項目が削除されます。必ず有効なタイム・スタンプを指定するようにしてください。 NULL パラメーターを指定してもデフォルトの動作はありません。

このパラメーターは、LSN を渡すときにも使用できるので、非アクティブ・ログの整理が可能です。

iEID 入力。履歴ファイルから単一の項目の整理をするときに使用できる、ユニークな ID を示します。

iCallerAction
入力。実行するアクションのタイプを指定します。有効な値 (db2ApiDf.h で定義) は、以下のとおりです。

DB2PRUNE_ACTION_HISTORY
履歴ファイルの項目を削除します。

DB2PRUNE_ACTION_LOG
アクティブ・ログ・パスからログ・ファイルを削除します。

iOptions
入力。有効な値 (db2ApiDf.h で定義) は、以下のとおりです。

DB2PRUNE_OPTION_FORCE
最後のバックアップの削除を強制します。

DB2PRUNE_OPTION_DELETE
履歴ファイルから整理されるログ・ファイルを削除します。

DB2PRUNE_OPTION_LSNSTRING
piString の値を LSN に指定します。これは、呼び出し側アクション DB2PRUNE_ACTION_LOG が指定されている場合に使用します。

REXX API 構文:

```
PRUNE RECOVERY HISTORY BEFORE :timestamp [WITH FORCE OPTION]
```

REXX API パラメーター:

timestamp
タイム・スタンプを含むホスト変数を示します。指定されたタイム・スタンプと等しいかまたは小さいタイム・スタンプを持つすべての項目が、履歴ファイルから削除されます。

WITH FORCE OPTION
指定した場合、最新のリストア・セット中の一部の項目がファイルから削除されることになる場合でも、指定されたタイム・スタンプに従って履歴ファイルの項目が削除されます。指定しない場合、入力時に指定したタイム・スタンプより小さいか等しい場合でも、最新のリストア・セットが保持されます。

使用上の注意:

履歴ファイルの項目を削除しても、実際のバックアップ、またはロード・ファイルは削除されません。それらのファイルを削除したい場合には、それを手作業で行って、それらのファイルがストレージ・メディア上で使用しているスペースを解放する必要があります。

注意:
最新の全データベース・バックアップを媒体から削除する (さらに、履歴ファイルから項目が削除される) 場合、すべての表スペース (カタログ表スペースおよびユーザー表スペースを含む) のバックアップを取るようにしてください。そのことを怠ると、データベースが回復不能になったり、データベース内のユーザー・データの一部が失われたりするおそれがあります。

関連資料:

- 111 ページの『db2HistoryUpdate - 履歴ファイルの更新』
- 106 ページの『db2HistoryOpenScan - 履歴ファイルのスキャンのオープン』
- 104 ページの『db2HistoryGetEntry - 履歴ファイルの次項目の入手』
- 102 ページの『db2HistoryCloseScan - 履歴ファイルのスキャンのクローズ』
- 453 ページの『SQLCA』

関連サンプル:

- 『dbrecov.sqc -- How to recover a database (C)』
- 『dbrecov.sqC -- How to recover a database (C++)』

db2QuerySatelliteProgress - サテライト同期の照会

サテライト同期セッションの状況をチェックします。

許可:

なし

必要な接続:

なし

API 組み込みファイル:

db2ApiDf.h

C API 構文:

```
/* File: db2ApiDf.h */
/* API: db2QuerySatelliteProgress */
/* ... */
SQL_API_RC SQL_API_FN
  db2QuerySatelliteProgress (
  db2Uint32 versionNumber,
  void *pParmStruct,
  struct sqlca *pSqlca);

typedef struct
{
  db2int32 oStep;
  db2int32 oSubstep;
  db2int32 oNumSubsteps;
  db2int32 oScriptStep;
  db2int32 oNumScriptSteps;
  char *poDescription;
```

```
  char *poError;
  char *poProgressLog;
} db2QuerySatelliteProgressStruct;
/* ... */
```

API パラメーター:

versionNumber
入力。 2 番目のパラメーター *pParmStruct* として渡される構造のバージョンとリリースのレベルを指定します。

pParmStruct
入力。 *db2QuerySatelliteProgressStruct* 構造を指すポインター。

pSqlca
出力。*sqlca* 構造へのポインター。

oStep
出力。同期セッション (db2ApiDf.h で定義されている) の現行ステップを示します。

oSubstep
出力。同期ステップ (*oStep*) をサブステップに分割できる場合、これは現行のサブステップになります。

oNumSubsteps
出力。同期セッションの現行ステップにサブステップ (*oSubstep*) が存在する場合、これは、同期ステップを構成するサブステップの合計数になります。

oScriptStep
出力。現行サブステップがスクリプトの実行である場合、このパラメーターはスクリプト実行の進行状況を報告します (可能な場合)。

oNumScriptSteps
出力。スクリプト・ステップが報告された場合、このパラメーターにはスクリプトの実行を構成するステップの合計数が入ります。

poDescription
出力。サテライトの同期セッションの状態の記述。

poError
出力。同期セッションでエラーが発生している場合、このパラメーターによってエラーの記述が渡されます。

poProgressLog
出力。このパラメーターはサテライトの同期セッションのログ全体を戻します。

関連資料:

• 453 ページの『SQLCA』

db2ReadLog - ログの非同期読み取り

現行のログ状態に関する情報を入手するために、ログ・レコードを DB2 UDB データベース・ログおよびログ・マネージャーから抽出します。この API を使用できるのは、リカバリー可能データベースの場合だけです。データベースがリカバリー可能なのは、データベースが RECOVERY に設定された *logretain*、または ON に設定された *userexit* で構成されている場合です。

許可:

以下のいずれかが必要です。

- *sysadm*
- *dbadm*

必要な接続:

データベース

API 組み込みファイル:

db2ApiDf.h

C API 構文:

```
/* File: db2ApiDf.h */
/* API: db2ReadLog */
/* ... */
SQL_API_RC SQL_API_FN
db2ReadLog (
  db2Uint32 versionNumber,
  void *pDB2ReadLogStruct,
  struct sqlca *pSqlca);

typedef SQL_STRUCTURE db2ReadLogStruct
{
  db2Uint32                   iCallerAction;
  SQLU_LSN                    *piStartLSN;
  SQLU_LSN                    *piEndLSN;
  char                        *poLogBuffer;
  db2Uint32                   iLogBufferSize;
  db2Uint32                   iFilterOption;
  db2ReadLogInfoStruct        *poReadLogInfo;

typedef SQL_STRUCTURE db2ReadLogInfoStruct
{
  SQLU_LSN                    initialLSN;
  SQLU_LSN                    firstReadLSN;
  SQLU_LSN                    nextStartLSN;
  db2Uint32                   logRecsWritten;
  db2Uint32                   logBytesWritten;
  SQLU_LSN                    firstReusedLSN;
  db2Uint32                   timeOfLSNReuse;
  db2TimeOfLog                currentTimeValue;
} db2ReadLogInfoStruct;

typedef SQL_STRUCTURE db2TimeOfLog
{
  db2Uint32                   seconds;
  db2Uint32                   accuracy;
} db2TimeOfLog;
/* ... */
```

API パラメーター:

versionNumber
入力。 2 番目のパラメーター *pDB2ReadLogStruct* として渡される構造のバージョンとリリースのレベルを指定します。

pDB2ReadLogStruct
入力。 *db2ReadLogStruct* を指すポインター。

pSqlca
出力。*sqlca* 構造へのポインター。

iCallerAction
入力。実行するアクションを指定します。

DB2READLOG_READ
開始ログ・シーケンス番号から終了ログ・シーケンス番号までのデータベース・ログを読み取り、この範囲内にあるログ・レコードを戻します。

DB2READLOG_READ_SINGLE
開始ログ・シーケンス番号によって識別される単一のログ・レコード (伝搬可能または伝搬不可のいずれでも) を読み取ります。

DB2READLOG_QUERY
データベース・ログを照会します。照会した結果は、*db2ReadLogInfoStruct* 構造を介して戻されます。

piStartLsn
入力。開始ログ・シーケンス番号は、ログの読み取りを開始する相対バイト・アドレスを指定します。この値は、実際のログ・レコードの始まりでなければなりません。

piEndLsn
入力。終了ログ・シーケンス番号は、ログの読み取りを終了する相対バイト・アドレスを指定します。この値は、*startLsn* の値より大きくなければなりません。実際のログ・レコードの終わりである必要はありません。

poLogBuffer
出力。指定した範囲内で読み取られた、伝搬可能なすべてのログ・レコードが順番に格納されるバッファー。このバッファーは、単一のログ・レコードを保持するのに十分な大きさでなければなりません。目安として、このバッファーは最低限 32 バイトでなければなりません。最大サイズは、要求された範囲のサイズによって異なってきます。バッファー内の各ログ・レコードには、接頭部として 6 バイトのログ・シーケンス番号 (LSN) が付けられます。これは、次のログ・レコードの LSN を示します。

iLogBufferSize
入力。バイト単位でログ・バッファーのサイズを指定します。

iFilterOption
入力。ログ・レコードを読み取るときに使用されるログ・レコード・フィルターのレベルを指定します。有効な値は以下のとおりです。

DB2READLOG_FILTER_OFF
指定された LSN 範囲内ですべてのログ・レコードを読み取ります。

DB2READLOG_FILTER_ON
伝搬可能とマークされた LSN 範囲内でログ・レコードのみを読み取ります。これは非同期ログ読み取り API の基本的な動作です。

poReadLogInfo
出力。呼び出しとデータベース・ログに関する情報を詳述する構造。

使用上の注意:

要求されるアクションがログの読み取りであれば、呼び出し側はログ・シーケンス番号の範囲と、ログ・レコードを保持するバッファーを提供します。この API は要求された LSN の範囲にあるログを順番に読み取ります。さらに、DATA CAPTURE オプションが CHANGES である表に関連付けられたログ・レコードと、現在アクティブなログ情報が入った db2ReadLogInfoStruct 構造を戻します。要求されたアクションが照会であれば、API は現在アクティブなログ情報が入った db2ReadLogInfoStruct 構造を戻します。

非同期ログ読み取りプログラムを使用するには、まずデータベース・ログを照会して有効な開始 LSN を探します。照会の呼び出しに続き、ログ情報の読み取り構造 (db2ReadLogInfoStruct) に、読み取りの呼び出しで使用される有効な開始 LSN (initialLSN メンバー内) が入ります。読み取りでの終了 LSN として使用される値は、次のうちの 1 つになります。

- initialLSN より大きい値
- 非同期ログ読み取りプログラムで現行ログの終わりとして解釈される、FFFF FFFF FFFF

開始および終了 LSN の範囲内で読み取られた伝搬可能なログ・レコードは、ログ・バッファーに戻されます。ログ・レコードには、その LSN は含まれません。これは、バッファーの中で、実際のログ・レコードの前に付けられます。db2ReadLog によって戻されるさまざまな DB2 ログ・レコードの詳細については、『DB2 UDB ログ・レコード』セクションで説明しています。

最初の読み取りの後、次の順番のログ・レコードを読み取るには、db2ReadLogStruct 構造で戻された nextStartLSN フィールドを使用します。この新しい開始 LSN と有効な終了 LSN を使用して、呼び出しをもう一度サブミットします。そうすると、次のブロックのレコードが読み取られます。SQLU_RLOG_READ_TO_CURRENT の *sqlca* コードは、現在アクティブであるログが最後まで読み取られたことを示します。

関連資料:

- 453 ページの『SQLCA』
- 234 ページの『db2Reorg - 再編成』

関連サンプル:

- 『dbrecov.sqc -- How to recover a database (C)』
- 『dbrecov.sqC -- How to recover a database (C++)』

db2ReadLogNoConn - データベース接続なしのログの読み取り

ログ・レコードを DB2 UDB データベース・ログから抽出し、現在のログ状態の情報をログ・マネージャーに照会します。この API の使用に先立ち、db2ReadLogNoConnInit を使用して、この API に入力パラメーターとして渡されるメモリーを割り振ります。この API の使用後は、db2ReadLogNoConnTerm を使用して、メモリーを割り振り解除します。

許可:

なし

必要な接続:

なし

API 組み込みファイル:

db2ApiDf.h

C API 構文:

```
/* File: db2ApiDf.h */
/* API: db2ReadLogNoConn */
/* ... */
SQL_API_RC SQL_API_FN
db2ReadLogNoConn (
  db2Uint32 versionNumber,
  void *pDB2ReadLogNoConnStruct,
  struct sqlca *pSqlca);

typedef SQL_STRUCTURE db2ReadLogNoConnStruct
{
  db2Uint32                   iCallerAction;
  SQLU_LSN                    *piStartLSN;
  SQLU_LSN                    *piEndLSN;
  char                        *poLogBuffer;
  db2Uint32                   iLogBufferSize;
  char                        *piReadLogMemPtr;
  db2ReadLogNoConnInfoStruct  *poReadLogInfo;
} db2ReadLogNoConnStruct;

typedef SQL_STRUCTURE db2ReadLogNoConnInfoStruct
{
  SQLU_LSN                    firstAvailableLSN;
  SQLU_LSN                    firstReadLSN;
  SQLU_LSN                    nextStartLSN;
  db2Uint32                   logRecsWritten;
  db2Uint32                   logBytesWritten;
  db2Uint32                   lastLogFullyRead;
  db2TimeOfLog                currentTimeValue;
} db2ReadLogNoConnInfoStruct;

/* ... */
```

API パラメーター:

versionNumber
入力。 2 番目のパラメーター *pDB2ReadLogNoConnStruct* として渡される構造のバージョンとリリースのレベルを指定します。

pDB2ReadLogNoConnStruct
入力。 *db2ReadLogNoConnStruct* 構造を指すポインター。

pSqlca
出力。*sqlca* 構造へのポインター。

iCallerAction
入力。実行するアクションを指定します。有効な値は以下のとおりです。

DB2READLOG_READ
開始ログ・シーケンス番号から終了ログ・シーケンス番号までのデータベース・ログを読み取り、この範囲内にあるログ・レコードを戻します。

DB2READLOG_READ_SINGLE
開始ログ・シーケンス番号によって識別される単一のログ・レコード (伝搬可能または伝搬不可のいずれでも) を読み取ります。

DB2READLOG_QUERY
データベース・ログを照会します。照会した結果は、db2ReadLogNoConnInfoStruct 構造を介して戻されます。

piStartLSN
入力。開始ログ・シーケンス番号は、ログの読み取りを開始する相対バイト・アドレスを指定します。この値は、実際のログ・レコードの始まりでなければなりません。

piEndLSN
入力。終了ログ・シーケンス番号は、ログの読み取りを終了する相対バイト・アドレスを指定します。この値は、*piStartLsn* の値より大きくなければなりません。実際のログ・レコードの終わりである必要はありません。

poLogBuffer
出力。指定した範囲内で読み取られた、伝搬可能なすべてのログ・レコードが順番に格納されるバッファー。このバッファーは、単一のログ・レコードを保持するのに十分な大きさでなければなりません。目安として、このバッファーは最低限 32 バイトでなければなりません。最大サイズは、要求された範囲のサイズによって異なってきます。バッファー内の各ログ・レコードには、接頭部として 6 バイトのログ・シーケンス番号 (LSN) が付けられます。これは、次のログ・レコードの LSN を示します。

iLogBufferSize
入力。バイト単位でログ・バッファーのサイズを指定します。

piReadLogMemPtr
入力。初期設定呼び出しで割り振られた、サイズ iReadLogMemoryLimit のメモリー・ブロック。このメモリーには、呼び出しのたびに API が必要とする永続データが含まれています。このメモリー・ブロックは、どのような方法であっても呼び出し側は再割り振りまたは変更してはなりません。

poReadLogInfo
出力。 *db2ReadLogNoConnInfoStruct* 構造を指すポインター。

firstAvailableLSN
使用可能なログ内で最初の使用可能な LSN。

firstReadLSN
この呼び出しでの最初の LSN 読み取り。

nextStartLSN
次の読み取り可能な LSN。

logRecsWritten
ログ・バッファー・フィールド *poLogBuffer* に書き込まれるログ・レコードの数。

logBytesWritten
ログ・バッファー・フィールド *poLogBuffer* に書き込まれるログ・バッファー・フィールドのバイト数。

lastLogFullyRead
読み取られて完了する最後のログ・ファイルを示す数。

使用上の注意:

db2ReadLogNoConn API は、db2ReadLogNoConnInit API を使用して割り振られるメモリー・ブロックを必要とします。メモリー・ブロックは、入力パラメーターとして、続くすべての db2ReadLogNoConn API に渡されなければならず、変更してはなりません。

ログの順次読み取りを要求する場合、API はログ・シーケンス番号 (LSN) の範囲および割り振られたメモリーを必要とします。 API は、初期設定されたときに指定されたフィルター・オプションおよび LSN 範囲に基づいて、ログ・レコードの順序を戻します。照会を要求する場合、ログ情報の読み取り構造に、読み取りの呼び出しで使用される有効な開始 LSN が入ります。読み取りでの終了 LSN として使用される値は、次のうちの 1 つになります。

- 呼び出し側が指定した startLSN の値より大きい値。
- 非同期ログ読み取りプログラムで、使用可能なログの終わりとして解釈される FFFF FFFF FFFF。

開始および終了 LSN の範囲内で読み取られた伝搬可能なログ・レコードは、ログ・バッファーに戻されます。ログ・レコードには、それの LSN は含まれません。 LSN は、バッファーの中で、実際のログ・レコードの前に付けられます。db2ReadLogNoConn によって戻されるさまざまな DB2 UDB ログ・レコードの詳細については、 『DB2 UDB ログ・レコード』セクションで説明しています。

最初の読み取りの後、次の順番のログ・レコードを読み取るには、db2ReadLogNoConnInfoStruct で戻された nextStartLSN 値を使用します。この新しい開始 LSN と有効な終了 LSN を使用して呼び出しをもう一度サブミットすると、次のレコード・ブロックが読み取られます。 SQLU_RLOG_READ_TO_CURRENT の sqlca コードは、使用可能なログ・ファイルが最後まで読み取られたことを示します。

この API がもう使用されない場合、db2ReadLogNoConnTerm を使用してメモリーを終了します。

関連資料:

- 453 ページの『SQLCA』

- 225 ページの『db2ReadLogNoConnInit - データベース接続なしのログ読み取りの初期設定』
- 227 ページの『db2ReadLogNoConnTerm - データベース接続なしのログ読み取りの終了』

db2ReadLogNoConnInit - データベース接続なしのログ読み取りの初期設定

db2ReadLogNoConn によって使用されるメモリーを割り振り、ログ・レコードを DB2 UDB データベース・ログから抽出します。また、ログ・マネージャーを照会して現行のログ状態に関する情報を取得できるようにします。

許可:

なし

必要な接続:

なし

API 組み込みファイル:

db2ApiDf.h

C API 構文:

```
/* File: db2ApiDf.h */
/* API: db2ReadLogNoConnInit */
/* ... */
SQL_API_RC SQL_API_FN
db2ReadLogNoConnInit (
  db2Uint32      versionNumber,
  void * pDB2ReadLogNoConnInitStruct,
  struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2ReadLogNoConnInitStruct
{
  db2Uint32                   iFilterOption;
  char                        *piLogFilePath;
  char                        *piOverflowLogPath;
  db2Uint32                   iRetrieveLogs;
  char                        *piDatabaseName;
  char                        *piNodeName;
  db2Uint32                   iReadLogMemoryLimit;
  char                        **poReadLogMemPtr;
} db2ReadLogNoConnInitStruct;
/* ... */
```

API パラメーター:

versionNumber
入力。 2 番目のパラメーター *pDB2ReadLogNoConnInitStruct* として渡される構造のバージョンとリリースのレベルを指定します。

pDB2ReadLogNoConnInitStruct
入力。 *db2ReadLogNoConnInitStruct* 構造を指すポインター。

pSqlca
出力。*sqlca* 構造へのポインター。

iFilterOption
入力。ログ・レコードを読み取るときに使用されるログ・レコード・フィルターのレベルを指定します。有効な値は以下のとおりです。

DB2READLOG_FILTER_OFF
指定された LSN 範囲内ですべてのログ・レコードを読み取ります。

DB2READLOG_FILTER_ON
伝搬可能とマークされた LSN 範囲内でログ・レコードのみを読み取ります。これは非同期ログ読み取り API の基本的な動作です。

piLogFilePath
入力。読み取られるログ・ファイルがある場所のパス。

piOverflowLogPath
入力。読み取られるログ・ファイルがある場所の代替パス。

iRetrieveLogs
入力。ログ・ファイル・パス、またはオーバーフロー・ログ・パスのどちらでも検索できないログ・ファイルを検索するために、ユーザー出口を呼び出すかどうかを指定するオプション。有効な値は以下のとおりです。

DB2READLOG_RETRIEVE_OFF
欠落したログ・ファイルを検索するために、ユーザー出口は呼び出しません。

DB2READLOG_RETRIEVE_LOGPATH
欠落したログ・ファイルを検索するために、指定されたログ・ファイル・パスにユーザー出口を呼び出します。

DB2READLOG_RETRIEVE_OVERFLOW
欠落したログ・ファイルを検索するために、指定されたオーバーフロー・ログ・パスにユーザー出口を呼び出します。

piDatabaseName
入力。読み取り中のリカバリー・ログを所有するデータベースの名前。これは、上記の検索オプションが指定された場合に必要です。

piNodeName
入力。読み取り中のリカバリー・ログを所有するノードの名前。これは、上記の検索オプションが指定された場合に必要です。

iReadLogMemoryLimit
入力。 API が内部に割り振る最大バイト数。

poReadLogMemPtr
出力。 API が割り振った、サイズ *iReadLogMemoryLimit* のメモリーのブロック。このメモリーには、呼び出しのたびに API が必要とする永続データが含まれています。このメモリー・ブロックは、どのような方法であっても呼び出し側は再割り振りまたは変更してはなりません。

使用上の注意:

db2ReadLogNoConnInit によって初期設定されたメモリーは変更してはなりません。

db2ReadLogNoConn がもう使用されない場合、db2ReadLogNoConnTerm を呼び出し、 db2ReadLogNoConnInit によって初期設定されたメモリーを割り振り解除します。

関連資料:

- 453 ページの『SQLCA』
- 222 ページの『db2ReadLogNoConn - データベース接続なしのログの読み取り』
- 227 ページの『db2ReadLogNoConnTerm - データベース接続なしのログ読み取りの終了』

db2ReadLogNoConnTerm - データベース接続なしのログ読み取りの終了

本来は db2ReadLogNoConnInit によって初期設定され、db2ReadLogNoConn によって使用されるメモリーを割り振り解除します。

許可:

| なし

必要な接続:

| なし

API 組み込みファイル:

db2ApiDf.h

C API 構文:

```
/* File: db2ApiDf.h */
/* API: db2ReadLogNoConnTerm */
/* ... */
SQL_API_RC SQL_API_FN
db2ReadLogNoConnTerm (
  db2Uint32      versionNumber,
  void * pDB2ReadLogNoConnTermStruct,
  struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2ReadLogNoConnTermStruct
{
  char                         **poReadLogMemPtr;
} db2ReadLogNoConnTermStruct;
/* ... */
```

API パラメーター:

| **versionNumber**
入力。 2 番目のパラメーター *pDB2ReadLogNoConnTermStruct* として渡される構造のバージョンとリリースのレベルを指定します。

pDB2ReadLogNoConnTermStruct
入力。 *db2ReadLogNoConnTermStruct* 構造を指すポインター。

pSqlca
出力。*sqlca* 構造へのポインター。

poReadLogMemPtr
出力。初期設定呼び出しで割り振られたメモリーのブロックを指すポインター。このポインターは解放され、NULL に設定されます。

関連資料:

- 453 ページの『SQLCA』
- 222 ページの『db2ReadLogNoConn - データベース接続なしのログの読み取り』
- 225 ページの『db2ReadLogNoConnInit - データベース接続なしのログ読み取りの初期設定』

db2Recover - データベースのリカバリー

データベースを、特定のポイント・イン・タイムまで、またはログの終わりまでリストアおよびロールフォワードします。

有効範囲:

パーティション・データベース環境では、この API はカタログ・パーティションからのみ呼び出すことができます。データベース・パーティション・サーバーが 1 つも指定されていない場合には、 db2nodes.cfg ファイルにリストされているすべてのデータベース・パーティション・サーバーに影響を与えます。特定のポイント・イン・タイムが指定される場合、この API はすべてのデータベース・パーティションに影響します。

許可:

既存のデータベースにリカバリーするには、次のいずれかが必要です。

- *sysadm*
- *sysctrl*
- *sysmaint*

新規のデータベースにリカバリーするには、次のいずれかが必要です。

- *sysadm*
- *sysctrl*

必要な接続:

既存のデータベースをリカバリーするには、データベース接続が必要です。この API を呼び出せば、指定したデータベースへの接続が自動的に確立され、リカバリー操作が終了すると接続が解放されます。新規のデータベースにリカバリーする場合、インスタンスおよびデータベース。データベースを作成するには、インスタンス・アタッチメントが必要です。

API 組み込みファイル:

db2ApiDf.h

C API 構文:

```
/* File: db2ApiDf.h */
/* API: db2Recover */
/* ... */
SQL_API_RC SQL_API_FN
  db2Recover (
       db2Uint32 versionNumber,
       void * pDB2RecovStruct,
       struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2RecoverStruct
{
  char                         *piSourceDBAlias;
  char                         *piUsername;
  char                         *piPassword;
  db2Uint32                    iRecoverCallerAction;
  db2Uint32                    iOptions;
  sqlint32                     *poNumReplies;
  struct sqlurf_info           *poNodeInfo;
  char                         *piStopTime;
  char                         *piOverflowLogPath;
  db2Uint32                    iNumChngLgOvrflw;
  struct sqlurf_newlogpath     *piChngLogOvrflw;
  db2int32                     iAllNodeFlag;
  db2int32                     iNumNodes;
  SQL_PDB_NODE_TYPE            *piNodeList;
  db2int32                     iNumNodeInfo;
  db2Uint32                    iRollforwardFlags;
  char                         *piHistoryFile;
  db2Uint32                    iNumChngHistoryFile;
  struct sqlu_histFile         *piChngHistoryFile;
} db2RecoverStruct;
/* ... */
```

汎用 API 構文:

```
/* File: db2ApiDf.h */
/* API: db2gRecover */
/* ... */
SQL_API_RC SQL_API_FN
  db2gRecover (
       db2Uint32 versionNumber,
       void * pDB2gRecoverStruct,
       struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2gRecoverStruct
{
  char                         *piSourceDBAlias;
  db2Uint32                    iSourceDBAliasLen;
  char                         *piUserName;
  db2Uint32                    iUserNameLen;
  char                         *piPassword;
  db2Uint32                    iPasswordLen;
  db2Uint32                    iRecoverCallerAction;
  db2Uint32                    iOptions;
  sqlint32                     *poNumReplies;
  struct sqlurf_info           *poNodeInfo;
  char                         *piStopTime;
  db2Uint32                    iStopTimeLen;
  char                         *piOverflowLogPath;
  db2Uint32                    iOverflowLogPathLen;
  db2Uint32                    iNumChngLgOvrflw;
  struct sqlurf_newlogpath     *piChngLogOvrflw;
  db2int32                     iAllNodeFlag;
  db2int32                     iNumNodes;
  SQL_PDB_NODE_TYPE            *piNodeList;
  db2int32                     iNumNodeInfo;
  db2Uint32                    iRollforwardFlags;
```

```
  char                        *piHistoryFile;
  db2Uint32                   iHistoryFileLen;
  db2Uint32                   iNumChngHistoryFile;
  struct sqlu_histFile        *piChngHistoryFile;
} db2gRecoverStruct;
/* ... */
```

API パラメーター:

versionNumber
入力。 2 番目のパラメーター *pDB2RecoverStruct* として渡される構造のバージョンとリリースのレベルを指定します。

pDB2RecoverStruct
入力。 *db2RecoverStruct* 構造を指すポインター。

pSqlca
出力。*sqlca* 構造へのポインター。

piSourceDBAlias
入力。リカバリーするデータベースのデータベース別名を含むストリング。

iSourceDBAliasLen
piSourceDBAlias の長さ (バイト単位)。

piUserName
入力。接続の試行時に使用されるユーザー名を含むストリングを指定します。 NULL にすることもできます。

iUserNameLen
piUsername の長さ (バイト単位)。

piPassword
入力。ユーザー名とともに使用されるパスワードを含むストリングを指定します。 NULL にすることもできます。

iPasswordLen
piPassword の長さ (バイト単位)。

iRecoverCallerAction
入力。有効な値は以下のとおりです。

DB2RESTORE_NOINTERRUPT
リストア操作を開始します。リストアを自動実行するよう指定します。通常ユーザーの介入を要求するシナリオは、呼び出し側への最初の戻りなしに試行されるか、エラーを生成します。この呼び出し側アクションは、たとえば、リストアに必要なメディアがすべて装てんされていることが明らかで、ユーティリティーによるプロンプトが必要とされない場合に使用してください。

DB2RESTORE_CONTINUE
ユーザーがユーティリティーによって要求された何らかのアクション (たとえば、新しいテープの装てん) を実行した後で、リストア操作を継続します。

DB2RESTORE_TERMINATE
ユーザーがユーティリティーによって要求された何らかのアクションの実行に失敗した場合、リストア操作を終了します。

DB2RESTORE_DEVICE_TERMINATE
リストア操作によって使用される装置のリストから特定の装置を除外します。特定の装置が入力でいっぱいになると、リストア・ユーティリティーは呼び出し側に警告を戻します。その場合、この呼び出し側アクションを指定してリストア・ユーティリティーを再び呼び出すことによって、警告が生成される原因となった装置を使用装置のリストから除外してください。

DB2RESTORE_PARM_CHK
リストア操作を実行することなく、パラメーターの妥当性を検査します。このオプションは、呼び出しが戻った後でデータベース接続を終了しません。この呼び出しが正常に戻った後、ユーザーが DB2RESTORE_CONTINUE で呼び出しを発行し、処置を進めることが期待されます。

DB2RESTORE_PARM_CHK_ONLY
リストア操作を実行することなく、パラメーターの妥当性を検査します。この呼び出しが戻る前に、この呼び出しによって確立したデータベース接続は終了し、後続する呼び出しは必要なくなります。

DB2RESTORE_TERMINATE_INCRE
完了する前に増分リストア操作を終了します。

DB2ROLLFORWARD_LOADREC_CONT
警告メッセージを生成した装置の使用を続けます (たとえば、新しいテープをマウントしたときなど)。

DB2ROLLFORWARD_DEVICE_TERM
警告メッセージを生成した装置の使用を停止します (たとえば、それ以上テープがない場合)。

DB2ROLLFORWARD_LOAD_REC_TERM
ロード・リカバリーに使用されているすべての装置を終了させます。

iOptions
入力。有効な値は以下のとおりです。

DB2RECOVER_EMPTY_FLAG
フラグが指定されていません。

DB2RECOVER_LOCAL_TIME
piStopTime によって停止時刻として指定された値が、 GMT ではなくローカル時間であることを示します。これはデフォルト設定です。

DB2RECOVER_GMT_TIME
このフラグは、*piStopTime* によって停止時刻として指定された値が、 GMT (グリニッジ標準時) であることを示します。

poNumReplies
出力。受信した応答の数。

poNodeInfo
出力。データベース・パーティション応答情報。

piStopTime
入力。 ISO 形式のタイム・スタンプを含む文字ストリング。このタイム・スタンプで設定された時刻を過ぎると、データベース・リカバリーは停止します。可能な限りロールフォワードしたい場合には、SQLUM_INFINITY_TIMESTAMP を指定してください。DB2ROLLFORWARD_QUERY、DB2ROLLFORWARD_PARM_CHECK、およびいずれかのロード・リカバリー (DB2ROLLFORWARD_LOADREC_) 呼び出し側アクションの場合は、NULL にすることができます。

iStopTimeLen
piStopTime の長さ (バイト単位)。

piOverflowLogPath
入力。使用される代替ログ・パスを指定します。このユーティリティーを使用する前に、ユーザーが、アクティブ・ログ・ファイルの他に、アーカイブ・ログ・ファイルを *logpath* 構成パラメーターで指定した場所に移動させることが必要です。このことは、ログ・パスに十分なスペースがない場合に問題になる可能性があります。その問題を解決するために、オーバーフロー・ログ・パスが備えられています。ロールフォワード・リカバリー中に、必要なログ・ファイルは、まずログ・パスで探索され、次にオーバーフロー・ログ・パスで探索されます。表スペースのロールフォワード・リカバリーに必要なログ・ファイルは、ログ・パスまたはオーバーフロー・ログ・パスのいずれかに置かれる可能性があります。呼び出し側がオーバーフロー・ログ・パスを指定しない場合、デフォルト値はログ・パスです。パーティション・データベース環境では、オーバーフロー・ログ・パスは有効な完全修飾パスでなければなりません。デフォルトのパスは、各データベース・パーティションのデフォルトのオーバーフロー・ログ・パスです。単一パーティション・データベース環境では、サーバーがローカルであれば、オーバーフロー・ログ・パスは相対パスにすることもできます。

iOverflowLogPathLen
piOverflowLogPath の長さ (バイト単位)。

iNumChngLgOvrflw
入力。パーティション・データベース環境のみ。変更されるオーバーフロー・ログ・パスの数。新しいログ・パスは、指定されたデータベース・パーティション・サーバーのデフォルトのオーバーフロー・ログ・パスだけをオーバーライドします。

piChngLogOvrflw
入力。パーティション・データベース環境のみ。変更されるオーバーフロー・ログ・パスの完全修飾名が入っている構造を指すポインター。新しいログ・パスは、指定されたデータベース・パーティション・サーバーのデフォルトのオーバーフロー・ログ・パスだけをオーバーライドします。

iAllNodeFlag
入力。パーティション・データベース環境のみ。ロールフォワード操作が、`db2nodes.cfg` で定義されているすべてのデータベース・パーティション・サーバーに適用されるかどうかを示します。有効な値は以下のとおりです。

DB2_NODE_LIST
piNodeList で渡されたリスト内のデータベース・パーティション・サーバーに適用されます。

DB2_ALL_NODES
すべてのデータベース・パーティション・サーバーに適用されます。 *piNodeList* は NULL でなければなりません。これはデフォルト値です。

DB2_ALL_EXCEPT
piNodeList で渡されたリスト内のデータベース・パーティション・サーバーを除いた、すべてのデータベース・パーティション・サーバーに適用されます。

DB2_CAT_NODE_ONLY
カタログ・パーティションにのみ適用されます。 *piNodeList* は NULL でなければなりません。

iNumNodes
入力。 *piNodeList* 配列内のデータベース・パーティション・サーバーの数を指定します。

piNodeList
入力。ロールフォワード操作を実行する対象のデータベース・パーティション・サーバー番号の配列を指すポインター。

iNumNodeInfo
入力。出力パラメーター *poNodeInfo* のサイズを定義します。これは、ロールフォワードされるそれぞれのデータベース・パーティションからの状況情報を保持するのに十分な大きさでなければなりません。単一パーティション・データベース環境では、このパラメーターは 1 に設定しなければなりません。このパラメーターの値は、この API が呼び出されるデータベース・パーティション・サーバーの数と同じにする必要があります。

RollforwardFlags
入力。ロールフォワード・フラグを指定します。有効な値 (`db2ApiDf.h` で定義) は、以下のとおりです。

DB2ROLLFORWARD_EMPTY_FLAG
フラグが指定されていません。

DB2ROLLFORWARD_LOCAL_TIME
ユーザーが GMT 時間ではなくユーザーの現地時間を使用して特定の時点にロールフォワードすることを可能にします。これによって、ユーザーがローカル・マシンで特定の時点にロールフォワードすることが容易になり、現地時間を GMT ポイント・イン・タイムに変換することによって生じる潜在的なユーザー・エラーの発生を除去します。

piHistoryFile
履歴ファイル。

iHistoryFileLen
piHistoryFile の長さ (バイト単位)。

iNumChngHistoryFile
リスト内の履歴ファイルの数。

piChngHistoryFile
履歴ファイルのリスト。

使用上の注意:

関連資料:

- 「コマンド・リファレンス」の『RECOVER DATABASE コマンド』

db2Reorg - 再編成

情報を短縮し、行や索引データを再構成してフラグメント化されたデータを除去することにより、表または表に定義されたすべての索引を再編成します。

許可:

以下のいずれかです。

- *sysadm*
- *sysctrl*
- *sysmaint*
- *dbadm*
- 表に対する CONTROL 特権

必要な接続:

データベース

API 組み込みファイル:

db2ApiDf.h

C API 構文:

```
/* File: db2ApiDf.h */
/* API: db2Reorg */
/* ... */
SQL_API_RC SQL_API_FN
  db2Reorg (
    db2Uint32 versionNumber,
    void *pReorgStruct,
    struct sqlca *pSqlca);

typedef SQL_STRUCTURE db2ReorgStruct
{
  db2Uint32 reorgType;
  db2Uint32 reorgFlags;
  db2int32 nodeListFlag;
  db2Uint32 numNodes;
  SQL_PDB_NODE_TYPE           *pNodeList;
  union db2ReorgObject reorgObject;
} db2ReorgStruct;

union db2ReorgObject
{
  struct db2ReorgTable tableStruct;
  struct db2ReorgIndexesAll indexesAllStruct;
};

typedef SQL_STRUCTURE db2ReorgTable
{
  char *pTableName;
  char *pOrderByIndex;
  char *pSysTempSpace;
} db2ReorgTable;
```

```
typedef SQL_STRUCTURE db2ReorgIndexesAll
{
  char *pTableName;
} db2ReorgIndexesAll;
/* ... */
```

汎用 API 構文:

```
/* File: db2ApiDf.h */
/* API: db2gReorg */
/* ... */
SQL_API_RC SQL_API_FN
  db2gReorg (
    db2Uint32 versionNumber,
    void *pReorgStruct,
    struct sqlca *pSqlca);

typedef SQL_STRUCTURE db2gReorgStruct
{
  db2Uint32 reorgType;
  db2Uint32 reorgFlags;
  db2int32 nodeListFlag;
  db2Uint32 numNodes;
  SQL_PDB_NODE_TYPE *pNodeList;
  union db2gReorgObject reorgObject;
} db2gReorgStruct;

typedef SQL_STRUCTURE db2gReorgNodes
{
  SQL_PDB_NODE_TYPE  nodeNum[SQL_PDB_MAX_NUM_NODE];
} db2gReorgNodes;

union db2gReorgObject
{
  struct db2gReorgTable tableStruct;
  struct db2gReorgIndexesAll indexesAllStruct;
};

typedef SQL_STRUCTURE db2gReorgTable
{
  db2Uint32 tableNameLen;
  char *pTableName;
  db2Uint32 orderByIndexLen;
  char *pOrderByIndex;
  db2Uint32 sysTempSpaceLen;
  char *pSysTempSpace;
} db2gReorgTable;

typedef SQL_STRUCTURE db2gReorgIndexesAll
{
  db2Uint32  tableNameLen;
  char       *pTableName;
} db2gReorgIndexesAll;
/* ... */
```

API パラメーター:

versionNumber

入力。 2 番目のパラメーター *pReorgStruct* として渡される構造のバージョンとリリースのレベルを指定します。

pReorgStruct

入力。 *db2ReorgStruct* 構造を指すポインター。

pSqlca
出力。 *sqlca* 構造を指すポインター。

reorgType
入力。再編成のタイプを指定します。有効な値 (db2ApiDf.h で定義) は、以下のとおりです。

DB2REORG_OBJ_TABLE_OFFLINE
表をオフラインで再編成します。

DB2REORG_OBJ_TABLE_INPLACE
表をインプレースで再編成します。

DB2REORG_OBJ_INDEXESALL
すべての索引を再編成します。

reorgFlags
入力。再編成オプション。有効な値 (db2ApiDf.h で定義) は、以下のとおりです。

DB2REORG_OPTION_NONE
デフォルトのアクション。

DB2REORG_LONGLOB
DB2REORG_OBJ_TABLE_OFFLINE が *reorgType* と指定されているときに使用される、長フィールドおよび LOB を再編成します。

DB2REORG_INDEXSCAN
DB2REORG_OBJ_TABLE_OFFLINE が *reorgType* と指定されているときに使用される、索引のスキャンの使用を再クラスターします。

DB2REORG_START_ONLINE
DB2REORG_OBJ_TABLE_INPLACE が *reorgType* と指定されているときに使用される、オンライン再編成を開始します。

DB2REORG_PAUSE_ONLINE
DB2REORG_OBJ_TABLE_INPLACE が *reorgType* と指定されているときに使用される、既存のオンライン再編成を休止します。

DB2REORG_STOP_ONLINE
DB2REORG_OBJ_TABLE_INPLACE が *reorgType* と指定されているときに使用される、既存のオンライン再編成を停止します。

DB2REORG_RESUME_ONLINE
DB2REORG_OBJ_TABLE_INPLACE が *reorgType* と指定されているときに使用される、休止されていたオンライン再編成を再開します。

DB2REORG_NOTRUNCATE_ONLINE
DB2REORG_OBJ_TABLE_INPLACE が *reorgType* と指定されているときに使用される、表の切り捨てを実行しません。

DB2REORG_ALLOW_NONE
表への読み取りまたは書き込みアクセスを許可しません。
DB2REORG_OBJ_TABLE_INPLACE が *reorgType* と指定されているときは、このパラメーターはサポートされません。

DB2REORG_ALLOW_WRITE
表への読み取りおよび書き込みアクセスを許可します。
DB2REORG_OBJ_TABLE_OFFLINE が *reorgType* と指定されているときは、このパラメーターはサポートされません。

DB2REORG_ALLOW_READ
表への読み取りアクセスのみを許可します。

DB2REORG_CLEANUP_NONE
DB2REORG_OBJ_INDEXESALL が *reorgType* と指定されているときは、クリーンアップは必要ありません。

DB2REORG_CLEANUP_ALL
DB2REORG_OBJ_INDEXESALL が *reorgType* と指定されているときに使用される、コミット済みの疑似削除キーおよびコミット済みの疑似空ページを除去することにより、表の索引をクリーンアップします。

DB2REORG_CLEANUP_PAGES
DB2REORG_OBJ_INDEXESALL が *reorgType* と指定されているときに使用される、コミット済みの疑似空ページのみクリーンアップします。ページにある疑似削除キーはクリーンアップしません。

DB2REORG_CONVERT_NONE
DB2REORG_OBJ_INDEXESALL が *reorgType* と指定されているときに使用される、変換は必要ありません。

DB2REORG_CONVERT
DB2REORG_OBJ_INDEXESALL が *reorgType* と指定されているときに使用される、タイプ 2 の索引に変換します。

nodeListFlag
入力。再編成するノードを指定します。有効な値 (`db2ApiDf.h` で定義) は、以下のとおりです。

DB2REORG_NODE_LIST
ノード・リスト配列内のすべてのノードにサブミットします。

DB2REORG_ALL_NODES
データベース・パーティション・グループ内のすべてのノードにサブミットします。

DB2REORG_ALL_EXCEPT
ノード・リスト・パラメーターによって指定されたノードを除き、すべてのノードにサブミットします。

numNodes
入力。ノード・リスト配列内のノード数。

pNodeList
ノード番号の配列を指すポインター。

reorgObject
入力。再編成されるオブジェクトのタイプを指定します。

tableStruct
表の再編成のオプションを指定します。

indexesAllStruct
索引の再編成のオプションを指定します。

tableNameLen
入力。 pTableName の長さ (バイト単位) を指定します。

pTableName
入力。再編成するオブジェクトの名前を指定します。

orderByIndexLen
入力。 pOrderByIndex の長さ (バイト単位) を指定します。

pOrderByIndex
入力。表の順序を決める索引を指定します。

sysTempSpaceLen
入力。 pSysTempSpace の長さ (バイト単位) を指定します。

pSysTempSpace
入力。一時オブジェクトを作成する SYSTEM TEMPORARY 表スペースを指定します。

使用上の注意:

表へのアクセスのパフォーマンス、索引のスキャン、および索引ページのプリフェッチの効果は、表データが何度も変更され、フラグメント化およびクラスター解除されたときに、悪影響を受ける可能性があります。 REORGCHK を使用して、表またはその索引が再編成の対象であるかを判別してください。すべての作業がコミットされ、すべてのオープン・カーソルはクローズされます。表またはその索引の再編成後、db2Runstats を使用して統計を更新し、 sqlarbnd を使用してこの表を使用しているパッケージを再バインドします。

表がいくつかのノードにパーティション分割されており、再編成が影響するノードのいずれかで失敗した場合には、失敗したノードだけが表の再編成をロールバックします。

注: 表の再編成が失敗した場合は、一時ファイルを削除しないでください。データベース・マネージャーは、これらの一時ファイルをデータベースのリカバリーに使用します。

索引の名前を指定した場合、データベース・マネージャーは、索引内の配列に基づいてデータを再編成します。パフォーマンスを最大にするには、 SQL 照会で頻繁に使用される索引を指定してください。索引の名前が指定されておらず、クラスタリング索引が存在する場合、データはクラスタリング索引に従って配列されます。

表の PCTFREE 値は、ページごとに指定されたフリー・スペースの量を決定します。この値が設定されていない場合、ユーティリティーは、それぞれのページにできるだけ大きなスペースを割り当てます。

表の再編成に続いて表スペースのロールフォワード・リカバリーを完了させるには、データと LONG 表スペースの両方で、ロールフォワードが使用可能でなければなりません。

表に COMPACT オプションを使用しない LOB 列がある場合、 LOB DATA 記憶オブジェクトは、表の再編成後、かなり大きくなる可能性があります。これは、行が再編成された順序と、使用された (SMS/DMS) 表スペースのタイプが原因で起こる可能性があります。

索引を再編成する場合、アクセス・オプションを使用して、他のトランザクションに、表への読み取り専用または読み取り/書き込みアクセスを許可します。表へのアクセスが許可されていない期間に、再編成された索引が使用可能になると、短いロックアウト期間が存在します。

索引の再作成が必要なために、読み取りまたは書き込みアクセス許可がある索引の再編成が失敗した場合、再編成はアクセスを許可せずに続行するように切り替えられます。メッセージは管理用通知ログおよび診断ログに書き込まれ、アクセス・モードの変更についてユーザーに警告します。

アクセスがない索引の再編成が失敗した場合、その索引は使用不可であり、次回の表アクセスで再作成する必要があります。

この API は、以下では使用できません。

- 索引拡張に基づくビューまたは索引
- DMS 表。ただしその表がある表スペースのオンライン・バックアップが実行されている間
- 宣言済み一時表

関連資料:

- 302 ページの『sqlarbnd - 再バインド』
- 453 ページの『SQLCA』
- 267 ページの『db2Runstats - 統計の実行』

関連サンプル:

- 『dbstat.sqb -- Reorganize table and run statistics (MF COBOL)』
- 『tbreorg.sqc -- How to reorganize a table and update its statistics (C)』
- 『tbreorg.sqC -- How to reorganize a table and update its statistics (C++)』

db2ResetAlertCfg - アラート構成のリセット

特定のオブジェクトのヘルス・インディケーター設定を、そのオブジェクト・タイプの現行デフォルトにリセットするか、またはオブジェクト・タイプの現行のデフォルトのヘルス・インディケーター設定を、インストール時のデフォルトにリセットします。

許可:

以下のいずれかです。

- *sysadm*
- *sysmaint*
- *sysctrl*

必要な接続:

インスタンス。インスタンス・アタッチが存在しない場合は、デフォルトのインスタンス・アタッチが作成されます。

API 組み込みファイル:

db2ApiDf.h

C API 構文:

```
/* File: db2ApiDf.h */
/* API:  db2ResetAlertCfg */
/* ... */
SQL_API_RC SQL_API_FN
db2ResetAlertCfg(
  db2Uint32 versionNumber,
  void *pParmStruct,
  struct sqlca *pSqlca );

typedef SQL_STRUCTURE db2ResetAlertCfgData
{
  db2Uint32             iObjType;
  char                 *piObjName;
  char                 *piDbname;
  db2Uint32             iIndicatorID;
} db2ResetAlertCfgData;
/* ... */
```

API パラメーター:

versionNumber
入力。 2 番目のパラメーター *pParmStruct* として渡される構造のバージョンとリリースのレベルを指定します。

pParmStruct
入力。 *db2ResetAlertCfgData* 構造を指すポインター。

pSqlca
出力。*sqlca* 構造へのポインター。

iObjType
入力。構成がリセットされるオブジェクトのタイプを指定します。有効な値 (db2ApiDf.h で定義) は、以下のとおりです。

- DB2ALERTCFG_OBJTYPE_DBM
- DB2ALERTCFG_OBJTYPE_DATABASES
- DB2ALERTCFG_OBJTYPE_TABLESPACES
- DB2ALERTCFG_OBJTYPE_TS_CONTAINERS
- DB2ALERTCFG_OBJTYPE_DATABASE
- DB2ALERTCFG_OBJTYPE_TABLESPACE
- DB2ALERTCFG_OBJTYPE_TS_CONTAINER

piObjName
入力。オブジェクト・タイプ *iObjType* が DB2ALERTCFG_OBJTYPE_TS_CONTAINER または DB2ALERTCFG_OBJTYPE_TABLESPACE に設定される場合、表スペースまたは表スペース・コンテナーの名前。データベース・コンテナーの名前は、<tablespace-numericalID>.<tablespace-containter-name> と定義されます。

piDbname

入力。オブジェクト・タイプ *iObjType* が DB2ALERTCFG_OBJTYPE_TS_CONTAINER、DB2ALERTCFG_OBJTYPE_TABLESPACE、および DB2ALERTCFG_OBJTYPE_DATABASE に設定される場合、構成がリセットされるデータベースの別名。

iIndicatorID

入力。構成のリセットが適用されるヘルス・インディケーター。

使用上の注意:

iObjType が DB2ALERTCFG_OBJTYPE_DBM、DB2ALERTCFG_OBJTYPE_DATABASES、DB2ALERTCFG_OBJTYPE_TABLESPACES、DB2ALERTCFG_OBJTYPE_TS_CONTAINERS である場合、または *piObjName* および *piDbName* が両方とも NULL である場合、オブジェクト・タイプの現行のデフォルトはリセットされます。 *iObjType* が DB2ALERTCFG_OBJTYPE_DATABASE、DB2ALERTCFG_OBJTYPE_TABLESPACE、DB2ALERTCFG_OBJTYPE_TS_CONTAINER である場合、および *piDbName* と *piObjName* (データベースには必要なし) が指定されている場合、特定のオブジェクトの現行の設定はリセットされます。

関連資料:

- 453 ページの『SQLCA』
- 75 ページの『db2GetAlertCfg - アラート構成の入手』
- 281 ページの『db2UpdateAlertCfg - アラート構成の更新』

db2ResetMonitor - モニターのリセット

呼び出しを発行しているアプリケーションについて、指定されたデータベースの、またはすべてのアクティブ・データベースのデータベース・システム・モニターのデータをリセットします。

有効範囲:

この API はインスタンス上の特定のデータベース・パーティション・サーバー、またはインスタンス上のすべてのデータベース・パーティションに影響を与えます。

許可:

以下のいずれかです。

- *sysadm*
- *sysctrl*
- *sysmaint*
- *sysmon*

必要な接続:

インスタンス。インスタンス・アタッチが存在しない場合は、デフォルトのインスタンス・アタッチが作成されます。

リモート・インスタンス (または別のローカル・インスタンス) 用のモニター・スイッチをリセットするには、まず最初にそのインスタンスにアタッチすることが必要です。

API 組み込みファイル:

db2ApiDf.h

C API 構文:

```
/* File: db2ApiDf.h */
/* API: db2ResetMonitor */
/* ... */
SQL_API_RC SQL_API_FN

db2ResetMonitor (
  db2Uint32 versionNumber,
  void *pParmStruct,
  struct sqlca *pSqlca);

typedef struct
{
        db2Uint32                       iResetAll;
        char                            *piDbAlias;
        db2Uint32                       iVersion;
        db2int32                        iNodeNumber;
}db2ResetMonitorData;
/* ... */
```

汎用 API 構文:

```
/* File: db2ApiDf.h */
/* API: db2gResetMonitor */
/* ... */
SQL_API_RC SQL_API_FN

db2gResetMonitor (
  db2Uint32 versionNumber,
  void *pParmStruct,
  struct sqlca *pSqlca);

typedef SQL_STRUCTURE db2gResetMonitorData
{
  db2Uint32                   iResetAll;
  char                        *piDbAlias;
  db2Uint32                   iDbAliasLength;
  db2Uint32                   iVersion;
  db2int32                    iNodeNumber;
} db2gResetMonitorData;
/* ... */
```

API パラメーター:

versionNumber
入力。 2 番目のパラメーター *pParmStruct* として渡される構造のバージョンとリリースのレベルを指定します。

pParmStruct
入力。 *db2ResetMonitorData* 構造を指すポインター。

pSqlca
出力。*sqlca* 構造へのポインター。

iResetAll
入力。リセット・フラグです。

piDbAlias
入力。データベースの別名を指すポインター。

iDbAliasLength
入力。 *piDbAlias* の長さ (バイト単位) を指定します。

iVersion
入力。収集するデータベース・モニター・データのバージョン ID です。データベース・モニターは、要求されたバージョンについて使用可能なデータのみを戻します。このパラメーターは、以下のいずれかのシンボリック定数に設定してください。

- SQLM_DBMON_VERSION1
- SQLM_DBMON_VERSION2
- SQLM_DBMON_VERSION5
- SQLM_DBMON_VERSION5_2
- SQLM_DBMON_VERSION6
- SQLM_DBMON_VERSION7
- SQLM_DBMON_VERSION8

注: バージョンとして SQLM_DBMON_VERSION1 が指定された場合、 API はリモートでは実行できません。

iNodeNumber
入力。要求の送信先となるデータベース・パーティション・サーバーを示します。この値に基づき、要求は現行のデータベース・パーティション・サーバー、すべてのデータベース・パーティション・サーバー、またはユーザーが指定したデータベース・パーティション・サーバーに対して処理されます。有効な値は以下のとおりです。

- SQLM_CURRENT_NODE
- SQLM_ALL_NODES
- *ノード値*

注: 独立型インスタンスの場合には、 SQLM_CURRENT_NODE を使用する必要があります。

使用上の注意:

各プロセス (アタッチ) には、モニター・データについての独自のプライベート・ビューがあります。あるユーザーがモニター・スイッチをリセットしたり、オフにしたりしても、他のユーザーはその影響を受けません。アプリケーションは、データベース・モニター関数を最初に呼び出すときに、データベース・マネージャー構成ファイルからデフォルトのスイッチ設定を継承します。これらの設定は、db2MonitorSwitches (モニター・スイッチの入手/更新) でオーバーライドできます。

すべてのアクティブ・データベースがリセットされる場合には、戻されるデータの整合性を保つために、一部のデータベース・マネージャー情報もリセットされます。

この API は、特定のデータ項目または特定のモニター・グループを選択的にリセットするためには使用できません。ただし、db2MonitorSwitches (モニター・スイッチの入手/更新) を使用して特定のグループのスイッチをいったんオフにしてからオンにすれば、そのグループをリセットすることができます。

関連資料:

- 89 ページの『db2GetSnapshot - スナップショットの入手』
- 211 ページの『db2MonitorSwitches - モニター・スイッチの入手/更新』
- 93 ページの『db2GetSnapshotSize - db2GetSnapshot 出力バッファーに必要なサイズの見積もり』
- 453 ページの『SQLCA』

db2Restore - データベースのリストア

db2Backup (データベースのバックアップ) を使用して、バックアップが取られていたデータベースが損傷または破壊された場合に、そのデータベースを再作成します。リストアされたデータベースは、バックアップ・コピーの作成時と同じ状態になります。このユーティリティーでは、(新しいデータベースへのリストアが可能であることに加えて) バックアップ・イメージ内のデータベース名とは異なる名前でデータベースをリストアすることが可能です。

このユーティリティーは、以前の 2 つのリリースで作成した DB2 データベースをリストアするためにも使用できます。

このユーティリティーは、表スペース・レベルのバックアップからリストアすることもできますし、データベース・バックアップ・イメージ内から表スペースをリストアすることもできます。

有効範囲:

この API は、それが呼び出されたデータベース・パーティションにのみ影響を与えます。

許可:

既存のデータベースにリストアするには、次のいずれかが必要です。

- *sysadm*
- *sysctrl*
- *sysmaint*

新規のデータベースにリストアするには、次のいずれかが必要です。

- *sysadm*
- *sysctrl*

必要な接続:

既存のデータベースにリストアする場合、データベース。この API を呼び出せば、指定したデータベースへの接続が自動的に確立され、リストア操作が終了すると接続が解放されます。

新規のデータベースにリストアする場合、インスタンスおよびデータベース。データベースを作成するには、インスタンス・アタッチメントが必要です。

現行のインスタンス (DB2INSTANCE 環境変数の値で定義) とは異なるインスタンスで新規のデータベースへのリストアを行うには、まず、新規のデータベースを存在させるインスタンスにアタッチすることが必要です。

API 組み込みファイル:

db2ApiDf.h

C API 構文:

```
/* File: db2ApiDf.h */
/* API: db2Restore */
/* ... */
SQL_API_RC SQL_API_FN
db2Restore (
  db2Uint32      versionNumber,
  void         *pDB2RestoreStruct,
  struct sqlca *pSqlca);
/* ... */

typedef SQL_STRUCTURE db2RestoreStruct
{
  char                        *piSourceDBAlias;
  char                        *piTargetDBAlias;
  char                        oApplicationId[SQLU_APPLID_LEN+1];
  char                        *piTimestamp;
  char                        *piTargetDBPath;
  char                        *piReportFile;
  struct db2TablespaceStruct  *piTablespaceList;
  struct db2MediaListStruct   *piMediaList;
  char                        *piUsername;
  char                        *piPassword;
  char                        *piNewLogPath;
  void                        *piVendorOptions;
  db2Uint32                   iVendorOptionsSize;
  db2Uint32                   iParallelism;
  db2Uint32                   iBufferSize;
  db2Uint32                   iNumBuffers;
  db2Uint32                   iCallerAction;
  db2Uint32                   iOptions;
  char                        *piComprLibrary;
  void                        *piComprOptions;
  db2Uint32                   iComprOptionsSize;
  char                        *piLogTarget;
} db2RestoreStruct;

typedef SQL_STRUCTURE db2TablespaceStruct
{
  char                        **tablespaces;
  db2Uint32                   numTablespaces;
} db2TablespaceStruct;

typedef SQL_STRUCTURE db2MediaListStruct
{
  char                        **locations;
```

```
  db2Uint32                   numLocations;
  char                        locationType;
} db2MediaListStruct;
/* ... */
```

汎用 API 構文:

```
/* File: db2ApiDf.h */
/* API: db2gRestore */
/* ... */
SQL_API_RC SQL_API_FN
db2gRestore (
  db2Uint32      versionNumber,
  void         *pDB2gRestoreStruct,
  struct sqlca *pSqlca);

typedef SQL_STRUCTURE db2gRestoreStruct
{
  char                        *piSourceDBAlias;
  db2Uint32                   iSourceDBAliasLen;
  char                        *piTargetDBAlias;
  db2Uint32                   iTargetDBAliasLen;
  char                        *poApplicationId;
  db2Uint32                   iApplicationIdLen;
  char                        *piTimestamp;
  db2Uint32                   iTimestampLen;
  char                        *piTargetDBPath;
  db2Uint32                   iTargetDBPathLen;
  char                        *piReportFile;
  db2Uint32                   iReportFileLen;
  struct db2gTablespaceStruct *piTablespaceList;
  struct db2gMediaListStruct  *piMediaList;
  char                        *piUsername;
  db2Uint32                   iUsernameLen;
  char                        *piPassword;
  db2Uint32                   iPasswordLen;
  char                        *piNewLogPath;
  db2Uint32                   iNewLogPathLen;
  void                        *piVendorOptions;
  db2Uint32                   iVendorOptionsSize;
  db2Uint32                   iParallelism;
  db2Uint32                   iBufferSize;
  db2Uint32                   iNumBuffers;
  db2Uint32                   iCallerAction;
  db2Uint32                   iOptions;
  char                        *piComprLibrary;
  db2Uint32                   iComprLibraryLen;
  void                        *piComprOptions;
  db2Uint32                   iComprOptionsSize;
  char                        *piLogTarget;
  db2Uint32                   iLogTargetLen;
} db2gRestoreStruct;

typedef SQL_STRUCTURE db2gTablespaceStruct
{
  struct db2Char              *tablespaces;
  db2Uint32                   numTablespaces;
} db2gTablespaceStruct;

typedef SQL_STRUCTURE db2gMediaListStruct
{
  struct db2Char              *locations;
  db2Uint32                   numLocations;
  char                        locationType;
} db2gMediaListStruct;

typedef SQL_STRUCTURE db2Char
```

```
{
   char            *pioData;
   db2Uint32       iLength;
   db2Uint32       oLength;
} db2Char;
/* ... */
```

API パラメーター:

versionNumber
入力。 2 番目のパラメーター *pDB2RestoreStruct* として渡される構造のバージョンとリリースのレベルを指定します。

pDB2RestoreStruct
入力。 *db2RestoreStruct* 構造を指すポインター。

pSqlca
出力。*sqlca* 構造へのポインター。

piSourceDBAlias
入力。ソース・データベース・バックアップ・イメージのデータベース別名を含むストリング。

iSourceDBAliasLen
入力。ソース・データベースの別名の長さを示す 4 バイトの符号なし整数 (バイト単位)。

piTargetDBAlias
入力。宛先データベースの別名を含むストリング。このパラメーターが NULL の場合、*piSourceDBAlias* が使用されます。

iTargetDBAliasLen
入力。宛先データベースの別名の長さを示す 4 バイトの符号なし整数 (バイト単位)。

oApplicationId
出力。 API によって、アプリケーションにサービスを提供しているエージェントを識別するストリングが戻されます。データベース・モニターを使用するバックアップ操作の進行状況に関する情報を取得することもできます。

poApplicationId
出力。長さ SQLU_APPLID_LEN+1 (`sqlutil` で定義) のバッファーを提供します。 API によって、アプリケーションにサービスを提供しているエージェントを識別するストリングが戻されます。データベース・モニターを使用するバックアップ操作の進行状況に関する情報を取得することもできます。

iApplicationIdLen
入力。 poApplicationId バッファーの長さを示す 4 バイトの符号なし整数 (バイト単位) です。 SQLU_APPLID_LEN+1 (`sqlutil` に定義) と等しくなければなりません。

piTimestamp
入力。バックアップ・イメージのタイム・スタンプを示すストリング。指定したソース内にバックアップ・イメージが 1 つしかない場合、このフィールドはオプションです。

iTimestampLen
入力。 piTimestamp バッファーの長さを示す 4 バイトの符号なし整数 (バイト単位)。

piTargetDBPath
入力。サーバー上の宛先データベースのディレクトリーの相対または完全修飾名を含むストリング。リストアされるバックアップのために新規のデータベースを作成する場合に使用します。そうでない場合は使用しません。

piReportFile
入力。ファイル名が指定されている場合、完全修飾名にしなければなりません。リストア時にリンク解除されるデータ・リンク・ファイル (高速調整の結果) が報告されます。

iReportFileLen
入力。 piReportFile バッファーの長さを示す 4 バイトの符号なし整数 (バイト単位) です。

piTablespaceList
入力。リストアされる表スペースのリストです。データベースまたは表スペース・バックアップ・イメージから、表スペースのサブセットをリストアする場合に使用されます。詳細については、*DB2TablespaceStruct* 構造を参照してください。以下の制限が適用されます。

- データベースはリカバリー可能でなければなりません。つまり、ログ保存またはユーザー出口が使用可能になっていなければなりません。
- リストアされるデータベースは、バックアップ・イメージの作成に使用されたのと同じデータベースでなければなりません。つまり、表スペース・リストア機能を使用して、データベースに表スペースを追加することはできません。
- ロールフォワード・ユーティリティーは、パーティション・データベース環境でリストアされる表スペースが、同じ表スペースを含む他のデータベース・パーティションと必ず同期化されるようにします。表スペースのリストア操作が要求され、*piTablespaceList* が NULL の場合、リストア・ユーティリティーはバックアップ・イメージ内のすべての表スペースのリストアを試みます。

バックアップ後に名前が変更された表スペースをリストアする場合、リストア・コマンドでは新しい表スペース名を使用しなければなりません。古い表スペース名を使用すると、その表スペースを見つけることができません。

piMediaList
入力。バックアップ・イメージのソース・メディア。提供される情報は、locationType フィールドの値によって異なります。 locationType に有効な値 (`sqlutil` で定義) は、以下のとおりです。

SQLU_LOCAL_MEDIA
ローカル装置 (テープ、ディスクまたはディスケットの組み合わせ)。

SQLU_XBSA_MEDIA
XBSA インターフェース。バックアップ・サービス API (XBSA) は、バックアップやアーカイブの目的でデータ・ストレージ管理を

必要とするアプリケーションまたは機能のための、オープン・アプリケーション・プログラミング・インターフェースです。

SQLU_TSM_MEDIA
TSM。ロケーション・ポインターが NULL に設定されている場合、 DB2 で提供される TSM 共用ライブラリーが使用されます。別のバージョンの TSM 共用ライブラリーが必要な場合には、SQLU_OTHER_MEDIA を使用し、共用ライブラリー名を入力してください。

SQLU_OTHER_MEDIA
ベンダー製品。ロケーション・フィールド内の共用ライブラリー名を提供します。

SQLU_USER_EXIT
ユーザー出口。追加の入力は必要ありません (サーバーが OS/2 上にある場合のみ使用可能です)。

piUsername
入力。接続の試行時に使用されるユーザー名を含むストリングを指定します。 NULL にすることもできます。

iUsernameLen
入力。 piUsername の長さを示す 4 バイトの符号なし整数 (バイト単位)。ユーザー名が提供されていない場合は、ゼロに設定してください。

piPassword
入力。ユーザー名とともに使用されるパスワードを含むストリングを指定します。 NULL にすることもできます。

iPasswordLen
入力。 piPassword の長さを示す 4 バイトの符号なし整数 (バイト単位)。パスワードが提供されていない場合は、ゼロに設定してください。

piNewLogPath
入力。リストア完了後、ロギングに使用されるパスを表すストリング。このフィールドが NULL の場合、デフォルトのログ・パスが使用されます。

iNewLogPathLen
入力。 *piNewLogPath* の長さを示す 4 バイトの符号なし整数 (バイト単位)。

piVendorOptions
入力。情報をアプリケーションからベンダー関数へ渡すのに使用されます。このデータ構造はフラットでなければなりません。つまり、間接のレベルはサポートされません。このデータについては、バイト反転が行われず、また、コード・ページがチェックされないことに注意してください。

iVendorOptionsSize
入力。 *piVendorOptions* の長さです。 65535 バイト以下でなければなりません。

iParallelism
入力。並列処理の度合い (バッファー・マニピュレーターの数) を指定します。最小値は 1 です。最大値は 1024 です。

iBufferSize
入力。バッファー・サイズを 4 KB の割り振り単位 (ページ) でバックアップします。最小値は 8 単位です。リストア用に入力するサイズは、バックアップ・イメージを作成するのに使用するバッファー・サイズと同じか、または整数倍でなければなりません。

iNumBuffers
入力。使用するリストア・バッファーの数を指定します。

iCallerAction
入力。実行するアクションを指定します。有効な値 (db2ApiDf で定義) は、以下のとおりです。

DB2RESTORE_RESTORE
リストア操作を開始します。

DB2RESTORE_NOINTERRUPT
リストアを開始します。リストアを自動実行するよう指定します。通常ユーザーの介入を要求するシナリオは、呼び出し側への最初の戻りなしに試行されるか、エラーを生成します。この呼び出し側アクションは、たとえば、リストアに必要なメディアがすべて装てんされていることが明らかで、ユーティリティーによるプロンプトが必要とされない場合に使用してください。

DB2RESTORE_CONTINUE
ユーザーがユーティリティーによって要求された何らかのアクション (たとえば、新しいテープの装てん) を実行した後で、リストアを継続します。

DB2RESTORE_TERMINATE
ユーザーがユーティリティーによって要求された何らかのアクションの実行に失敗した場合、リストアを終了します。

DB2RESTORE_DEVICE_TERMINATE
リストアによって使用される装置のリストから特定の装置を除外します。特定の装置が入力でいっぱいになると、リストア・ユーティリティーは呼び出し側に警告を戻します。その場合、この呼び出し側アクションを指定してリストアを再び呼び出すことによって、警告が生成される原因となった装置を使用装置のリストから除外してください。

DB2RESTORE_PARM_CHK
リストアを実行することなく、パラメーターの妥当性を検査します。このオプションは、呼び出しが戻った後でデータベース接続を終了しません。この呼び出しが正常に戻った後、ユーザーが DB2RESTORE_CONTINUE で呼び出しを発行し、処置を進めることが期待されます。

DB2RESTORE_PARM_CHK_ONLY
リストアを実行することなく、パラメーターの妥当性を検査します。この呼び出しが戻る前に、この呼び出しによって確立したデータベース接続は終了し、後続する呼び出しは必要なくなります。

DB2RESTORE_TERMINATE_INCRE
完了前に増分リストア操作を終了します。

DB2RESTORE_RESTORE_STORDEF
最初の呼び出し。表スペース・コンテナーの再定義が要求されます。

DB2RESTORE_STORDEF_NOINTERRUPT
最初の呼び出し。リストアは割り込まれずに実行されます。表スペース・コンテナーの再定義が要求されます。

iOptions
入力。リストア・プロパティーのビットマップ。オプションは組み合わされて、ビット単位 OR 演算子を使用して *iOptions* の値を生成します。有効な値 (db2ApiDf で定義) は、以下のとおりです。

DB2RESTORE_OFFLINE
オフライン・リストア操作を実行します。

DB2RESTORE_ONLINE
オンライン・リストア操作を実行します。

DB2RESTORE_DB
データベースにあるすべての表スペースをリストアします。これはオフラインで実行する必要があります。

DB2RESTORE_TABLESPACE
バックアップ・イメージから、*piTablespaceList* パラメーターにリストされた表スペースのみリストアします。これはオンラインまたはオフラインで実行できます。

DB2RESTORE_HISTORY
履歴ファイルだけをリストアします。

DB2RESTORE_COMPR_LIB
圧縮ライブラリーをリストアすることを指定します。このオプションは、他のリストア・タイプと同時に使用することはできません。バックアップ・イメージの中にオブジェクトが存在している場合、それはデータベース・ディレクトリーの中にリストアされます。バックアップ・イメージの中にオブジェクトが存在していない場合、リストア操作は失敗します。

DB2RESTORE_LOGS
バックアップ・イメージに含まれている一連のログ・ファイルだけをリストアすることを指定します。バックアップ・イメージの中にログ・ファイルが含まれていない場合、リストア操作は失敗します。このオプションを指定する場合は、*piLogTarget* パラメーターも指定する必要があります。

DB2RESTORE_INCREMENTAL
手動累積リストア操作を実行します。

DB2RESTORE_AUTOMATIC
自動累積 (増分) リストア操作を実行します。
DB2RESTORE_INCREMENTAL とともに指定しなければなりません。

DB2RESTORE_DATALINK
調整操作を実行します。定義された DATALINK 列を含む表では、RECOVERY YES オプションを指定する必要があります。

DB2RESTORE_NODATALINK
調整操作を実行しません。 DATALINK 列を持つ表は、DataLink_Roconcile_pending (DRP) 状態に入れられます。定義された DATALINK 列を含む表では、RECOVERY YES オプションを指定する必要があります。

DB2RESTORE_ROLLFWD
データベースのリストアが成功した後、データベースをロールフォワード・ペンディング状態にします。

DB2RESTORE_NOROLLFWD
データベースのリストアが成功した後、データベースをロールフォワード・ペンディング状態にしません。バックアップがオンラインで実行される場合、または表スペース・レベルのリストアの場合は、この値は指定できません。リストアが成功した後で、データベースがロールフォワード・ペンディング状態にある場合は、db2Rollforward (データベースのロールフォワード) を実行してからでなければ、データベースを使用できません。

piComprLibrary
入力。バックアップ・イメージが圧縮されている場合、バックアップ・イメージの解凍を実行するために使用する外部ライブラリーの名前を示します。この名前は、サーバー上の 1 個のファイルを参照する完全修飾パスでなければなりません。値が NULL ポインターであるか、空ストリングへのポインターである場合、 DB2 は、イメージに保管されたライブラリーを使用しようとします。バックアップが圧縮されていなかった場合、このパラメーターの値は無視されます。指定されたライブラリーが見つからない場合、リストアは失敗します。

piComprLibraryLen
入力。 piComprLibrary で指定したライブラリー名の長さを示す 4 バイトの符号なし整数 (バイト単位) です。ライブラリー名が提供されていない場合は、ゼロに設定してください。

piComprOptions
入力。バイナリー・データのうち、解凍ライブラリーの初期設定ルーチンに渡すブロックを記述します。 DB2 はこのストリングをクライアントからサーバーに直接渡すため、バイト反転やコード・ページ変換の問題がある場合は圧縮ライブラリーで処理する必要があります。データ・ブロックの最初の文字が '@' なら、データの残りの部分は、サーバー上に存在するファイルの名前を指定するものとして解釈されます。その場合 DB2 は、piComprOptions および iComprOptionsSize の内容をそのファイルの内容およびサイズで置き換え、そのようにして得られる新しい値を初期設定ルーチンに渡します。

iComprOptionsSize
入力。 piComprOptions として渡されるデータ・ブロックのサイズを表す 4 バイトの符号なし整数。 piComprOptions が NULL ポインターである場合に限り、iComprOptionsSize はゼロになります。

piLogTarget
入力。バックアップ・イメージからログ・ファイルを抽出する際のターゲット・ディレクトリーとして使用する、データベース・サーバー上の既存のディレクトリーの絶対パス。このパラメーターを指定する場合、バックアップ・イメージ内のログ・ファイルは、そのターゲット・ディレクトリー内に抽出されます。このパラメーターを指定しない場合、バックアップ・イメージ内のログ・ファイルは抽出されません。バックアップ・イメージからログ・ファイルだけを抽出する場合は、DB2RESTORE_LOGS パラメーターを使用してください。

iLogTargetLen
入力。 *piLogTarget* 内のパスの長さを示す 4 バイトの符号なし整数 (バイト単位) です。

tablespaces
バックアップを取る表スペースのリストを指すポインター。 C の場合、リストは NULL で終了するストリングです。一般的には、*db2Char* 構造のリストです。

numTablespaces
tablespaces パラメーター内の項目数。

locations
メディア・ロケーションのリストを指すポインター。 C の場合、リストは NULL で終了するストリングです。一般的には、db2Char 構造のリストです。

numLocations
locations パラメーター内の項目数。

locationType
メディア・タイプを示す文字。有効な値 (`sqlutil` で定義) は、以下のとおりです。

SQLU_LOCAL_MEDIA
ローカル装置 (テープ、ディスク、ディスケット、または名前付きパイプ)

SQLU_XBSA_MEDIA
XBSA インターフェース。

SQLU_TSM_MEDIA
Tivoli Storage Manager。

SQLU_OTHER_MEDIA
ベンダー・ライブラリー。

SQLU_USER_EXIT
ユーザー出口 (サーバーが OS/2 上にある場合のみ使用可能です)。

pioData
文字データ・バッファーを指すポインター。

iLength
入力。 *pioData* バッファーのサイズ

oLength
出力。将来の利用のために予約されています。

使用上の注意:

オフラインのリストアの場合、このユーティリティーは、排他モードでデータベースに接続します。リストアされるデータベースにアプリケーション (呼び出し側のアプリケーションを含む) がすでに接続している場合、このユーティリティーは失敗します。さらに、リストア・ユーティリティーが、リストアの実行に使用されており、アプリケーション (呼び出し側のアプリケーションを含む) が同じワークステーション上の任意のデータベースにすでに接続されている場合、要求は失敗します。接続が成功すると、 API はリストアが完了するまで他のアプリケーションを締め出します。

現行のデータベース構成ファイルは、それが使用不能でない限り、バックアップ・コピーによって置換されません。ファイルが置換される場合には、警告メッセージが戻されます。

db2Backup (データベースのバックアップ) を使用して、データベースまたは表スペースをバックアップしなければなりません。

呼び出し側アクションが DB2RESTORE_NOINTERRUPT の場合、リストアはアプリケーションにプロンプトを出すことなく継続されます。呼び出し側のアクションが DB2RESTORE_RESTORE で、ユーティリティーが既存のデータベースにリストアしようとしている場合、ユーティリティーは、何らかのユーザー介入を要求するメッセージとともに、アプリケーションに制御を戻します。ユーザーとの対話の処理が終了した後、後続の呼び出しにおいて処理が継続される (DB2RESTORE_CONTINUE) か、終了 (DB2RESTORE_TERMINATE) されるかを示す呼び出し側アクションを設定して、RESTORE DATABASE を再び呼び出します。このユーティリティーは処理を終了させ、 *sqlca* で SQLCODE を戻します。

終了時に装置をクローズするには、呼び出し側アクションを DB2RESTORE_DEVICE_TERMINATE に設定してください。たとえば、2 つのテープ装置を使用して 3 つのテープ・ボリュームからリストアを行う場合、テープの 1 つがリストアされると、アプリケーションは、API からテープの終わりを示す SQLCODE とともに制御を受け取ります。ここで、アプリケーションはユーザーに別のテープを装てんするよう要求しますが、テープが「もうない」ことをユーザーが示すと、メディア装置の終了を通知する呼び出し側アクション SQLUD_DEVICE_TERMINATE を指定して API に戻ります。これで、デバイス・ドライバーは終了しますが、リストアに関連する残りの装置は、リストア・セット内の全セグメントがリストアされるまで処理済みの入力を保持し続けます (バックアップ処理中に、リストア・セット内のセグメントの数が最後のメディア装置に置かれます)。この呼び出し側アクションは、テープ以外の装置 (ベンダーによってサポートされる装置) でも使用できます。

アプリケーションに戻る前にパラメーター・チェックを実行したい場合には、呼び出し側アクションを DB2RESTORE_PARM_CHK に設定してください。

宛先変更したリストアを実行する場合には、呼び出し側アクションを DB2RESTORE_RESTORE_STORDEF に設定し、 『sqlbstsc - 表スペース・コンテナーの設定』との組み合わせで使用してください。

データベース・リストアの重要な段階でシステム障害が発生した場合、正常なリストアが実行されるまで、ユーザーはデータベースに正常に接続することができません。接続を試みたときにエラー・メッセージが戻されることによって、この状態であることが検出されます。バックアップされたデータベースがロールフォワード・リカバリーに使用できるよう構成されておらず、さらにログ保存パラメーターとユーザー出口パラメーターのいずれかが使用可能になっている、使用可能な現行の構成ファイルがある場合、ユーザーは、リストアに続いて、データベースの新しいバックアップをとるか、またはこれらのパラメーターを使用不能にしてから、データベースに接続することが必要です。

リストアが失敗すると、リストアされたデータベースはドロップされませんが (既存のデータベース以外へのリストアの場合を除き)、使用不能になります。

バックアップ上の履歴ファイルをリストアするようにリストア・タイプで指定されている場合、それはデータベースの既存の履歴ファイル上にリストアされます。事実上、リストアされるバックアップの後に履歴ファイルに加えられた変更は、すべて消去されることになります。このことが望ましくない場合は、履歴ファイルを新規またはテスト・データベースにリストアさせることにより、実行された更新を破棄することなく、履歴ファイルの内容を表示できるようにしてください。

バックアップ操作時にデータベースのロールフォワード・リカバリーが使用可能だった場合には、 db2Restore の実行が成功した後に db2Rollforward を発行することによって、データベースを損傷または破壊の発生前の状態に戻すことができます。データベースがリカバリー可能な場合、リストアの完了後に、デフォルトでペンディング状態がロールフォワードされます。

データベース・バックアップ・イメージがオフラインで作成されており、呼び出し側がリストア後のデータベースの ロールフォワードを必要としない場合、リストア用に DB2RESTORE_NOROLLFWD オプションを使用できます。これにより、リストア後にデータベースがすぐに使用可能になります。バックアップ・イメージがオンラインで作成されている場合は、呼び出し元はリストアが完了した時点で、対応するログ・レコードを使用してロールフォワードを行わなければなりません。

| ログ・ファイルを含むバックアップ・イメージからログ・ファイルをリストアする
| 場合は、 LOGTARGET オプションを指定し、それに DB2 サーバー上に存在する
| 有効な完全修飾パス名を指定する必要があります。それらの条件が満たされている
| 場合、リストア・ユーティリティーは、イメージ内のログ・ファイルをターゲット・パスに書き込みます。ログを含まないバックアップ・イメージのリストアで
| LOGTARGET を指定した場合、リストア操作で表スペース・データのリストアが試
| 行される前にエラーが戻されます。また、LOGTARGET に無効なパスや読み取り専
| 用パスが指定された場合も、リストアからエラーが戻されます。

| リストア・コマンド発行時点に LOGTARGET パス内にログ・ファイルが存在して
| いる場合、ユーザーに対して警告プロンプトが戻されます。 WITHOUT
| PROMPTING が指定されている場合、この警告は戻されません。

LOGTARGET を指定したリストア時に、なんらかの理由で抽出できないログ・ファイルがあった場合、リストアは失敗し、エラーが戻されます。バックアップ・イメージから抽出するログ・ファイルの中に、 LOGTARGET パス内に既存のファイルと同じ名前のものが 1 つでもあると、リストア操作は失敗し、エラーが戻されます。リストア・ユーティリティーは、LOGTARGET ディレクトリー内に既存のログ・ファイルを上書きしません。

保管されているログ・セットだけをバックアップ・イメージからリストアすることも可能です。ログ・ファイルだけをリストアすることを指定するには、LOGTARGET パスに加えて LOGS オプションを指定します。 LOGTARGET パスを指定しないで LOGS オプションを指定すると、エラーになります。この操作モードでログ・ファイルをリストアしようとして問題が発生した場合、そのリストアは即座に終了し、エラーが戻されます。

自動増分リストア時には、リストア操作のターゲット・イメージに含まれているログだけがバックアップ・イメージから取り出されます。増分リストア処理中に参照される中間イメージに含まれるログが、それらの中間バックアップ・イメージから抽出されることはありません。手動増分リストア時には、LOGTARGET パスは、最終リストア・コマンドを発行する場合にのみ指定してください。

バックアップが圧縮されているなら、そのことは DB2 によって検出され、リストア前に自動的にデータが解凍されます。 db2Restore API でライブラリーが指定されている場合、データの解凍にはそれが使用されます。そうでない場合、バックアップ・イメージにライブラリーが格納されているなら、それが使用されます。そうでない場合、データは解凍できず、リストアは失敗します。

バックアップ・イメージから圧縮ライブラリーをリストアする場合 (リストアのタイプとして DB2RESTORE_COMPR_LIB を指定して明示的に、または圧縮バックアップの通常のリストアを実行することにより暗黙的に)、そのリストア操作は、バックアップが作成されたのと同じプラットフォームおよびオペレーティング・システム上で実行する必要があります。バックアップ作成時のプラットフォームとリストア操作実行時のプラットフォームが違っていると、それらの 2 つのシステムの間のクロスプラットフォーム・リストアが DB2 で通常にサポートされている場合でも、リストア操作は失敗します。

関連資料:

- 390 ページの『sqlemgdb - データベースの移行』
- 257 ページの『db2Rollforward - データベースのロールフォワード』
- 453 ページの『SQLCA』
- 28 ページの『db2Backup - データベースのバックアップ』
- 36 ページの『db2CfgGet - 構成パラメーターの入手』

関連サンプル:

- 『dbrecov.sqc -- How to recover a database (C)』
- 『dbrecov.sqC -- How to recover a database (C++)』

db2Rollforward - データベースのロールフォワード

データベース・ログ・ファイルに記録されたトランザクションを適用することによって、データベースをリカバリーします。この API は、データベースまたは表スペースのバックアップがリストアされた後、あるいはメディア・エラーが原因で表スペースがデータベースによってオフラインにされた場合に呼び出されます。ロールフォワード・リカバリーを用いてデータベースをリカバリーするには、前もってデータベースがリカバリー可能でなければなりません (すなわち、データベース構成パラメーター *logarchmeth1* が**オン**に設定されていなければなりません)。

有効範囲:

パーティション・データベース環境では、この API はカタログ・パーティションからのみ呼び出すことができます。データベースまたは表スペースのポイント・イン・タイムを指定したロールフォワード呼び出しは、 db2nodes.cfg ファイルにリストされているすべてのデータベース・パーティション・サーバーに影響を与えます。ログの終わりを指定したデータベースまたは表スペース・ロールフォワード呼び出しは、指定されたデータベース・パーティション・サーバーに影響を与えます。データベース・パーティション・サーバーが 1 つも指定されていない場合には、 db2nodes.cfg ファイルにリストされているすべてのデータベース・パーティション・サーバーに影響を与えます。特定のデータベース・パーティション・サーバーでロールフォワードが必要ない場合、そのデータベース・パーティション・サーバーは無視されます。

許可:

以下のいずれかが必要です。

- *sysadm*
- *sysctrl*
- *sysmaint*

必要な接続:

なし。この API によってデータベース接続が確立されます。

API 組み込みファイル:

db2ApiDf.h

C API 構文:

```
/* File: db2ApiDf.h */
/* API: db2Rollforward */
/* ... */
SQL_API_RC SQL_API_FN
db2Rollforward (
  db2Uint32 versionNumber,
  void *pDB2RollforwardStruct,
  struct sqlca *pSqlca);

typedef SQL_STRUCTURE db2RollforwardStruct
{
  struct db2RfwdInputStruct   *piRfwdInput;
  struct db2RfwdOutputStruct  *poRfwdOutput;
} db2RollforwardStruct;

typedef SQL_STRUCTURE db2RfwdInputStruct
```

```
{
  sqluint32                   iVersion;
  char                        *piDbAlias;
  db2Uint32                   iCallerAction;
  char                        *piStopTime;
  char                        *piUserName;
  char                        *piPassword;
  char                        *piOverflowLogPath;
  db2Uint32                   iNumChngLgOvrflw;
  struct sqlurf_newlogpath    *piChngLogOvrflw;
  db2Uint32                   iConnectMode;
  struct sqlu_tablespace_bkrst_list *piTablespaceList;
  db2int32                    iAllNodeFlag;
  db2int32                    iNumNodes;
  SQL_PDB_NODE_TYPE           *piNodeList;
  db2int32                    iNumNodeInfo;
  char                        *piDroppedTblID;
  char                        *piExportDir;
  db2Uint32                   iRollforwardFlags;
} db2RfwdInputStruct;

typedef SQL_STRUCTURE db2RfwdOutputStruct
{
  char                        *poApplicationId;
  sqlint32                    *poNumReplies;
  struct sqlurf_info          *poNodeInfo;
} db2RfwdOutputStruct;

typedef SQL_STRUCTURE sqlurf_newlogpath
{
  SQL_PDB_NODE_TYPE           nodenum;
   unsigned short             pathlen;
   char                       logpath[SQL_LOGPATH_SZ+SQL_LOGFILE_NAME_SZ+1];
} sqlurf_newlogpath;

typedef SQL_STRUCTURE sqlu_tablespace_bkrst_list
{
  long                         num_entry;
  struct sqlu_tablespace_entry *tablespace;
} sqlu_tablespace_bkrst_list;

typedef SQL_STRUCTURE sqlu_tablespace_entry
{
  sqluint32                   reserve_len;
  char                        tablespace_entry[SQLU_MAX_TBS_NAME_LEN+1];
  char                        filler[1];
} sqlu_tablespace_entry;

typedef SQL_STRUCTURE sqlurf_info
{
   SQL_PDB_NODE_TYPE nodenum;
   sqlint32        state;
   unsigned char   nextarclog[SQLUM_ARCHIVE_FILE_LEN+1];
   unsigned char   firstarcdel[SQLUM_ARCHIVE_FILE_LEN+1];
   unsigned char   lastarcdel[SQLUM_ARCHIVE_FILE_LEN+1];
   unsigned char   lastcommit[SQLUM_TIMESTAMP_LEN+1];
} sqlurf_info;
/* ... */
```

汎用 API 構文:

```
/* File: db2ApiDf.h */
/* API: db2Rollforward */
/* ... */
SQL_API_RC SQL_API_FN
db2gRollforward (
  db2Uint32 versionNumber,
```

```
  void *pDB2gRollforwardStruct,
  struct sqlca *pSqlca);

typedef SQL_STRUCTURE db2gRollforwardStruct
{
  struct db2gRfwdInputStruct  *piRfwdInput;
  struct db2RfwdOutputStruct  *poRfwdOutput;
} db2gRollforwardStruct;

SQL_STRUCTURE db2gRfwdInputStruct
{
  db2Uint32                   iDbAliasLen;
  db2Uint32                   iStopTimeLen;
  db2Uint32                   iUserNameLen;
  db2Uint32                   iPasswordLen;
  db2Uint32                   iOvrflwLogPathLen;
  db2Uint32                   iDroppedTblIDLen;
  db2Uint32                   iExportDirLen;
  sqluint32                   iVersion;
  char                        *piDbAlias;
  db2Uint32                   iCallerAction;
  char                        *piStopTime;
  char                        *piUserName;
  char                        *piPassword;
  char                        *piOverflowLogPath;
  db2Uint32                   iNumChngLgOvrflw;
  struct sqlurf_newlogpath    *piChngLogOvrflw;
  db2Uint32                   iConnectMode;
  struct sqlu_tablespace_bkrst_list *piTablespaceList;
  db2int32                    iAllNodeFlag;
  db2int32                    iNumNodes;
  SQL_PDB_NODE_TYPE           *piNodeList;
  db2int32                    iNumNodeInfo;
  char                        *piDroppedTblID;
  char                        *piExportDir;
  db2Uint32                   iRollforwardFlags;
} db2gRfwdInputStruct;

typedef SQL_STRUCTURE db2RfwdOutputStruct
{
  char                        *poApplicationId;
  sqlint32                    *poNumReplies;
  struct sqlurf_info          *poNodeInfo;
} db2RfwdOutputStruct;

typedef SQL_STRUCTURE sqlurf_newlogpath
{
  SQL_PDB_NODE_TYPE           nodenum;
   unsigned short             pathlen;
   char                       logpath[SQL_LOGPATH_SZ+SQL_LOGFILE_NAME_SZ+1];
} sqlurf_newlogpath;

typedef SQL_STRUCTURE sqlu_tablespace_bkrst_list
{
  long                         num_entry;
  struct sqlu_tablespace_entry *tablespace;
} sqlu_tablespace_bkrst_list;

typedef SQL_STRUCTURE sqlu_tablespace_entry
{
  sqluint32                   reserve_len;
  char                        tablespace_entry[SQLU_MAX_TBS_NAME_LEN+1];
  char                        filler[1];
} sqlu_tablespace_entry;

typedef SQL_STRUCTURE sqlurf_info
{
```

```
   SQL_PDB_NODE_TYPE nodenum;
   sqlint32        state;
   unsigned char   nextarclog[SQLUM_ARCHIVE_FILE_LEN+1];
   unsigned char   firstarcdel[SQLUM_ARCHIVE_FILE_LEN+1];
   unsigned char   lastarcdel[SQLUM_ARCHIVE_FILE_LEN+1];
   unsigned char   lastcommit[SQLUM_TIMESTAMP_LEN+1];
} sqlurf_info;
/* ... */
```

API パラメーター:

versionNumber
入力。 2 番目のパラメーターとして渡される構造のバージョンとリリースのレベルを指定します。

pDB2RollforwardStruct
入力。 *db2RollforwardStruct* 構造を指すポインター。

pSqlca
出力。*sqlca* 構造へのポインター。

piRfwdInput
入力。 *db2RfwdInputStruct* 構造を指すポインター。

poRfwdOutput
出力。 *db2RfwdOutputStruct* 構造を指すポインター。

iDbAliasLen
入力。データベース別名の長さ (バイト単位) を指定します。

iStopTimeLen
入力。停止時刻パラメーターの長さ (バイト単位) を指定します。停止時刻が提供されていない場合は、ゼロに設定してください。

iUserNameLen
入力。ユーザー名の長さ (バイト単位) を指定します。ユーザー名が提供されていない場合は、ゼロに設定してください。

iPasswordLen
入力。パスワードの長さ (バイト単位) を指定します。パスワードが提供されていない場合は、ゼロに設定してください。

iOverflowLogPathLen
入力。オーバーフロー・ログ・パスの長さ (バイト単位) を指定します。オーバーフロー・ログ・パスが提供されていない場合は、ゼロに設定してください。

iVersion
入力。ロールフォワード・パラメーターのバージョン ID。
SQLUM_RFWD_VERSION として定義されています。

piDbAlias
入力。データベース別名を含むストリング。これは、システム・データベース・ディレクトリーにカタログされている別名です。

iCallerAction
入力。実行するアクションを指定します。有効な値 (db2ApiDf.h で定義) は、以下のとおりです。

DB2ROLLFORWARD_ROLLFWD
piStopTime で指定された時点までロールフォワードします。データベースのロールフォワードの場合、データベースは*ロールフォワード・ペンディング* 状態のままになります。表スペースのロールフォワードの場合、表スペースは*ロールフォワード処理* 状態のままになります。

DB2ROLLFORWARD_STOP
ロールフォワード・リカバリーを終了します。新しいログ・レコードは処理されず、コミットされていないトランザクションはバックアウトされます。データベースまたは表スペースの*ロールフォワード・ペンディング* 状態はオフになります。同義語は DB2ROLLFORWARD_RFWD_COMPLETE です。

DB2ROLLFORWARD_RFWD_STOP
piStopTime で指定された時点までロールフォワードし、ロールフォワード・リカバリーを終了します。データベースまたは表スペースの*ロールフォワード・ペンディング* 状態はオフになります。同義語は DB2ROLLFORWARD_RFWD_COMPLETE です。

DB2ROLLFORWARD_QUERY
nextarclog、*firstarcdel*、 *lastarcdel*、および *lastcommit* の照会値。データベース状況とノード番号を戻します。

DB2ROLLFORWARD_PARM_CHECK
ロールフォワードを実行することなく、パラメーターの妥当性を検査します。

DB2ROLLFORWARD_CANCEL
現在実行中のロールフォワード操作を取り消します。データベースまたは表スペースは、リカバリー・ペンディング状態に置かれます。

注: このオプションは、ロールフォワードが実際に実行中であるときには使用できません。ロールフォワードが休止されている (つまり、STOP を待っている) 場合、あるいはロールフォワード中にシステム障害が発生した場合に使用できます。このオプションの使用に際しては、注意が必要です。

データベースのロールフォワードには、テープ装置を使用したロード・リカバリーが必要とされる場合があります。装置に関してユーザーの介入が必要な場合、ロールフォワード API は警告メッセージを戻します。以下の 3 つのアクションのいずれかを指定して、再び API を呼び出すことができます。

DB2ROLLFORWARD_LOADREC_CONT
警告メッセージを生成した装置の使用を続けます (たとえば、新しいテープをマウントしたときなど)。

DB2ROLLFORWARD_DEVICE_TERM
警告メッセージを生成した装置の使用を停止します (たとえば、それ以上テープがない場合)。

DB2ROLLFORWARD_LOAD_REC_TERM
ロード・リカバリーに使用されているすべての装置を終了させます。

piStopTime
入力。 ISO 形式のタイム・スタンプを含む文字ストリング。このタイム・スタンプで設定された時刻を過ぎると、データベース・リカバリーは停止します。可能な限りロールフォワードしたい場合には、SQLUM_INFINITY_TIMESTAMP を指定してください。DB2ROLLFORWARD_QUERY、DB2ROLLFORWARD_PARM_CHECK、およびいずれかのロード・リカバリー (DB2ROLLFORWARD_LOADREC_*xxx*) 呼び出し側アクションの場合は、NULL にすることができます。

piUserName
入力。アプリケーションのユーザー名を含むストリング。 NULL にすることもできます。

piPassword
入力。提供されたユーザー名 (ある場合) のパスワードを含むストリング。 NULL にすることもできます。

piOverflowLogPath
入力。使用される代替ログ・パスを指定します。このユーティリティーを使用する前に、アクティブ・ログ・ファイルの他に、アーカイブ・ログ・ファイルをユーザーが *logpath* に移動させることが必要です。このことは、*logpath* に十分なスペースがない場合に問題になる可能性があります。その問題を解決するために、オーバーフロー・ログ・パスが備えられています。ロールフォワード・リカバリー中に、必要なログ・ファイルは、まず *logpath* で探索され、次にオーバーフロー・ログ・パスで探索されます。表スペースのロールフォワード・リカバリーに必要なログ・ファイルは、*logpath* またはオーバーフロー・ログ・パスのいずれかに置かれる可能性があります。呼び出し側がオーバーフロー・ログ・パスを指定しない場合、デフォルト値は *logpath* です。パーティション・データベース環境では、オーバーフロー・ログ・パスは有効な完全修飾パスでなければなりません。デフォルトのパスは、各ノードのデフォルトのオーバーフロー・ログ・パスです。単一パーティション・データベース環境では、サーバーがローカルであれば、オーバーフロー・ログ・パスは相対パスにすることもできます。

iNumChngLgOvrflw
入力。パーティション・データベース環境のみ。変更されるオーバーフロー・ログ・パスの数。新しいログ・パスは、指定されたデータベース・パーティション・サーバーのデフォルトのオーバーフロー・ログ・パスだけをオーバーライドします。

piChngLogOvrflw
入力。パーティション・データベース環境のみ。変更されるオーバーフロー・ログ・パスの完全修飾名が入っている構造を指すポインター。新しいログ・パスは、指定されたデータベース・パーティション・サーバーのデフォルトのオーバーフロー・ログ・パスだけをオーバーライドします。

iConnectMode
入力。有効な値 (db2ApiDf.h で定義) は、以下のとおりです。

DB2ROLLFORWARD_OFFLINE
オフライン・ロールフォワード。データベースのロールフォワード・リカバリーの場合には、必ずこの値を指定してください。

DB2ROLLFORWARD_ONLINE
オンライン・ロールフォワード。

piTablespaceList
入力。ログの終わりまで、または指定された時点までロールフォワードされる表スペースの名前が入っている構造を指すポインター。指定されない場合には、ロールフォワードを必要とする表スペースが選択されます。

iAllNodeFlag
入力。パーティション・データベース環境のみ。ロールフォワード操作が、db2nodes.cfg で定義されているすべてのデータベース・パーティション・サーバーに適用されるかどうかを示します。有効な値は以下のとおりです。

DB2_NODE_LIST
piNodeList で渡されたリスト内のデータベース・パーティション・サーバーに適用されます。

DB2_ALL_NODES
すべてのデータベース・パーティション・サーバーに適用されます。 *piNodeList* は NULL でなければなりません。これはデフォルト値です。

DB2_ALL_EXCEPT
piNodeList で渡されたリスト内のデータベース・パーティション・サーバーを除いた、すべてのデータベース・パーティション・サーバーに適用されます。

DB2_CAT_NODE_ONLY
カタログ・パーティションにのみ適用されます。 *piNodeList* は NULL でなければなりません。

iNumNodes
入力。 *piNodeList* 配列内のデータベース・パーティション・サーバーの数を指定します。

piNodeList
入力。ロールフォワード操作を実行する対象のデータベース・パーティション・サーバー番号の配列を指すポインター。

iNumNodeInfo
入力。出力パラメーター *poNodeInfo* のサイズを定義します。これは、ロールフォワードされるそれぞれのデータベース・パーティションからの状況情報を保持するのに十分な大きさでなければなりません。単一パーティション・データベース環境では、このパラメーターは 1 に設定しなければなりません。このパラメーターの値は、この API が呼び出されるデータベース・パーティション・サーバーの数と同じにする必要があります。

piDroppedTblID
入力。リカバリーが実行されているドロップ済み表の ID を含むストリング。

piExportDir
入力。ドロップした表データのエクスポート先のディレクトリー。

RollforwardFlags
入力。ロールフォワード・フラグを指定します。有効な値 (db2ApiDf.h で定義) は、以下のとおりです。

DB2ROLLFORWARD_EMPTY_FLAG
フラグが指定されていません。

DB2ROLLFORWARD_LOCAL_TIME
ユーザーが GMT 時間ではなくユーザーの現地時間を使用して特定の時点にロールフォワードすることを可能にします。これによって、ユーザーがローカル・マシンで特定の時点にロールフォワードすることが容易になり、現地時間を GMT ポイント・イン・タイムに変換することによって生じる潜在的なユーザー・エラーの発生を除去します。

DB2ROLLFORWARD_NO_RETRIEVE
ユーザーがアーカイブ・ログの検索を使用不可にすることを許可することによって、待機マシン上でどのログ・ファイルがロールフォワードされるべきかを制御します。ログ・ファイルのロールフォワードを制御することによって、待機マシンが実動マシンより X 時間遅れていることを確認でき、ユーザーが両システムに影響を与えることを防ぐことができます。待機システムがアーカイブにアクセスせず、たとえば TSM がアーカイブの場合に、元のマシンがファイルを検索することだけを許可する場合に、このオプションは役立ちます。また、実動システムがファイルをアーカイブし、待機システムが同じファイルを検索している間、待機システムが不完全なログ・ファイルを検索するという可能性も除去します。

poApplicationId
出力。アプリケーション ID。

poNumReplies
出力。受信した応答の数。

poNodeInfo
出力。データベース・パーティション応答情報。

nodenum
ノード番号。

pathlen
新規 logpath の長さ。

logpath
新規オーバーフロー・ログ・パス。

num_entry
tablespace フィールドによって示されたリスト内の項目数。

tablespace
sqlu_tablepsace_entry 構造を指すポインター。

reserve_len
tablespace_entry フィールドに指定された文字ストリングの長さ。 C 言語以外の場合です。

tablespace_entry
表スペース名。

state 状態情報。

nextarclog
次に必要とされるアーカイブ・ログ・ファイルの戻された名前を保管するバッファー。 DB2ROLLFORWARD_QUERY 以外の呼び出し側アクションが指定された場合、このフィールドに戻される値は、ファイルのアクセス時にエラーが発生したことを示すものです。考えられる原因は、次のとおりです。

- ファイルが、データベース・ログ・ディレクトリー内に見つからなかった、またはオーバーフロー・ログ・パス・パラメーターで指定したパスになかった。
- ログ・アーカイブ方式が、アーカイブ・ファイルを戻すことに失敗した。

firstarcdel
リカバリーに必要ではなくなった最初のアーカイブ・ログ・ファイルの戻された名前を保管するバッファー。このファイルおよび lastarcdel (を含む) までのすべてのファイルは、ディスクを空けるために移動することができます。

たとえば、firstarcdel および lastarcdel に戻される値が S0000001.LOG および S0000005.LOG である場合、以下のログ・ファイルを移動できます。

- `S0000001.LOG`
- `S0000002.LOG`
- `S0000003.LOG`
- `S0000004.LOG`
- `S0000005.LOG`

lastarcdel
データベース・ログ・ディレクトリーから除去できる最新のアーカイブ・ログ・ファイルの戻された名前を保管するバッファー。

lastcommit
ISO 形式のタイム・スタンプを含むストリング。この値は、ロールフォワード操作の終了後に、最後にコミットされたトランザクションのタイム・スタンプを表します。

使用上の注意:

データベース・マネージャーは、アーカイブおよびログ・ファイルに格納された情報を使用して、最後のバックアップのときにデータベースで実行されたトランザクションを再構築します。

この API の呼び出し時に実行されるアクションは、呼び出し前のデータベースの *rollforward_pending* フラグによって異なります。これは db2CfgGet (構成パラメーターの入手) を使用して照会できます。データベースがロールフォワード・ペンディ

ング状態にある場合、 *rollforward_pending* フラグは DATABASE に設定されています。 1 つまたは複数の表スペースが SQLB_ROLLFORWARD_PENDING または SQLB_ROLLFORWARD_IN_PROGRESS 状態にある場合、フラグは TABLESPACE に設定されています。データベースも表スペースもロールフォワードする必要がない場合は、 *rollforward_pending* フラグが NO に設定されています。

この API の呼び出し時にデータベースがロールフォワード・ペンディング状態にある場合は、データベースがロールフォワードされます。表スペースは、異常状態によって 1 つまたは複数の表スペースがオフラインにならない限り、データベースのロールフォワードが正常に終了すると正常の状態に戻ります。 *rollforward_pending* フラグが TABLESPACE に設定されている場合には、ロールフォワード・ペンディング状態にある表スペース、あるいは名前によって要求された表スペースだけがロールフォワードされます。

注: 表スペースのロールフォワードが異常終了してしまった場合、ロールフォワード中だった表スペースは、 SQLB_ROLLFORWARD_IN_PROGRESS 状態に置かれます。次に ROLLFORWARD DATABASE を呼び出したときには、 SQLB_ROLLFORWARD_IN_PROGRESS 状態にあるそれらの表スペースだけが処理されます。選択された表スペース名のセットに SQLB_ROLLFORWARD_IN_PROGRESS 状態のすべての表スペースが含まれているのではない場合には、要求されていない表スペースが SQLB_RESTORE_PENDING 状態に置かれます。

データベースがロールフォワード・ペンディング状態になく、特定の時点が指定されない場合には、ロールフォワード進行中状態にある表スペースがログの終わりまでロールフォワードされます。ロールフォワード進行中状態の表スペースがない場合には、ロールフォワード・ペンディング状態にある表スペースがログの終わりまでロールフォワードされます。

この API は、ログ・ファイルの読み取りを、バックアップ・イメージに一致するログ・ファイルから始めます。ログ・ファイルをロールフォワードする前に、 DB2ROLLFORWARD_QUERY 呼び出し側アクションを指定してこの API を呼び出すと、このログ・ファイルの名前を判別することができます。

ログ・ファイル内のトランザクションは、データベースに再適用されます。ログは、情報が使用可能である限り、あるいは停止時刻パラメーターで指定された時刻まで、順方向に処理されます。

以下のイベントが生じると、リカバリーが停止します。

- ログ・ファイルがこれ以上見つからない
- ログ・ファイル内のタイム・スタンプが、停止時刻パラメーターで指定された完了タイム・スタンプを超えた。
- ログ・ファイルの読み取り中に、エラーが発生した。

一部のトランザクションは、リカバリーされない可能性があります。 *lascommit* で戻された値は、最後にコミットされ、データベースに適用されたトランザクションのタイム・スタンプを示します。

アプリケーションまたは人為エラーが原因でデータベースのリカバリーが必要となった場合、エラーが発生する前の時点でリカバリーを停止することを指示するために、 *piStopTime* にタイム・スタンプ値を指定することができます。これは、データ

ベースの全ロールフォワード・リカバリーと、表スペースの特定の時点までのロールフォワードに適用されます。また、このことにより、前回失敗したリカバリーの試みで判別された、ログ読み取りエラーが発生する前の時点でリカバリーを停止させることも可能になります。

rollforward_recovery フラグが DATABASE に設定されている場合、ロールフォワード・リカバリーが終了するまで、データベースは使用できません。DB2ROLLFORWARD_STOP または DB2ROLLFORWARD_RFWRD_STOP の呼び出し側アクションを指定してこの API を呼び出すことにより、データベースのロールフォワード・ペンディング状態をオフにすれば、リカバリーを終了させることができます。*rollforward_recovery* フラグが TABLESPACE になれば、データベースを使用できるようになります。ただし、SQLB_ROLLFORWARD_PENDING および SQLB_ROLLFORWARD_IN_PROGRESS 状態の表スペースは、表スペースのロールフォワード・リカバリーを実行するための API が呼び出されるまで使用不能になります。表スペースをある時点までロールフォワードすると、表スペースは、正常なロールフォワード後にバックアップ・ペンディング状態に置かれます。

RollforwardFlags オプションが DB2ROLLFORWARD_LOCAL_TIME に設定されている場合、ユーザーに戻されるすべてのメッセージは現地時間で示されます。パーティション・データベース環境の場合、すべての時間はサーバー、またはカタログ・パーティション上で変換されます。タイム・スタンプ・ストリングはサーバー上で GMT に変換されるので、時間はクライアントではなくサーバーの時間帯に基づく現地時間となります。クライアントの時間帯とサーバーの時間帯とが異なる場合、サーバーの現地時間を使用してください。これはコントロール・センターの現地時間オプションとは異なります。そのオプションは、クライアントの現地時間を使用します。タイム・スタンプ・ストリングが夏時間調整のための時間変更に接近している場合、停止時刻が時間変更の前か後かを判別して、それを適切に指定することが大切です。

関連資料:

- 453 ページの『SQLCA』
- 244 ページの『db2Restore - データベースのリストア』

関連サンプル:

- 『dbrecov.sqc -- How to recover a database (C)』
- 『dbrecov.sqC -- How to recover a database (C++)』

db2Runstats - 統計の実行

表または関連する索引の両方または一方の特性についての統計を更新します。これらの特性には、レコード数、ページ数、レコードの平均長などがあります。オプティマイザーは、データへのアクセス・パスを判別するとき、これらの統計を使用します。

表に数多くの更新が加えられたときや、表を再編成した後、あるいは新規の索引を作成した後などに、このユーティリティーを呼び出してください。

統計は、API が実行されるデータベース・パーティションに存在する表パーティションに基づいて収集されます。グローバル表統計は、あるデータベース・パーティ

ションで取得された値に、表が完全に保管されているデータベース・パーティションの数を掛けることによって導出されます。グローバル統計は、カタログ表に保管されます。

API が呼び出されるデータベース・パーティションは、表のパーティションを含んでいる必要はありません。

- 表のパーティションを含むデータベース・パーティションから API が呼び出されると、ユーティリティーはこのデータベース・パーティションで実行されます。
- 表パーティションを含まないデータベース・パーティションから API が呼び出されると、要求は、表パーティションを保持しているデータベース・パーティション・グループ内の最初のデータベース・パーティションに送られます。その後、このデータベース・パーティションでユーティリティーが実行されます。

有効範囲:

この API は、db2nodes.cfg ファイル内の任意のデータベース・パーティション・サーバーから呼び出すことができます。この API は、カタログ・データベース・パーティション上のカタログを更新するために使用できます。

許可:

以下のいずれかです。

- *sysadm*
- *sysctrl*
- *sysmaint*
- 表に対する CONTROL 特権
- LOAD

必要な接続:

データベース

API 組み込みファイル:

db2ApiDf.h

C API 構文:

```
/* File: db2ApiDf.h */
/* API: db2Runstats */
/* ... */
SQL_API_RC SQL_API_FN
   db2Runstats (
    db2Uint32 versionNumber,
    db2RunstatsData *data,
    struct sqlca *sqlca);

typedef SQL_STRUCTURE db2RunstatsData
{
  double                      iSamplingOption;
  unsigned char               *piTablename;
  db2ColumnData               **piColumnList;
  db2ColumnDistData           **piColumnDistributionList;
  db2ColumnGrpData            **piColumnGroupList;
  unsigned char               **piIndexList;
  db2Uint32                   iRunstatsFlags;
  db2int16                    iNumColumns;
  db2int16                    iNumColdist;
  db2int16                    iNumColGroups;
```

```
  db2int16                    iNumIndexes;
  db2int16                    iParallelismOption;
  db2int16                    iTableDefaultFreqValues;
  db2int16                    iTableDefaultQuantiles;
  db2Uint32                   iUtilImpactPriority;
  db2Uint32                   iSamplingRepeatable;
} db2RunstatsData;

typedef SQL_STRUCTURE db2ColumnData
{
  unsigned char               *piColumnName;
  db2int16                    iColumnFlags;
} db2ColumnData;

typedef SQL_STRUCTURE db2ColumnDistData
{
  unsigned char               *piColumnName;
  db2int16                    iNumFreqValues;
  db2int16                    iNumQuantiles;
} db2ColumnDistData;

typedef SQL_STRUCTURE db2ColumnGrpData
{
  unsigned char               **piGroupColumnNames;
  db2int16                    iGroupSize;
  db2int16                    iNumFreqValues;
  db2int16                    iNumQuantiles;
} db2ColumnGrpData;
/* ... */
```

汎用 API 構文:

```
/* File: db2ApiDf.h */
/* API: db2gRunstats */
/* ... */
SQL_API_RC SQL_API_FN
   db2gRunstats (
    db2Uint32 versionNumber,
    db2gRunstatsData *data,
    struct sqlca *sqlca);

typedef SQL_STRUCTURE db2gRunstatsData
{
  double                      iSamplingOption;
  unsigned char               *piTablename;
  db2gColumnData              **piColumnList;
  db2gColumnDistData          **piColumnDistributionList;
  db2gColumnGrpData           **piColumnGroupList;
  unsigned char               **piIndexList;
  db2Uint16                   *piIndexNamesLen;
  db2Uint32                   iRunstatsFlags;
  db2Uint16                   iTablenameLen;
  db2int16                    iNumColumns;
  db2int16                    iNumColdist;
  db2int16                    iNumColGroups;
  db2int16                    iNumIndexes;
  db2int16                    iParallelismOption;
  db2int16                    iTableDefaultFreqValues;
  db2int16                    iTableDefaultQuantiles;
  db2Uint32                   iSamplingRepeatable;
  db2Uint32                   iUtilImpactPriority;
  db2Uint32                   iSamplingRepeatable;
} db2gRunstatsData;

typedef SQL_STRUCTURE db2gColumnData
{
  unsigned char               *piColumnName;
```

```
  db2Uint16                   iColumnNameLen;
  db2int16                    iColumnFlags;
} db2gColumnData;

typedef SQL_STRUCTURE db2gColumnDistData
{
  unsigned char               *piColumnName;
  db2Uint16                   iColumnNameLen;
  db2int16                    iNumFreqValues;
  db2int16                    iNumQuantiles;
} db2gColumnDistData;

typedef SQL_STRUCTURE db2gColumnGrpData
{
  unsigned char               **piGroupColumnNames;
  db2Uint16                   *piGroupColumnNamesLen;
  db2int16                    iGroupSize;
  db2int16                    iNumFreqValues;
  db2int16                    iNumQuantiles;
} db2gColumnGrpData;
/* ... */
```

API パラメーター:

versionNumber
入力。 2 番目のパラメーター *data* として渡される構造のバージョンとリリースのレベルを指定します。

data 入力。 *db2RunstatsData* 構造を指すポインター。

sqlca
出力。 *sqlca* 構造を指すポインター。

iSamplingOption
入力。表データのサンプルに対する統計を収集することを示します。 *iSamplingOption* はサンプルのサイズをパーセンテージ P で表します。この値は、100 以下の正数でなければなりません (1 と 0 の間の値でも差し支えありません)。たとえば、値 0.01 は 1 % 100 分の 1 を表し、平均で 10 000 行につき 1 行がサンプルに含まれることを意味します。 DB2 では、値 0 または 100 は、 DB2RUNSTATS_SAMPLING_SYSTEM が指定されたかどうかにかかわらず、表のサンプリングが指定されていないものとして扱われます。 100 より大きい値、または 0 より小さい値は、 DB2 ではエラー (SQL1197N) として扱われます。サンプリング・タイプとして、BERNOULLI および SYSTEM が可能です。サンプリング・タイプの指定は、 *iRunstatsFlags* の DB2RUNSTATS_SAMPLING_SYSTEM の設定によって制御されます。

piTablename
入力。統計が収集される表の完全修飾名を指すポインター。名前は別名にすることができます。行タイプの場合、*piTablename* は階層のルート表の名前でなければなりません。

piColumnList
入力。 *db2ColumnData* エレメントの配列。この配列の各エレメントは、以下の 2 つのサブエレメントで構成されています。

- 統計が収集される列の名前を表すストリング。
- 列の統計オプションを示すフラグ・フィールド。

iNumColumns がゼロの場合、*piColumnList* は無視されます (提供されている場合)。

piColumnDistributionList

入力。 *db2ColumnDistData* エレメントの配列。これらのエレメントは、特定の列 (複数の場合もある) の分散統計の収集が必要な場合に提供されます。この配列の各エレメントは、以下の 3 つのサブエレメントで構成されています。

- 分散統計が収集される列の名前を表すストリング。
- 収集する頻度 (数値)。
- 収集する変位値の数。

piColumnList に現れない *piColumnDistributionList* 内に現れる列には、列に収集された基本的な列統計があります。これは、最初から *piColumnList* 内にこれらの列が組み込まれているのと同じ効果があります。 *iNumColdist* がゼロの場合、*piColumnDistributionList* は無視されます。

piColumnGroupList

入力。 *db2ColumnGrpData* エレメントの配列。これらのエレメントは、列のグループで列統計を収集する場合に提供されます。つまり、各行に対するグループの各列の値は連結され、単一の値として処理されます。
各 *db2ColumnGrpData* は 3 つの整数フィールドとストリングの配列で構成されます。最初の整数フィールドは、ストリング *piGroupColumns* の配列内のストリング数を表します。この配列内の各ストリングには、1 つの列名が入っています。たとえば、列の組み合わせ統計が列グループ (c1、c2) および (c3、c4、c5) で収集される場合、 *piGroupColumns* には 2 つの *db2ColumnGrpData* エレメントが存在します。

最初の *db2ColumnGrpData* エレメントでは、 *piGroupSize* = 2 およびストリングの配列には、2 つのエレメントとして c1 と c2 が入っています。

2 番目の *db2ColumnGrpData* エレメントでは、 *piGroupSize* = 3 およびストリングの配列には、3 つのエレメントとして c3、c4、および c5 が入っています。

2 番目および 3 番目の整数フィールドは、列グループに関して分散統計を収集するときの、頻度および変位値の数値をそれぞれ表します。これは、現在サポートされていません。

piColumnList に現れない *piColumnGroupList* 内に現れる列には、列に収集された基本的な列統計があります。これは、最初から *piColumnList* 内にこれらの列が組み込まれているのと同じ効果があります。 *iNumColGroups* がゼロの場合、*piColumnGroupList* は無視されます。

piIndexList

入力。ストリングの配列。それぞれのストリングには、完全修飾された索引名が 1 つ含まれます。 *NumIndexes* がゼロの場合、 *piIndexList* は無視されます。

piIndexNamesLen

入力。索引リストにある索引名のそれぞれの長さを示す値の配列 (バイト単位)。 NumIndexes がゼロの場合、*piIndexNamesLen* は無視されます。

iRunstatsFlags

入力。統計オプションを指定するために使用されるビット・マスク・フィールド。有効な値は以下のとおりです。

DB2RUNSTATS_ALL_COLUMNS

表のすべての列で統計を収集します。このオプションは、列、列分散、列グループ、または索引構造リストの組み合わせで指定することができます。これは、表のすべての列で統計を収集したいが、特定の列に対して統計オプションを提供したい場合に役立ちます。

DB2RUNSTATS_KEY_COLUMNS

表で定義されたすべての索引を構成する列でのみ統計を収集します。このオプションは、列、列分散、列グループ、または索引構造リストの組み合わせで指定することができます。これは、表のすべてのキー列で統計を収集したいが、非キー列でも統計を収集したい場合、または特定のキー列の統計オプションを提供したい場合に役立ちます。

DB2RUNSTATS_DISTRIBUTION

分散統計を収集します。このオプションは、
DB2RUNSTATS_ALL_COLUMNS および
DB2RUNSTATS_KEY_COLUMNS とともにのみ使用できます。
DB2RUNSTATS_ALL_COLUMNS とともに使用される場合、分散統計は表のすべての列に対して収集されます。
DB2RUNSTATS_KEY_COLUMNS とともに使用される場合、分散統計は表に定義されたすべての索引を構成するすべての列に対して収集されます。 DB2RUNSTATS_ALL_COLUMNS および
DB2RUNSTATS_KEY_COLUMNS で使用される場合、基本統計は表のすべて列について収集され、分散統計は表で定義されたすべての索引を構成する列についてのみ収集されます。

DB2RUNSTATS_ALL_INDEXES

表で定義されたすべての索引で統計を収集します。

DB2RUNSTATS_EXT_INDEX

詳細な索引統計を収集します。このオプションは、
DB2RUNSTATS_ALL_INDEXES または索引名 (*piIndexList* および *iNumIndexes* > 0) の明示的なリストとともに指定する必要があります。

DB2RUNSTATS_EXT_INDEX_SAMPLED

抽出方式を使用して詳細な索引統計を収集します。このオプションは、DB2RUNSTATS_ALL_INDEXES または索引名 (*piIndexList* および *iNumIndexes* > 0) の明示的なリストとともに指定する必要があります。 DB2RUNSTATS_EXT_INDEX は、同時に指定された場合は無視されます。

DB2RUNSTATS_ALLOW_READ

統計の収集中に、他のユーザーが読み取り専用アクセスを行えるようにします。デフォルトでは、読み取りアクセスおよび書き込みアクセスが許可されます。

DB2RUNSTATS_SAMPLING_SYSTEM
データ・ページのうち、 *iSamplingOption* パラメーターを使ってユーザーが指定したパーセンテージの関する統計を収集します。SYSTEM サンプリングの場合、各ページを個別に扱い、ページは確率 P/100 (P は *iSamplingOption* の値) で含まれ、確率 1-P/100 で除外されます。したがって、*iSamplingOption* の値が 10 (つまり 10 % のサンプル) であれば、各ページは 0.1 の確率で含められ、0.9 の確率で除外されます。

DB2RUNSTATS_SAMPLING_SYSTEM が指定されない場合、DB2 は、サンプリング方式として BERNOULLI サンプリングを使用することを想定します。 BERNOULLI サンプリングの場合、各行を個別に扱い、行は確率 P/100 (P は *iSamplingOption* の値) で含まれ、確率 1-P/100 で除外されます。

SYSTEM サンプリングと BERNOULLI サンプリングのどちらも、DB2RUNSTATS_SAMPLING_REPEAT フラグが指定されない限り、統計収集を実行するたびに、通常はそれぞれ異なる表サンプルが生成されます。

DB2RUNSTATS_SAMPLING_REPEAT
iSamplingRepeatable パラメーターを介してシードが受け渡されることを示します。 *iSamplingRepeatable* 値は、データ・サンプルを生成する際のシードとして使用されます。さらに、サンプリング比率を指示するために *iSamplingOption* パラメーターを指定する必要もあります。

DB2RUNSTATS_USE_PROFILE
表のカタログにすでに登録済みの統計プロファイルを使って表の統計を収集します。 *iRunstatsFlags* ビット・マスク内でこのフラグによって USE PROFILE オプションが指定される場合、*db2RunstatsData* 内の他のすべてのオプションは無視されます。

DB2RUNSTATS_SET_PROFILE
指定された統計オプションを記録するプロファイルをカタログに生成および保管し、その同じオプションを使って統計を収集します。

DB2RUNSTATS_SET_PROFILE_ONLY
指定された統計オプションを記録するプロファイルをカタログに生成および保管しますが、実際には表の統計を収集しません。

DB2RUNSTATS_UPDATE_PROFILE
カタログ内の既存の統計プロファイルを変更し、更新後のプロファイルのオプションを使って統計を収集します。

DB2RUNSTATS_UPDATE_PROFILE_ONLY
カタログ内の既存の統計プロファイルを変更しますが、実際には表の統計を収集しません。

iTablenameLen
入力。表名の長さを示す値 (バイト単位)。

iNumColumns
入力。 piColumnList リストで指定された項目数。

iNumColdist
入力。 *piColumnDistributionList* リストで指定された項目数。

iNumColGroups
入力。 *piColumnGroupList* リストで指定された項目数。

iNumIndexes
入力。 *piIndexList* リストで指定された項目数。

iParallelismOption
入力。将来の使用のために予約済み。有効な値は 0 です。

iTableDefaultFreqValues
入力。表について収集する頻度のデフォルトの回数を指定します。有効な値は以下のとおりです。

n 列レベルで他の値が指定されていない場合、n の頻度で収集されます。

0 列レベルで他の値が指定されていない場合、収集の頻度はありません。

-1 収集する頻度に、デフォルトのデータベース構成パラメーター NUM_FREQVALUES を使用します。

iTableDefaultQuantiles
入力。表について収集する変位値のデフォルトの数値を指定します。有効な値は以下のとおりです。

n 列レベルで他の値が指定されていない場合、n 個の変位値が収集されます。

0 列レベルで他の値が指定されない場合、変位値は収集されません。

-1 収集する変位値の数について、デフォルトのデータベース構成パラメーター NUM_QUANTILES を使用します。

iUtilImpactPriority
入力。 runstats 呼び出しの優先度。有効な値の範囲は 0 から 100 です。0 はスロットルなし、 100 は可能な限り最も高い優先度をそれぞれ表します。

piColumnName
入力。列名を表すストリングを指すポインター。

iColumnNameLen
入力。列名の長さを示す値 (バイト単位)。

iColumnFlags
入力。列の統計オプションを指定するために使用されるビット・マスク・フィールド。有効な値は以下のとおりです。

DB2RUNSTATS_COLUMN_LIKE_STATS
列で LIKE 統計を収集します。

iNumFreqValues
入力。列で収集する頻度。有効な値は以下のとおりです。

n 列で n の頻度で収集します。

-1 表の頻度のデフォルト値を使用します。たとえばこれには、設定されている場合は *iTableDefaultFreqValues* や、またはデータベース構成パラメーター NUM_FREQVALUES があります。

iNumQuantiles
入力。列に関して収集する変位値の数。有効な値は以下のとおりです。

n 列に関して n 個の変位値を収集します。

-1 変位値の表デフォルト数を使用します。これには、たとえば *iTableDefaultQuantiles* (設定されている場合) や、データベース構成パラメーター NUM_QUANTILES があります。

piGroupColumnNames
入力。ストリングの配列。各ストリングは、統計が収集される列グループの一部である列名を表します。

piGroupColumnNamesLen
入力。列名リストにある列名のそれぞれの長さを示す値の配列 (バイト単位)。

iGroupSize
入力。列グループ内の列の数。有効な値は以下のとおりです。

n 列グループは n 列で構成されています。

iNumFreqValues
入力。将来の使用のために予約済み。

iNumQuantiles
入力。将来の使用のために予約済み。

iSamplingRepeatable
入力。表サンプリングに使用するシードを表す、負でない整数。負のシードを渡した場合、エラー (SQL1197N) が発生します。このシードを使用するには、 DB2RUNSTATS_SAMPLING_REPEAT フラグを設定しておく必要があります。このオプションを *iSamplingOption* パラメーターとともに使用すれば、これ以降の統計収集で同じサンプル・データを生成することができます。ただし、反復可能な要求が最後に実行された後で、何らかのアクティビティーによって表データが変更された場合には、繰り返し要求ごとにサンプル・セットが異なる可能性があります。さらに、一貫した結果を生成するためには、サンプルを取得する方式 (BERNOULLI または SYSTEM) が同じでなければなりません。

使用上の注意:

db2Runstats は、以下のような場合に統計を更新するために使用してください。

- 表が何回も修正されている場合 (たとえば、数多くの更新が行われている場合や、大量のデータが挿入または削除されている場合など)
- 表が再編成されている場合
- 新しい索引が作成されている場合

統計が更新された後、sqlabndx - バインドを使用してパッケージを再バインドすることによって、表への新しいアクセス・パスを作成することができます。

索引統計を要求したときに、索引を含む表についての統計がそれまで実行されていなかった場合、表と索引の両方に関する統計が計算されます。

db2Runstats API が索引でのみ統計を収集している場合は、以前に収集された分散統計は保存されます。そうでない場合は、API は以前に収集された分散統計をドロップします。

この API を呼び出した後、アプリケーションは COMMIT を発行して、ロックを解除する必要があります。

新しいアクセス・プランが生成されるようにするには、この API を呼び出した後、ターゲット表を参照するパッケージを再バインドする必要があります。

表に対してのみこの API を実行すると、結果として表レベルの統計が、既存の索引レベルの統計と不整合な状況になる場合があります。たとえば、索引レベルの統計が特定の表で収集され、後にかなりの行数がこの表から削除された場合、表に対してのみこの API を発行すると、不整合状態の FIRSTKEYCARD よりも小さい表カーディナリティーという結果に終わる場合があります。それと同様に、索引に対してのみこの API を発行する場合、既存の表レベル統計が整合性がない状態のままになることがあります。たとえば、表レベルの統計が特定の表で収集され、後でかなりの行数がこの表から削除された場合、索引に対してのみ db2Runstats API を発行すると、いくつかの列が表カーディナリティーより大きい COLCARD を持つという結果に終わる場合があります。そのような不整合が検出された場合、警告が戻されます。

関連資料:

- 294 ページの『sqlabndx - バインド』
- 453 ページの『SQLCA』
- 「コマンド・リファレンス」の『REORGCHK コマンド』
- 36 ページの『db2CfgGet - 構成パラメーターの入手』
- 234 ページの『db2Reorg - 再編成』

関連サンプル:

- 『dbstat.sqb -- Reorganize table and run statistics (MF COBOL)』
- 『tbreorg.sqc -- How to reorganize a table and update its statistics (C)』
- 『tbreorg.sqC -- How to reorganize a table and update its statistics (C++)』

db2SetSyncSession - サテライト同期セッションの設定

サテライトの同期セッションを設定します。同期セッションはサテライトで実行されるユーザー・アプリケーションのバージョンと関連付けられます。アプリケーションの各バージョンは特定のデータベース構成でサポートされ、特定のデータ・セット (それぞれ中央サイトで同期化できる) を操作します。

許可:

なし

必要な接続:

なし

API 組み込みファイル:

db2ApiDf.h

C API 構文:

```
/* File: db2ApiDf.h */
/* API: db2SetSyncSession */
/* ... */
SQL_API_RC SQL_API_FN
  db2SetSyncSession (
    db2Uint32 versionNumber,
    void *pParmStruct,
    struct sqlca *pSqlca);

typedef struct
{
  char *piSyncSessionID;
} db2SetSyncSessionStruct;
/* ... */
```

API パラメーター:

versionNumber
入力。 2 番目のパラメーター *pParmStruct* として渡される構造のバージョンとリリースのレベルを指定します。

pParmStruct
入力。 *db2SetSyncSessionStruct* 構造を指すポインター。

pSqlca
出力。*sqlca* 構造へのポインター。

piSyncSessionID
入力。サテライトが使用する同期セッションの ID を指定します。指定された値は、サテライト・コントロール・サーバーで定義されているように、サテライトのグループの適切なアプリケーション・バージョンと一致する必要があります。

関連資料:

• 453 ページの『SQLCA』

db2SetWriteForDB - 入出力の設定または再開

データベースの入出力書き込みの中断や、ディスクへの入出力書き込みの再開を設定します。ミラーの分割が起こる前に、データベースの入出力書き込みを中断しなければなりません。問題が発生するのを防ぐために、同じ接続を保持し、書き込みを中断および再開してください。

許可:

以下のいずれかです。

- *sysadm*
- *sysctrl*

- *sysmaint*

必要な接続:

データベース

API 組み込みファイル:

db2ApiDf.h

C API 構文:

```
/* File: db2ApiDf.h */
/* API: db2SetWriteForDB */
/* ... */
SQL_API_RC SQL_API_FN
db2SetWriteForDB (
  db2Uint32 versionNumber,
  void *pParmStruct,
  struct sqlca *pSqlca);

typedef struct db2SetWriteDbStruct
{
  db2int32                    iOption;
  char                        *piTablespaceNames;
} db2SetWriteDbStruct;
/* ... */
```

API パラメーター:

version
入力。 2 番目のパラメーター *pParmStruct* として渡される構造のバージョンとリリースのレベルを指定します。

pParmStruct
入力。 *db2SetWriteDbStruct* 構造を指すポインター。

pSqlca
出力。*sqlca* 構造へのポインター。

iOption
入力。アクションを指定します。有効な値は以下のとおりです。

DB2_DB_SUSPEND_WRITE
ディスクへの入出力書き込みを中断します。

DB2_DB_RESUME_WRITE
ディスクへの入出力書き込みを再開します。

piTablespaceNames
入力。将来の利用のために予約されています。

db2SyncSatellite - サテライトの同期

サテライトを同期化します。

許可:

なし

必要な接続:

なし

API 組み込みファイル:

db2ApiDf.h

C API 構文:

```
/* File: db2ApiDf.h */
/* API: db2SyncSatellite */
/* ... */
SQL_API_RC SQL_API_FN
db2SyncSatellite (
  db2Uint32 versionNumber,
  void *pParmStruct,
  struct sqlca *pSqlca);
/* ... */
```

API パラメーター:

versionNumber
入力。 2 番目のパラメーター *pParmStruct* として渡される構造のバージョンとリリースのレベルを指定します。

pParmStruct
入力。 NULL に設定してください。

pSqlca
出力。*sqlca* 構造へのポインター。

関連資料:

• 453 ページの『SQLCA』

db2SyncSatelliteStop - サテライト同期の停止

サテライトの現在アクティブな同期セッションを停止します。セッションは、このサテライトの同期化を db2SyncSatellite を呼び出して再開できるように停止されます。

許可:

なし

必要な接続:

なし

API 組み込みファイル:

db2ApiDf.h

C API 構文:

```
/* File: db2ApiDf.h */
/* API: db2SyncSatelliteStop */
/* ... */
SQL_API_RC SQL_API_FN
db2SyncSatelliteStop (
```

```
  db2Uint32 VersionNumber,
  void *pParmStruct,
  struct sqlca *pSqlca);
/* ... */
```

API パラメーター:

versionNumber
入力。 2 番目のパラメーター *pParmStruct* として渡される構造のバージョンとリリースのレベルを指定します。

pParmStruct
入力。 NULL に設定してください。

pSqlca
出力。*sqlca* 構造へのポインター。

関連資料:

- 453 ページの『SQLCA』
- 278 ページの『db2SyncSatellite - サテライトの同期』

db2SyncSatelliteTest - サテライト同期のテスト

サテライトの同期化機能をテストします。

許可:

なし

必要な接続:

なし

API 組み込みファイル:

db2ApiDf.h

C API 構文:

```
/* File: db2ApiDf.h */
/* API: db2SyncSatelliteTest */
/* ... */
SQL_API_RC SQL_API_FN
db2SyncSatelliteTest (
  db2Uint32 versionNumber,
  void *pParmStruct,
  struct sqlca *pSqlca);
/* ... */
```

API パラメーター:

versionNumber
入力。 2 番目のパラメーター *pParmStruct* として渡される構造のバージョンとリリースのレベルを指定します。

pParmStruct
入力。 NULL に設定してください。

pSqlca
出力。*sqlca* 構造へのポインター。

関連資料:

• 453 ページの『SQLCA』

db2UpdateAlertCfg - アラート構成の更新

ヘルス・インディケーターのアラート構成設定を更新します。

許可:

以下のいずれかです。

- *sysadm*
- *sysctrl*
- *sysmaint*

必要な接続:

インスタンス。インスタンス・アタッチが存在しない場合は、デフォルトのインスタンス・アタッチが作成されます。

API 組み込みファイル:

db2ApiDf.h

C API 構文:

```
/* File: db2ApiDf.h */
/* API: db2UpdateAlertCfg */
/* ... */
SQL_API_RC SQL_API_FN
db2UpdateAlertCfg (
  db2Uint32 versionNumber,
  void *pParmStruct,
  struct sqlca *pSqlca);

typedef SQL_STRUCTURE db2UpdateAlertCfgData
{
   db2Uint32                 iObjType;
   char                      *piObjName;
   char                      *piDbName;
   db2Uint32                 iIndicatorID;
   db2Uint32                 iNumIndAttribUpdates;
   struct db2AlertAttrib     *piIndAttribUpdates;
   db2Uint32                 iNumActionUpdates;
   struct db2AlertActionUpdate *piActionUpdates;
   db2Uint32                 iNumActionDeletes;
   struct db2AlertActionDelete *piActionDeletes;
   db2Uint32                 iNumNewActions;
   struct db2AlertActionNew  *piNewActions;
} db2UpdateAlertCfgData;

typedef SQL_STRUCTURE db2AlertAttrib
{
   db2Uint32                 iAttribID;
   char                      *piAttribValue;
} db2AlertAttrib;

typedef SQL_STRUCTURE db2AlertActionUpdate
{
```

```
   db2Uint32                iActionType;
   char                     *piActionName;
   db2Uint32                iCondition;
   db2Uint32                iNumParmUpdates;
   struct db2AlertAttrib    *piParmUpdates;
} db2AlertActionUpdate;

typedef SQL_STRUCTURE db2AlertActionDelete
{
   db2Uint32                iActionType;
   char                     *piName;
   db2Uint32                iCondition;
} db2AlertActionDelete;

typedef SQL_STRUCTURE db2AlertActionNew
{
   db2Uint32                iActionType;
   struct db2AlertScriptAction *piScriptAttribs;
   struct db2AlertTaskAction *piTaskAttribs;
} db2AlertActionNew;

typedef SQL_STRUCTURE db2AlertScriptAction
{
   db2Uint32                scriptType;
   db2Uint32                condition;
   char                     *pPathName;
   char                     *pWorkingDir;
   char                     *pCmdLineParms;
   char                     stmtTermChar;
   char                     *pUserID;
   char                     *pPassword;
   char                     *pHostName;
} db2AlertScriptAction;

typedef SQL_STRUCTURE db2AlertTaskAction
{
   char                     *pTaskName;
   db2Uint32                condition;
   char                     *pUserID;
   char                     *pPassword;
   char                     *pHostName;
} db2AlertTaskAction;
/* ... */
```

API パラメーター:

versionNumber
入力。 2 番目のパラメーター *pParmStruct* として渡される構造のバージョンとリリースのレベルを指定します。

pParmStruct
入力。 *db2UpdateAlertCfgData* 構造を指すポインター。

pSqlca
出力。*sqlca* 構造へのポインター。

iObjType
入力。構成が要求されるオブジェクトのタイプを指定します。有効な値は以下のとおりです。

- DB2ALERTCFG_OBJTYPE_DBM
- DB2ALERTCFG_OBJTYPE_DATABASES
- DB2ALERTCFG_OBJTYPE_TABLESPACES

- DB2ALERTCFG_OBJTYPE_TS_CONTAINERS
- DB2ALERTCFG_OBJTYPE_DATABASE
- DB2ALERTCFG_OBJTYPE_TABLESPACE
- DB2ALERTCFG_OBJTYPE_TS_CONTAINER

piObjName
入力。オブジェクト・タイプ *iObjType* が、DB2ALERTCFG_OBJTYPE_TABLESPACE または DB2ALERTCFG_OBJTYPE_TS_CONTAINER に設定される場合、表スペースまたは表スペース・コンテナーの名前。そうでない場合は、NULL を設定します。

piDbName
入力。オブジェクト・タイプ *iObjType* が DB2ALERTCFG_OBJTYPE_TS_CONTAINER、DB2ALERTCFG_OBJTYPE_TABLESPACE、および DB2ALERTCFG_OBJTYPE_DATABASE に設定される場合、構成が要求されるデータベースの別名。そうでない場合は、NULL を設定します。

iIndicatorID
入力。構成の更新が適用されるヘルス・インディケーター。

iNumIndAttribUpdates
入力。 *iIndicatorID* ヘルス・インディケーター用に更新されるアラート属性の数。

piIndAttribUpdates
入力。 *db2AlertAttrib* 構造配列を指すポインター。

iNumActionUpdates
入力。 *iIndicatorID* ヘルス・インディケーター用に更新されるアラート・アクションの数。

piActionUpdates
入力。 *db2AlertActionUpdate* 構造配列を指すポインター。

iNumActionDeletes
入力。 *iIndicatorID* ヘルス・インディケーターから削除されるアラート・アクションの数。

piActionDeletes
入力。 *db2AlertActionDelete* 構造配列を指すポインター。

iNumNewActions
入力。 *iIndicatorID* ヘルス・インディケーターに追加される新しいアラート・アクションの数。

piNewActions
入力。 *db2AlertActionNew* 構造配列を指すポインター。

iAttribID
入力。更新されるアラート属性を指定します。有効な値は以下のとおりです。

- DB2ALERTCFG_ALARM
- DB2ALERTCFG_WARNING

- DB2ALERTCFG_SENSITIVITY
- DB2ALERTCFG_ACTIONS_ENABLED
- DB2ALERTCFG_THRESHOLD_CHECK

piAttribValue
入力。アラート属性の新しい値。有効な値は以下のとおりです。

- DB2ALERTCFG_ALARM
- DB2ALERTCFG_WARNING
- DB2ALERTCFG_SENSITIVITY
- DB2ALERTCFG_ACTIONS_ENABLED
- DB2ALERTCFG_THRESHOLD_CHECK

iActionType
入力。アラート・アクションを指定します。有効な値は以下のとおりです。

- DB2ALERTCFG_ACTIONTYPE_SCRIPT
- DB2ALERTCFG_ACTIONTYPE_TASK

piActionName
入力。アラート・アクションの名前。スクリプト・アクションの名前が、スクリプトの絶対パス名です。タスク・アクションの名前は、<task-numberical-ID>.<task-numberical-suffix> の形式のストリングです。

iCondition
アクションを実行する状態。ヘルス・インディケーターに基づく有効な値は以下のとおりです。

- DB2ALERTCFG_CONDITION_ALL
- DB2ALERTCFG_CONDITION_WARNING
- DB2ALERTCFG_CONDITION_ALARM

ヘルス・インディケーターに基づく状態の場合は、`sqlmon` で定義された数値を使用します。

iNumParmUpdates
入力。 *piParmUpdates* 配列で更新されるアクション属性の数。

piParmUpdates
入力。 *db2AlertAttrib* 構造を指すポインター。

piName
入力。アラート・アクションまたはスクリプト・アクションの名前。スクリプト・アクションの名前は、スクリプトの絶対パス名です。一方、タスク・アクションの名前は、<task-numerical-ID>.<task-numerical-suffix> の形式のストリングです。

piScriptAttribs
入力。 *db2AlertScriptAction* 構造を指すポインター。

piTaskAttribs
入力。 *db2AlertTaskAction* 構造を指すポインター。

scriptType
スクリプトが DB2 コマンド・スクリプトなのか、またはオペレーティング・システム・スクリプトなのかを指定します。有効な値は以下のとおりです。

- DB2ALERTCFG_SCRIPTTYPE_DB2CMD
- DB2ALERTCFG_SCRIPTTYPE_OS

condition
アクションを実行する状態。ヘルス・インディケーターに基づく有効な値は以下のとおりです。

- DB2ALERTCFG_CONDITION_ALL
- DB2ALERTCFG_CONDITION_WARNING
- DB2ALERTCFG_CONDITION_ALARM

ヘルス・インディケーターに基づく状態の場合は、`sqlmon` で定義された数値を使用します。

pPathName
実行するスクリプトの絶対パス名。

pWorkingDir
スクリプトが実行されるディレクトリーの絶対パス名。

pCmdLineParms
scriptType が DB2ALERTCFG_SCRIPTTYPE_OSCMD である場合のコマンド行パラメーター。

stmtTermChar
scriptType が DB2ALERTCFG_SCRIPTTYPE_OS である場合に、 DB2 コマンドの各ステートメントを終了する文字。

pUserID
スクリプトを実行するユーザー・アカウント。

pPassword
pUserId 用の有効なパスワード。

pHostName
スクリプトまたはタスクを実行するホスト名。 *pHostName* の説明については、『db2GetAlertCfg』を参照してください。

pTaskName
タスク名。

関連資料:

- 75 ページの『db2GetAlertCfg - アラート構成の入手』
- 239 ページの『db2ResetAlertCfg - アラート構成のリセット』

db2UpdateAlternateServerForDB - データベースの代替サーバーの更新

データベース別名に関連した代替サーバーをシステム・データベース・ディレクトリー内で更新します。

有効範囲:

この API はシステム・データベース・ディレクトリーに影響を与えます。

許可:

以下のいずれかが必要です。

- *sysadm*
- *sysctrl*

必要な接続:

なし

API 組み込みファイル:

db2ApiDf.h

C API 構文:

```
/* File: db2ApiDf.h */
/* API: db2UpdateAlternateServerForDB */
/* ... */
SQL_API_RC SQL_API_FN
  db2UpdateAlternateServerForDB (
       db2Uint32 versionNumber,
       void * pParmStruct,
       struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2UpdateAltServerStruct
{
   char                     *piDbAlias;
   char                     *piHostName;
   char                     *piPort;
} db2UpdateAltServerStruct;
  /* ... */
```

汎用 API 構文:

```
/* File: db2ApiDf.h */
/* API: db2gUpdateAlternateServerForDB */
/* ... */
SQL_API_RC SQL_API_FN
  db2gUpdateAlternateServerForDB (
       db2Uint32 versionNumber,
       void * pParmStruct,
       struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2gUpdateAltServerStruct
{
   db2Uint32                iDbAlias_len;
   char                     *piDbAlias;
   db2Uint32                iHostName_len;
   char                     *piHostName;
   db2Uint32                iPort_len;
   char                     *piPort;
} db2gUpdateAltServerStruct;
  /* ... */
```

API パラメーター:

versionNumber
入力。 2 番目のパラメーター *pParmStruct* として渡される構造のバージョンとリリースのレベルを指定します。

pParmStruct
入力。 *db2UpdateAltServerStruct* 構造を指すポインター。

pSqlca
出力。*sqlca* 構造へのポインター。

iDbAlias_len
入力。 *piDbAlias* の長さ (バイト単位)。

piDatabaseAlias
入力。データベースの別名を含むストリングを指定します。

iHostName_len
入力。 *piHostName* の長さ (バイト単位)。

piHostName
入力。データベースの代替サーバーが常駐しているノードのホスト名または IP アドレスを含む文字列。ホスト名は、TCP/IP ネットワークで認識されるノードの名前です。ホスト名の最大長は 255 文字です。

iPort_len
入力。 *piPort* の長さ (バイト単位)。

piPort 入力。代替サーバーのデータベース・マネージャー・インスタンスのポート番号。ポート番号の最大長は 14 文字です。

使用上の注意:

この API は、システム・データベース・ディレクトリーにのみ適用されます。

この API は、サーバーでのみ使用してください。クライアントでこれを発行すると、無視されて、警告 SQL1889W が出されます。

LDAP (Lightweight Directory Access Protocol) サポートが現行のマシン上で使用可能である場合、データベースの代替サーバーは、自動的に LDAP ディレクトリーで更新されます。

関連資料:

- 340 ページの『sqlecadb - データベースのカタログ』
- 411 ページの『sqleuncd - データベースのアンカタログ』
- 453 ページの『SQLCA』
- 167 ページの『db2LdapUpdateAlternateServerForDB - データベースの代替サーバーの LDAP 更新』

db2UpdateContact - 連絡先の更新

連絡先の属性を更新します。連絡先は、通知メッセージが送信されるユーザーです。連絡先は、システムでローカルに定義することも、グローバル・リストで定義することもできます。 DB2 Administration Server (DAS) の構成パラメーター *contact_host* の設定は、リストがローカルかグローバルかを判別します。

許可:

なし

必要な接続:

インスタンス。インスタンス・アタッチが存在しない場合は、デフォルトのインスタンス・アタッチが作成されます。

API 組み込みファイル:

db2ApiDf.h

C API 構文:

```
/* File: db2ApiDf.h */
/* API: db2UpdateContact */
/* ... */
SQL_API_RC SQL_API_FN
db2UpdateContact (
  db2Uint32 versionNumber,
  void *pParmStruct,
  struct sqlca *pSqlca);

typedef SQL_STRUCTURE db2UpdateContactData
{
   char                       *piUserid;
   char                       *piPassword;
   char                       *piContactName;
   db2Uint32                  iNumAttribsUpdated;
   struct db2ContactAttrib    *piAttribs;
} db2UpdateContactData;

typedef SQL_STRUCTURE db2ContactAttrib
{
   db2Uint32                  iAttribID;
   char                       *piAttribValue;
} db2ContactAttrib;
/* ... */
```

API パラメーター:

versionNumber
入力。 2 番目のパラメーター *pParmStruct* として渡される構造のバージョンとリリースのレベルを指定します。

pParmStruct
入力。 *db2UpdateContactData* 構造を指すポインター。

pSqlca
出力。*sqlca* 構造へのポインター。

piContactName
入力。更新される連絡先の名前を指定します。

iNumAttribsUpdated
入力。更新される属性の数。

piAttribs
入力。 *db2ContactAttrib* 構造を指すポインター。

iAttribID
入力。連絡先の属性を指定します。有効な値は以下のとおりです。

- DB2CONTACT_ADDRESS
- DB2CONTACT_TYPE
- DB2CONTACT_MAXPAGELEN
- DB2CONTACT_DESCRIPTION

piAttribValue
入力。連絡先の属性の新しい値。

関連資料:

- 453 ページの『SQLCA』
- 「*管理ガイド: パフォーマンス*」の『contact_host - 「連絡先リストのロケーション」構成パラメーター』
- 17 ページの『db2AddContact - 連絡先の追加』
- 60 ページの『db2DropContact - 連絡先のドロップ』
- 83 ページの『db2GetContacts - 連絡先の入手』

db2UpdateContactGroup - 連絡先グループの更新

連絡先グループの属性を更新します。連絡先グループには、通知メッセージが送信されるユーザーのリストが入っています。連絡先グループは、システムでローカルに定義することも、グローバル・リストで定義することもできます。 DB2 Administration Server (DAS) の構成パラメーター *contact_host* の設定は、リストがローカルかグローバルかを判別します。

許可:

なし。

必要な接続:

なし。

API 組み込みファイル:

db2ApiDf.h

C API 構文:

```
/* File: db2ApiDf.h */
/* API: db2UpdateContactGroup */
/* ... */
```

```
SQL_API_RC SQL_API_FN
db2UpdateContactGroup (
  db2Uint32 versionNumber,
  void *pParmStruct,
  struct sqlca *pSqlca);

typedef SQL_STRUCTURE db2UpdateContactGroupData
{
   char                       *piUserId;
   char                       *piPassword;
   char                       *piGroupName;
   db2Uint32                  iNumNewContacts;
   struct db2ContactTypeData *piNewContacts;
   db2Uint32                  iNumDroppedContacts;
   struct db2ContactTypeData *piDroppedContacts;
   char                       *piNewDescription;
} db2UpdateContactGroupData;

typedef SQL_STRUCTURE db2ContactTypeData
{
   db2Uint32                  contactType;
   char                       *pName;
} db2ContactTypeData;
/* ... */
```

API パラメーター:

versionNumber
入力。 2 番目のパラメーター *pParmStruct* として渡される構造のバージョンとリリースのレベルを指定します。

pParmStruct
入力。 *db2ResetMonitorData* 構造を指すポインター。

pSqlca
出力。*sqlca* 構造へのポインター。

piUserid
入力。ユーザー名。

piPassword
入力。 *piUserid* 用のパスワード。

piGroupName
入力。更新する連絡先グループの名前。

iNumNewContacts
入力。グループに追加される新しい連絡先の数。

piNewContacts
入力。 *db2ContactTypeData* 構造を指すポインター。

iNumDroppedContacts
入力。ドロップされるグループ内の連絡先の数。

piDroppedContacts
入力。 *db2ContactTypeData* 構造を指すポインター。

piNewDescription
入力。グループの新しい説明。古い説明を変更しない場合は、このパラメーターを NULL に設定します。

contactType
連絡先のタイプを指定します。有効な値は以下のとおりです。

- DB2CONTACT_SINGLE
- DB2CONTACT_GROUP

pName
連絡先グループ名、または *contactType* が DB2CONTACT_SINGLE に設定されている場合は連絡先の名前。

関連資料:

- 453 ページの『SQLCA』
- 「*管理ガイド: パフォーマンス*」の『contact_host - 「連絡先リストのロケーション」構成パラメーター』
- 18 ページの『db2AddContactGroup - 連絡先グループの追加』
- 61 ページの『db2DropContactGroup - 連絡先グループのドロップ』
- 80 ページの『db2GetContactGroup - 連絡先グループの入手』
- 81 ページの『db2GetContactGroups - 連絡先グループの入手』

db2UpdateHealthNotificationList - ヘルス通知リストの更新

インスタンスによって発行されるヘルス・アラートについての通知の連絡先リストを更新します。

許可:

以下のいずれかです。

- *sysadm*
- *sysctrl*
- *sysmaint*

必要な接続:

インスタンス。インスタンス・アタッチが存在しない場合は、デフォルトのインスタンス・アタッチが作成されます。

API 組み込みファイル:

db2ApiDf.h

C API 構文:

```
/* File: db2ApiDf.h */
/* API: db2UpdateHealthNotificationList */
/* ... */
SQL_API_RC SQL_API_FN
db2UpdateHealthNotificationList (
  db2Uint32 versionNumber,
  void *pParmStruct,
  struct sqlca *pSqlca);

typedef SQL_STRUCTURE db2UpdateHealthNotificationListData
{
   db2Uint32                 iNumUpdates;
   struct db2HealthNotificationListUpdate *piUpdates;
} db2UpdateHealthNotificationListData;
```

```
typedef SQL_STRUCTURE db2HealthNotificationListUpdate
{
   db2Uint32                iUpdateType;
   struct db2ContactTypeData *piContact;
} db2HealthNotificationListUpdate;

typedef SQL_STRUCTURE db2ContactTypeData
{
   db2Uint32                contactType;
   char                     *pName;
} db2ContactTypeData;
/* ... */
```

API パラメーター:

versionNumber
入力。 2 番目のパラメーター *pParmStruct* として渡される構造のバージョンとリリースのレベルを指定します。

pParmStruct
入力。 *db2UpdateHealthNotificationListData* 構造を指すポインター。

pSqlca
出力。*sqlca* 構造へのポインター。

iNumUpdates
入力。更新の数。

piUpdates
入力。 *db2HealthNotificationListUpdate* 構造を指すポインター。

iUpdateType
入力。更新のタイプを指定します。有効な値は以下のとおりです。

- DB2HEALTHNOTIFICATIONLIST_ADD
- DB2HEALTHNOTIFICATIONLIST_DROP

piContact
入力。 *db2ContactTypeData* 構造を指すポインター。

contactType
連絡先のタイプを指定します。有効な値は以下のとおりです。

- DB2CONTACT_SINGLE
- DB2CONTACT_GROUP

pName
contactType が DB2CONTACT_GROUP に設定されている場合は連絡先グループ名、または *ioContactType* が DB2CONTACT_SINGLE に設定されている場合は連絡名。

関連資料:

- 453 ページの『SQLCA』
- 84 ページの『db2GetHealthNotificationList - ヘルス通知リストの入手』

db2UtilityControl - ユーティリティー制御

実行されているユーティリティーの優先順位を制御します。これを使用して、ユーティリティーの起動をスロットルまたは非スロットルすることができます。

許可:

sysadm

必要な接続:

インスタンス

API 組み込みファイル:

db2ApiDf.h

C API 構文:

```
/* File: db2ApiDf.h */
/* API: db2UtilityControl */
/* ... */
SQL_API_RC SQL_API_FN
db2UtilityControl (
  db2Uint32 version,
  void *pUtilityControlStruct,
  struct sqlca *pSqlca );

typedef struct
{
  db2Uint32 iId,
  db2Uint32 iAttribute,
  void *pioValue
} db2UtilityControlStruct;
/* ... */
```

汎用 API 構文:

```
/* File: db2ApiDf.h */
/* API: db2gUtilityControl */
/* ... */
SQL_API_RC SQL_API_FN
db2gUtilityControl (
  db2Uint32 version,
  void *pgUtilityControlStruct,
  struct sqlca *pSqlca );

typedef struct
{
  db2Uint32 iId,
  db2Uint32 iAttribute,
  void *pioValue
} db2gUtilityControlStruct;
/* ... */
```

API パラメーター:

version
入力。 2 番目のパラメーター *pUtilitlyControlStruct* として渡される構造のバージョンとリリースのレベルを指定します。

pUtilityControlStruct
入力。 *db2UtilityControlStruct* 構造を指すポインター。

pSqlca
出力。*sqlca* 構造へのポインター。

iId 入力。変更するユーティリティーの ID を指定します。

iAttribute
入力。変更する属性を指定します。有効な値 (db2ApiDf.h で定義) は、以下のとおりです。

DB2UTILCTRL_PRIORITY_ATTRIB
ユーティリティーのスロットル優先度を変更します。

pioValue
入力。 *iAttribute* パラメーターと関連した新規の属性値を指定します。

注: *iAttribute* パラメーターが DB2UTILCTRL_PRIORITY_ATTRIB に設定されている場合、 *pioValue* パラメーターは、優先順位が入っている *db2Uint32* を指すようにする必要があります。

使用上の注意:

指定した *iId* を持つ既存ユーティリティーが存在しない場合、SQL1153N が戻されます。これは無効な引き数が指定されて関数が呼び出されたか、またはユーティリティーが完了したことを示す場合があります。

ユーティリティーがスロットルをサポートしていない場合、 SQL1154N が戻されます。

---

sqlabndx - バインド

バインド・ユーティリティーを呼び出し、プリコンパイラーによって生成されたバインド・ファイルに保管された SQL ステートメントを作成します。また、データベースに保管されるパッケージを作成します。

有効範囲:

この API は、db2nodes.cfg の任意のデータベース・パーティション・サーバーから呼び出すことができます。 It updates the データベース catalogs on the catalog partition. この効果はすべてのデータベース・パーティション・サーバーで可視になります。

許可:

以下のいずれかです。

- *sysadm* または *dbadm* 権限。
- パッケージが存在していない場合、および以下のいずれかの場合には、BINDADD 特権。
  - パッケージのスキーマ名が存在していない場合、データベースに対する IMPLICIT_SCHEMA 権限。

  - パッケージのスキーマ名が存在している場合、そのスキーマに対する CREATEIN 特権。
- パッケージが存在している場合、そのスキーマに対する ALTERIN 特権。
- パッケージが存在している場合、そのパッケージに対する BIND 特権。

アプリケーション内の静的 SQL ステートメントをコンパイルするために必要な特権もすべて必要です。グループに認可された特権が、静的ステートメントの許可の検査に使用されることはありません。 *sysadm* 権限は持っていてもバインドを完了するための明示的な特権を持っていない場合、データベース・マネージャーによって明示的な *dbadm* 権限が自動的に付与されます。

必要な接続:

データベース

API 組み込みファイル:

sql.h

C API 構文:

```
/* File: sql.h */
/* API: sqlabndx */
/* ... */
SQL_API_RC SQL_API_FN
  sqlabndx (
    _SQLOLDCHAR *pBindFileName,
    _SQLOLDCHAR *pMsgFileName,
    struct sqlopt *pBindOptions,
    struct sqlca *pSqlca);
/* ... */
```

汎用 API 構文:

```
/* File: sql.h */
/* API: sqlgbndx */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgbndx (
    unsigned short MsgFileNameLen,
    unsigned short BindFileNameLen,
    struct sqlca *pSqlca,
    struct sqlopt *pBindOptions,
    _SQLOLDCHAR *pMsgFileName,
    _SQLOLDCHAR *pBindFileName);
/* ... */
```

API パラメーター:

MsgFileNameLen
入力。メッセージ・ファイル名の長さを示す 2 バイトの符号なし整数 (バイト単位) です。

BindFileNameLen
入力。バインド・ファイル名の長さを示す 2 バイトの符号なし整数 (バイト単位) です。

pSqlca
出力。*sqlca* 構造へのポインター。

pBindOptions

入力。 API へ BIND オプションを渡すのに使用される構造を示します。この構造の詳細については、SQLOPT を参照してください。

pMsgFileName

入力。エラー、警告、および情報メッセージの宛先を含むストリングです。オペレーティング・システム・ファイルまたは標準装置のパスおよび名前を指定できます。ファイルがすでに存在する場合、そのファイルは上書きされます。存在していない場合は、新たに作成されます。

pBindFileName

入力。バインド・ファイル名、またはバインド・ファイル名のリストを含むファイルの名前を示すストリングです。バインド・ファイル名には、拡張子 .bnd を含めなければなりません。それらのファイルのパスで指定することができます。

バインド・リスト・ファイルの前には アットマーク (@) を付けます。たとえば、完全に修飾されたバインド・リスト・ファイル名は以下のようになります。

```
/u/user1/bnd/@all.lst
```

バインド・リスト・ファイルには 1 つまたは複数のバインド・ファイル名を含めてください。また、バインド・リスト・ファイルには拡張子 .lst を付けてください。

最初を除くすべてのバインド・ファイル名の前にプラス記号 (+) を付けます。このバインド・ファイル名は 2 行以上になることがあります。たとえば、バインド・リスト・ファイル all.lst には以下のものを含めることができます。

```
mybind1.bnd+mybind2.bnd+
mybind3.bnd+
mybind4.bnd
```

リスト・ファイルのバインド・ファイル名にパス指定を使用することもできます。パスが指定されていない場合、データベース・マネージャーはバインド・リスト・ファイルからパス情報を得ます。

REXX API 構文:

この API は、SQLDB2 インターフェースを使って、REXX から呼び出すことができます。

使用上の注意:

バインド処理は、アプリケーション・プログラムのソース・ファイルのプリコンパイル処理の一部として実行することもできますし、別のプロセスとして後から実行することもできます。バインドを別のプロセスとして実行するときは BIND を使用します。

パッケージを作成するのに使われた名前はバインド・ファイルに保管されます。その名前は、その名前が生成されたソース・ファイルの名前を基にして付けられます (既存のパスや拡張子は取り除かれます)。たとえば、プリコンパイルされた myapp.sqc というソース・ファイルは、 myapp.bnd というデフォルトのバインド・

ファイルと、 MYAPP というデフォルトのパッケージ名を生成します。 (ただし、バインド・ファイル名およびパッケージ名は、 sqlaprep の SQL_BIND_OPT および SQL_PKG_OPT オプションを使用して、プリコンパイル時にオーバーライドすることができます。)

BIND は、ユーザーが開始したトランザクションのもとで実行されます。バインドの実行後、 BIND は COMMIT (バインドが正常終了した場合) 命令か ROLLBACK (バインドが異常終了した場合) 命令を発行して、現行のトランザクションを終了させ、別のトランザクションを開始します。

致命的エラーまたは 100 を超えるエラーが生じた場合、バインドは停止してしまいます。致命的エラーがバインド中に発生した場合、BIND はバインドを停止し、すべてのファイルをクローズしようとします。また、パッケージは廃棄されます。

アプリケーション・プログラムのバインド処理には、この解説書では説明されていない前提条件や制限事項があります。たとえば、アプリケーションは V8 クライアントから V8 サーバーへはバインドできず、 V7 サーバーに対して実行されます。

BIND オプションのタイプおよび値は sql で定義されています。

関連資料:

- 300 ページの『sqlaprep - プログラムのプリコンパイル』
- 453 ページの『SQLCA』
- 454 ページの『SQLCHAR』
- 493 ページの『SQLOPT』

関連サンプル:

- 『dbpkg.sqc -- How to work with packages (C)』
- 『dbsample.sqc -- Creates a sample database (C)』
- 『dbpkg.sqC -- How to work with packages (C++)』

sqlaintp - エラー・メッセージの入手

sqlca 構造の *sqlcode* フィールドが指定したエラー状態と関連のあるメッセージを検索します。

許可:

なし

必要な接続:

なし

API 組み込みファイル:

sql.h

C API 構文:

```
/* File: sql.h */
/* API: sqlaintp */
/* ... */
SQL_API_RC SQL_API_FN
  sqlaintp (
    char *pBuffer,
    short BufferSize,
    short LineWidth,
    struct sqlca *pSqlca);
/* ... */
```

汎用 API 構文:

```
/* File: sql.h */
/* API: sqlgintp */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgintp (
    short BufferSize,
    short LineWidth,
    struct sqlca *pSqlca,
    _SQLOLDCHAR *pBuffer);
/* ... */
```

API パラメーター:

BufferSize
入力。検索したメッセージ・テキストを保留するストリング・バッファーのサイズ (バイト単位) です。

LineWidth
入力。メッセージ・テキストの各行ごとの最大行幅を示します。ワード境界で改行されます。値ゼロは、メッセージ・テキストが改行されることなく戻されることを示します。

pSqlca
出力。*sqlca* 構造へのポインター。

pBuffer
出力。メッセージ・テキストが配置されるストリング・バッファーを指すポインター。メッセージをバッファーに合わせて切り捨てる必要がある場合、切り捨ては NULL ストリング終止符を見込んで行われます。

REXX API 構文:

```
GET MESSAGE INTO :msg [LINEWIDTH width]
```

REXX API パラメーター:

msg テキスト・メッセージが入れられる REXX 変数。

width
テキスト・メッセージの各行ごとの最大行幅。ワード境界で改行されます。*width* が指定されない場合または 0 に設定された場合、メッセージ・テキストは改行されずに戻されます。

使用上の注意:

呼び出しごとに 1 つのメッセージが戻されます。

改行 (改行 - LF、または復帰/改行 - CR/LF) の順序列は、各メッセージの末尾に置かれます。

正の行幅が指定されている場合、その行幅を超えないように、改行の順序列がワード間に挿入されます。

あるワードが行幅よりも長い場合、その行に入るだけの文字が入ります。改行が挿入され、入りきらなかった残りの文字は次の行に移されます。

マルチスレッド・アプリケーションでは、sqlaintp は有効なコンテキストに追加する必要があります。そうでない場合は、SQLCODE -1445 のメッセージ・テキストを取得できません。

戻りコード:

コード メッセージ

+i フォーマット設定メッセージのバイト数を示す正の整数です。呼び出し側が入力したバッファー・サイズよりもこの数の方が大きい場合、メッセージは切り捨てられます。

-1 メッセージ書式化サービスを機能させるには、利用可能なメモリーが不十分です。要求されたメッセージは、戻されません。

-2 エラーはありません。 *sqlca* に、エラー・コード (SQLCODE = 0) は含まれていませんでした。

-3 メッセージ・ファイルがアクセス不能または正しくありません。

-4 行幅が 0 未満です。

-5 無効 *sqlca*、不良バッファー・アドレス、または不良バッファー長を示します。

戻りコードが -1 または -3 の場合、メッセージ・バッファーにはその問題に関する、より詳細な情報が含まれています。

関連資料:

- 417 ページの『sqlogstt - SQLSTATE メッセージの入手』
- 453 ページの『SQLCA』

関連サンプル:

- 『checkerr.cbl -- Checks for and prints to the screen SQL warnings and errors (IBM COBOL)』
- 『dbcfg.sqc -- Configure database and database manager configuration parameters (C)』
- 『utilapi.c -- Error-checking utility for non-embedded SQL samples in C (C)』
- 『dbcfg.sqC -- Configure database and database manager configuration parameters (C++)』
- 『utilapi.C -- Checks for and prints to the screen SQL warnings and errors (C++)』

sqlaprep - プログラムのプリコンパイル

組み込み SQL ステートメントを含むアプリケーション・プログラム・ソース・ファイルを処理します。 SQL ステートメントのホスト言語呼び出しが入った修正済みソース・ファイルが作成され、デフォルトではデータベース内にパッケージが作成されます。

有効範囲:

この API は、db2nodes.cfg の任意のデータベース・パーティション・サーバーから呼び出すことができます。カタログ・パーティションのデータベース・カタログを更新します。この効果はすべてのデータベース・パーティション・サーバーで可視になります。

許可:

以下のいずれかです。

- *sysadm* または *dbadm* 権限。
- パッケージが存在していない場合、および以下のいずれかの場合には、BINDADD 特権。
  - パッケージのスキーマ名が存在していない場合、データベースに対する IMPLICIT_SCHEMA 権限。
  - パッケージのスキーマ名が存在している場合、そのスキーマに対する CREATEIN 特権。
- パッケージが存在している場合、そのスキーマに対する ALTERIN 特権。
- パッケージが存在している場合、そのパッケージに対する BIND 特権。

アプリケーション内の静的 SQL ステートメントをコンパイルするために必要な特権もすべて必要です。グループに認可された特権が、静的ステートメントの許可の検査に使用されることはありません。 *sysadm* 権限は持っていてもバインドを完了するための明示的な特権を持っていない場合、データベース・マネージャーによって明示的な *dbadm* 権限が自動的に付与されます。

必要な接続:

データベース

API 組み込みファイル:

sql.h

C API 構文:

```
/* File: sql.h */
/* API: sqlaprep */
/* ... */
SQL_API_RC SQL_API_FN
  sqlaprep (
    _SQLOLDCHAR *pProgramName,
    _SQLOLDCHAR *pMsgFileName,
    struct sqlopt *pPrepOptions,
    struct sqlca *pSqlca);
/* ... */
```

汎用 API 構文:

```
/* File: sql.h */
/* API: sqlgprep */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgprep (
    unsigned short MsgFileNameLen,
    unsigned short ProgramNameLen,
    struct sqlca *pSqlca,
    struct sqlopt *pPrepOptions,
    _SQLOLDCHAR *pMsgFileName,
    _SQLOLDCHAR *pProgramName);
/* ... */
```

API パラメーター:

MsgFileNameLen
入力。メッセージ・ファイル名の長さを示す 2 バイトの符号なし整数 (バイト単位) です。

ProgramNameLen
入力。プログラム名の長さを示す 2 バイトの符号なし整数 (バイト単位) です。

pSqlca
出力。*sqlca* 構造へのポインター。

pPrepOptions
入力。 API へプリコンパイル・オプションを渡すのに使用される構造を示します。この構造の詳細については、SQLOPT を参照してください。

pMsgFileName
入力。エラー、警告、および情報メッセージの宛先を含むストリングです。オペレーティング・システム・ファイルまたは標準装置のパスおよび名前を指定できます。ファイルがすでに存在する場合、そのファイルは上書きされます。存在していない場合は、新たに作成されます。

pProgramName
入力。プリコンパイルされるアプリケーションの名前を含むストリングです。以下の拡張子を使用してください。

- .sqb - COBOL アプリケーションの場合
- .sqc - C アプリケーションの場合
- .sqC - UNIX C++ アプリケーションの場合
- .sqf - FORTRAN アプリケーションの場合
- .sqx - C++ アプリケーションの場合

TARGET オプションを使用する場合は、入力ファイル名の拡張子をこの定義済みリストから入力する必要がありません。

UNIX ベース・システム上で組み込み SQL が含まれている C++ アプリケーションを使用する場合の優先拡張子は、 sqC です。ただし、UNIX ベース・システムでは、大文字小文字を区別しないシステム用に開発された sqx 表記も許容されています。

REXX API 構文:

この API は、SQLDB2 インターフェースを使って、REXX から呼び出すことができます。

使用上の注意:

修正されたソース・ファイルが作成されますが、これには SQL ステートメントと同じホスト言語ステートメントが入っています。デフォルトでは、接続がすでに確立されているデータベース内にパッケージが作成されます。パッケージ名は、プログラム・ファイル名 (拡張子を除く、大文字になります) と同じ最大 8 文字までです。

一度データベースに接続されると、 **sqlaprep** は開始済みのトランザクションの下で実行します。プリコンパイルの発行後、PRECOMPILE PROGRAM は COMMIT または ROLLBACK 命令を発行して、現行のトランザクションを終了し、別のトランザクションを開始します。

1 つでも致命的エラーが発生するか、または 100 を超えるエラーが発生すると、プリコンパイルは停止してしまいます。致命的エラーが発生すると、PRECOMPILE PROGRAM はプリコンパイルを停止し、すべてのファイルをクローズしようとします。また、パッケージは廃棄されます。

プリコンパイル・オプションのタイプおよび値は `sql.h` で定義されています。

関連資料:

- 294 ページの『sqlabndx - バインド』
- 453 ページの『SQLCA』
- 493 ページの『SQLOPT』

関連サンプル:

- 『dbpkg.sqc -- How to work with packages (C)』
- 『dbpkg.sqC -- How to work with packages (C++)』

sqlarbnd - 再バインド

バインド・ファイルを用いずに、データベースに保管されているパッケージを再作成できるようにします。

許可:

以下のいずれかです。

- *sysadm* または *dbadm* 権限。
- スキーマに対する ALTERIN 特権。
- パッケージに対する BIND 特権。

SYSCAT.PACKAGES システム・カタログ表の BOUNDBY 列に記録された許可 ID (最も新しくパッケージをバインドした人の ID) が、再バインドを実行する際のバインド者の許可 ID として使用されます。また、その ID はデフォルトの *schema* としてパッケージ内の表参照にも使用されます。このデフォルトの修飾子は、再バイ

ンド要求を実行するユーザーの許可 ID とは異なる場合があることに注意してください。 REBIND は、パッケージの作成時に指定されたのと同じ BIND オプションを使用します。

必要な接続:

データベース

API 組み込みファイル:

sql.h

C API 構文:

```
/* File: sql.h */
/* API: sqlarbnd */
/* ... */
SQL_API_RC SQL_API_FN
sqlarbnd (
  char *pPackageName,
  struct sqlca *pSqlca,
  struct sqlopt *pRebindOptions);
/* ... */
```

汎用 API 構文:

```
/* File: sql.h */
/* API: sqlgrbnd */
/* ... */
SQL_API_RC SQL_API_FN
sqlgrbnd (
  unsigned short PackageNameLen,
  char *pPackageName,
  struct sqlca *pSqlca,
  struct sqlopt *pRebindOptions);
/* ... */
```

API パラメーター:

PackageNameLen
入力。パッケージ名の長さを示す 2 バイトの符号なし整数 (バイト単位) です。

pPackageName
入力。再バインドするパッケージを指定する、修飾子付きまたは修飾子なしの名前を含むストリングです。修飾子なしのパッケージ名には、現行の許可 ID によって暗黙的に修飾子が付けられます。この名前にはパッケージのバージョンは含まれていません。空ストリングではないバージョンのパッケージを指定するには、 SQL_OPT_VERSION 再 BIND オプションを使用して、バージョン ID を指定しなければなりません。

pSqlca
出力。*sqlca* 構造へのポインター。

pRebindOptions
入力。 *SQLOPT* 構造を指すポインター。これは API に再 BIND オプションを渡すために使用します。この構造の詳細については、SQLOPT を参照してください。

REXX API 構文:

この API は、SQLDB2 インターフェースを使って、REXX から呼び出すことができます。

使用上の注意:

再バインドが正常に行われても、 REBIND がトランザクションを自動的にコミットすることはありません。ですから、ユーザー自身がトランザクションを明示的にコミットする必要があります。しかし、このことにより「what if」分析が可能になります。つまり、特定の統計を更新した後、変更した内容を見るためにパッケージの再バインドを試行できるようになります。さらに、1 作業単位内で複数の再バインドを実行することも可能になります。

この API の特徴を、以下にいくつか示します。

- パッケージを短時間で再作成できます。この API を使用することにより、元のバインド・ファイルがなくても、システム内の変更を利用できるようになります。たとえば、特定の SQL ステートメントが新しく作成した索引を利用できそうな場合、 REBIND によってパッケージを再作成できます。 REBIND によって、db2Runstats の実行後にパッケージを再作成することもできます。その結果、新規の統計を利用できるようになります。
- 作動不能パッケージを再作成できます。作動不能パッケージは、バインド・ユーティリティーまたは再バインド・ユーティリティーのどちらかを呼び出すことにより、明示的に再バインドしなければなりません。あるパッケージに依存する機能インスタンスがドロップされると、そのパッケージは作動不能としてマークされます (SYSCAT.PACKAGES システム・カタログの VALID 列が X に設定されます)。再バインド保存オプションは、作動不能パッケージ用にはサポートされていません。
- 無効パッケージの再バインドに関する制御がユーザーに与えられます。無効パッケージは、その実行時にデータベース・マネージャーによって自動的に (暗黙的に) 再バインドされます。それにより、無効パッケージに関する最初の SQL 要求を実行するのに著しい遅延が生じることがあります。その場合、初期遅延をなくし、予期しない SQL エラー・メッセージが戻されることのないようにするためには (暗黙的な再バインドが失敗すると戻されることがある)、無効なパッケージをシステムに自動的に再バインドさせるのではなく、ユーザー自身が無効パッケージを明示的に再バインドするようにしてください。たとえば、移行が行われると、データベースに保管されているすべてのパッケージが DB2 バージョン 5 の移行処理によって無効にされます。これに多数のパッケージが関係している場合、一度にすべての無効パッケージを明示的に再バインドしてください。この明示的な再バインドは、BIND、REBIND、または **db2rbind** ツールを使用して行うことができます。

パッケージを明示的に再バインドするのに、 BIND と REBIND のどちらを使用すべきかは、環境によって異なります。特に BIND を使用する理由がない限り、REBIND を使用するようにしてください。それは、REBIND の方が BIND よりもパフォーマンスの点で非常に優れているためです。ただし、以下の場合には必ず BIND を使用してください。

- プログラムに修正が加えられている場合 (たとえば、 SQL ステートメントが追加または削除された場合、またはパッケージがそのプログラムの実行可能モジュールと一致しない場合など)。
- 再バインドにおいて BIND オプションのいずれかを変更したい場合。 REBIND は BIND オプションをサポートしていません。たとえば、バインド処理の過程として、付与されたパッケージに対する特権を持ちたい場合には、 BIND を使用しなければなりません。これには SQL_GRANT_OPT オプションがあるからです。
- パッケージが現在ではデータベース内に存在していない場合。
- すべての バインド・エラーを検出することが必要な場合。 REBIND の場合、検出された最初のエラーだけが戻され、その後終了してしまいます。それに対し、BIND コマンドはバインド処理中に発生した最初の 100 エラーを戻してきます。

REBIND は DB2 Connect によってサポートされています。

他のユーザーが使用中のパッケージ上で REBIND を実行しても、そのユーザーの論理作業単位が終了するまでは、再バインドは行われません。再バインド中、排他ロックが SYSCAT.PACKAGES システム・カタログ表にあるそのパッケージのレコードに保留されているからです。

REBIND を実行すると、データベース・マネージャーによって、SYSCAT.STATEMENTS システム・カタログ表に保管されている SQL ステートメントを基にしたパッケージが再作成されます。

同じパッケージ番号と作成者を持つバージョンが多く存在する場合は、 1 つのバージョンのみが一度にバインドされます。 SQL_OPT_VERSION 再 BIND オプションを使用して指定されない場合、 VERSION のデフォルトは "" となります。再バインド要求で指定された名前と作成者に一致する名前および作成者を持つパッケージが 1 つのみであったとしても、 VERSION が明示的または暗黙的に指定された VERSION と一致しない場合は、再バインドされません。

REBIND を実行してエラーが発生した場合、処理は停止し、エラー・メッセージが戻されます。

SQL_EXPLSNAP_OPT か SQL_EXPLAIN_OPT が YES または ALL (カタログ内の EXPLAIN_SNAPSHOT と EXPLAIN_MODE 列を調べてください) に設定されていれば、 REBIND が実行されている間、その Explain 表を使用することができます。使用される Explain 表は、REBIND を要求したユーザーのものであり、最初にバインドを実行したユーザーのものではありません。

再 BIND オプションのタイプおよび値は `sql.h` で定義されています。

関連タスク:

- 「*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*」の『REXX における SQLEXEC、SQLDBS、および SQLDB2 の登録』

関連資料:

- 294 ページの『sqlabndx - バインド』
- 453 ページの『SQLCA』
- 493 ページの『SQLOPT』

- 「コマンド・リファレンス」の『REBIND コマンド』
- 「コマンド・リファレンス」の『db2rbind - すべてのパッケージの再バインド・コマンド』
- 267 ページの『db2Runstats - 統計の実行』

関連サンプル:

- 『dbpkg.sqc -- How to work with packages (C)』
- 『dbsample.sqc -- Creates a sample database (C)』
- 『dbpkg.sqC -- How to work with packages (C++)』
- 『rebind.sqb -- How to rebind a package (IBM COBOL)』

sqlbctcq - 表スペース・コンテナー照会のクローズ

表スペース・コンテナーの照会要求を終了し、関連したリソースを解放します。

許可:

以下のいずれかです。

- *sysadm*
- *sysctrl*
- *sysmaint*
- *dbadm*

必要な接続:

データベース

API 組み込みファイル:

sqlutil.h

C API 構文:

```
/* File: sqlutil.h */
/* API: sqlbctcq */
/* ... */
SQL_API_RC SQL_API_FN
  sqlbctcq (
    struct sqlca *pSqlca);
/* ... */
```

汎用 API 構文:

```
/* File: sqlutil.h */
/* API: sqlgctcq */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgctcq (
    struct sqlca *pSqlca);
/* ... */
```

API パラメーター:

pSqlca
出力。*sqlca* 構造へのポインター。

関連資料:

- 315 ページの『sqlbotcq - 表スペース・コンテナー照会のオープン』
- 308 ページの『sqlbftcq - 表スペース・コンテナー照会の取り出し』
- 324 ページの『sqlbtcq - 表スペース・コンテナー照会』
- 321 ページの『sqlbstsc - 表スペース・コンテナーの設定』
- 453 ページの『SQLCA』

関連サンプル:

- 『tabscont.sqb -- How to get tablespace container information (IBM COBOL)』
- 『tspace.sqb -- How to copy and free memory in a tablespace (IBM COBOL)』
- 『tsinfo.sqc -- How to get information at the table space level (C)』
- 『tsinfo.sqC -- How to get information at the table space level (C++)』

sqlbctsq - 表スペース照会のクローズ

表スペースの照会要求を終了し、関連したリソースを解放します。

許可:

以下のいずれかです。

- *sysadm*
- *sysctrl*
- *sysmaint*
- *dbadm*
- *load*

必要な接続:

データベース

API 組み込みファイル:

sqlutil.h

C API 構文:

```
/* File: sqlutil.h */
/* API: sqlbctsq */
/* ... */
SQL_API_RC SQL_API_FN
  sqlbctsq (
    struct sqlca *pSqlca);
/* ... */
```

汎用 API 構文:

```
/* File: sqlutil.h */
/* API: sqlgctsq */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgctsq (
    struct sqlca *pSqlca);
/* ... */
```

API パラメーター:

pSqlca
出力。*sqlca* 構造へのポインター。

関連資料:

- 317 ページの『sqlbotsq - 表スペース照会のオープン』
- 310 ページの『sqlbftpq - 表スペース照会の取り出し』
- 313 ページの『sqlbmtsq - 表スペース照会』
- 311 ページの『sqlbgtss - 表スペース統計の入手』
- 320 ページの『sqlbstpq - 単一の表スペース照会』
- 453 ページの『SQLCA』

関連サンプル:

- 『tabspace.sqb -- How to get tablespace information (IBM COBOL)』
- 『tspace.sqb -- How to copy and free memory in a tablespace (IBM COBOL)』
- 『tsinfo.sqc -- How to get information at the table space level (C)』
- 『tsinfo.sqC -- How to get information at the table space level (C++)』

sqlbftcq - 表スペース・コンテナー照会の取り出し

指定された行数の表スペース・コンテナーの照会データを取り出します。各行はコンテナー用のデータで構成されます。

有効範囲:

パーティション・データベース環境では、現行のデータベース・パーティション上の表スペースだけがリストされます。

許可:

以下のいずれかです。

- *sysadm*
- *sysctrl*
- *sysmaint*
- *dbadm*

必要な接続:

データベース

API 組み込みファイル:

sqlutil.h

C API 構文:

```
/* File: sqlutil.h */
/* API: sqlbftcq */
/* ... */
SQL_API_RC SQL_API_FN
  sqlbftcq (
    struct sqlca *pSqlca,
```

```
    sqluint32 MaxContainers,
    struct SQLB_TBSCONTQRY_DATA *pContainerData,
    sqluint32 *pNumContainers);
/* ... */
```

汎用 API 構文:

```
/* File: sqlutil.h */
/* API: sqlgftcq */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgftcq (
    struct sqlca *pSqlca,
    sqluint32 MaxContainers,
    struct SQLB_TBSCONTQRY_DATA *pContainerData,
    sqluint32 *pNumContainers);
/* ... */
```

API パラメーター:

pSqlca
出力。*sqlca* 構造へのポインター。

MaxContainers
入力。ユーザー割り当ての出力域 (*pContainerData* によって指される) が保留できるデータの最大行数です。

pContainerData
出力。データを照会するための構造である、出力域を指すポインター。この構造の詳細については、SQLB-TBSCONTQRY-DATA を参照してください。この API の呼び出し側は、スペースをこれらの構造の *MaxContainers* に割り振るとともに、 *pContainerData* をこのスペースを指すように設定する必要があります。 API はこのスペースを使用して、表スペース・コンテナーのデータを戻します。

pNumContainers
出力。戻される出力の行数を示します。

使用上の注意:

ユーザーには、 *pContainerData* パラメーターによって指されるメモリーを割り振ったり解放したりする責任があります。この API を使用できるのは、sqlbotcq 呼び出しが正常に行われた後だけです。この API を繰り返し呼び出して、sqlbotcq が生成するリストを取り出すこともできます。

関連資料:

- 315 ページの『sqlbotcq - 表スペース・コンテナー照会のオープン』
- 306 ページの『sqlbctcq - 表スペース・コンテナー照会のクローズ』
- 324 ページの『sqlbtcq - 表スペース・コンテナー照会』
- 321 ページの『sqlbstsc - 表スペース・コンテナーの設定』
- 453 ページの『SQLCA』
- 448 ページの『SQLB-TBSCONTQRY-DATA』

関連サンプル:

- 『tabscont.sqb -- How to get tablespace container information (IBM COBOL)』

- 『tspace.sqb -- How to copy and free memory in a tablespace (IBM COBOL)』
- 『tsinfo.sqc -- How to get information at the table space level (C)』
- 『tsinfo.sqC -- How to get information at the table space level (C++)』

sqlbftpq - 表スペース照会の取り出し

指定された行数の表スペースの照会データを取り出します。各行は表スペース用のデータで構成されます。

有効範囲:

パーティション・データベース環境では、現行のデータベース・パーティション上の表スペースだけがリストされます。

許可:

以下のいずれかです。

- *sysadm*
- *sysctrl*
- *sysmaint*
- *dbadm*
- *load*

必要な接続:

データベース

API 組み込みファイル:

sqlutil.h

C API 構文:

```
/* File: sqlutil.h */
/* API: sqlbftpq */
/* ... */
SQL_API_RC SQL_API_FN
  sqlbftpq (
    struct sqlca *pSqlca,
    sqluint32 MaxTablespaces,
    struct SQLB_TBSPQRY_DATA *pTablespaceData,
    sqluint32 *pNumTablespaces);
/* ... */
```

汎用 API 構文:

```
/* File: sqlutil.h */
/* API: sqlgftpq */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgftpq (
    struct sqlca *pSqlca,
    sqluint32 MaxTablespaces,
    struct SQLB_TBSPQRY_DATA *pTablespaceData,
    sqluint32 *pNumTablespaces);
/* ... */
```

API パラメーター:

pSqlca
出力。*sqlca* 構造へのポインター。

MaxTablespaces
入力。ユーザー割り当ての出力域 (*pTablespaceData* によって指される) が保留できるデータの最大行数です。

pTablespaceData
入力/出力。データを照会するための構造である、出力域を指すポインター。この構造の詳細については、SQLB-TBSPQRY-DATA を参照してください。この API の呼び出し側は、以下のことを行う必要があります。

- スペースをこれらの構造の *MaxTablespaces* に割り振る。
- 構造を初期化する。
- 最初の構造の TBSPQVER を SQLB_TBSPQRY_DATA_ID に設定する。
- *pTablespaceData* がこのスペースを指すように設定する。 API はこのスペースを使用して、表スペースのデータを戻します。

pNumTablespaces
出力。戻される出力の行数を示します。

使用上の注意:

ユーザーには、 *pTablespaceData* パラメーターによって指されるメモリーを割り振ったり解放したりする責任があります。この API を使用できるのは、sqlbotsq 呼び出しが正常に行われた後だけです。この API を繰り返し呼び出して、sqlbotsq が生成するリストを取り出すこともできます。

関連資料:

- 317 ページの『sqlbotsq - 表スペース照会のオープン』
- 307 ページの『sqlbctsq - 表スペース照会のクローズ』
- 313 ページの『sqlbmtsq - 表スペース照会』
- 311 ページの『sqlbgtss - 表スペース統計の入手』
- 320 ページの『sqlbstpq - 単一の表スペース照会』
- 453 ページの『SQLCA』
- 449 ページの『SQLB-TBSPQRY-DATA』

関連サンプル:

- 『tabspace.sqb -- How to get tablespace information (IBM COBOL)』
- 『tspace.sqb -- How to copy and free memory in a tablespace (IBM COBOL)』
- 『tsinfo.sqc -- How to get information at the table space level (C)』
- 『tsinfo.sqC -- How to get information at the table space level (C++)』

sqlbgtss - 表スペース統計の入手

表スペースの使用率に関する情報を提供します。

有効範囲:

パーティション・データベース環境では、現行のデータベース・パーティション上の表スペースだけがリストされます。

許可:

以下のいずれかです。

- *sysadm*
- *sysctrl*
- *sysmaint*
- *dbadm*
- *load*

必要な接続:

データベース

API 組み込みファイル:

sqlutil.h

C API 構文:

```
/* File: sqlutil.h */
/* API: sqlbgtss */
/* ... */
SQL_API_RC SQL_API_FN
  sqlbgtss (
    struct sqlca *pSqlca,
    sqluint32 TablespaceId,
    struct SQLB_TBS_STATS *pTablespaceStats);
/* ... */
```

汎用 API 構文:

```
/* File: sqlutil.h */
/* API: sqlggtss */
/* ... */
SQL_API_RC SQL_API_FN
  sqlggtss (
    struct sqlca *pSqlca,
    sqluint32 TablespaceId,
    struct SQLB_TBS_STATS *pTablespaceStats);
/* ... */
```

API パラメーター:

pSqlca
出力。*sqlca* 構造へのポインター。

TablespaceId
入力。照会される単一の表スペースの ID を示します。

pTablespaceStats
出力。ユーザー割り当ての *SQLB_TBS_STATS* 構造を指すポインター。表スペースに関する情報は、この構造に戻されます。

使用上の注意:

戻されるフィールドとそれらの意味については、SQLB-TBS-STATS を参照してください。

関連資料:

- 317 ページの『sqlbotsq - 表スペース照会のオープン』
- 310 ページの『sqlbftpq - 表スペース照会の取り出し』
- 307 ページの『sqlbctsq - 表スペース照会のクローズ』
- 313 ページの『sqlbmtsq - 表スペース照会』
- 320 ページの『sqlbstpq - 単一の表スペース照会』
- 453 ページの『SQLCA』
- 447 ページの『SQLB-TBS-STATS』

関連サンプル:

- 『tabspace.sqb -- How to get tablespace information (IBM COBOL)』
- 『tspace.sqb -- How to copy and free memory in a tablespace (IBM COBOL)』
- 『tsinfo.sqc -- How to get information at the table space level (C)』
- 『tsinfo.sqC -- How to get information at the table space level (C++)』

sqlbmtsq - 表スペース照会

表スペース照会データへの 1 回呼び出しインターフェースを提供します。データベース内のすべての表スペースの照会データが、1 つの配列に戻されます。

有効範囲:

パーティション・データベース環境では、現行のデータベース・パーティション上の表スペースだけがリストされます。

許可:

以下のいずれかです。

- *sysadm*
- *sysctrl*
- *sysmaint*
- *dbadm*
- *load*

必要な接続:

データベース

API 組み込みファイル:

sqlutil.h

C API 構文:

```
/* File: sqlutil.h */
/* API: sqlbmtsq */
/* ... */
SQL_API_RC SQL_API_FN
  sqlbmtsq (
    struct sqlca *pSqlca,
    sqluint32 *pNumTablespaces,
```

```
    struct SQLB_TBSPQRY_DATA ***pppTablespaceData,
    sqluint32 reserved1,
    sqluint32 reserved2);
/* ... */
```

汎用 API 構文:

```
/* File: sqlutil.h */
/* API: sqlgmtsq */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgmtsq (
    struct sqlca *pSqlca,
    sqluint32 *pNumTablespaces,
    struct SQLB_TBSPQRY_DATA ***pppTablespaceData,
    sqluint32 reserved1,
    sqluint32 reserved2);
/* ... */
```

API パラメーター:

pSqlca
出力。*sqlca* 構造へのポインター。

pNumTablespaces
出力。接続されているデータベース内にある表スペースの合計数を示します。

pppTablespaceData
出力。呼び出し側が API にポインターのアドレスを提供します。表スペース照会データのスペースは API によって割り振られ、そのスペースを指すポインターが呼び出し側に戻されます。ポインターは、呼び出しから戻る際、表スペース照会データの完全な集合へのポインターである *SQLB_TBSPQRY_DATA* の配列を指します。

reserved1
入力。常に SQLB_RESERVED1 です。

reserved2
入力。常に SQLB_RESERVED2 です。

使用上の注意:

この API は、より低レベルなサービス、つまり以下の 3 つの API を利用します。

- sqlbotsq
- sqlbftpq
- sqlbctsq

それにより、表スペース照会データのすべてを一度に入手できます。

十分な量のメモリーが利用可能であれば、この関数は表スペースの数を戻すとともに、表スペース照会データのメモリー・ロケーションを指すポインターも戻します。 sqlefmem を呼び出してこのメモリーを解放するのは、ユーザーの責任です。

十分な量のメモリーを利用できない場合、この関数は表スペースの数を戻すだけで、メモリーは割り振られません。そうした事態が考えられる場合には、sqlbotsq、sqlbftpq、および sqlbctsq を使用して、完全ではないそのリストを直ちに取り出してください。

関連資料:

- 317 ページの『sqlbotsq - 表スペース照会のオープン』
- 310 ページの『sqlbftpq - 表スペース照会の取り出し』
- 307 ページの『sqlbctsq - 表スペース照会のクローズ』
- 311 ページの『sqlbgtss - 表スペース統計の入手』
- 320 ページの『sqlbstpq - 単一の表スペース照会』
- 371 ページの『sqlefmem - メモリーの解放』
- 453 ページの『SQLCA』

関連サンプル:

- 『dbrecov.sqc -- How to recover a database (C)』
- 『tsinfo.sqc -- How to get information at the table space level (C)』
- 『dbrecov.sqC -- How to recover a database (C++)』
- 『tsinfo.sqC -- How to get information at the table space level (C++)』
- 『tspace.sqb -- How to copy and free memory in a tablespace (IBM COBOL)』

sqlbotcq - 表スペース・コンテナー照会のオープン

表スペース・コンテナーの照会操作を実行し、現在、表スペースにあるコンテナーの数を戻します。

許可:

以下のいずれかです。

- *sysadm*
- *sysctrl*
- *sysmaint*
- *dbadm*

必要な接続:

データベース

API 組み込みファイル:

sqlutil.h

C API 構文:

```
/* File: sqlutil.h */
/* API: sqlbotcq */
/* ... */
SQL_API_RC SQL_API_FN
  sqlbotcq (
    struct sqlca *pSqlca,
    sqluint32 TablespaceId,
    sqluint32 *pNumContainers);
/* ... */
```

汎用 API 構文:

```
/* File: sqlutil.h */
/* API: sqlgotcq */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgotcq (
    struct sqlca *pSqlca,
    sqluint32 TablespaceId,
    sqluint32 *pNumContainers);
/* ... */
```

API パラメーター:

pSqlca
出力。*sqlca* 構造へのポインター。

TablespaceId
入力。コンテナー・データを必要とする表スペースの ID です。特殊な ID SQLB_ALL_TABLESPACES (sqlutil.h で) を指定すると、データベース全体を包含するコンテナーの完全なリストが作成されます。

pNumContainers
出力。指定した表スペース内のコンテナーの数です。

使用上の注意:

通常、この API の後には sqlbftcq が 1 回以上呼び出され、その後 sqlbctcq が 1 回呼び出されます。

アプリケーションは以下に挙げる API を呼び出して、表スペースで使用しているコンテナーに関する情報を取り出すことができます。

- sqlbtcq

  コンテナー情報の完全なリストを取り出します。この API は、全コンテナーの情報を保留するのに必要なスペースを割り振り、この情報を指すポインターを戻します。この API を使用して、コンテナーのリストをスキャンし、特定の情報を入手することもできます。この API を使用することは、以下の 3 つの API (sqlbotcq、 sqlbftcq、および sqlbctcq) を呼び出すのとほぼ同じです。異なるのは、この API の場合、出力情報用のメモリーが自動的に割り振られるという点です。この API を呼び出した後は、必ず sqlefmem を呼び出して、メモリーを解放する必要があります。

- sqlbotcq
- sqlbftcq
- sqlbctcq

  前述の 3 つの API は、SQL カーソルと同様の働きをします。つまり、OPEN/FETCH/CLOSE パラダイムを使用します。呼び出し元は、取り出しのための出力域を提供する必要があります。 SQL カーソルの場合とは異なり、一度にアクティブにできる表スペース・コンテナーの照会の数は 1 つだけです。この API の集合を使用して、表スペース・コンテナーのリストをスキャンし、特定の情報を入手することもできます。これらの API を使用することにより、アプリケーションのメモリー要件を制御することができます (sqlbtcq と比較)。

sqlbotcq を呼び出すと、現行コンテナー情報のスナップショットがアプリケーションを保守するエージェントに形成されます。アプリケーションによって 2 回目の表スペース・コンテナーの照会呼び出し (sqlbtcq または sqlbotcq) が発行された場合、このスナップショットは更新された情報に置き換えられます。

ロッキングは実行されないので、スナップショットの生成後に別のアプリケーションによって加えられた変更は、バッファー内の情報に反映されない可能性もあります。この情報はトランザクションの一部ではありません。

スナップショット・バッファーには、表スペースの照会用のものと表スペース・コンテナーの照会用のものとがあります。それらのバッファーは互いに独立したものです。

関連資料:

- 308 ページの『sqlbftcq - 表スペース・コンテナー照会の取り出し』
- 306 ページの『sqlbctcq - 表スペース・コンテナー照会のクローズ』
- 324 ページの『sqlbtcq - 表スペース・コンテナー照会』
- 321 ページの『sqlbstsc - 表スペース・コンテナーの設定』
- 371 ページの『sqlefmem - メモリーの解放』
- 453 ページの『SQLCA』

関連サンプル:

- 『tabscont.sqb -- How to get tablespace container information (IBM COBOL)』
- 『tspace.sqb -- How to copy and free memory in a tablespace (IBM COBOL)』
- 『tsinfo.sqc -- How to get information at the table space level (C)』
- 『tsinfo.sqC -- How to get information at the table space level (C++)』

sqlbotsq - 表スペース照会のオープン

表スペースの照会操作を実行し、現在データベースにある表スペースの数を戻します。

許可:

以下のいずれかです。

- *sysadm*
- *sysctrl*
- *sysmaint*
- *dbadm*
- *load*

必要な接続:

データベース

API 組み込みファイル:

sqlutil.h

C API 構文:

```
/* File: sqlutil.h */
/* API: sqlbotsq */
/* ... */
SQL_API_RC SQL_API_FN
  sqlbotsq (
    struct sqlca *pSqlca,
    sqluint32 TablespaceQueryOptions,
    sqluint32 *pNumTablespaces);
/* ... */
```

汎用 API 構文:

```
/* File: sqlutil.h */
/* API: sqlgotsq */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgotsq (
    struct sqlca *pSqlca,
    sqluint32 TablespaceQueryOptions,
    sqluint32 *pNumTablespaces);
/* ... */
```

API パラメーター:

pSqlca
出力。*sqlca* 構造へのポインター。

TablespaceQueryOptions
入力。処理する表スペースを示します。有効な値 (sqlutil で定義) は、以下のとおりです。

SQLB_OPEN_TBS_ALL
データベースにあるすべての表スペースを処理します。

SQLB_OPEN_TBS_RESTORE
ユーザーのエージェントがリストアする表スペースだけを処理します。

pNumTablespaces
出力。接続されているデータベース内の表スペースの数を示します。

使用上の注意:

通常、この API の後には sqlbftpq が 1 回以上呼び出され、その後 sqlbctsq が 1 回呼び出されます。

アプリケーションは以下に挙げる API を呼び出して、現在定義されている表スペースに関する情報を取り出すことができます。

- sqlbstpq

  指定された表スペースに関する情報を取り出します。ただ 1 つの表スペース項目だけが (呼び出し側で提供したスペースに) 戻されます。表スペースの ID がわかっている場合に、この API を使用してください。また、必要なのはその表スペースに関する情報だけです。

- sqlbmtsq

  すべての表スペースに関する情報を取り出します。この API は、全表スペースの情報を保留するのに必要なスペースを割り振り、この情報を指すポインターを戻

します。この API を使用して、特定の情報の探索時に、表スペースのリストをスキャンすることもできます。この API を使用することは、以下に挙げる 3 つの API を呼び出すのとほぼ同じです。異なるのは、この API の場合、出力情報用のメモリーが自動的に割り振られるという点です。この API を呼び出した後は、必ず sqlefmem を呼び出して、メモリーを解放する必要があります。

- sqlbotsq
- sqlbftpq
- sqlbctsq

  前述の 3 つの API は、SQL カーソルと同様の働きをします。つまり、OPEN/FETCH/CLOSE パラダイムを使用します。呼び出し元は、取り出しのための出力域を提供する必要があります。 SQL カーソルの場合とは異なり、一度にアクティブにできる表スペースの照会の数は 1 つだけです。この API の集合を使用して、特定の情報の探索時に、表スペースのリストをスキャンすることもできます。これらの API の集合を使用することにより、アプリケーションのメモリー要件を制御することができます (sqlbmtsq と比較)。

sqlbotsq を呼び出すと、現行の表スペース情報のスナップショットが、アプリケーションを保守するエージェントのバッファーに入れられます。アプリケーションによって 2 回目の表スペースの照会呼び出し (sqlbmtsq または sqlbotsq) が発行された場合、このスナップショットは更新された情報で置き換えられます。

ロッキングは実行されないので、別のアプリケーションによってその後に加えられた変更は、バッファー内の情報に反映されない可能性もあります。この情報はトランザクションの一部ではありません。

スナップショット・バッファーには、表スペースの照会用のものと表スペース・コンテナーの照会用のものとがあります。それらのバッファーは互いに独立したものです。

関連資料:

- 310 ページの『sqlbftpq - 表スペース照会の取り出し』
- 307 ページの『sqlbctsq - 表スペース照会のクローズ』
- 313 ページの『sqlbmtsq - 表スペース照会』
- 320 ページの『sqlbstpq - 単一の表スペース照会』
- 371 ページの『sqlefmem - メモリーの解放』
- 453 ページの『SQLCA』

関連サンプル:

- 『tabspace.sqb -- How to get tablespace information (IBM COBOL)』
- 『tspace.sqb -- How to copy and free memory in a tablespace (IBM COBOL)』
- 『tsinfo.sqc -- How to get information at the table space level (C)』
- 『tsinfo.sqC -- How to get information at the table space level (C++)』

sqlbstpq - 単一の表スペース照会

現在定義されている単一の表スペースに関する情報を検索します。

有効範囲:

パーティション・データベース環境では、現行のデータベース・パーティション上の表スペースだけがリストされます。

許可:

以下のいずれかです。

- *sysadm*
- *sysctrl*
- *sysmaint*
- *dbadm*
- *load*

必要な接続:

データベース

API 組み込みファイル:

sqlutil.h

C API 構文:

```
/* File: sqlutil.h */
/* API: sqlbstpq */
/* ... */
SQL_API_RC SQL_API_FN
  sqlbstpq (
    struct sqlca *pSqlca,
    sqluint32 TablespaceId,
    struct SQLB_TBSPQRY_DATA *pTablespaceData,
    sqluint32 reserved);
/* ... */
```

汎用 API 構文:

```
/* File: sqlutil.h */
/* API: sqlgstpq */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgstpq (
    struct sqlca *pSqlca,
    sqluint32 TablespaceId,
    struct SQLB_TBSPQRY_DATA *pTablespaceData,
    sqluint32reserved);
/* ... */
```

API パラメーター:

pSqlca
出力。*sqlca* 構造へのポインター。

TablespaceId
入力。照会する表スペースの ID です。

pTablespaceData
入力/出力。表スペース情報が戻される、ユーザー提供の *SQLB_TBSPQRY_DATA* 構造を指すポインター。この API の呼び出し側は、構造を初期化し、 TBSPQVER を (sqlutil にある) SQLB_TBSPQRY_DATA_ID に設定する必要があります。

reserved
入力。常に SQLB_RESERVED1 です。

使用上の注意:

この API は、照会したい表スペースの ID が分かっている場合に、単一の表スペースに関する情報を検索します。この API は、OPEN TABLESPACE QUERY、FETCH、および CLOSE API の組み合わせの代替手段といえます。この API は、これら 3 つの API よりも、パフォーマンスの点で優れています。目的の表スペースの ID が事前に分からない場合には、この API ではなく、これら 3 つの API の組み合わせを使用して、その表スペースをスキャンします。表スペース ID は、システム・カタログから探し出すことができます。戻される項目が 1 つだけなので、エージェントのスナップショットは取られません。その項目が直接戻されます。

詳細については、『sqlbotsq』を参照してください。

関連資料:

- 317 ページの『sqlbotsq - 表スペース照会のオープン』
- 310 ページの『sqlbftpq - 表スペース照会の取り出し』
- 307 ページの『sqlbctsq - 表スペース照会のクローズ』
- 313 ページの『sqlbmtsq - 表スペース照会』
- 311 ページの『sqlbgtss - 表スペース統計の入手』
- 453 ページの『SQLCA』

関連サンプル:

- 『tabspace.sqb -- How to get tablespace information (IBM COBOL)』
- 『tspace.sqb -- How to copy and free memory in a tablespace (IBM COBOL)』
- 『tsinfo.sqc -- How to get information at the table space level (C)』
- 『tsinfo.sqC -- How to get information at the table space level (C++)』

sqlbstsc - 表スペース・コンテナーの設定

この API は、ユーザーがデータベースをリストアするよう*リダイレクトされた* リストアの実行を容易にします。オペレーティング・システムの記憶域コンテナーの異なるセットが必要になります。

表スペースが*記憶域定義ペンディング* または*記憶域定義許可済み* 状態の場合に、この API を使用してください。これらの状態はリストア操作中、データベース・ページをリストアする直前に起こり得るものです。

許可:

以下のいずれかです。

- *sysadm*
- *sysctrl*

必要な接続:

データベース

API 組み込みファイル:

sqlutil.h

C API 構文:

```
/* File: sqlutil.h */
/* API: sqlbstsc */
/* ... */
SQL_API_RC SQL_API_FN
  sqlbstsc (
    struct sqlca *pSqlca,
    sqluint32 SetContainerOptions,
    sqluint32 TablespaceId,
    sqluint32 NumContainers,
    struct SQLB_TBSCONTQRY_DATA *pContainerData);
/* ... */
```

汎用 API 構文:

```
/* File: sqlutil.h */
/* API: sqlgstsc */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgstsc (
    struct sqlca *pSqlca,
    sqluint32 SetContainerOptions,
    sqluint32 TablespaceId,
    sqluint32 NumContainers,
    struct SQLB_TBSCONTQRY_DATA *pContainerData);
/* ... */
```

API パラメーター:

pSqlca
出力。*sqlca* 構造へのポインター。

SetContainerOptions
入力。このフィールドは追加のオプションを指定するために使用します。有効な値 (sqlutil で定義) は、以下のとおりです。

SQLB_SET_CONT_INIT_STATE
ロールフォワードの実行時に、表スペースの更新操作を再実行します。

SQLB_SET_CONT_FINAL_STATE
ロールフォワードの実行時に、ログにおける表スペースの更新操作を無視します。

TablespaceId
入力。変更する表スペースの ID です。

NumContainers
入力。 *pContainerData* によって指される構造が保留する行数を示します。

pContainerData
入力。コンテナーの仕様を示します。 *SQLB_TBSCONTQRY_DATA* 構造が使

用されますが、 *contType*、*totalPages*、 *name*、および *nameLen* (C 以外の言語用) フィールドだけが使用されます。他のすべてのフィールドは無視されます。

使用上の注意:

この API は db2Restore とともに使用されます。

データベース、または 1 つまたは複数の表スペースのバックアップを取ることにより、バックアップしようとする表スペースが使用しているすべての表スペース・コンテナーのレコードを保持できます。リストア中、バックアップにリストされたすべてのコンテナーについて、現在存在しているかどうか、およびアクセス可能であるかどうかがチェックされます。何らかの理由で 1 つでもアクセス不能なコンテナーがあると、リストアは失敗してしまいます。そのような場合であってもリストアを行えるようにするため、リストア中は表スペース・コンテナーのリダイレクトがサポートされています。このサポートには、表スペース・コンテナーの追加、変更、もしくは除去が含まれます。そのようなコンテナーをユーザーが追加、変更、または除去できるようにするのが、この API です。

通常、この API は以下のような順序のアクションで使用します。

1. *CallerAction* を SQLUD_RESTORE_STORDEF に設定して、db2Restore を呼び出す。

   リストア・ユーティリティーによって、コンテナーの一部がアクセス不能であることを示す *sqlcode* が戻されます。
2. sqlbstsc を呼び出して、表スペース・コンテナーの定義を設定する。*SetContainerOptions* パラメーターは SQLB_SET_CONT_FINAL_STATE に設定します。
3. *CallerAction* を SQLUD_CONTINUE に設定して、もう一度 sqlurst を呼び出す。

上記の手順により、新規の表スペース・コンテナーの定義をリストアに使えるようになります。また、リストアの完了後に db2Rollforward を呼び出すと、ログにある表スペース・コンテナーの追加操作は無視されます。

この API を使用する際には、コンテナー・リストの設定時に、リストアまたはロールフォワードが元のデータをすべてそれら新規のコンテナーに置き換えられるよう十分なディスク・スペースが必要であることに注意してください。十分なスペースがないと、それらの表スペースは、十分なディスク・スペースが利用可能になるまで、 *recovery pending* 状態のままになります。賢明なデータベース管理者であれば、ディスク使用率のレコードを保持しているはずです。その後、リストアまたはロールフォワード操作が必要な場合は、必要なディスク・スペースが認識されます。

関連資料:

- 257 ページの『db2Rollforward - データベースのロールフォワード』
- 453 ページの『SQLCA』
- 28 ページの『db2Backup - データベースのバックアップ』
- 244 ページの『db2Restore - データベースのリストア』

関連サンプル:

- 『dbrecov.sqc -- How to recover a database (C)』
- 『dbrecov.sqC -- How to recover a database (C++)』
- 『tabscont.sqb -- How to get tablespace container information (IBM COBOL)』
- 『tspace.sqb -- How to copy and free memory in a tablespace (IBM COBOL)』

sqlbtcq - 表スペース・コンテナー照会

表スペース・コンテナー照会データへの 1 回呼び出しインターフェースを提供します。特定の表スペース内のすべてのコンテナー、またはすべての表スペース内のすべてのコンテナーの照会データが、 1 つの配列に戻されます。

有効範囲:

パーティション・データベース環境では、現行のデータベース・パーティション上の表スペースだけがリストされます。

許可:

以下のいずれかです。

- *sysadm*
- *sysctrl*
- *sysmaint*
- *dbadm*

必要な接続:

データベース

API 組み込みファイル:

sqlutil.h

C API 構文:

```
/* File: sqlutil.h */
/* API: sqlbtcq */
/* ... */
SQL_API_RC SQL_API_FN
  sqlbtcq (
    struct sqlca *pSqlca,
    sqluint32 TablespaceId,
    sqluint32 *pNumContainers,
    struct SQLB_TBSCONTQRY_DATA **ppContainerData);
/* ... */
```

汎用 API 構文:

```
/* File: sqlutil.h */
/* API: sqlgtcq */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgtcq (
    struct sqlca *pSqlca,
```

```
    sqluint32 TablespaceId,
    sqluint32 *pNumContainers,
    struct SQLB_TBSCONTQRY_DATA **ppContainerData);
/* ... */
```

API パラメーター:

pSqlca
出力。*sqlca* 構造へのポインター。

TablespaceId
入力。コンテナー・データを必要とする表スペースの ID を示します。特殊な ID SQLB_ALL_TABLESPACES (sqlutil で定義) を指定すると、データベース全体を包含する全コンテナーのリストが作成されます。

pNumContainers
出力。表スペース内のコンテナーの数を示します。

ppContainerData
出力。呼び出し側が API を、 *SQLB_TBSCONTQRY_DATA* 構造を指すポインターのアドレスで提供します。表スペース・コンテナー照会データのスペースは API によって割り振られ、そのスペースを指すポインターが呼び出し側に戻されます。呼び出しの戻りで、 *SQLB_TBSCONTQRY_DATA* 構造を指すポインターが、表スペース・コンテナー照会データの完全な集合を指します。

使用上の注意:

この API は、より低レベルなサービス、つまり以下の 3 つの API を利用します。

- sqlbotcq
- sqlbftcq
- sqlbctcq

それにより、表スペース・コンテナー照会データのすべてを一度に入手できます。

十分な量のメモリーが利用可能であれば、この関数はコンテナーの数を戻すとともに、表スペース・コンテナー照会データのメモリー・ロケーションを指すポインターも戻します。 sqlefmem を呼び出してこのメモリーを解放するのは、ユーザーの責任です。

十分な量のメモリーを利用できない場合、この関数はコンテナーの数を戻すだけで、メモリーは割り振られません。そうした事態が考えられる場合には、sqlbotcq、sqlbftcq、および sqlbctcq を使用して、完全ではないそのリストをすぐに取り出してください。

関連資料:

- 315 ページの『sqlbotcq - 表スペース・コンテナー照会のオープン』
- 308 ページの『sqlbftcq - 表スペース・コンテナー照会の取り出し』
- 306 ページの『sqlbctcq - 表スペース・コンテナー照会のクローズ』
- 321 ページの『sqlbstsc - 表スペース・コンテナーの設定』
- 371 ページの『sqlefmem - メモリーの解放』
- 453 ページの『SQLCA』

関連サンプル:

- 『dbrecov.sqc -- How to recover a database (C)』
- 『tsinfo.sqc -- How to get information at the table space level (C)』
- 『dbrecov.sqC -- How to recover a database (C++)』
- 『tsinfo.sqC -- How to get information at the table space level (C++)』
- 『tabscont.sqb -- How to get tablespace container information (IBM COBOL)』
- 『tspace.sqb -- How to copy and free memory in a tablespace (IBM COBOL)』

---

sqlcspqy - DRDA 未確定トランザクションのリスト

LU 6.2 プロトコルで接続されたパートナー LU 間で、未確定のトランザクションのリストを提供します。

許可:

sysadm

必要な接続:

インスタンス。

API 組み込みファイル:

sqlxa.h

C API 構文:

```
/* File: sqlxa.h */
/* API: sqlcspqy */
/* ... */
extern int SQL_API_FN sqlcspqy(SQLCSPQY_INDOUBT     **indoubt_data,
                               sqlint32             *indoubt_count,
                               struct sqlca          *sqlca);
/* ... */
```

API パラメーター:

indoubt_data
出力。戻された配列を指すポインター。

indoubt_count
出力。戻された配列内のエレメント数を示します。

pSqlca
出力。*sqlca* 構造へのポインター。

使用上の注意:

分散作業単位内でコーディネーターと参加者の間に通信がなくなるときに、 DRDA 未確定トランザクションが発生します。

分散作業単位により、ユーザーまたはアプリケーションは、単一作業単位内の複数のロケーションにあるデータを読み取ったり更新したりすることができます。ただし、そのような作業には、2 フェーズ・コミットが必要です。

最初のフェーズで、すべての参加者がコミットの準備をするよう要求します。次のフェーズで、トランザクションのコミットまたはロールバックを行います。最初のフェーズの後にコーディネーターか参加者が使用できなくなると、分散トランザクションが未確定になります。

LIST DRDA INDOUBT TRANSACTIONS を発行する前に、アプリケーション処理は同期点マネージャー (SPM) インスタンスに接続されていなければなりません。SPM_NAME を CONNECT ステートメント上の *dbalias* として使用します。SPM_NAME は、データベース・マネージャーの構成パラメーターです。

関連資料:

- 「*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』
- 「*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』
- 453 ページの『SQLCA』

sqle_activate_db - データベースの活動化

指定されたデータベースを活動化し、必要なデータベースのサービスをすべて開始します。こうして、データベースの接続が使用可能になり、任意のアプリケーションが使用できるようになります。

有効範囲:

この API は、すべてのデータベース・パーティション・サーバーで、指定されたデータベースを活動化します。データベースの活動化中に 1 つまたは複数のデータベース・パーティション・サーバーでエラーが起きた場合、警告が戻されます。データベースは、API が正常に実行したすべてのデータベース・パーティション・サーバー上で活動化状態のままになります。

注: エラーが起きたのがコーディネーター・パーティションかカタログ・パーティションの場合、 API は否定の *sqlcode* を戻し、どのデータベース・パーティション・サーバー上でもデータベースは活動化されません。

許可:

以下のいずれかです。

- *sysadm*
- *sysctrl*
- *sysmaint*

必要な接続:

なし。 ACTIVATE DATABASE を呼び出しているアプリケーションには、既存のデータベース接続は全くありません。

API 組み込みファイル:

sqlenv.h

C API 構文:

```
/* File: sqlenv.h */
/* API: sqle_activate_db */
/* ... */
SQL_API_RC SQL_API_FN
  sqle_activate_db (
    char *pDbAlias,
    char *pUserName,
    char *pPassword,
    void *pReserved,
    struct sqlca *pSqlca);
/* ... */
```

汎用 API 構文:

```
/* File: sqlenv.h */
/* API: sqlg_activate_db */
/* ... */
SQL_API_RC SQL_API_FN
  sqlg_activate_db (
    unsigned short DbAliasLen,
    unsigned short UserNameLen,
    unsigned short PasswordLen,
    char *pDbAlias,
    char *pUserName,
    char *pPassword,
    void *pReserved,
    struct sqlca *pSqlca);
/* ... */
```

API パラメーター:

DbAliasLen
入力。データベース別名の長さを示す 2 バイトの符号なし整数 (バイト単位) です。

UserNameLen
入力。ユーザー名の長さを示す 2 バイトの符号なし整数 (バイト単位) です。ユーザー名が提供されていない場合は、ゼロに設定してください。

PasswordLen
入力。パスワードの長さを示す 2 バイトの符号なし整数 (バイト単位) です。パスワードが提供されていない場合は、ゼロに設定してください。

pDbAlias
入力。データベースの別名を指すポインター。

pUserName
入力。データベースを開始させるユーザー ID を指すポインター。 NULL にすることもできます。

pPassword
入力。ユーザー名用のパスワードを指すポインター。 NULL にすることもできます。しかし、ユーザー名が指定されている場合は、それを指定する必要があります。

pReserved
将来の利用のために予約されています。

pSqlca
出力。*sqlca* 構造へのポインター。

REXX API 構文:

この API は、SQLDB2 インターフェースを使って、REXX から呼び出すことができます。

使用上の注意:

データベースを開始しておらず、アプリケーション内で DB2 CONNECT TO (または暗黙接続) がある場合、アプリケーションはデータベース・マネージャーが必要なデータベースを開始する間待機しなければなりません。このような場合、最初のアプリケーションで作業をする前に、データベースの初期設定で時間がかかってしまいます。しかし、いったん最初のアプリケーションでデータベースを開始したら、他のアプリケーションはデータベースに接続するだけで使用できるようになります。

データベース管理者は、 ACTIVATE DATABASE を使用して選択したデータベースを開始させることができます。こうすると、アプリケーションがデータベースの初期設定で時間を浪費してしまうことを避けられます。

ACTIVATE DATABASE で初期設定されたデータベースは、 sqle_deactivate_db または db2InstanceStop を使用しないとシャットダウンできません。活動化されたデータベースのリストを入手するには、db2GetSnapshot を呼び出してください。

データベースが DB2 CONNECT TO (または暗黙接続) で開始され、続いてその同じデータベースに ACTIVATE DATABASE が発行される場合、そのデータベースを遮断するには DEACTIVATE DATABASE を使用しなければなりません。

再始動させる必要があるデータベース (たとえば、矛盾状態にあるデータベース) での処理時には、 ACTIVATE DATABASE は DB2 CONNECT TO (または暗黙接続) と同じような働きをします。 ACTIVATE DATABASE で初期設定される前に、データベースは再始動します。

関連タスク:

- 「*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*」の『REXX における SQLEXEC、SQLDBS、および SQLDB2 の登録』

関連資料:

- 89 ページの『db2GetSnapshot - スナップショットの入手』
- 329 ページの『sqle_deactivate_db - データベースの非活動化』
- 453 ページの『SQLCA』
- 「*コマンド・リファレンス*」の『ACTIVATE DATABASE コマンド』

sqle_deactivate_db - データベースの非活動化

指定したデータベースを停止させます。

有効範囲:

パーティション・データベース環境では、この API がすべてのデータベース・パーティション・サーバーにある、指定されたデータベースを非活動化します。 1 つま

たは複数のデータベース・パーティション・サーバーでエラーが起きた場合、警告が戻されます。一部のデータベース・パーティション・サーバーではデータベースの非活動化が正常に行われても、エラーが起きたデータベース・パーティション・サーバーではデータベースは活動化されたままになります。

注: エラーが起きたのがコーディネーター・パーティションかカタログ・パーティションの場合、 API は否定の *sqlcode* を戻し、データベースが非活動化されたデータベース・パーティション・サーバー上での再活動化は行われません。

許可:

以下のいずれかです。

- *sysadm*
- *sysctrl*
- *sysmaint*

必要な接続:

なし。 DEACTIVATE DATABASE を呼び出しているアプリケーションには、既存のデータベース接続は全くありません。

API 組み込みファイル:

sqlenv.h

C API 構文:

```
/* File: sqlenv.h */
/* API: sqle_deactivate_db */
/* ... */
SQL_API_RC SQL_API_FN
  sqle_deactivate_db (
    char *pDbAlias,
    char *pUserName,
    char *pPassword,
    void *pReserved,
    struct sqlca *pSqlca);
/* ... */
```

汎用 API 構文:

```
/* File: sqlenv.h */
/* API: sqlg_deactivate_db */
/* ... */
SQL_API_RC SQL_API_FN
  sqlg_deactivate_db (
    unsigned short DbAliasLen,
    unsigned short UserNameLen,
    unsigned short PasswordLen,
    char *pDbAlias,
    char *pUserName,
    char *pPassword,
    void *pReserved,
    struct sqlca *pSqlca);
/* ... */
```

API パラメーター:

DbAliasLen
入力。データベース別名の長さを示す 2 バイトの符号なし整数 (バイト単位) です。

UserNameLen
入力。ユーザー名の長さを示す 2 バイトの符号なし整数 (バイト単位) です。ユーザー名が提供されていない場合は、ゼロに設定してください。

PasswordLen
入力。パスワードの長さを示す 2 バイトの符号なし整数 (バイト単位) です。パスワードが提供されていない場合は、ゼロに設定してください。

pDbAlias
入力。データベースの別名を指すポインター。

pUserName
入力。データベースを停止させるときに使用するユーザー ID を指すポインター。 NULL にすることもできます。

pPassword
入力。ユーザー名用のパスワードを指すポインター。 NULL にすることもできます。しかし、ユーザー名が指定されている場合は、それを指定する必要があります。

pReserved
将来の利用のために予約されています。

pSqlca
出力。*sqlca* 構造へのポインター。

REXX API 構文:

この API は、SQLDB2 インターフェースを使って、REXX から呼び出すことができます。

使用上の注意:

ACTIVATE DATABASE で初期設定されたデータベースは、 DEACTIVATE DATABASE を使用しないと遮断できません。データベース・マネージャーを停止する前に、db2InstanceStop によって、活動化されたデータベースがすべて自動的に停止されます。データベースが ACTIVATE DATABASE で初期設定されたのであれば、 DB2 CONNECT RESET ステートメント (0 に等しいカウンター) では遮断しないので、 DEACTIVATE DATABASE を使用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 クライアント・アプリケーションのプログラミング*」の『REXX における SQLEXEC、SQLDBS、および SQLDB2 の登録』

関連資料:

- 327 ページの『sqle_activate_db - データベースの活動化』
- 453 ページの『SQLCA』
- 「*コマンド・リファレンス*」の『DEACTIVATE DATABASE コマンド』

sqleaddn - ノードの追加

新しいデータベース・パーティション・サーバーをパーティション・データベース環境に追加します。この API は、インスタンスで現在定義されているデータベースをすべて対象にして、新規データベース・パーティション・サーバー上にデータベース・パーティションを作成します。ユーザーは、任意のシステム TEMPORARY 表スペース用のソース・データベース・パーティション・サーバーをデータベースとともに作成するか、またはシステム TEMPORARY 表スペースを作成しないかを指定することができます。この API は追加するデータベース・パーティション・サーバーから発行する必要があり、データベース・パーティション・サーバー上でのみ発行可能です。

有効範囲:

この API は、それが実行されるデータベース・パーティション・サーバーにのみ影響を与えます。

許可:

以下のいずれかです。

- *sysadm*
- *sysctrl*

必要な接続:

なし

API 組み込みファイル:

sqlenv.h

C API 構文:

```
/* File: sqlenv.h */
/* API: sqleaddn */
/* ... */
SQL_API_RC SQL_API_FN
  sqleaddn (
    void *pAddNodeOptions,
    struct sqlca *pSqlca);
/* ... */
```

汎用 API 構文:

```
/* File: sqlenv.h */
/* API: sqlgaddn */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgaddn (
    unsigned short addnOptionsLen,
    struct sqlca *pSqlca,
    void *pAddNodeOptions);
/* ... */
```

API パラメーター:

addnOptionsLen
入力。オプション *sqle_addn_options* 構造の長さを示す 2 バイトの符号なし整数 (バイト単位) です。

pSqlca

出力。*sqlca* 構造へのポインター。

pAddNodeOptions

入力。オプション *sqle_addn_options* 構造を指すポインター。 SYSTEM TEMPORARY 表スペース定義がある場合は、この構造によってそのソース・データベース・パーティション・サーバーを指定します。この定義はノードの追加操作中に作成されたすべてのデータベース・パーティションに関するものです。何も指定されていない場合 (つまり、NULL ポインターが指定されている場合)、 SYSTEM TEMPORARY 表スペース定義はカタログ・パーティションの定義と同じになります。

REXX API 構文:

この API は、SQLDB2 インターフェースを使って、REXX から呼び出すことができます。

使用上の注意:

新規のデータベース・パーティション・サーバーを追加する前に、十分なストレージを確保しておいてください。このストレージには、システム上に存在するすべてのデータベース用のコンテナーを作成する必要があります。

ノードの追加操作によって、新規データベース・パーティション・サーバー上に、インスタンス内にあるすべてのデータベース用の空のデータベース・パーティションが作成されます。新規のデータベース・パーティション用の構成パラメーターはデフォルト値に設定されます。

データベース・パーティションをローカルに作成している間にノードの追加操作が失敗すると、クリーンアップ段階に入ります。この処理では、作成されたデータベースがすべてローカルでドロップされます。これは、追加されたデータベース・パーティション・サーバー (つまり、ローカル・データベース・パーティション・サーバー) だけからデータベース・パーティションが除去されるということです。その他のデータベース・パーティション・サーバー上に存在しているデータベース・パーティションは影響を受けません。これが失敗した場合、クリーンアップは終了し、エラーが戻されます。

ALTER DATABASE PARTITION GROUP ステートメントを使用して、データベース・パーティション・サーバーを、データベース・パーティション・グループに追加してからでなければ、新規のデータベース・パーティション・サーバー上に、ユーザー・データを含めることはできません。

データベース作成操作またはデータベースのドロップ操作が進行中の場合、この API は正常に実行されません。操作が一度完了してしまうと、この API をもう一度呼び出すことができます。

データベース・パーティションとともにシステム TEMPORARY 表スペースが作成された場合、表スペース定義を検索するために、sqleaddn はパーティション・データベース環境で他のデータベース・パーティション・サーバーと通信する必要があります。 *start_stop_time* データベース・マネージャー構成パラメーターを使用して、時間 (分) を指定します。他のデータベース・パーティション・サーバーはこの

時間内で表スペース定義に応答する必要があります。時間を超過すると、API は失敗します。 *start_stop_time* の値を大きくして、API をもう一度呼び出してください。

関連タスク:

- 「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」の『REXX における SQLEXEC、SQLDBS、および SQLDB2 の登録』

関連資料:

- 「*SQL リファレンス 第 2 巻*」の『ALTER DATABASE PARTITION GROUP ステートメント』
- 348 ページの『sqlecrea - データベースの作成』
- 368 ページの『sqledrpn - ノード・ドロップの検査』
- 453 ページの『SQLCA』
- 459 ページの『SQLE-ADDN-OPTIONS』

sqleatcp - パスワードのアタッチと変更

インスタンス・レベルの関数 (たとえば、 CREATE DATABASE や FORCE APPLICATION) が実行されるノードを、アプリケーションが指定できるようにします。このノードには、現行のインスタンス (**DB2INSTANCE** 環境変数の値で定義された) はもちろん、同一ワークステーション上の別のインスタンスを指定することもできますし、リモート・ワークステーション上のインスタンスでもかまいません。指定されたノードへの論理的なインスタンスの接続が確立されますが、物理的な接続がまだ存在していない場合には、そのノードへの物理的な通信接続が開始されます。

注: この API は、アタッチ中のインスタンスのユーザー・パスワードのオプションを変更できるようにして、 sqleatin API の機能を拡張します。

許可:

なし

必要な接続:

この API はインスタンスのアタッチを確立します。

API 組み込みファイル:

sqlenv.h

C API 構文:

```
/* File: sqlenv.h */
/* API: sqleatcp */
/* ... */
SQL_API_RC SQL_API_FN
  sqleatcp (
    char *pNodeName,
    char *pUserName,
```

```
    char *pPassword,
    char *pNewPassword,
    struct sqlca *pSqlca);
/* ... */
```

汎用 API 構文:

```
/* File: sqlenv.h */
/* API: sqlgatcp */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgatcp (
    unsigned short NewPasswordLen,
    unsigned short PasswordLen,
    unsigned short UserNameLen,
    unsigned short NodeNameLen,
    struct sqlca *pSqlca,
    char *pNewPassword,
    char *pPassword,
    char *pUserName,
    char *pNodeName);
/* ... */
```

API パラメーター:

NewPasswordLen
入力。新規パスワードの長さを示す 2 バイトの符号なし整数 (バイト単位) です。新規パスワードが提供されていない場合は、ゼロに設定してください。

PasswordLen
入力。パスワードの長さを示す 2 バイトの符号なし整数 (バイト単位) です。パスワードが提供されていない場合は、ゼロに設定してください。

UserNameLen
入力。ユーザー名の長さを示す 2 バイトの符号なし整数 (バイト単位) です。ユーザー名が提供されていない場合は、ゼロに設定してください。

NodeNameLen
入力。ノード名の長さを示す 2 バイトの符号なし整数 (バイト単位) です。ノード名が提供されていない場合は、ゼロに設定してください。

pSqlca
出力。*sqlca* 構造へのポインター。

pNewPassword
入力。指定されたユーザー名の新規パスワードを含むストリングを指定します。パスワード変更が必要でない場合は、NULL に設定してください。

pPassword
入力。指定されたユーザー名のパスワードを含むストリングを示します。NULL にすることもできます。

pUserName
入力。アタッチが認証されるユーザー名を含むストリングを指定します。NULL にすることもできます。

pNodeName
入力。アタッチしたいインスタンスの別名を含むストリングを指定します。このインスタンスには、ローカル・ノード・ディレクトリーに一致する項目

がなければなりません。唯一の例外は、ローカル・インスタンス (**DB2INSTANCE** 環境変数で指定された) です。このローカル・インスタンスは、アタッチのオブジェクトとして指定することはできますが、ノード・ディレクトリー内でノード名として使用することはできません。 NULL にすることもできます。

REXX API 構文:

この API を REXX から直接呼び出すことはできません。ただし、REXX プログラマーは、 DB2 コマンド行プロセッサーを呼び出して ATTACH コマンドを実行することにより、この関数を利用できます。

使用上の注意:

注: ノード・ディレクトリー内のノード名は、インスタンスの別名と見なすことができます。

接続要求が成功すると、 *sqlca* の *sqlerrmc* フィールドには、 16 進数 FF で区切られた 9 つのトークンが含まれるようになります (CONNECT 要求が正常に実行されたときに戻されるトークンと同様です)。

1. アプリケーション・サーバーの国地域コード
2. アプリケーション・サーバーのコード・ページ
3. 許可 ID
4. ノード名 (API で指定された)
5. サーバーの ID およびプラットフォーム・タイプ
6. サーバーで開始されたエージェントのエージェント ID
7. エージェントの索引
8. サーバーのノード数
9. サーバーがパーティション・データベース・サーバーの場合はパーティションの数

ノード名が長さゼロのストリングまたは NULL の場合、現行のアタッチ状態に関する情報が戻されます。アタッチが存在していない場合、sqlcode 1427 が戻されます。存在する場合には、そのアタッチに関する情報が *sqlca* の *sqlerrmc* フィールドに戻されます (上述のとおりです)。

アタッチが実行されなかった場合、インスタンス・レベルの API が **DB2INSTANCE** 環境変数で指定された現行のインスタンスに対して実行されます。

特定の関数 (**db2start**、**db2stop**、およびすべてのディレクトリー・サービスなど) は、リモートには実行されません。つまり、それらの関数が影響を与えるのは、**DB2INSTANCE** 環境変数の値で定義されたローカル・インスタンス環境に対してのみです。

アタッチが存在し、かつ API がノード名を指定して発行された場合、現行のアタッチはドロップされ、新規ノードへのアタッチが試行されます。

ユーザー名およびパスワードが認証される箇所、およびパスワードが変更される箇所は、ターゲット・インスタンスの認証タイプによって異なります。

アタッチが確立されたノードは、sqlesetc API を呼び出すことによっても指定できます。

関連資料:

- 406 ページの『sqlesetc - クライアントの設定』
- 337 ページの『sqleatin - アタッチ』
- 370 ページの『sqledtin - 切り離し』
- 453 ページの『SQLCA』
- 463 ページの『SQLE-CONN-SETTING』

関連サンプル:

- 『dbinst.cbl -- Attach to and detach from an instance (IBM COBOL)』
- 『inattach.c -- Attach to and detach from an instance (C)』
- 『inattach.C -- Attach to and detach from an instance (C++)』

sqleatin - アタッチ

インスタンス・レベルの関数 (たとえば、 CREATE DATABASE や FORCE APPLICATION) が実行されるノードを、アプリケーションが指定できるようにします。このノードには、現行のインスタンス (**DB2INSTANCE** 環境変数の値で定義された) はもちろん、同一ワークステーション上の別のインスタンスを指定することもできますし、リモート・ワークステーション上のインスタンスでもかまいません。指定されたノードへの論理的なインスタンスの接続が確立されますが、物理的な接続がまだ存在していない場合には、そのノードへの物理的な通信接続が開始されます。

注: パスワード変更が必要な場合は、sqleatin API ではなく、sqleatcp API を使用してください。

許可:

なし

必要な接続:

この API はインスタンスのアタッチを確立します。

API 組み込みファイル:

sqlenv.h

C API 構文:

```
/* File: sqlenv.h */
/* API: sqleatin */
/* ... */
SQL_API_RC SQL_API_FN
  sqleatin (
    char *pNodeName,
    char *pUserName,
    char *pPassword,
    struct sqlca *pSqlca);
/* ... */
```

汎用 API 構文:

```
/* File: sqlenv.h */
/* API: sqlgatin */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgatin (
    unsigned short PasswordLen,
    unsigned short UserNameLen,
    unsigned short NodeNameLen,
    struct sqlca *pSqlca,
    char *pPassword,
    char *pUserName,
    char *pNodeName);
/* ... */
```

API パラメーター:

PasswordLen
入力。パスワードの長さを示す 2 バイトの符号なし整数 (バイト単位) です。パスワードが提供されていない場合は、ゼロに設定してください。

UserNameLen
入力。ユーザー名の長さを示す 2 バイトの符号なし整数 (バイト単位) です。ユーザー名が提供されていない場合は、ゼロに設定してください。

NodeNameLen
入力。ノード名の長さを示す 2 バイトの符号なし整数 (バイト単位) です。ノード名が提供されていない場合は、ゼロに設定してください。

pSqlca
出力。*sqlca* 構造へのポインター。

pPassword
入力。指定されたユーザー名のパスワードを含むストリングを示します。NULL にすることもできます。

pUserName
入力。アタッチが認証されるユーザー名を含むストリングを指定します。NULL にすることもできます。

pNodeName
入力。アタッチしたいインスタンスの別名を含むストリングを指定します。このインスタンスには、ローカル・ノード・ディレクトリーに一致する項目がなければなりません。唯一の例外は、ローカル・インスタンス (**DB2INSTANCE** 環境変数で指定された) です。このローカル・インスタンスは、アタッチのオブジェクトとして指定することはできますが、ノード・ディレクトリー内でノード名として使用することはできません。 NULL にすることもできます。

REXX API 構文:

```
ATTACH [TO nodename [USER username USING password]]
```

REXX API パラメーター:

nodename
ユーザーがアタッチしたいインスタンスの別名。このインスタンスには、ローカル・ノード・ディレクトリーに一致する項目がなければなりません。唯

一の例外は、ローカル・インスタンス (**DB2INSTANCE** 環境変数で指定された) です。このローカル・インスタンスは、アタッチのオブジェクトとして指定することはできますが、ノード・ディレクトリー内でノード名として使用することはできません。

username
インスタンスにアタッチする際に使用される名前。

password
ユーザー名の認証に使用されるパスワード。

使用上の注意:

注: ノード・ディレクトリー内のノード名は、インスタンスの別名と見なすことができます。

接続要求が成功すると、 *sqlca* の *sqlerrmc* フィールドには、 16 進数 FF で区切られた 9 つのトークンが含まれるようになります (CONNECT 要求が正常に実行されたときに戻されるトークンと同様です)。

1. アプリケーション・サーバーの国地域コード
2. アプリケーション・サーバーのコード・ページ
3. 許可 ID
4. ノード名 (API で指定された)
5. サーバーの ID およびプラットフォーム・タイプ
6. サーバーで開始されたエージェントのエージェント ID
7. エージェントの索引
8. サーバーのノード数
9. サーバーがパーティション・データベース・サーバーの場合はパーティションの数

ノード名が長さゼロのストリングまたは NULL の場合、現行のアタッチ状態に関する情報が戻されます。アタッチが存在していない場合、sqlcode 1427 が戻されます。存在する場合には、そのアタッチに関する情報が *sqlca* の *sqlerrmc* フィールドに戻されます (上述のとおりです)。

アタッチが実行されなかった場合、インスタンス・レベルの API が **DB2INSTANCE** 環境変数で指定された現行のインスタンスに対して実行されます。

特定の関数 (**db2start**、**db2stop**、およびすべてのディレクトリー・サービスなど) は、リモートには実行されません。つまり、それらの関数が影響を与えるのは、**DB2INSTANCE** 環境変数の値で定義されたローカル・インスタンス環境に対してのみです。

アタッチが存在し、かつ API がノード名を指定して発行された場合、現行のアタッチはドロップされ、新規ノードへのアタッチが試行されます。

ユーザー名およびパスワードが認証される箇所は、ターゲット・インスタンスの認証タイプによって異なります。

アタッチが確立されたノードは、sqlesetc API を呼び出すことによっても指定できます。

関連資料:

- 406 ページの『sqlesetc - クライアントの設定』
- 370 ページの『sqledtin - 切り離し』
- 334 ページの『sqleatcp - パスワードのアタッチと変更』
- 453 ページの『SQLCA』
- 463 ページの『SQLE-CONN-SETTING』

関連サンプル:

- 『dbinst.cbl -- Attach to and detach from an instance (IBM COBOL)』
- 『inattach.c -- Attach to and detach from an instance (C)』
- 『utilapi.c -- Error-checking utility for non-embedded SQL samples in C (C)』
- 『inattach.C -- Attach to and detach from an instance (C++)』
- 『utilapi.C -- Checks for and prints to the screen SQL warnings and errors (C++)』

sqlecadb - データベースのカタログ

システム・データベース・ディレクトリーに、データベースのロケーション情報を保管します。データベースは、ローカル・ワークステーションかリモート・ノードのどちらかに位置付けることができます。

有効範囲:

この API はシステム・データベース・ディレクトリーに影響を与えます。パーティション・データベース環境で、システム・データベース・ディレクトリーにローカル・データベースをカタログする場合は、データベースが存在しているデータベース・パーティション・サーバーからこの API を呼び出す必要があります。

許可:

以下のいずれかです。

- *sysadm*
- *sysctrl*

必要な接続:

なし

API 組み込みファイル:

sqlenv.h

C API 構文:

```
/* File: sqlenv.h */
/* API: sqlecadb */
/* ... */
SQL_API_RC SQL_API_FN
  sqlecadb (
    _SQLOLDCHAR *pDbName,
```

```
    _SQLOLDCHAR *pDbAlias,
    unsigned char Type,
    _SQLOLDCHAR *pNodeName,
    _SQLOLDCHAR *pPath,
    _SQLOLDCHAR *pComment,
    unsigned short Authentication,
    _SQLOLDCHAR *pPrincipal,
    struct sqlca *pSqlca);
/* ... */
```

汎用 API 構文:

```
/* File: sqlenv.h */
/* API: sqlgcadb */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgcadb (
    unsigned short PrinLen,
    unsigned short CommentLen,
    unsigned short PathLen,
    unsigned short NodeNameLen,
    unsigned short DbAliasLen,
    unsigned short DbNameLen,
    struct sqlca *pSqlca,
    _SQLOLDCHAR *pPrinName,
    unsigned short Authentication,
    _SQLOLDCHAR *pComment,
    _SQLOLDCHAR *pPath,
    _SQLOLDCHAR *pNodeName,
    unsigned char Type,
    _SQLOLDCHAR *pDbAlias,
    _SQLOLDCHAR *pDbName);
/* ... */
```

API パラメーター:

PrinLen
入力。プリンシパル名の長さを示す 2 バイトの符号なし整数 (バイト単位) です。プリンシパルが提供されていない場合は、ゼロに設定してください。認証が SQL_AUTHENTICATION_KERBEROS として指定されている場合に限り、この値をゼロ以外にする必要があります。

CommentLen
入力。コメントの長さを示す 2 バイトの符号なし整数 (バイト単位) です。コメントが提供されていない場合は、ゼロに設定してください。

PathLen
入力。ローカル・データベース・ディレクトリーのパスの長さを示す 2 バイトの符号なし整数 (バイト単位) です。パスが提供されていない場合は、ゼロに設定してください。

NodeNameLen
入力。ノード名の長さを示す 2 バイトの符号なし整数 (バイト単位) です。ノード名が提供されていない場合には、ゼロに設定してください。

DbAliasLen
入力。データベース別名の長さを示す 2 バイトの符号なし整数 (バイト単位) です。

DbNameLen
入力。データベース名の長さを示す 2 バイトの符号なし整数 (バイト単位) です。

pSqlca
出力。*sqlca* 構造へのポインター。

pPrinName
入力。データベースが存在している DB2 サーバーのプリンシパル名を含むストリングです。認証が SQL_AUTHENTICATION_KERBEROS の場合に限り、この値を指定してください。

Authentication
入力。データベース用に指定される認証タイプが入ります。認証はユーザーが本人かどうかを検査するプロセスです。データベース・オブジェクトへのアクセスはユーザーの認証によって決まります。有効な値 (sqlenv から) は以下のとおりです。

SQL_AUTHENTICATION_SERVER
認証が、宛先データベースを含むノードで行われるということを指定します。

SQL_AUTHENTICATION_CLIENT
認証が、アプリケーションを呼び出すノードで行われるということを指定します。

SQL_AUTHENTICATION_KERBEROS
認証が、kerberos セキュリティー・メカニズムを使用するということを指定します。

SQL_AUTHENTICATION_NOT_SPECIFIED
認証は指定されません。

SQL_AUTHENTICATION_SVR_ENCRYPT
認証が宛先データベースを含むノードで行われることと、認証のパスワードが暗号化されることを指定します。

SQL_AUTHENTICATION_DATAENC
宛先データベースを含むノードで認証が行われること、および接続でデータ暗号化を使用する必要があることを指定します。

SQL_AUTHENTICATION_GSSPLUGIN
外部 GSS API ベースのプラグイン・セキュリティー機構を使って認証が行われることを指定します。

このパラメーターは SQL_AUTHENTICATION_NOT_SPECIFIED に設定することができます。ただし、DB2 バージョン 1 サーバーに存在するデータベースをカタログする場合は設定できません。

データベース・カタログに認証タイプを指定すると、接続中のパフォーマンスが向上するという結果になります。

pComment
入力。データベースのオプション・コメントを含むストリングを指定します。 NULL ストリングはコメントがないことを示します。コメント・ストリングの最大長は 30 文字です。

pPath

入力。 UNIX ベースのシステムの場合は、カタログするデータベースが存在するパス名を指定するストリングを指定します。最大長は 215 文字です。

Windows オペレーティング・システムの場合、このストリングはカタログするデータベースが存在するドライブ名を指定します。

NULL ポインターが提供されている場合、データベース・マネージャーの構成パラメーター *dftdbpath* で指定されたデフォルトのデータベースが指定されたものと見なされます。

pNodeName

入力。データベースが位置するノードの名前を含むストリングを指定します。 NULL にすることもできます。

注: *pPath* も *pNodeName* も指定されていない場合、データベースはローカルであり、データベースのロケーションはデータベース・マネージャー構成パラメーター *dftdbpath* で指定されたロケーションであるものと見なされます。

Type 入力。データベースが間接であるか、リモートであるか、それとも DCE を通してカタログするかを指定する単一の文字。有効な値 (`sqlenv` で定義) は、以下のとおりです。

SQL_INDIRECT
データベースが今のインスタンスに存在するよう指定します。

SQL_REMOTE
データベースが別のインスタンスに存在するよう指定します。

SQL_DCE
データベースが DCE を通してカタログされるよう指定します。

pDbAlias

入力。データベースの別名を含むストリングを指定します。

pDbName

入力。データベース名を含むストリングを指定します。

REXX API 構文:

```
CATALOG DATABASE dbname [AS alias] [ON path|AT NODE nodename]
[AUTHENTICATION authentication] [WITH "comment"]

CATALOG GLOBAL DATABASE db_global_name AS alias
USING DIRECTORY {DCE} [WITH "comment"]
```

REXX API パラメーター:

dbname

カタログするデータベースの名前。

alias データベースの代替名を指定します。別名が指定されない場合は、データベース名が別名として使用されます。

path カタログするデータベースが存在するパスを指定します。

nodename
カタログするデータベースが存在するリモート・ワークステーションの名前を指定します。

注: *path* も *nodename* も指定されていない場合、データベースはローカルであり、データベースのロケーションはデータベース・マネージャー構成パラメーター *dftdbpath* で指定されたロケーションであるものと見なされます。

authentication
認証が行われる場所を指定します。有効な値は以下のとおりです。

SERVER
認証がターゲット・データベースを含むノードで行われます。これがデフォルト値です。

CLIENT
認証がアプリケーションを呼び出すノードで行われます。

KERBEROS
認証が、kerberos セキュリティー・メカニズムを使用するということを指定します。

NOT_SPECIFIED
認証は指定されません。

SVR_ENCRYPT
認証が宛先データベースを含むノードで行われることと、認証のパスワードが暗号化されることを指定します。

DATAENC
宛先データベースを含むノードで認証が行われること、および接続でデータ暗号化を使用する必要があることを指定します。

GSSPLUGIN
外部 GSS API ベースのプラグイン・セキュリティー機構を使って認証が行われることを指定します。

comment
システム・データベース・ディレクトリー内のデータベースまたはデータベース項目について記述します。コメント・ストリングの最大長は 30 文字です。復帰文字や改行文字は許可されません。コメント・テキストは必ず二重引用符で囲んでください。

db_global_name
DCE 名前スペース内のデータベースを固有に識別する完全修飾名を示します。

DCE 使用されるグローバル・ディレクトリー・サービスを示します。

REXX の例:

```
call SQLDBS 'CATALOG GLOBAL DATABASE /.../cell1/subsys/database/DB3
AS dbtest USING DIRECTORY DCE WITH "Sample Database"'
```

使用上の注意:

ローカル・ノードやリモート・ノードにあるデータベースをカタログする場合、以前にアンカタログしたデータベースを再カタログする場合、または 1 つのデータベースに対して複数の別名を保持する場合 (データベースのロケーションにかかわらず)、 CATALOG DATABASE を使用してください。

データベースの作成時に、DB2 は自動的にそれらをカタログします。また、データベースの項目をローカル・データベース・ディレクトリーに、その他の項目をシステム・データベース・ディレクトリーにカタログします。リモート・クライアント (または、同じマシンの別のインスタンスから実行しているクライアント) からデータベースを作成した場合、クライアント・インスタンスでは、システム・データベース・ディレクトリーにも項目が作成されます。

(DB2INSTANCE 環境変数の値で定義された) 現行インスタンスで作成されたデータベースは、*間接* としてカタログされます。その他のインスタンスで作成されたデータベースは、 (物理的には同一のマシンに存在している場合であっても) リモートとしてカタログされます。

システム・データベース・ディレクトリーが存在しない場合、CATALOG DATABASE は自動的にそれを作成します。システム・データベース・ディレクトリーは、使用中のデータベース・マネージャー・インスタンスを含むパスに保管されます。システム・データベース・ディレクトリーは、データベースの外部で保守されます。ディレクトリーに含まれる項目は、次のとおりです。

- 別名
- 認証タイプ
- コメント
- データベース
- 項目タイプ
- ローカル・データベース・ディレクトリー (ローカル・データベースをカタログするとき)
- ノード名 (リモート・データベースをカタログするとき)
- リリース情報

タイプ・パラメーターを `SQL_INDIRECT` に設定してデータベースをカタログすると、入力された認証パラメーターの値は無視され、ディレクトリーの認証は `SQL_AUTHENTICATION_NOT_SPECIFIED` に設定されます。

ディレクトリーのキャッシュが使用可能にされている場合、データベース、ノード、および DCS ディレクトリー・ファイルはメモリーにキャッシュされます。アプリケーションのディレクトリー・キャッシュは、最初のディレクトリー検索時に作成されます。キャッシュはアプリケーションがディレクトリー・ファイルのどれかを修正したときにのみ最新にされるため、他のアプリケーションが行ったディレクトリーの変更は、アプリケーションを再始動するまで有効にならないことがあります。 DB2 の共用キャッシュを最新にするには (サーバーのみ)、データベース・マネージャーを停止させてから (**db2stop**)、再始動 (**db2start**) させてください。別のアプリケーション用のディレクトリー・キャッシュを最新にするには、そのアプリケーションを停止させてから再始動させてください。

関連資料:

- 53 ページの『db2DbDirCloseScan - データベース・ディレクトリー・スキャンのクローズ』
- 54 ページの『db2DbDirGetNextEntry - データベース・ディレクトリーの次項目の入手』
- 58 ページの『db2DbDirOpenScan - データベース・ディレクトリー・スキャンのオープン』
- 411 ページの『sqleuncd - データベースのアンカタログ』
- 453 ページの『SQLCA』

関連サンプル:

- 『dbcat.cbl -- Catalog to and uncatalog from a database (IBM COBOL)』
- 『ininfo.c -- Set and get information at the instance level (C)』
- 『ininfo.C -- Set and get information at the instance level (C++)』

sqlecran - ノードでのデータベースの作成

API を呼び出すデータベース・パーティション・サーバー上だけでデータベースを作成します。この API は汎用目的ではありません。たとえば、あるデータベース・パーティション・サーバーのデータベース・パーティションが損傷して再作成が必要な場合に、 db2Restore とともに使用する必要があります。この API を不適切な仕方で使用すると、システム内に不整合が生じるので、注意して使用してください。

注: この API を (損傷したという理由で) ドロップされたデータベース・パーティションを再作成するために使用した場合、このデータベース・パーティション・サーバーのデータベースはリストア・ペンディング状態になります。データベース・パーティションを再作成したら、ただちにデータベースをこのデータベース・パーティション・サーバー上にリストアする必要があります。

有効範囲:

この API は、それが呼び出されたデータベース・パーティション・サーバーにのみ影響を与えます。

許可:

以下のいずれかです。

- *sysadm*
- *sysctrl*

必要な接続:

インスタンス。データベースを別のデータベース・パーティション・サーバーで作成する場合、まずそのデータベース・パーティション・サーバーにアタッチすることが必要になります。データベース接続は、この API によって処理中に一時的に確立されます。

API 組み込みファイル:

sqlenv.h

C API 構文:

```
/* File: sqlenv.h */
/* API: sqlecran */
/* ... */
SQL_API_RC SQL_API_FN
  sqlecran (
    char *pDbName,
    void *pReserved,
    struct sqlca *pSqlca);
/* ... */
```

汎用 API 構文:

```
/* File: sqlenv.h */
/* API: sqlgcran */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgcran (
    unsigned short reservedLen,
    unsigned short dbNameLen,
    struct sqlca *pSqlca,
    void *pReserved,
    char *pDbName);
/* ... */
```

API パラメーター:

reservedLen
入力。 *pReserved* の長さのために予約されています。

dbNameLen
入力。データベース名の長さを示す 2 バイトの符号なし整数 (バイト単位) です。

pSqlca
出力。*sqlca* 構造へのポインター。

pReserved
入力。 NULL に設定されたスペア・ポインター、またはゼロを指すスペア・ポインター。将来の利用のために予約されています。

pDbName
入力。作成されるデータベースの名前を含むストリングを指定します。NULL にはしないでください。

REXX API 構文:

この API は、SQLDB2 インターフェースを使って、REXX から呼び出すことができます。

使用上の注意:

データベースが正常に作成されると、リストア・ペンディング状態になります。このデータベースを使用するには、その前にデータベースをこのデータベース・パーティション・サーバー上にリストアする必要があります。

関連タスク:

- 「*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*」の『REXX における SQLEXEC、SQLDBS、および SQLDB2 の登録』

関連資料:

- 348 ページの『sqlecrea - データベースの作成』
- 362 ページの『sqledpan - ノードでのデータベースのドロップ』
- 453 ページの『SQLCA』
- 244 ページの『db2Restore - データベースのリストア』

sqlecrea - データベースの作成

オプションでユーザー定義の照合シーケンスを持つ新規データベースを初期設定し、 3 つの初期表スペースやシステム表を作成し、さらにはリカバリー・ログを割り当てます。

有効範囲:

パーティション・データベース環境では、この API は db2nodes.cfg ファイルにリストされているすべてのデータベース・パーティション・サーバーに影響を与えます。

この API が呼び出されるデータベース・パーティション・サーバーは、新規のデータベース用のカタログ・パーティションになります。

許可:

以下のいずれかです。

- *sysadm*
- *sysctrl*

必要な接続:

インスタンス。データベースを別の (リモート) ノードで作成すると、まずそのノードにアタッチすることが必要になります。データベース接続は、この API によって処理中に一時的に確立されます。

API 組み込みファイル:

sqlenv.h

C API 構文:

```
/* File: sqlenv.h */
/* API: sqlecrea */
/* ... */
SQL_API_RC SQL_API_FN
  sqlecrea (
    char *pDbName,
    char *pLocalDbAlias,
    char *pPath,
    struct sqledbdesc *pDbDescriptor,
    struct sqledbterritoryinfo *pTerritoryInfo,
    char Reserved2,
    void *pReserved1,
    struct sqlca *pSqlca);
/* ... */
```

汎用 API 構文:

```
/* File: sqlenv.h */
/* API: sqlgcrea */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgcrea (
    unsigned short PathLen,
    unsigned short LocalDbAliasLen,
    unsigned short DbNameLen,
    struct sqlca *pSqlca,
    void *pReserved1,
    unsigned short Reserved2,
    struct sqledbterritoryinfo *pTerritoryInfo,
    struct sqledbdesc *pDbDescriptor,
    char *pPath,
    char *pLocalDbAlias,
    char *pDbName);
/* ... */
```

API パラメーター:

PathLen
入力。パスの長さを示す 2 バイトの符号なし整数 (バイト単位) です。パスが提供されていない場合は、ゼロに設定してください。

LocalDbALiasLen
入力。ローカル・データベース別名の長さを示す 2 バイトの符号なし整数 (バイト単位) です。ローカル別名が提供されていない場合は、ゼロに設定してください。

DbNameLen
入力。データベース名の長さを示す 2 バイトの符号なし整数 (バイト単位) です。

pSqlca
出力。*sqlca* 構造へのポインター。

pReserved1
入力。 NULL に設定されたスペア・ポインター、またはゼロを指すスペア・ポインター。

Reserved2
入力。将来の利用のために予約されています。

pTerritoryInfo
入力。 *sqledbterritoryinfo* 構造を指すポインター。データベースのロケールおよびコード・セットが含まれます。 NULL にすることもできます。

pDbDescriptor
入力。データベースの作成時に使用されるデータベース記述ブロックを指すポインター。データベース記述ブロックは、照合シーケンスのように、永久にデータベースの構成ファイルに保管される値を提供するために使用することができます。 NULL にすることもできます。 Unicode データベース用にサポートされる照合シーケンスについては、データベース記述ブロック (SQLEDBDESC) に関するトピックを参照してください。

pPath
入力。 UNIX ベースのシステムの場合は、データベースを作成するパスを指定します。パスが指定されない場合、データベースは、データベース・マネージャー構成ファイルに指定されているデフォルトのデータベース・パス

(*dftdbpath* パラメーター) に作成されます。 Windows オペレーティング・システムの場合は、データベースの作成先のドライブ名を指定します。 NULL にすることもできます。

注: パーティション・データベース環境の場合は、 NFS がマウントされたディレクトリーにはデータベースを作成しないでください。パスが指定されていない場合、 NFS がマウントされたパスに *dftdbpath* データベース・マネージャー構成パラメーターが指定されないようにしてください。 (たとえば、UNIX ベースのシステムの場合には、インスタンスの所有者の $HOME ディレクトリーを指定しないでください。) パーティション・データベース環境でこの API に指定されたパスを相対パスにすることはできません。

pLocalDbAlias
入力。クライアントのシステム・データベース・ディレクトリーに置かれる別名を含むストリングを指定します。 NULL にすることもできます。ローカル別名が指定されない場合、データベース名はデフォルトのものになります。

pDbName
入力。データベース名を含むストリングを指定します。これはシステム・データベース・ディレクトリーにカタログされるデータベース名です。データベースは、サーバーのシステム・データベース・ディレクトリーに正常に作成されると、データベース名と同じデータベース別名を持つシステム・データベース・ディレクトリーに自動的にカタログされます。 NULL にはしないでください。

REXX API 構文:

```
CREATE DATABASE dbname [ON path] [ALIAS dbalias]
[USING CODESET codeset TERRITORY territory]
[COLLATE USING {SYSTEM | IDENTITY | USER :udcs}]
[NUMSEGS numsegs] [DFT_EXTENT_SZ dft_extentsize]
[CATALOG TABLESPACE <tablespace_definition>]
[USER TABLESPACE <tablespace_definition>]
[TEMPORARY TABLESPACE <tablespace_definition>]
[WITH comment]

<tablespace_definition> は以下を表します。
MANAGED BY {
SYSTEM USING :SMS_string |
DATABASE USING :DMS_string }
[ EXTENTSIZE number_of_pages ]
[ PREFETCHSIZE number_of_pages ]
[ OVERHEAD number_of_milliseconds ]
[ TRANSFERRATE number_of_milliseconds ]
```

REXX API パラメーター:

dbname
データベースの名前。

dbalias
データベースの別名。

path データベースを作成するパス。

パスが指定されない場合、データベースは、データベース・マネージャー構成ファイルに指定されているデフォルトのデータベース・パス (*dftdbpath* 構成パラメーター) に作成されます。

注: パーティション・データベース環境の場合は、 NFS がマウントされたディレクトリーにはデータベースを作成しないでください。パスが指定されていない場合、 NFS がマウントされたパスに *dftdbpath* データベース・マネージャー構成パラメーターが指定されないようにしてください。 (たとえば、UNIX ベースのシステムの場合には、インスタンスの所有者の $HOME ディレクトリーを指定しないでください。) パーティション・データベース環境でこの API に指定されたパスを相対パスにすることはできません。

codeset
データベースに入力されたデータに使用するコード・セットを表します。

territory
データベースに入力されたデータに使用する地域コード (ロケール) を表します。

SYSTEM
データベース・テリトリーに基づいた照合シーケンス。

IDENTITY
ストリングの各バイトのバイナリー順序によって判別される照合シーケンス。ストリングは、左端のバイトから始まって、バイトごとに比較されます。

USER udcs
照合シーケンスは、アプリケーションを呼び出すことにより、照合シーケンスを定義する 256 バイトのストリングを含むホスト変数に指定されます。

numsegs
DAT、IDX、および LF ファイルを保管するために作成および使用されるセグメント・ディレクトリーの数を表します。

dft_extentsize
データベースの表スペースのデフォルトの *extent size* を指定します。

SMS_string
表スペースに属する予定の 1 つまたは複数のコンテナーを識別するコンパウンド REXX ホスト変数を表します。ここに表スペース・データが格納されます。以下の項目において、XXX はホスト変数名を表しています。各ディレクトリー名の長さは 254 バイトを超えることはできないことに注意してください。

XXX.0 指定されたディレクトリーの数。

XXX.1 SMS 表スペースの最初のディレクトリー名。

XXX.2 SMS 表スペースの 2 番目のディレクトリー名。

XXX.3 以降、3 番目、4 番目 ... と続きます。

DMS_string
表スペースに属する予定の 1 つまたは複数のコンテナーを識別するコンパウンド REXX ホスト変数を表します。ここに表スペース・データが格納さ

れます。コンテナー・サイズ (4KB ページの数で指定) およびタイプ (ファイルまたは装置) が格納されます。指定された装置 (ファイルではない) は、前もって存在していなければなりません。以下の項目において、XXX はホスト変数名を表しています。それぞれのコンテナー名の長さは 254 バイトを超えてはならないことに注意してください。

XXX.0 REXX ホスト変数内のストリングの数 (最初のレベル・エレメントの数)。

XXX.1.1
最初のコンテナーのタイプ (file または device)。

XXX.1.2
最初のファイル名または装置名。

XXX.1.3
最初のコンテナーのサイズ (ページ単位)。

XXX.2.1
2 番目のコンテナーのタイプ (file または device)。

XXX.2.2
2 番目のファイル名または装置名。

XXX.2.3
2 番目のコンテナーのサイズ (ページ単位)。

XXX.3.1
以降、3 番目、4 番目 ... と続きます。

EXTENTSIZE number_of_pages
次のコンテナーにスキップする前に、コンテナーに書き込まれることになる 4KB ページの数。

PREFETCHSIZE number_of_pages
データのプリフェッチを実行する際に、表スペースから読み取られることになる 4KB ページの数。

OVERHEAD number_of_milliseconds
入出力制御装置のオーバーヘッド、ディスク・シーク、および待ち時間を指定する数 (ミリ秒単位)。

TRANSFERRATE number_of_milliseconds
1 つの 4KB ページをメモリーに読み取る時間を指定する数 (ミリ秒単位)。

comment
システム・ディレクトリー内のデータベースまたはデータベース項目のコメント。コメント中に復帰文字または改行文字を使用しないでください。コメント文は必ず二重引用符で囲んでください。コメントは最大 30 文字まで記述できます。

使用上の注意:

CREATE DATABASE は以下の事柄を行います。

- 指定されたサブディレクトリーにデータベースを作成します。パーティション・データベース環境では、 db2nodes.cfg にリストされたすべてのデータベース・パーティション・サーバー上にデータベースを作成し、各データベース・パーテ

ィション・サーバーで指定されたサブディレクトリーの下に $DB2INSTANCE/NODE*xxxx* ディレクトリーを作成します。 *xxxx* はローカル・データベース・パーティション・サーバー番号を表します。単一パーティション環境では、指定されたサブディレクトリーの下に $DB2INSTANCE/NODE0000 ディレクトリーを作成します。

- システム・カタログ表およびリカバリー・ログを作成します。
- 以下のデータベース・ディレクトリー内のデータベースをカタログします。
  - *pPath* によって示されたパスにあるサーバーのローカル・データベース・ディレクトリー。または、パスが指定されていない場合は、データベース・マネージャーのシステム構成ファイルで定義されたデフォルトのデータベース・パスが使用されます。ローカル・データベース・ディレクトリーは、データベースを含む各ファイル・システムに存在します。
  - アタッチされたインスタンス用のサーバーのシステム・データベース・ディレクトリー。作成されるディレクトリー項目にはデータベース名やデータベース別名が含まれます。

    API がリモート・クライアントから呼び出された場合は、クライアントのシステム・データベース・ディレクトリーもデータベース名やデータベース別名で更新されます。

  システム・データベース・ディレクトリーとローカル・データベース・ディレクトリーがどちらも存在しない場合、そのどちらかを作成します。指定されていれば、コメントおよびコード設定値は両方のディレクトリーに入れられます。
- 指定されたコード設定、区域、および照合シーケンスを保管します。照合シーケンスがユニークな重みから成っているか、または ID 順序である場合、フラグがデータベース構成ファイルで設定されます。
- SYSCAT、SYSFUN、SYSIBM、および SYSIBM を所有者とする SYSSTAT という各スキーマを作成します。この API が呼び出されるデータベース・パーティション・サーバーは、新規のデータベース用のカタログ・パーティションになります。 2 つのデータベース・パーティション・グループ (IBMDEFAULTGROUP および IBMCATGROUP) が自動的に作成されます。
- 事前に定義されたデータベース・マネージャー・バインド・ファイルのデータベースにバインドします (これらは db2ubind.lst にリストされています)。これらの 1 つまたは複数のファイルのバインドが正常に行わなかった場合、sqlecrea は SQLCA に警告を戻し、失敗したバインドに関する情報を提供します。バインドが失敗した場合、ユーザーは修正処置を行った後、失敗したファイルを手動でバインドできます。データベースはどのような場合にでも作成されます。バインドを行うとき、 PUBLIC に認可された CREATEIN 特権を伴って、 NULLID と呼ばれているスキーマが暗黙的に作成されます。
- SYSCATSPACE、TEMPSPACE1、および USERSPACE1 表スペースを作成します。 SYSCATSPACE 表スペースはカタログ・パーティションにのみ作成されます。すべてのデータベース・パーティションには同じ表スペース定義があります。
- 以下の権限を付与します。

- データベース作成者に対する DBADM 権限と、CONNECT、 CREATETAB、 BINDADD、 CREATE_NOT_FENCED、 IMPLICIT_SCHEMA、および LOAD の各種特権。
- PUBLIC に対する CONNECT、 CREATETAB、 BINDADD、 IMPLICIT_SCHEMA の各種特権。
- PUBLIC に対する USERSPACE1 表スペース上の USE 特権。
- PUBLIC に対する各システム・カタログ上の SELECT 特権。
- 正常実行したバインド・ユーティリティーごとに、PUBLIC に対する BIND および EXECUTE 特権。
- SYSFUN スキーマのすべての関数について、PUBLIC に対する EXECUTE WITH GRANT 特権。
- SYSIBM スキーマのすべてのプロシージャーについて、PUBLIC に対する EXECUTE 特権。

dbadm 権限を使用すると、これらの権限を他のユーザーまたは PUBLIC に GRANT (または取り消し) することができます。データベース上で *sysadm* または *dbadm* 権限を付与された別の管理者が上記の特権を取り消したとしても、データベース作成者はそれらの特権を保持します。

パーティション・データベース環境では、データベース・マネージャーはすべてのデータベース・パーティション・サーバーの指定されたパスまたはデフォルト・パスの下に、 `$DB2INSTANCE/NODE`*xxxx* サブディレクトリーを作成します。 *xxxx* は `db2nodes.cfg` ファイルで定義されたノード番号です (つまり、ノード 0 は `NODE0000` になります)。 `SQL00001` から `SQL`*nnnnn* のサブディレクトリーはこのパスにあります。これにより、別々のデータベース・パーティション・サーバーと関連付けられたデータベース・オブジェクトが、それぞれ別々のディレクトリーに保管されるようになります (指定されたパスまたはデフォルト・パスの下にあるサブディレクトリー `$DB2INSTANCE` をすべてのデータベース・パーティション・サーバーが共有している場合でも、そのようになります)。

Windows および AIX では、コード・セット名の長さは最大 9 文字に制限されています。たとえば、`ISO8859-15` というコード・セット名の代わりに、`ISO885915` のようにします。

アプリケーションがすでにデータベースに接続されている場合、 CREATE DATABASE は失敗します。

データベース記述ブロック構造が正しく設定されていない場合、エラー・メッセージが戻されます。

データベース記述ブロックの「目印」は、シンボリック値 `SQLE_DBDESC_2` (`sqlenv` で定義) に設定しなければなりません。以下のユーザー定義の照合シーケンスの例は、ホスト言語組み込みファイルで使用できます。

sqle819a データベースのコード・ページが 819 (ISO Latin/1) である場合、この順序により、ホスト CCSID 500 (EBCDIC 国際標準) に従ってソートが行われます。

sqle819b データベースのコード・ページが 819 (ISO Latin/1) である場合、この順序により、ホスト CCSID 037 (EBCDIC 米国英語) に従ってソートが行われます。

sqle850a データベースのコード・ページが 850 (ASCII Latin/1) である場合、この順序により、ホスト CCSID 500 (EBCDIC 国際標準) に従ってソートが行われます。

sqle850b データベースのコード・ページが 850 (ASCII Latin/1) である場合、この順序により、ホスト CCSID 037 (EBCDIC 米国英語) に従ってソートが行われます。

sqle932a データベースのコード・ページが 932 (ASCII 日本語) である場合、この順序により、ホスト CCSID 5035 (EBCDIC 日本語) に従ってソートが行われます。

sqle932b データベースのコード・ページが 932 (ASCII 日本語) である場合、この順序により、ホスト CCSID 5026 (EBCDIC 日本語) に従ってソートが行われます。

CREATE DATABASE 中に指定された照合シーケンスは後で変更することはできません。データベースのすべての文字比較は指定された照合シーケンスを使用します。これは索引の構造と照会の結果に影響します。

新規のデータベースに異なる別名を定義するには、 **sqlecadb** を使用してください。

関連資料:

- 294 ページの『sqlabndx - バインド』
- 340 ページの『sqlecadb - データベースのカタログ』
- 366 ページの『sqledrpd - データベースのドロップ』
- 346 ページの『sqlecran - ノードでのデータベースの作成』
- 362 ページの『sqledpan - ノードでのデータベースのドロップ』
- 474 ページの『SQLEDBTERRITORYINFO』
- 453 ページの『SQLCA』
- 475 ページの『SQLEDBDESC』
- 「コマンド・リファレンス」の『CREATE DATABASE コマンド』

関連サンプル:

- 『db_udcs.cbl -- How to use user-defined collating sequence (IBM COBOL)』
- 『dbconf.cbl -- Update database configuration (IBM COBOL)』
- 『ebcdicdb.cbl -- Create a database with EBCDIC 037 standard collating sequence (IBM COBOL)』
- 『dbcreate.c -- Create and drop databases (C)』
- 『dbrecov.sqc -- How to recover a database (C)』
- 『dbsample.sqc -- Creates a sample database (C)』
- 『dbcreate.C -- Create and drop databases (C++)』
- 『dbrecov.sqC -- How to recover a database (C++)』

sqlectnd - ノードのカタログ

DB2 サーバーのインスタンスへのアクセスに使用する通信プロトコルに基づいて、そのインスタンスの位置に関する情報をノード・ディレクトリーに保管します。この情報は、アプリケーションとサーバー・インスタンスの間でデータベース接続を確立するのに必要です。

許可:

以下のいずれかです。

- *sysadm*
- *sysctrl*

必要な接続:

なし

API 組み込みファイル:

sqlenv.h

C API 構文:

```
/* File: sqlenv.h */
/* API: sqlectnd */
/* ... */
SQL_API_RC SQL_API_FN
  sqlectnd (
    struct sqle_node_struct *pNodeInfo,
    void *pProtocolInfo,
    struct sqlca *pSqlca);
/* ... */
```

汎用 API 構文:

```
/* File: sqlenv.h */
/* API: sqlgctnd */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgctnd (
    struct sqlca *pSqlca,
    struct sqle_node_struct *pNodeInfo,
    void *pProtocolInfo);
/* ... */
```

API パラメーター:

pNodeInfo
入力。ノード・ディレクトリー構造を指すポインター。

pProtocolInfo
入力。プロトコル構造を指す以下のポインター

- SQLE-NODE-CPIC
- SQLE-NODE-IPXSPX
- SQLE-NODE-LOCAL
- SQLE-NODE-NETB
- SQLE-NODE-NPIPE
- SQLE-NODE-TCPIP

pSqlca
出力。*sqlca* 構造へのポインター。

REXX API 構文:

```
CATALOG APPC NODE nodename DESTINATION symbolic_destination_name
[SECURITY {NONE|SAME|PROGRAM}]
[WITH comment]
```

REXX API パラメーター:

nodename
カタログするノードの別名。

symbolic_destination_name
リモート・パートナー・ノードのシンボリック宛先名。

comment
そのノード・ディレクトリー項目と関連したオプション・コメント。コメントには CR/LF 文字を含めないようにしてください。最大長は 30 文字です。コメント・テキストは必ず二重引用符で囲んでください。

REXX API 構文:

```
CATALOG IPXSPX NODE nodename REMOTE file_server SERVER objectname
[WITH comment]
```

REXX API パラメーター:

nodename
カタログするノードの別名。

file_server
データベース・マネージャー・インスタンスのインターネットワーク・アドレスを登録する、 NetWare ファイル・サーバーの名前。インターネットワーク・アドレスは、 NetWare ファイル・サーバーのバインダリーに保管され、 *objectname* を使用してアクセスされます。

objectname
データベース・マネージャー・サーバー・インスタンスは、 NetWare ファイル・サーバー上ではオブジェクト *objectname* と表されます。サーバーの IPX/SPX インターネットワーク・アドレスはこのオブジェクトに保管され、このオブジェクトから検索されます。

comment
そのノード・ディレクトリー項目と関連したオプション・コメント。コメントには CR/LF 文字を含めないようにしてください。最大長は 30 文字です。コメント・テキストは必ず二重引用符で囲んでください。

REXX API 構文:

```
CATALOG LOCAL NODE nodename INSTANCE instance_name [WITH comment]
```

REXX API パラメーター:

nodename
カタログするノードの別名。

instance_name
カタログするインスタンスの名前。

comment
そのノード・ディレクトリー項目と関連したオプション・コメント。コメントには CR/LF 文字を含めないようにしてください。最大長は 30 文字です。コメント・テキストは必ず二重引用符で囲んでください。

REXX API 構文:

```
CATALOG NETBIOS NODE nodename REMOTE server_nname ADAPTER adapternum
[WITH comment]
```

REXX API パラメーター:

nodename
カタログするノードの別名。

server_nname
リモート・ワークステーションの名前。サーバー・インスタンスのデータベース・マネージャー構成ファイルに検出されたワークステーション名 (*nname*) です。

adapternum
ローカル LAN アダプター番号。

comment
そのノード・ディレクトリー項目と関連したオプション・コメント。コメントには CR/LF 文字を含めないようにしてください。最大長は 30 文字です。コメント・テキストは必ず二重引用符で囲んでください。

REXX API 構文:

```
CATALOG NPIPE NODE nodename REMOTE computer_name INSTANCE instance_name
```

REXX API パラメーター:

nodename
カタログするノードの別名。

computer_name
ターゲット・データベースが存在するノードのコンピューター名。

instance_name
カタログするインスタンスの名前。

REXX API 構文:

```
CATALOG TCPIP NODE nodename REMOTE hostname SERVER servicename
[WITH comment]
```

REXX API パラメーター:

nodename
カタログするノードの別名。

hostname
ターゲット・データベースが存在するノードのホスト名。

servicename
リモート・ノード上にあるデータベース・マネージャー・インスタンスのサービス名か、このサービス名と関連があるポート番号のどちらかです。

comment
そのノード・ディレクトリー項目と関連したオプション・コメント。コメントには CR/LF 文字を含めないようにしてください。最大長は 30 文字です。コメント・テキストは必ず二重引用符で囲んでください。

使用上の注意:

ノード・ディレクトリーが存在していない場合、 DB2 はこの API への最初の呼び出しでノード・ディレクトリーを作成します。 Windows オペレーティング・システムの場合、そのノード・ディレクトリーは使用中のインスタンスのディレクトリーに格納されます。 UNIX ベースのシステムの場合、DB2 インストール・ディレクトリー (たとえば、sqllib) に格納されます。

ディレクトリーのキャッシュが使用可能にされている場合、データベース、ノード、および DCS ディレクトリー・ファイルはメモリーにキャッシュされます。アプリケーションのディレクトリー・キャッシュは、最初のディレクトリー検索時に作成されます。キャッシュはアプリケーションがディレクトリー・ファイルのどれかを修正したときにのみ最新にされるため、他のアプリケーションが行ったディレクトリーの変更は、アプリケーションを再始動するまで有効にならないことがあります。 DB2 の共用キャッシュを最新にするには (サーバーのみ)、データベース・マネージャーを停止させてから (**db2stop**)、再始動 (**db2start**) させてください。別のアプリケーション用のディレクトリー・キャッシュを最新にするには、そのアプリケーションを停止させてから再始動させてください。

関連資料:

- 392 ページの『sqlencls - ノード・ディレクトリー・スキャンのクローズ』
- 393 ページの『sqlengne - ノード・ディレクトリー次項目の入手』
- 395 ページの『sqlenops - ノード・ディレクトリー・スキャンのオープン』
- 413 ページの『sqleuncn - ノードのアンカタログ』
- 468 ページの『SQLE-NODE-CPIC』
- 470 ページの『SQLE-NODE-NETB』
- 471 ページの『SQLE-NODE-STRUCT』
- 472 ページの『SQLE-NODE-TCPIP』
- 453 ページの『SQLCA』
- 468 ページの『SQLE-NODE-IPXSPX』
- 469 ページの『SQLE-NODE-LOCAL』
- 471 ページの『SQLE-NODE-NPIPE』

関連サンプル:

- 『ininfo.c -- Set and get information at the instance level (C)』
- 『ininfo.C -- Set and get information at the instance level (C++)』
- 『nodecat.cbl -- Get node directory information (IBM COBOL)』

sqledcgd - データベース・コメントの変更

システム・データベース・ディレクトリーまたはローカル・データベース・ディレクトリー内のデータベース・コメントを変更します。現行のコメント関連テキストは、新規のコメント・テキストと置き換えることができます。

有効範囲:

この API は、それが発行されたデータベース・パーティション・サーバーにのみ影響を与えます。

許可:

以下のいずれかです。

- *sysadm*
- *sysctrl*

必要な接続:

なし

API 組み込みファイル:

sqlenv.h

C API 構文:

```
/* File: sqlenv.h */
/* API: sqledcgd */
/* ... */
SQL_API_RC SQL_API_FN
  sqledcgd (
    _SQLOLDCHAR *pDbAlias,
    _SQLOLDCHAR *pPath,
    _SQLOLDCHAR *pComment,
    struct sqlca *pSqlca);
/* ... */
```

汎用 API 構文:

```
/* File: sqlenv.h */
/* API: sqlgdcgd */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgdcgd (
    unsigned short CommentLen,
    unsigned short PathLen,
    unsigned short DbAliasLen,
    struct sqlca *pSqlca,
    _SQLOLDCHAR *pComment,
    _SQLOLDCHAR *pPath,
    _SQLOLDCHAR *pDbAlias);
/* ... */
```

API パラメーター:

CommentLen

入力。コメントの長さを示す 2 バイトの符号なし整数 (バイト単位) です。コメントが提供されていない場合は、ゼロに設定してください。

PathLen
入力。パス・パラメーターの長さを示す 2 バイトの符号なし整数 (バイト単位) です。パスが提供されていない場合は、ゼロに設定してください。

DbAliasLen
入力。データベース別名の長さを示す 2 バイトの符号なし整数 (バイト単位) です。

pSqlca
出力。*sqlca* 構造へのポインター。

pComment
入力。データベースのオプション・コメントを含むストリングを指定します。 NULL ストリングはコメントがないことを示します。既存のデータベース・コメントに変更が加えられないことを示す場合もあります。

pPath
入力。ローカル・データベース・ディレクトリーが存在するパスを含むストリングを指定します。指定されたパスが NULL ポインターである場合、システム・データベース・ディレクトリーが使用されます。

コメントは、API が実行されるデータベース・パーティション・サーバーで、ローカル・データベース・ディレクトリーまたはシステム・データベース・ディレクトリーでの変更だけが行われます。すべてのデータベース・パーティション・サーバーのデータベース・コメントを変更するには、データベース・パーティション・サーバーごとに API を実行します。

pDbAlias
入力。データベース別名を含むストリング。これは、システム・データベース・ディレクトリーにカタログされる名前です。パスが指定されている場合には、ローカル・データベース・ディレクトリーにカタログされる名前です。

REXX API 構文:

```
CHANGE DATABASE database_alias COMMENT [ON path] WITH comment
```

REXX API パラメーター:

database_alias
コメントが変更されるデータベースの別名を指定します。

システム・データベース・ディレクトリーでコメントを変更するには、データベース別名を指定する必要があります。

データベースが存在するパスを (*path* パラメーターで) 指定した場合、データベースの名前 (別名ではなく) を入力してください。ローカル・データベース・ディレクトリーでコメントを変更するには、この方式を使用してください。

path データベースが存在するパス。

comment
システム・データベース・ディレクトリーまたはローカル・データベース・ディレクトリーの項目を記述します。カタログしたデータベースについての記述を補足する、あらゆるコメントを入力することができます。コメント・

ストリングの最大長は 30 文字です。復帰文字や改行文字は許可されません。コメント・テキストは必ず二重引用符で囲んでください。

使用上の注意:

既存のコメント・テキストは、新規のテキストに置き換えられます。情報を追加する場合、既存のコメント・テキストに続けて新規テキストを入力してください。

データベース別名と関連する項目のコメントだけが修正されます。データベース名が同じでも、別名が異なるその他の項目には影響しません。

パスを指定した場合、データベース別名をローカル・データベース・ディレクトリーにカタログしてください。また、パスを指定しなかった場合は、データベース別名をシステム・データベース・ディレクトリーにカタログしてください。

関連資料:

- 340 ページの『sqlecadb - データベースのカタログ』
- 53 ページの『db2DbDirCloseScan - データベース・ディレクトリー・スキャンのクローズ』
- 348 ページの『sqlecrea - データベースの作成』
- 54 ページの『db2DbDirGetNextEntry - データベース・ディレクトリーの次項目の入手』
- 58 ページの『db2DbDirOpenScan - データベース・ディレクトリー・スキャンのオープン』

関連サンプル:

- 『dbcmt.cbl -- Change a database comment in the database directory (IBM COBOL)』
- 『ininfo.c -- Set and get information at the instance level (C)』
- 『ininfo.C -- Set and get information at the instance level (C++)』

sqledpan - ノードでのデータベースのドロップ

指定されたデータベース・パーティション・サーバーでデータベースをドロップします。パーティション・データベース環境でのみ実行可能です。

有効範囲:

この API は、それが呼び出されたデータベース・パーティション・サーバーにのみ影響を与えます。

許可:

以下のいずれかです。

- *sysadm*
- *sysctrl*

必要な接続:

なし。インスタンス・アタッチは呼び出しの期間中に確立されます。

API 組み込みファイル:

sqlenv.h

C API 構文:

```
/* File: sqlenv.h */
/* API: sqledpan */
/* ... */
SQL_API_RC SQL_API_FN
  sqledpan (
    char *pDbAlias,
    void *pReserved,
    struct sqlca *pSqlca);
/* ... */
```

汎用 API 構文:

```
/* File: sqlenv.h */
/* API: sqlgdpan */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgdpan (
    unsigned short Reserved1,
    unsigned short DbAliasLen,
    struct sqlca *pSqlca,
    void *pReserved2,
    char *pDbAlias);
/* ... */
```

API パラメーター:

Reserved1
将来の利用のために予約されています。

DbAliasLen
入力。データベース別名の長さを示す 2 バイトの符号なし整数 (バイト単位) です。

pSqlca
出力。*sqlca* 構造へのポインター。

pReserved2
NULL に設定されたスペア・ポインター、またはゼロを指すスペア・ポインター。将来の利用のために予約されています。

pDbAlias
入力。ドロップされるデータベースの別名を含むストリングを指定します。これは、システム・データベース・ディレクトリーにある実際のデータベース名を参照するための名前です。

REXX API 構文:

この API は、SQLDB2 インターフェースを使って、REXX から呼び出すことができます。

使用上の注意:

この API を不適切な仕方で使用すると、システム内に不整合が生じるので、注意して使用してください。

関連タスク:

- 「*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*」の『REXX における SQLEXEC、SQLDBS、および SQLDB2 の登録』

関連資料:

- 366 ページの『sqledrpd - データベースのドロップ』
- 346 ページの『sqlecran - ノードでのデータベースの作成』
- 453 ページの『SQLCA』

sqledreg - 登録解除

ネットワーク・ファイル・サーバーから DB2 サーバーの登録を解除します。 DB2 サーバーのネットワーク・アドレスが、ファイル・サーバーの指定されたレジストリーから除去されます。

許可:

なし

必要な接続:

なし

API 組み込みファイル:

sqlenv.h

C API 構文:

```
/* File: sqlenv.h */
/* API: sqledreg */
/* ... */
SQL_API_RC SQL_API_FN
  sqledreg (
    unsigned short Registry,
    void *pRegisterInfo,
    struct sqlca *pSqlca);
/* ... */
```

汎用 API 構文:

```
/* File: sqlenv.h */
/* API: sqlgdreg */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgdreg (
    unsigned short Registry,
    void *pRegisterInfo,
    struct sqlca *pSqlca);
/* ... */
```

API パラメーター:

Registry
入力。ネットワーク・ファイル・サーバー上で DB2 サーバーの登録を解除

する位置を示します。このリリースでは、 SQL_NWBINDERY レジストリー (sqlenv で定義された NetWare ファイル・サーバー・バインダリー) だけがサポートされています。

pRegisterInfo
入力。 *sqle_reg_nwbindery* 構造を指すポインター。この構造では、呼び出し側はネットワーク・ファイル・サーバーで有効なユーザー名とパスワードを指定します。

pSqlca
出力。*sqlca* 構造へのポインター。

REXX API 構文:

この API は、SQLDB2 インターフェースを使って、REXX から呼び出すことができます。

使用上の注意:

Registry の値が SQL_NWBINDERY の場合、この API は、 *sqle_reg_nwbindery* 構造で指定された NetWare ユーザー名とパスワードを使用して、データベース・マネージャー構成ファイルで指定された NetWare ファイル・サーバー (FILESERVER) にログオンします。データベース・マネージャー構成ファイルで指定されたオブジェクト名 (OBJECTNAME) は、 NetWare ファイル・サーバー・バインダリーから削除されます。

指定した NetWare ユーザー名およびパスワードには、監視またはそれと同等の権限が必要です。

この API は、 DB2 サーバーからローカルに発行しなければなりません。リモートにはサポートされていません。

IPX/SPX フィールドを再構成する場合、または DB2 サーバーの IPX/SPX インターネットワーク・アドレスを変更する場合は、変更を行う前にまずネット作業ファイル・サーバーから DB2 サーバーの登録を解除し、次に変更を行って、最後に登録し直してください。

関連タスク:

- 「*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*」の『REXX における SQLEXEC、SQLDBS、および SQLDB2 の登録』

関連資料:

- 400 ページの『sqleregs - 登録』
- 453 ページの『SQLCA』
- 473 ページの『SQLE-REG-NWBINDERY』
- 「*コマンド・リファレンス*」の『DEREGISTER コマンド』

sqledrpd - データベースのドロップ

データベースの内容とそのすべてのログ・ファイルを削除し、データベースをアンカタログし、さらにデータベースのサブディレクトリーを削除します。

有効範囲:

デフォルトは、この API は db2nodes.cfg ファイルにリストされているすべてのデータベース・パーティション・サーバーに影響を与えます。

許可:

以下のいずれかです。

- *sysadm*
- *sysctrl*

必要な接続:

インスタンス。リモート・データベースを呼び出す場合、その前に ATTACH を呼び出す必要はありません。データベースがリモートとしてカタログされている場合、リモート・ノードへのインスタンス・アタッチは、呼び出しの期間中に確立されます。

API 組み込みファイル:

sqlenv.h

C API 構文:

```
/* File: sqlenv.h */
/* API: sqledrpd */
/* ... */
SQL_API_RC SQL_API_FN
  sqledrpd (
    _SQLOLDCHAR *pDbAlias,
    struct sqlca *pSqlca);
/* ... */
```

汎用 API 構文:

```
/* File: sqlenv.h */
/* API: sqlgdrpd */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgdrpd (
    unsigned short Reserved1,
    unsigned short DbAliasLen,
    struct sqlca *pSqlca,
    _SQLOLDCHAR *pReserved2,
    _SQLOLDCHAR *pDbAlias);
/* ... */
```

API パラメーター:

Reserved1
将来の利用のために予約されています。

DbAliasLen
入力。データベース別名の長さを示す 2 バイトの符号なし整数 (バイト単位) です。

pSqlca
出力。*sqlca* 構造へのポインター。

pReserved2
NULL に設定されたスペア・ポインター、またはゼロを指すスペア・ポインター。将来の利用のために予約されています。

pDbAlias
入力。ドロップされるデータベースの別名を含むストリングを指定します。これは、システム・データベース・ディレクトリーにある実際のデータベース名を参照するための名前です。

REXX API 構文:

```
DROP DATABASE dbalias
```

REXX API パラメーター:

dbalias
ドロップするデータベースの別名。

使用上の注意:

sqledrpd はすべてのユーザー・データとログ・ファイルを削除します。リストア操作の後でロールフォワード・リカバリー用のログ・ファイルが必要な場合は、この API を呼び出す前にファイルを保管する必要があります。

データベースは使用中であってはなりません。データベースをドロップする前に、すべてのユーザーをデータベースから切断しなければなりません。

ドロップするには、データベースがシステム・データベース・ディレクトリーにカタログされている必要があります。指定されたデータベース別名だけが、システム・データベース・ディレクトリーから除去されます。同じデータベースに対して他の別名が存在する場合、その項目はそのままです。ドロップされるデータベースがローカル・データベース・ディレクトリーの最後の項目である場合、ローカル・データベース・ディレクトリーは自動的に削除されます。

この API がリモート・クライアント (または同一マシンの別のインスタンス) から呼び出される場合、指定された別名はクライアントのシステム・データベース・ディレクトリーから除去されます。それに対応するデータベース名は、サーバーのシステム・データベース・ディレクトリーから除去されます。

この API は DATALINK 列を介してリンクされているすべてのファイルをリンク解除します。リンク解除操作は DB2 Data Links Manager で非同期に実行されるので、その効果が即座に DB2 Data Links Manager で現れなかったり、リンク解除されたファイルが他の操作に即時に使用できるようにならなかったりする場合があります。 API が呼び出されると、そのデータベースへのすべての DB2 Data Links Manager 構成は使用可能になるはずです。そうでない場合、データベースのドロップは失敗します。

関連資料:

- 340 ページの『sqlecadb - データベースのカタログ』
- 348 ページの『sqlecrea - データベースの作成』

- 411 ページの『sqleuncd - データベースのアンカタログ』
- 346 ページの『sqlecran - ノードでのデータベースの作成』
- 362 ページの『sqledpan - ノードでのデータベースのドロップ』
- 453 ページの『SQLCA』

関連サンプル:

- 『dbconf.cbl -- Update database configuration (IBM COBOL)』
- 『dbcreate.c -- Create and drop databases (C)』
- 『dbcreate.C -- Create and drop databases (C++)』

sqledrpn - ノード・ドロップの検査

データベース・パーティション・サーバーがデータベースによって使用されているかどうかを検査します。データベース・パーティション・サーバーをドロップできるかどうかを示すメッセージが戻されます。

有効範囲:

この API は、それが発行されたデータベース・パーティション・サーバーにのみ影響を与えます。

許可:

以下のいずれかです。

- *sysadm*
- *sysctrl*

API 組み込みファイル:

sqlenv.h

C API 構文:

```
/* File: sqlenv.h */
/* API: sqledrpn */
/* ... */
SQL_API_RC SQL_API_FN
  sqledrpn (
    unsigned short Action,
    void *pReserved,
    struct sqlca *pSqlca);
/* ... */
```

汎用 API 構文:

```
/* File: sqlenv.h */
/* API: sqlgdrpn */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgdrpn (
    unsigned short Reserved1,
    struct sqlca *pSqlca,
    void *pReserved2,
    unsigned short Action);
/* ... */
```

API パラメーター:

Reserved1

pReserved2 の長さのために予約されています。

pSqlca

出力。*sqlca* 構造へのポインター。

pReserved2

NULL に設定されたスペア・ポインター、または 0 を指すスペア・ポインター。将来の利用のために予約されています。

Action

要求されたアクション。有効値は以下のとおりです。

SQL_DROPNODE_VERIFY

REXX API 構文:

この API は、SQLDB2 インターフェースを使って、REXX から呼び出すことができます。

使用上の注意:

データベース・パーティション・サーバーが使用されていないことを示すメッセージが戻された場合は、 **db2stop** コマンドと DROP NODENUM を使用して、db2nodes.cfg ファイルからそのデータベース・パーティション・サーバーの項目をドロップします。そうすると、パーティション・データベース環境からデータベース・パーティション・サーバーがドロップされます。

データベース・パーティション・サーバーが使用されていることを示すメッセージが戻された場合は、以下の処置を実行してください。

1. ドロップされるデータベース・パーティション・サーバーには、インスタンス内の各データベース用にデータベース・パーティションがあります。これらのデータベース・パーティションのいずれかにデータが含まれている場合は、データベース・パーティションを使用するデータベース・パーティション・グループを再分散します。データベース・パーティション・グループを再分散し、ドロップされないデータベース・パーティション・サーバーにあるデータベース・パーティションにデータを移動させます。

   データベース・パーティション・グループの再分散後、データベース・パーティションを使用する各データベース・パーティション・グループからデータベース・パーティションをドロップします。データベース・パーティションをデータベース・パーティション・グループからドロップするには、 sqludrdt API のノード・ドロップ・オプション、または ALTER DATABASE PARTITION GROUP ステートメントを使用できます。
2. データベース・パーティション・サーバーで定義されているイベント・モニターをドロップします。
3. sqledrpn を再実行して、データベース・パーティション・サーバーのデータベース・パーティションが使用されていないことを確認します。

関連タスク:

- 「*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*」の『REXX における SQLEXEC、SQLDBS、および SQLDB2 の登録』

関連資料:

- 332 ページの『sqleaddn - ノードの追加』
- 453 ページの『SQLCA』

sqledtin - 切り離し

論理インスタンス接続を除去します。この層を使用した論理接続がほかにない場合、物理通信接続も終了します。

許可:

なし

必要な接続:

なし。既存のインスタンス・アタッチを除去します。

API 組み込みファイル:

sqlenv.h

C API 構文:

```
/* File: sqlenv.h */
/* API: sqledtin */
/* ... */
SQL_API_RC SQL_API_FN
  sqledtin (
    struct sqlca *pSqlca);
/* ... */
```

汎用 API 構文:

```
/* File: sqlenv.h */
/* API: sqlgdtin */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgdtin (
    struct sqlca *pSqlca);
/* ... */
```

API パラメーター:

pSqlca
出力。*sqlca* 構造へのポインター。

REXX API 構文:

```
DETACH
```

関連資料:

- 337 ページの『sqleatin - アタッチ』
- 453 ページの『SQLCA』

関連サンプル:

- 『dbinst.cbl -- Attach to and detach from an instance (IBM COBOL)』
- 『inattach.c -- Attach to and detach from an instance (C)』

- 『utilapi.c -- Error-checking utility for non-embedded SQL samples in C (C)』
- 『inattach.C -- Attach to and detach from an instance (C++)』
- 『utilapi.C -- Checks for and prints to the screen SQL warnings and errors (C++)』

sqlefmem - メモリーの解放

DB2 API によって割り振られたメモリーを、呼び出し側に代わって解放します。この API は、sqlbtcq および sqlbmtsq とともに使用するよう意図されています。

許可:

なし

必要な接続:

なし

API 組み込みファイル:

sqlenv.h

C API 構文:

```
/* File: sqlenv.h */
/* API: sqlefmem */
/* ... */
SQL_API_RC SQL_API_FN
  sqlefmem (
    struct sqlca *pSqlca,
    void *pBuffer);
/* ... */
```

汎用 API 構文:

```
/* File: sqlenv.h */
/* API: sqlgfmem */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgfmem (
    struct sqlca *pSqlca,
    void *pBuffer);
/* ... */
```

API パラメーター:

pSqlca
出力。*sqlca* 構造へのポインター。

pBuffer
入力。解放されるメモリーを指すポインター。

関連資料:

- 313 ページの『sqlbmtsq - 表スペース照会』
- 324 ページの『sqlbtcq - 表スペース・コンテナー照会』
- 453 ページの『SQLCA』

関連サンプル:

- 『dbrecov.sqc -- How to recover a database (C)』
- 『tsinfo.sqc -- How to get information at the table space level (C)』
- 『dbrecov.sqC -- How to recover a database (C++)』
- 『tsinfo.sqC -- How to get information at the table space level (C++)』
- 『tabscont.sqb -- How to get tablespace container information (IBM COBOL)』
- 『tabspace.sqb -- How to get tablespace information (IBM COBOL)』
- 『tspace.sqb -- How to copy and free memory in a tablespace (IBM COBOL)』

sqlefrce - アプリケーションの強制終了

システムからローカルまたはリモートのユーザーやアプリケーションを強制終了し、サーバー上での保守を可能にします。

重要: 割り込みできない操作 (たとえばデータベース・リストア) を強制終了する場合、データベースが利用可能になるには、その操作の再実行が正常終了しなければなりません。

有効範囲:

この API は db2nodes.cfg ファイルにリストされているすべてのデータベース・パーティション・サーバーに影響を与えます。

パーティション・データベース環境では、強制終了されているアプリケーションのコーディネーター・パーティションからこの API を発行する必要はありません。この API は、パーティション・データベース環境ではどのデータベース・パーティション・サーバーからでも発行できます。

許可:

以下のいずれかです。

- *sysadm*
- *sysctrl*
- *sysmaint*

必要な接続:

インスタンス。リモート・サーバーからユーザーを強制終了する場合、最初にそのサーバーにアタッチする必要があります。アタッチがない場合、この API はローカルで実行されます。

API 組み込みファイル:

sqlenv.h

C API 構文:

```
/* File: sqlenv.h */
/* API: sqlefrce */
/* ... */
SQL_API_RC SQL_API_FN
  sqlefrce (
    long NumAgentIds,
```

```
    sqluint32 *pAgentIds,
    unsigned short ForceMode,
    struct sqlca *pSqlca);
/* ... */
```

汎用 API 構文:

```
/* File: sqlenv.h */
/* API: sqlgfrce */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgfrce (
    struct sqlca *pSqlca,
    unsigned short ForceMode,
    sqluint32 *pAgentIds,
    long NumAgentIds);
/* ... */
```

API パラメーター:

pSqlca
出力。*sqlca* 構造へのポインター。

ForceMode
入力。 sqlefrce API の操作モードを指定する整数です。非同期モードだけがサポートされています。つまり、API は指定されたすべてのユーザーが終了していなくても戻されます。 API が正常に出されるかエラーが発生するとすぐに戻ります。その結果、アプリケーション強制終了の呼び出しが完了してから、指定されたユーザーが終了するまでに若干時間がかかることがあります。

このパラメーターは、 SQL_ASYNCH (sqlenv で定義) に設定しなければなりません。

pAgentIds
入力。符号なしの長整数の配列を指すポインター。各項目は、対応するデータベース・ユーザーのエージェント ID を示します。

NumAgentIds
入力。終了するユーザーの合計数を示す整数です。この数はエージェント ID の配列のエレメント数と同じにする必要があります。

このパラメーターが (sqlenv で定義された) SQL_ALL_USERS に設定された場合、データベース接続またはインスタンス接続を使用するすべてのアプリケーションが強制終了されます。このパラメーターがゼロに設定された場合、エラーが戻されます。

REXX API 構文:

```
FORCE APPLICATION {ALL | :agentidarray} [MODE ASYNC]
```

REXX API パラメーター:

ALL すべてのアプリケーションが切断されます。これには、データベース接続を使用するアプリケーション、およびインスタンス接続を使用するアプリケーションが含まれます。

agentidarray
終了されるエージェント ID のリストを含むコンパウンド REXX ホスト変数。以下の項目において、XXX はホスト変数の名前を表しています。

XXX.0 終了されるエージェントの数。

XXX.1 最初のエージェント ID。

XXX.2 2 番目のエージェント ID。

XXX.3 以降、3 番目、4 番目 ... と続きます。

ASYNC
現在サポートされている唯一のモード。 sqlefrce は指定されたすべてのアプリケーションが終了しなくても戻されます。

使用上の注意:

db2stop は強制終了の間は実行できません。データベース・マネージャーはアクティブのままなので、その後のデータベース・マネージャー操作は **db2start** を呼び出さなくても処理できます。

データベースの保全性を確保するため、終了できるのは、アイドル中のユーザー、または割り込み可能なデータベース操作を実行中のユーザーだけです。

FORCE を出した後でも、データベースはまだ接続要求を受諾します。すべてのユーザーを完全に強制終了するために、追加の FORCE が必要になる場合があります。

強制終了されるユーザーのエージェント ID を収集するには、データベース・システム・モニター機能を使用します。

強制終了モードが SQL_ASYNCH (許可されている唯一の値) に設定されている場合、API は呼び出しアプリケーションにすぐに戻ります。

最小の検証が強制終了されるエージェント ID の配列上で実行されます。ユーザーは、指定したエレメントの合計数を含む配列をポインターが指していることを確認する必要があります。 *NumAgentIds* が SQL_ALL_USERS に設定されている場合、その配列は無視されます。

ユーザーが接続終了したとき、データベースの整合性を確認するために ROLLBACK が実行されます。

強制切断できるすべてのユーザーを強制切断します。指定されたエージェント ID が 1 つ以上見つからない場合、 *sqlca* 構造の *sqlcode* が 1230 に設定されます。たとえば、エージェント ID の収集と sqlefrce の呼び出しの間にユーザーがサインオフすると、エージェント ID が見つからない場合があります。 API を呼び出すユーザーは決して強制終了されません。

エージェント ID は繰り返し使用することができます。そして、そのエージェント ID が、データベース・システム・モニターによる収集の後に、アプリケーションを強制終了するために使用されることもあります。したがって、あるユーザーがサインオフした場合、別のユーザーがサインオンして、そのサインオフしたユーザーと同じエージェント ID を、この再利用プロセスによって獲得することができます。しかし、このときに、間違ったユーザーを強制終了してしまうおそれもあります。

関連資料:

• 89 ページの『db2GetSnapshot - スナップショットの入手』

- 337 ページの『sqleatin - アタッチ』
- 370 ページの『sqledtin - 切り離し』
- 453 ページの『SQLCA』

関連サンプル:

- 『dbconn.sqc -- How to connect to and disconnect from a database (C)』
- 『dbsample.sqc -- Creates a sample database (C)』
- 『instart.c -- Stop and start the current local instance (C)』
- 『dbconn.sqC -- How to connect to and disconnect from a database (C++)』
- 『instart.C -- Stop and start the current local instance (C++)』
- 『dbstop.cbl -- How to stop a database manager (IBM COBOL)』

sqlegdad - DCS データベースのカタログ

リモート・データベースに関する情報を、データベース接続サービス (DCS) ディレクトリーに保管します。このようなデータベースには、 DB2 Connect などのアプリケーション・リクエスター (AR) を介してアクセスします。システム・データベース・ディレクトリー内のデータベース名と一致する名前が DCS ディレクトリー項目にある場合、指定した AR を呼び出して、データベースが存在するリモート・サーバーに SQL 要求を転送します。

許可:

以下のいずれかです。

- *sysadm*
- *sysctrl*

必要な接続:

なし

API 組み込みファイル:

sqlenv.h

C API 構文:

```
/* File: sqlenv.h */
/* API: sqlegdad */
/* ... */
SQL_API_RC SQL_API_FN
  sqlegdad (
    struct sql_dir_entry *pDCSDirEntry,
    struct sqlca *pSqlca);
/* ... */
```

汎用 API 構文:

```
/* File: sqlenv.h */
/* API: sqlggdad */
/* ... */
SQL_API_RC SQL_API_FN
  sqlggdad (
    struct sqlca *pSqlca,
    struct sql_dir_entry *pDCSDirEntry);
/* ... */
```

API パラメーター:

pSqlca
出力。*sqlca* 構造へのポインター。

pDCSDirEntry
入力。 *sql_dir_entry* (データベース接続サービス・ディレクトリー) 構造を指すポインター。

REXX API 構文:

```
CATALOG DCS DATABASE dbname [AS target_dbname]
[AR arname] [PARMS parms] [WITH comment]
```

REXX API パラメーター:

dbname
追加されるディレクトリー項目のローカル・データベース名。

target_dbname
ターゲット・データベース名。

arname
アプリケーション・クライアント名。

parms
パラメーター・ストリング。指定する場合、このストリングは二重引用符 (″) で囲む必要があります。

comment
項目に関連したコメント。最大長は 30 文字です。コメントは二重引用符 (″) で囲んでください。

使用上の注意:

DB2 Connectプログラムは、以下のような DRDA アプリケーション・サーバーへの接続を提供します。

- System/370 および System/390 アーキテクチャーのホスト・コンピューター上の DB2 for OS/390 データベース。
- System/370 および System/390 アーキテクチャーのホスト・コンピューター上の DB2 for VM and VSE データベース。
- Application System/400 (AS/400) ホスト・コンピューター上の OS/400 データベース。

データベース・マネージャーは、データベース接続サービス・ディレクトリーがなければ自分で作成します。このディレクトリーは、使用しているデータベース・マネージャー・インスタンスを含むパスに保管されます。また、データベースの外側で保持されます。

データベースは、システム・データベース・ディレクトリーにリモート・データベースとしてもカタログしなければなりません。

注: ディレクトリーのキャッシュが使用可能にされている場合、データベース、ノード、および DCS ディレクトリー・ファイルはメモリーにキャッシュされます。アプリケーションのディレクトリー・キャッシュは、最初のディレクトリ

一検索時に作成されます。キャッシュはアプリケーションがディレクトリー・ファイルのどれかを修正したときにのみ最新にされるため、他のアプリケーションが行ったディレクトリーの変更は、アプリケーションを再始動するまで有効にならないことがあります。 DB2 の共用キャッシュを最新にするには (サーバーのみ)、データベース・マネージャーを停止させてから (**db2stop**)、再始動 (**db2start**) させてください。別のアプリケーション用のディレクトリー・キャッシュを最新にするには、そのアプリケーションを停止させてから再始動させてください。

関連資料:

- 377 ページの『sqlegdcl - DCS ディレクトリー・スキャンのクローズ』
- 380 ページの『sqlegdge - データベース用 DCS ディレクトリー項目の入手』
- 382 ページの『sqlegdgt - DCS ディレクトリー項目の入手』
- 384 ページの『sqlegdsc - DCS ディレクトリー・スキャンのオープン』
- 378 ページの『sqlegdel - DCS データベースのアンカタログ』
- 453 ページの『SQLCA』
- 445 ページの『SQL-DIR-ENTRY』

関連サンプル:

- 『dcscat.cbl -- Get information for a DCS directory in a database (IBM COBOL)』
- 『ininfo.c -- Set and get information at the instance level (C)』
- 『ininfo.C -- Set and get information at the instance level (C++)』

sqlegdcl - DCS ディレクトリー・スキャンのクローズ

sqlegdsc (DCS ディレクトリー・スキャンのオープン) によって割り振られたリソースを解放します。

許可:

なし

必要な接続:

なし

API 組み込みファイル:

sqlenv.h

C API 構文:

```
/* File: sqlenv.h */
/* API: sqlegdcl */
/* ... */
SQL_API_RC SQL_API_FN
  sqlegdcl (
    struct sqlca *pSqlca);
/* ... */
```

汎用 API 構文:

```
/* File: sqlenv.h */
/* API: sqlggdcl */
/* ... */
SQL_API_RC SQL_API_FN
  sqlggdcl (
    struct sqlca *pSqlca);
/* ... */
```

API パラメーター:

pSqlca
出力。*sqlca* 構造へのポインター。

REXX API 構文:

```
CLOSE DCS DIRECTORY
```

関連資料:

- 382 ページの『sqlegdgt - DCS ディレクトリー項目の入手』
- 384 ページの『sqlegdsc - DCS ディレクトリー・スキャンのオープン』
- 453 ページの『SQLCA』

関連サンプル:

- 『dcscat.cbl -- Get information for a DCS directory in a database (IBM COBOL)』
- 『ininfo.c -- Set and get information at the instance level (C)』
- 『ininfo.C -- Set and get information at the instance level (C++)』

sqlegdel - DCS データベースのアンカタログ

データベース接続サービス (DCS) ディレクトリーから項目を削除します。

許可:

以下のいずれかです。

- *sysadm*
- *sysctrl*

必要な接続:

なし

API 組み込みファイル:

sqlenv.h

C API 構文:

```
/* File: sqlenv.h */
/* API: sqlegdel */
/* ... */
SQL_API_RC SQL_API_FN
  sqlegdel (
    struct sql_dir_entry *pDCSDirEntry,
    struct sqlca *pSqlca);
/* ... */
```

汎用 API 構文:

```
/* File: sqlenv.h */
/* API: sqlggdel */
/* ... */
SQL_API_RC SQL_API_FN
  sqlggdel (
    struct sqlca *pSqlca,
    struct sql_dir_entry *pDCSDirEntry);
/* ... */
```

API パラメーター:

pSqlca
出力。*sqlca* 構造へのポインター。

pDCSDirEntry
入力/出力。データベース接続サービス・ディレクトリー構造を指すポインター。この構造の *ldb* フィールドには、削除するデータベースのローカル名を入れてください。一致するローカル・データベース名がある DCS ディレクトリー項目は、削除前にこの構造にコピーされます。

REXX API 構文:

```
UNCATALOG DCS DATABASE dbname [USING :value]
```

REXX API パラメーター:

dbname
削除されるディレクトリー項目のローカル・データベース名。

value ディレクトリー項目情報が戻されるコンパウンド REXX ホスト変数。以下の項目において、XXX はホスト変数名を表しています。名前が指定されなかった場合、名前 SQLGWINF が使用されます。

XXX.0	変数内のエレメント数 (常に 7)
XXX.1	RELEASE
XXX.2	LDB
XXX.3	TDB
XXX.4	AR
XXX.5	PARMS
XXX.6	COMMENT
XXX.7	RESERVED

使用上の注意:

DCS データベースは、sqleuncd API を使用してアンカタログできるリモート・データベースとして、システム・データベース・ディレクトリーにもカタログされています。

DCS ディレクトリー内のデータベースを再カタログするには、sqlegdad API を使用してください。

ノード上でカタログされているデータベースをリストするには、sqlegdsc、sqlegdgt、および sqlegdcl API を使用してください。

ディレクトリーのキャッシュが *dir_cache* 構成パラメーターを使用して使用可能にされている場合、データベース、ノード、および DCS ディレクトリー・ファイルはメモリーにキャッシュされます。アプリケーションのディレクトリー・キャッシュは、最初のディレクトリー検索時に作成されます。キャッシュはアプリケーションがディレクトリー・ファイルのどれかを修正したときにのみ最新にされるため、他のアプリケーションが行ったディレクトリーの変更は、アプリケーションを再始動するまで有効にならないことがあります。 DB2 の共用キャッシュを最新にするには (サーバーのみ)、データベース・マネージャーを停止させてから (**db2stop**)、再始動 (**db2start**) させてください。別のアプリケーション用のディレクトリー・キャッシュを最新にするには、そのアプリケーションを停止させてから再始動させてください。

関連資料:

- 375 ページの『sqlegdad - DCS データベースのカタログ』
- 377 ページの『sqlegdcl - DCS ディレクトリー・スキャンのクローズ』
- 380 ページの『sqlegdge - データベース用 DCS ディレクトリー項目の入手』
- 382 ページの『sqlegdgt - DCS ディレクトリー項目の入手』
- 384 ページの『sqlegdsc - DCS ディレクトリー・スキャンのオープン』
- 411 ページの『sqleuncd - データベースのアンカタログ』
- 453 ページの『SQLCA』
- 445 ページの『SQL-DIR-ENTRY』
- 36 ページの『db2CfgGet - 構成パラメーターの入手』

関連サンプル:

- 『dcscat.cbl -- Get information for a DCS directory in a database (IBM COBOL)』
- 『ininfo.c -- Set and get information at the instance level (C)』
- 『ininfo.C -- Set and get information at the instance level (C++)』

sqlegdge - データベース用 DCS ディレクトリー項目の入手

データベース接続サービス (DCS) ディレクトリーにある特定の項目の情報を戻します。

許可:

なし

必要な接続:

なし

API 組み込みファイル:

sqlenv.h

C API 構文:

```
/* File: sqlenv.h */
/* API: sqlegdge */
/* ... */
SQL_API_RC SQL_API_FN
  sqlegdge (
    struct sql_dir_entry *pDCSDirEntry,
    struct sqlca *pSqlca);
/* ... */
```

汎用 API 構文:

```
/* File: sqlenv.h */
/* API: sqlggdge */
/* ... */
SQL_API_RC SQL_API_FN
  sqlggdge (
    struct sqlca *pSqlca,
    struct sql_dir_entry *pDCSDirEntry);
/* ... */
```

API パラメーター:

pSqlca
出力。*sqlca* 構造へのポインター。

pDCSDirEntry
入力/出力。データベース接続サービス・ディレクトリー構造へのポインター。この構造の *ldb* フィールドは、検索する DCS ディレクトリー項目のあるデータベースのローカル名で充てんしてください。構造内の残りのフィールドは、この API の戻りに埋め込まれます。

REXX API 構文:

```
GET DCS DIRECTORY ENTRY FOR DATABASE dbname [USING :value]
```

REXX API パラメーター:

dbname
取得するディレクトリー項目のローカル・データベース名を指定します。

value ディレクトリー項目情報が戻されるコンパウンド REXX ホスト変数。以下の項目において、XXX はホスト変数名を表しています。名前が指定されなかった場合、名前 SQLGWINF が使用されます。

XXX.0	変数内のエレメント数 (常に 7)
XXX.1	RELEASE
XXX.2	LDB
XXX.3	TDB
XXX.4	AR
XXX.5	PARMS
XXX.6	COMMENT
XXX.7	RESERVED

関連資料:

- 375 ページの『sqlegdad - DCS データベースのカタログ』
- 377 ページの『sqlegdcl - DCS ディレクトリー・スキャンのクローズ』

- 382 ページの『sqlegdgt - DCS ディレクトリー項目の入手』
- 384 ページの『sqlegdsc - DCS ディレクトリー・スキャンのオープン』
- 378 ページの『sqlegdel - DCS データベースのアンカタログ』
- 445 ページの『SQL-DIR-ENTRY』

関連サンプル:

- 『dcscat.cbl -- Get information for a DCS directory in a database (IBM COBOL)』
- 『ininfo.c -- Set and get information at the instance level (C)』
- 『ininfo.C -- Set and get information at the instance level (C++)』

sqlegdgt - DCS ディレクトリー項目の入手

データベース接続サービス (DCS) ディレクトリー項目のコピーを、アプリケーションが提供したバッファーへ転送します。

許可:

なし

必要な接続:

なし

API 組み込みファイル:

sqlenv.h

C API 構文:

```
/* File: sqlenv.h */
/* API: sqlegdgt */
/* ... */
SQL_API_RC SQL_API_FN
  sqlegdgt (
    short *pNumEntries,
    struct sql_dir_entry *pDCSDirEntries,
    struct sqlca *pSqlca);
/* ... */
```

汎用 API 構文:

```
/* File: sqlenv.h */
/* API: sqlggdgt */
/* ... */
SQL_API_RC SQL_API_FN
  sqlggdgt (
    struct sqlca *pSqlca,
    short *pNumEntries,
    struct sql_dir_entry *pDCSDirEntries);
/* ... */
```

API パラメーター:

pSqlca
出力。*sqlca* 構造へのポインター。

pNumEntries
入力/出力。呼び出し側のバッファーにコピーされる項目数を示す短整数を指すポインター。実際にコピーされる項目の数が戻されます。

pDCSDirEntries
出力。収集された DCS ディレクトリー項目が、 API 呼び出しの戻りに保留される場合のバッファーを指すポインター。バッファーには、*pNumEntries* パラメーターで指定された数の項目を保留するだけの十分な大きさが必要です。

REXX API 構文:

```
GET DCS DIRECTORY ENTRY [USING :value]
```

REXX API パラメーター:

value ディレクトリー項目情報が戻されるコンパウンド REXX ホスト変数。以下の項目において、XXX はホスト変数名を表しています。名前が指定されなかった場合、名前 SQLGWINF が使用されます。

XXX.0	変数内のエレメント数 (常に 7)
XXX.1	RELEASE
XXX.2	LDB
XXX.3	TDB
XXX.4	AR
XXX.5	PARMS
XXX.6	COMMENT
XXX.7	RESERVED

使用上の注意:

GET DCS DIRECTORY ENTRIES を発行する前に、項目のカウントを戻す sqlegdsc (DCS ディレクトリー・スキャンのオープン) を呼び出す必要があります。

すべての項目が呼び出し側にコピーされた場合、データベース接続サービス・ディレクトリー・スキャンは自動的にクローズします。また、すべてのリソースが解放されます。

項目が残っている場合、さらにこの API を呼び出すか、 CLOSE DCS DIRECTORY SCAN を呼び出して、システム・リソースを解放してください。

関連資料:

- 377 ページの『sqlegdcl - DCS ディレクトリー・スキャンのクローズ』
- 380 ページの『sqlegdge - データベース用 DCS ディレクトリー項目の入手』
- 384 ページの『sqlegdsc - DCS ディレクトリー・スキャンのオープン』
- 453 ページの『SQLCA』
- 445 ページの『SQL-DIR-ENTRY』

関連サンプル:

- 『dcscat.cbl -- Get information for a DCS directory in a database (IBM COBOL)』

- 『ininfo.c -- Set and get information at the instance level (C)』
- 『ininfo.C -- Set and get information at the instance level (C++)』

sqlegdsc - DCS ディレクトリー・スキャンのオープン

データベース接続サービス・ディレクトリー項目のコピーをメモリーに保管するとともに、項目の数を戻します。このコピーは、ディレクトリーがオープンする時点のディレクトリーのスナップショットです。

この API への呼び出しの後にディレクトリー自体に変更が加えられることがあっても、このコピーが更新されることはありません。項目を検索するには、sqlegdgt (DCS ディレクトリー項目の入手) を使用します。この API の呼び出しに関連付けられたリソースを解放するには、 sqlegdcl (DCS ディレクトリー・スキャンのクローズ) を使用します。

許可:

なし

必要な接続:

なし

API 組み込みファイル:

sqlenv.h

C API 構文:

```
/* File: sqlenv.h */
/* API: sqlegdsc */
/* ... */
SQL_API_RC SQL_API_FN
  sqlegdsc (
    short *pNumEntries,
    struct sqlca *pSqlca);
/* ... */
```

汎用 API 構文:

```
/* File: sqlenv.h */
/* API: sqlggdsc */
/* ... */
SQL_API_RC SQL_API_FN
  sqlggdsc (
    struct sqlca *pSqlca,
    short *pNumEntries);
/* ... */
```

API パラメーター:

pSqlca
出力。*sqlca* 構造へのポインター。

pNumEntries
出力。ディレクトリー項目の数が戻される 2 バイト域のアドレスを示します。

REXX API 構文:

```
OPEN DCS DIRECTORY
```

使用上の注意:

スキャンの呼び出し側では、戻された値 *pNumEntries* を使用して、項目を受け取るのに十分なメモリーを割り振ります。コピーがすでに保留されているのにスキャン呼び出しを受け取る場合、直前のコピーは解放され、新規のコピーが収集されます。

関連資料:

- 377 ページの『sqlegdcl - DCS ディレクトリー・スキャンのクローズ』
- 380 ページの『sqlegdge - データベース用 DCS ディレクトリー項目の入手』
- 382 ページの『sqlegdgt - DCS ディレクトリー項目の入手』
- 453 ページの『SQLCA』

関連サンプル:

- 『dcscat.cbl -- Get information for a DCS directory in a database (IBM COBOL)』
- 『ininfo.c -- Set and get information at the instance level (C)』
- 『ininfo.C -- Set and get information at the instance level (C++)』

sqlegins - インスタンスの入手

DB2INSTANCE 環境変数の値を戻します。

許可:

なし

必要な接続:

なし

API 組み込みファイル:

sqlenv.h

C API 構文:

```
/* File: sqlenv.h */
/* API: sqlegins */
/* ... */
SQL_API_RC SQL_API_FN
  sqlegins (
    _SQLOLDCHAR *pInstance,
    struct sqlca *pSqlca);
/* ... */
```

汎用 API 構文:

```
/* File: sqlenv.h */
/* API: sqlggins */
/* ... */
SQL_API_RC SQL_API_FN
```

```
  sqlggins (
    struct sqlca *pSqlca,
    _SQLOLDCHAR *pInstance);
/* ... */
```

API パラメーター:

pSqlca
出力。*sqlca* 構造へのポインター。

pInstance
出力。データベース・マネージャー・インスタンス名が配置されているストリング・バッファーを指すポインター。 このバッファーの長さは、NULL 終了文字の 1 バイトも含め、少なくとも 9 バイトなければなりません。

REXX API 構文:

```
GET INSTANCE INTO :instance
```

REXX API パラメーター:

instance
データベース・マネージャー・インスタンス名が配置される REXX ホスト変数。

使用上の注意:

DB2INSTANCE 環境変数内の値が、ユーザーのアタッチするインスタンスである必要はありません。

ユーザーが現在アタッチしているインスタンスを識別するには、 *sqlca* 構造の場合を除き、 NULL 引き数を指定して sqleatin (アタッチ) を呼び出してください。

関連資料:

- 337 ページの『sqleatin - アタッチ』
- 453 ページの『SQLCA』

関連サンプル:

- 『dbinst.cbl -- Attach to and detach from an instance (IBM COBOL)』
- 『ininfo.c -- Set and get information at the instance level (C)』
- 『ininfo.C -- Set and get information at the instance level (C++)』

sqleintr - 割り込み

要求を停止させます。この API は、アプリケーションの制御の切れ目シグナル・ハンドラーから呼び出されます。この制御の切れ目シグナル・ハンドラーは、デフォルトに設定することができ、 sqleisig (シグナル・ハンドラーのインストール) や、プログラマーにより提供されるルーチンを用いてインストールすることができます。また、適切なオペレーティング・システム呼び出しを使用してインストールすることもできます。

許可:

なし

必要な接続:

なし

API 組み込みファイル:

sqlenv.h

C API 構文:

```
/* File: sqlenv.h */
/* API: sqleintr */
/* ... */
SQL_API_RC SQL_API_FN
  sqleintr (
    void);
/* ... */
```

汎用 API 構文:

```
/* File: sqlenv.h */
/* API: sqlgintr */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgintr (
    void);
/* ... */
```

API パラメーター:

なし

REXX API 構文:

```
INTERRUPT
```

例:

```
   call SQLDBS 'INTERRUPT'
```

使用上の注意:

割り込みハンドラーからは、 **sqleintr** 以外のデータベース・マネージャー API も呼び出さないようにしてください。しかし、システムがそのことを行わずに済むよう保護することはありません。

コミットまたはロールバックの状態にあるデータベース・トランザクションはすべて、割り込みを行うことができません。

割り込まれたデータベース・マネージャー要求は、割り込まれたことを示すコードを戻します。

以下の表は、割り込み操作が他の API で実行されるときのアクションを示しています。

表 21. INTERRUPT アクション

データベース活動	アクション
BACKUP	ユーティリティーが取り消されます。メディアにあるデータが未完了の可能性があります。
BIND	バインドが取り消されます。パッケージ作成がロールバックされます。
COMMIT	なし。 COMMIT は完了します。
CREATE DATABASE/CREATE DATABASE AT NODE/ADD NODE/DROP NODE VERIFY	ある特定の時点以降、これらの API は割り込み不能になります。その時点以前に割り込み呼び出しを受け取った場合、データベースは作成されません。割り込み呼び出しを受け取るのがその時点以降の場合には、割り込みは無視されます。
DROP DATABASE/DROP DATABASE AT NODE	なし。 API は完了します。
EXPORT/IMPORT/RUNSTATS	ユーティリティーが取り消されます。データベースは、ロールバックを更新します。
FORCE APPLICATION	なし。 FORCE APPLICATION は完了します。
LOAD	ユーティリティーが取り消されます。表内のデータは未完了の可能性があります。
PREP	プリコンパイルは取り消されます。パッケージ作成がロールバックされます。
REORGANIZE TABLE	割り込みはコピーが完了するまで遅れます。表へアクセスする次の試みで、索引の再作成が再開します。
RESTORE	ユーティリティーが取り消されます。 DROP DATABASE が実行されます。表スペース・レベルのリストアには不適切です。
ROLLBACK	なし。 ROLLBACK は完了します。
ディレクトリー・サービス	ディレクトリーは、整合した状態を保ちます。ユーティリティー機能は、実行される場合と、されない場合があります。
SQL データ定義ステートメント	データベース・トランザクションは、 SQL ステートメントの呼び出し前の既存の状態に設定されます。
他の SQL ステートメント	データベース・トランザクションは、 SQL ステートメントの呼び出し前の既存の状態に設定されます。

sqleisig - シグナル・ハンドラーのインストール

デフォルトの割り込み (通常、 Ctrl+C または Ctrl+BREAK あるいはその両方) シグナル・ハンドラーをインストールします。このデフォルトのハンドラーが割り込みシグナルを検出すると、シグナルがリセットされ、sqleintr が呼び出されます。

許可:

なし

必要な接続:

なし

API 組み込みファイル:

sqlenv.h

C API 構文:

```
/* File: sqlenv.h */
/* API: sqleisig */
/* ... */
SQL_API_RC SQL_API_FN
  sqleisig (
    struct sqlca *pSqlca);
/* ... */
```

汎用 API 構文:

```
/* File: sqlenv.h */
/* API: sqlgisig */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgisig (
    struct sqlca *pSqlca);
/* ... */
```

API パラメーター:

pSqlca
出力。*sqlca* 構造へのポインター。

REXX API 構文:

```
INSTALL SIGNAL HANDLER
```

使用上の注意:

アプリケーションがシグナル・ハンドラーを所持しておらず、割り込みを受け取る場合、アプリケーションは終了します。この API は、単純なシグナル処理を備えています。アプリケーションに高度な割り込み処理要件がない場合に、この API を使用することができます。

割り込みシグナル・ハンドラーを正しく機能させるために、この API を呼び出してください。

アプリケーションがより精巧な割り込み処理スキーマを必要とする場合、 sqleintr API も呼び出すことができるシグナル処理ルーチンを開発することができます。オペレーティング・システム呼び出しまたは言語に固有のライブラリー・シグナル関数を使用してください。カスタマイズされたシグナル・ハンドラーによって実行されるデータベース・マネージャー操作は、 sqleintr API だけに限ってください。オペレーティング・システム・プログラミングの技法および慣例に確実に従って、以前にインストールしたシグナル・ハンドラーが正しく機能するようにしてください。

関連資料:

- 386 ページの『sqleintr - 割り込み』
- 453 ページの『SQLCA』

関連サンプル:

- 『dbcmt.cbl -- Change a database comment in the database directory (IBM COBOL)』

sqlemgdb - データベースの移行

以前のバージョン (バージョン 2.x 以降) の DB2 データベースを現行の形式に変換します。

許可:

sysadm

必要な接続:

この API によってデータベース接続が確立されます。

API 組み込みファイル:

sqlenv.h

C API 構文:

```
/* File: sqlenv.h */
/* API: sqlemgdb */
/* ... */
SQL_API_RC SQL_API_FN
  sqlemgdb (
    _SQLOLDCHAR *pDbAlias,
    _SQLOLDCHAR *pUserName,
    _SQLOLDCHAR *pPassword,
    struct sqlca *pSqlca);
/* ... */
```

汎用 API 構文:

```
/* File: sqlenv.h */
/* API: sqlgmgdb */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgmgdb (
    unsigned short PasswordLen,
    unsigned short UserNameLen,
    unsigned short DbAliasLen,
    struct sqlca *pSqlca,
    _SQLOLDCHAR *pPassword,
    _SQLOLDCHAR *pUserName,
    _SQLOLDCHAR *pDbAlias);
/* ... */
```

API パラメーター:

PasswordLen
入力。パスワードの長さを示す 2 バイトの符号なし整数 (バイト単位) です。パスワードが提供されていない場合は、ゼロに設定してください。

UserNameLen
入力。ユーザー名の長さを示す 2 バイトの符号なし整数 (バイト単位) です。ユーザー名が提供されていない場合は、ゼロに設定してください。

DbAliasLen
入力。データベース別名の長さを示す 2 バイトの符号なし整数 (バイト単位) です。

pSqlca
出力。*sqlca* 構造へのポインター。

pPassword
入力。提供されたユーザー名 (ある場合) のパスワードを含むストリング。NULL にすることもできます。

pUserName
入力。アプリケーションのユーザー名を含むストリング。 NULL にすることもできます。

pDbAlias
入力。システム・データベース・ディレクトリーにカタログされているデータベースの別名を含むストリングを指定します。

REXX API 構文:

```
MIGRATE DATABASE dbalias [USER username USING password]
```

REXX API パラメーター:

dbalias
移行するデータベースの別名

username
データベースの再始動に使用されるユーザー名を指定します。

password
ユーザー名の認証に使用されるパスワード。

使用上の注意:

この API はデータベースを新しいバージョンに移行するだけで、移行したデータベースを以前の古いバージョンに変換することはできません。

移行の前にデータベースをカタログする必要があります。

関連資料:

- 453 ページの『SQLCA』

関連サンプル:

- 『dbmigrat.c -- Migrate a database (C)』
- 『dbmigrat.C -- Migrate a database (C++)』
- 『migrate.cbl -- Demonstrates how to migrate to a database (IBM COBOL)』

sqlencls - ノード・ディレクトリー・スキャンのクローズ

sqlenops (ノード・ディレクトリーのオープン) によって割り振られたリソースを解放します。

許可:

なし

必要な接続:

なし

API 組み込みファイル:

sqlenv.h

C API 構文:

```
/* File: sqlenv.h */
/* API: sqlencls */
/* ... */
SQL_API_RC SQL_API_FN
  sqlencls (
    unsigned short Handle,
    struct sqlca *pSqlca);
/* ... */
```

汎用 API 構文:

```
/* File: sqlenv.h */
/* API: sqlgncls */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgncls (
    unsigned short Handle,
    struct sqlca *pSqlca);
/* ... */
```

API パラメーター:

Handle
入力。関連する OPEN NODE DIRECTORY SCAN API から戻される ID です。

pSqlca
出力。*sqlca* 構造へのポインター。

REXX API 構文:

```
CLOSE NODE DIRECTORY :scanid
```

REXX API パラメーター:

scanid
OPEN NODE DIRECTORY SCAN API によって戻された *scanid* を含むホスト変数。

関連資料:

- 393 ページの『sqlengne - ノード・ディレクトリー次項目の入手』
- 395 ページの『sqlenops - ノード・ディレクトリー・スキャンのオープン』
- 453 ページの『SQLCA』

関連サンプル:

- 『ininfo.c -- Set and get information at the instance level (C)』
- 『ininfo.C -- Set and get information at the instance level (C++)』
- 『nodecat.cbl -- Get node directory information (IBM COBOL)』

sqlengne - ノード・ディレクトリー次項目の入手

sqlenops (ノード・ディレクトリー・スキャンのオープン) が呼び出された後、ノード・ディレクトリーにある次項目を戻します。この API への以降の呼び出しは、追加の項目を戻します。

許可:

なし

必要な接続:

なし

API 組み込みファイル:

sqlenv.h

C API 構文:

```
/* File: sqlenv.h */
/* API: sqlengne */
/* ... */
SQL_API_RC SQL_API_FN
  sqlengne (
    unsigned short Handle,
    struct sqleninfo **ppNodeDirEntry,
    struct sqlca *pSqlca);
/* ... */
```

汎用 API 構文:

```
/* File: sqlenv.h */
/* API: sqlgngne */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgngne (
    unsigned short Handle,
    struct sqleninfo **ppNodeDirEntry,
    struct sqlca *pSqlca);
/* ... */
```

API パラメーター:

Handle

入力。 sqlenops (ノード・ディレクトリー・スキャンのオープン) から戻される ID です。

ppNodeDirEntry

出力。 *sqleninfo* 構造を指すポインターのアドレスを示します。この API の呼び出し側が構造用のメモリーを提供する必要はありません。提供する必要があるのは、ポインターだけです。 API からの戻りで、このポインターは sqlenops (ノード・ディレクトリー・スキャンのオープン) によって割り振られたノード・ディレクトリーのコピーにあるノード・ディレクトリーの次項目を指すようになります。

pSqlca

出力。*sqlca* 構造へのポインター。

REXX API 構文:

```
GET NODE DIRECTORY ENTRY :scanid [USING :value]
```

REXX API パラメーター:

scanid

OPEN NODE DIRECTORY SCAN API によって戻された ID を含む REXX ホスト変数。

value ノード項目情報が戻されるコンパウンド REXX ホスト変数。名前が指定されなかった場合、名前 SQLNINFO が使用されます。以下の項目において、XXX はホスト変数名を表しています (対応するフィールド名は API によって戻される構造から取られています)。

XXX.0	変数内のエレメント数 (通常 16)
XXX.1	NODENAME
XXX.2	LOCALLU
XXX.3	PARTNERLU
XXX.4	MODE
XXX.5	COMMENT
XXX.6	RESERVED
XXX.7	PROTOCOL (プロトコル・タイプ)
XXX.8	ADAPTER (NetBIOS アダプター番号)
XXX.9	RESERVED
XXX.10	SYMDESTNAME (シンボリック宛先名)
XXX.11	SECURITY (セキュリティー・タイプ)
XXX.12	HOSTNAME
XXX.13	SERVICENAME
XXX.14	FILESERVER
XXX.15	OBJECTNAME
XXX.16	INSTANCE (ローカル・インスタンス名)

使用上の注意:

ノード・ディレクトリー項目情報バッファーにあるすべてのフィールドは、右方にブランクが埋め込まれます。

この API が呼び出されるとき、スキャンする項目がもはや存在していないならば、*sqlca* の *sqlcode* 値は 1014 に設定されます。

この API を *pNumEntries* に指定された回数呼び出すことにより、全ディレクトリーをスキャンすることができます。

関連資料:

- 392 ページの『sqlencls - ノード・ディレクトリー・スキャンのクローズ』
- 395 ページの『sqlenops - ノード・ディレクトリー・スキャンのオープン』
- 453 ページの『SQLCA』
- 480 ページの『SQLENINFO』

関連サンプル:

- 『ininfo.c -- Set and get information at the instance level (C)』
- 『ininfo.C -- Set and get information at the instance level (C++)』
- 『nodecat.cbl -- Get node directory information (IBM COBOL)』

sqlenops - ノード・ディレクトリー・スキャンのオープン

ノード・ディレクトリーのコピーをメモリーに保管するとともに、項目の数を戻します。このコピーは、ディレクトリーがオープンする時点のディレクトリーのスナップショットです。後になってディレクトリー自体に変更が加えられることがあっても、このコピーが更新されることはありません。

ノード・ディレクトリーの中でノード項目に関する情報を調べるには、 sqlengne (次のノード・ディレクトリー項目) を使用します。スキャンをクローズするには、sqlencls (ノード・ディレクトリー・スキャンのクローズ) を使用します。このことを行うと、ディレクトリーのコピーがメモリーから除去されます。

許可:

なし

必要な接続:

なし

API 組み込みファイル:

sqlenv.h

C API 構文:

```
/* File: sqlenv.h */
/* API: sqlenops */
/* ... */
SQL_API_RC SQL_API_FN
  sqlenops (
```

```
    unsigned short *pHandle,
    unsigned short *pNumEntries,
    struct sqlca *pSqlca);
/* ... */
```

汎用 API 構文:

```
/* File: sqlenv.h */
/* API: sqlgnops */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgnops (
    unsigned short *pHandle,
    unsigned short *pNumEntries,
    struct sqlca *pSqlca);
/* ... */
```

API パラメーター:

pHandle
出力。この API から戻された ID です。この ID は、sqlengne (次のノード・ディレクトリー項目の入手) および sqlencls (ノード・ディレクトリー・スキャンのクローズ) に渡されなければなりません。

pNumEntries
出力。ディレクトリー項目の数が戻される 2 バイト域のアドレスを示します。

pSqlca
出力。*sqlca* 構造へのポインター。

REXX API 構文:

```
OPEN NODE DIRECTORY USING :value
```

REXX API パラメーター:

value ノード・ディレクトリー情報が戻されるコンパウンド REXX 変数。以下の項目において、XXX はホスト変数名を表しています。

XXX.0 変数内のエレメント数 (常に 2)。

XXX.1 *scanid* の数を含む REXX ホスト変数を指定します。

XXX.2 ディレクトリー内に含まれる項目の数。

使用上の注意:

この API が割り振った記憶域は、sqlencls (ノード・ディレクトリー・スキャンのクローズ) により解放されます。

ノード・ディレクトリーに対して、複数のノード・ディレクトリー・スキャンを発行することができます。ただし同じ結果になるとは限りません。次に走査をオープンするまでの間に、ディレクトリーが変更されている場合もあります。

プロセスごとに最大 8 つのノード・ディレクトリー・スキャンをオープンすることができます。

関連資料:

• 392 ページの『sqlencls - ノード・ディレクトリー・スキャンのクローズ』

- 393 ページの『sqlengne - ノード・ディレクトリー次項目の入手』
- 453 ページの『SQLCA』

関連サンプル:

- 『ininfo.c -- Set and get information at the instance level (C)』
- 『ininfo.C -- Set and get information at the instance level (C++)』
- 『nodecat.cbl -- Get node directory information (IBM COBOL)』

sqleqryc - クライアントの照会

アプリケーション・プロセスの現行の接続設定を戻します。

許可:

なし

必要な接続:

なし

API 組み込みファイル:

sqlenv.h

C API 構文:

```
/* File: sqlenv.h */
/* API: sqleqryc */
/* ... */
SQL_API_RC SQL_API_FN
  sqleqryc (
    struct sqle_conn_setting *pConnectionSettings,
    unsigned short NumSettings,
    struct sqlca *pSqlca);
/* ... */
```

汎用 API 構文:

```
/* File: sqlenv.h */
/* API: sqlgqryc */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgqryc (
    struct sqle_conn_setting *pConnectionSettings,
    unsigned short NumSettings,
    struct sqlca *pSqlca);
/* ... */
```

API パラメーター:

pConnectionSettings

入力/出力。接続設定のタイプおよび値を指定する *sqle_conn_setting* 構造を指すポインター。ユーザーは、*NumSettings* 個の接続設定構造の配列を定義し、この配列内の各エレメントの *type* フィールドを設定して、 5 つある接続設定オプションの 1 つを指定します。戻り時に、各エレメントの *value* フィールドには、指定したオプションの現行設定が含まれます。

NumSettings
入力。戻される接続オプション値の数を示す任意の整数 (0 から 7) を指定します。

pSqlca
出力。*sqlca* 構造へのポインター。

REXX API 構文:

```
QUERY CLIENT INTO :output
```

REXX API パラメーター:

output
アプリケーション・プロセスの現行の接続設定に関する情報を含むコンパウンド REXX ホスト変数。以下の項目において、XXX はホスト変数名を表しています。

XXX.1 CONNECTION タイプの現行接続設定。

XXX.2 SQLRULES の現行接続設定。

XXX.3 COMMIT の発行時にどの接続が解放されるのかを示す現行接続設定。

XXX.4 SYNCPOINT オプションの現行接続設定。 2 フェーズ・コミットのセマンティクスを適用するためにトランザクション・マネージャーが使用されるべきかどうか、単一のトランザクション内で複数のデータベースがアクセスされる場合に、更新されるデータベースが 1 つだけであることをデータベース・マネージャーが確認するべきかどうか、あるいはこのようなオプションがいずれも使用されないかを示します。

XXX.5 NETBIOS アダプターに関連して同時に存在する接続の最大数を示す、現行接続設定。

XXX.6 据え置かれた PREPARE の現行接続設定。

使用上の注意:

アプリケーション・プロセスの接続設定は、実行中にいつでも照会できます。

QUERY CLIENT が正常に出されると、 *sqle_conn_setting* 構造のフィールドには、アプリケーション・プロセスの現行の接続設定が含まれます。 SET CLIENT がまだ呼び出されていない場合、設定値には、 SQL ステートメントがすでに処理されている場合にのみ、プリコンパイル・オプションの値が使われます。そうでない場合には、プリコンパイル・オプションのデフォルト値が使われます。

関連資料:

- 406 ページの『sqlesetc - クライアントの設定』
- 399 ページの『sqleqryi - クライアント情報の照会』
- 453 ページの『SQLCA』
- 463 ページの『SQLE-CONN-SETTING』

関連サンプル:

- 『cli_info.c -- Set and get information at the client level (C)』
- 『cli_info.C -- Set and get information at the client level (C++)』
- 『client.cbl -- How to set and query a client (IBM COBOL)』

sqleqryi - クライアント情報の照会

既存のクライアント情報を戻します。この API はデータベース別名の指定を許可するため、アプリケーションは特定の接続と関連したクライアント情報を照会することができます。 sqleseti API が前に確立された値でない場合、NULL を戻します。

特定の接続が要求されると、この API はその接続に対する最新の値を戻します。すべての接続が指定されると、API はすべての接続に関連する値を戻します。この値は、sqleseti の最新の呼び出しで渡された値です (すべての接続を指定する)。

許可:

なし

必要な接続:

なし

API 組み込みファイル:

sqlenv.h

C API 構文:

```
/* File: sqlenv.h */
/* API: sqleqryi */
/* ... */
SQL_API_RC SQL_API_FN
  sqleqryi (
    unsigned short DbAliasLen,
    char *pDbAlias,
    unsigned short NumItems,
    struct sqle_client_info*pClient_Info,
    struct sqlca *pSqlca);
/* ... */
```

汎用 API 構文:

```
/* File: sqlenv.h */
/* API: sqleqryi */
/* ... */
SQL_API_RC SQL_API_FN
  sqleqryi (
    unsigned short DbAliasLen,
    char *pDbAlias,
    unsigned short NumItems,
    struct sqle_client_info*pClient_Info,
    struct sqlca *pSqlca);
/* ... */
```

API パラメーター:

DbAliasLen
入力。データベース別名の長さを示す 2 バイトの符号なし整数 (バイト単

位) です。ゼロより大きい値が指定される場合、 *pDbAlias* は別名を指さなければなりません。この別名の sqleseti への最新の呼び出し (または、長さゼロの別名を指定する sqleseti への呼び出し) と関連する設定を戻します。ゼロが指定されると、長さゼロの別名を指定する sqleseti への最新の呼び出しと関連する設定を戻します。

pDbAlias
入力。データベース別名を含むストリングを指すポインター。

NumItems
入力。修正される項目の数を指定します。最小値は 1 です。

pClient_Info
入力。 *NumItems* の *sqle_client_info* 構造の配列を指すポインター。その構造のそれぞれには、戻される値を示すタイプ・フィールドと、その戻される値を指すポインターが含まれています。ポインターが指す領域は、要求されている値を十分収容できる大きさでなければなりません。

pSqlca
出力。*sqlca* 構造へのポインター。

使用上の注意:

これらの設定は、実行中にいつでも照会できます。 API 呼び出しが正常に終了すると、現行の設定は指定された領域に戻ります。 sqleseti API への呼び出しを介して設定されていないフィールドには、長さゼロ、および NULL 終了ストリング (¥0) を戻します。

関連資料:

- 408 ページの『sqleseti - クライアント情報の設定』
- 453 ページの『SQLCA』
- 460 ページの『SQLE-CLIENT-INFO』

関連サンプル:

- 『cli_info.c -- Set and get information at the client level (C)』
- 『cli_info.C -- Set and get information at the client level (C++)』

sqleregs - 登録

ネットワーク・サーバー上で DB2 サーバーを登録します。 DB2 サーバーのネットワーク・アドレスは、ファイル・サーバー上の指定したレジストリーに保管されます。そのアドレスは、ファイル・サーバーにおいて、 IPX/SPX 通信プロトコルを使用するクライアント・アプリケーションによって検索することができます。

許可:

なし

必要な接続:

なし

API 組み込みファイル:

sqlenv.h

C API 構文:

```
/* File: sqlenv.h */
/* API: sqleregs */
/* ... */
SQL_API_RC SQL_API_FN
  sqleregs (
    unsigned short Registry,
    void *pRegisterInfo,
    struct sqlca *pSqlca);
/* ... */
```

汎用 API 構文:

```
/* File: sqlenv.h */
/* API: sqlgregs */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgregs (
    unsigned short Registry,
    void *pRegisterInfo,
    struct sqlca *pSqlca);
/* ... */
```

API パラメーター:

Registry
入力。 DB2 サーバーを登録するネットワーク・ファイル・サーバーを指定します。このリリースでは、 SQL_NWBINDERY (sqlenv で定義された NetWare ファイル・サーバー・バインダリー) だけがサポートされます。

pRegisterInfo
入力。 *sqle_reg_nwbindery* 構造を指すポインター。この構造では、呼び出し側がネットワーク・ファイル・サーバーにおいて有効なユーザー名とパスワードを指定します。

pSqlca
出力。*sqlca* 構造へのポインター。

REXX API 構文:

この API は、SQLDB2 インターフェースを使って、REXX から呼び出すことができます。

使用上の注意:

この API は、DB2 サーバー・マシン (この API が呼び出されたマシン) の IPX/SPX アドレスを判別した後、データベース・マネージャー構成ファイルに指定された *objectname* の値を使用して NetWare ファイル・サーバー・バインダリーにオブジェクトを作成します。 DB2 サーバーの IPX/SPX アドレスは、そのオブジェクトの特性として保管されます。クライアントが IPX/SPX ファイル・サーバー・アドレッシングを用いて DB2 データベースに接続するためには、ノード・ディレクトリーにある IPX/SPX ノードを (サーバーで指定されたものと同じ FILESERVER および OBJECTNAME を使用して) カタログする必要があります。

指定した NetWare ユーザー名およびパスワードには、管理権限またはそれと同等の権限が必要です。

この API は、 DB2 サーバーからローカルに発行しなければなりません。リモートにはサポートされていません。

DB2 のインストールおよび構成が完了したら、 DB2 サーバーをネットワーク・ファイル・サーバーに一度登録する必要があります (IPX/SPX クライアントが*直接アドレッシング* によってこの DB2 サーバーに接続する場合だけは除きます)。その後、IPX/SPX フィールドを再構成するか、または DB2 サーバーの IPX/SPX インターネットワーク・アドレスが変更される場合には、まずネット作業ファイル・サーバー上で DB2 サーバーの登録を解除し、次に変更を行って、最後に登録し直してください。

関連タスク:

- 「*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*」の『REXX における SQLEXEC、SQLDBS、および SQLDB2 の登録』

関連資料:

- 364 ページの『sqledreg - 登録解除』
- 453 ページの『SQLCA』
- 473 ページの『SQLE-REG-NWBINDERY』
- 「*コマンド・リファレンス*」の『REGISTER コマンド』

sqlesact - 会計情報ストリングの設定

アプリケーションの次の接続要求とともに、 DRDA サーバーに送られる会計情報を提供します。

許可:

なし

必要な接続:

なし

API 組み込みファイル:

sqlenv.h

C API 構文:

```
/* File: sqlenv.h */
/* API: sqlesact */
/* ... */
SQL_API_RC SQL_API_FN
  sqlesact (
    char *pAccountingString,
    struct sqlca *pSqlca);
/* ... */
```

汎用 API 構文:

```
/* File: sqlenv.h */
/* API: sqlgsact */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgsact (
    unsigned short AccountingStringLen,
    char *pAccountingString,
    struct sqlca *pSqlca);
/* ... */
```

API パラメーター:

AccountingStringLen
入力。会計情報ストリングの長さを示す 2 バイトの符号なし整数 (バイト単位) を指定します。

pAccountingString
入力。会計情報を含むストリングを指定します。

pSqlca
出力。*sqlca* 構造へのポインター。

使用上の注意:

会計情報を接続要求とともに送りたい場合には、データベースに接続する前に、アプリケーションからこの API を呼び出す必要があります。 API を再び呼び出して別のデータベースに接続するまでは、会計情報ストリングに変更を加えることができます。接続しない場合には、アプリケーションが終了するまで、現行の値が有効のままになります。会計情報ストリングは、最大 SQL_ACCOUNT_STR_SZ (sqlenv で定義) で指定されたバイト数の長さにまですることができます。それよりも長い場合は、切り捨てられます。 DRDA サーバーへの伝送時に、会計情報ストリングが正しく変換されるようにするため、文字 A から Z、0 から 9、および下線記号 (_) だけを使用するようにしてください。

関連資料:

- 408 ページの『sqleseti - クライアント情報の設定』
- 453 ページの『SQLCA』

関連サンプル:

- 『setact.cbl -- How to set accounting string (IBM COBOL)』

sqlesdeg - ランタイム次数の設定

指定されたアクティブ・アプリケーションの SQL ステートメントに、パーティション内並列処理での最大ランタイムの度合いを設定します。この API は、CREATE INDEX の並列処理には影響を与えません。

有効範囲:

この API は db2nodes.cfg ファイルにリストされているすべてのデータベース・パーティション・サーバーに影響を与えます。

許可:

以下のいずれかです。

- *sysadm*
- *sysctrl*

必要な接続:

インスタンス。リモート・サーバーにおける最大のランタイム並列処理を変更するには、まずそのサーバーにアタッチすることが必要です。アタッチが存在しない場合、SET RUNTIME DEGREE ステートメントは失敗します。

API 組み込みファイル:

sqlenv.h

C API 構文:

```
/* File: sqlenv.h */
/* API: sqlesdeg */
/* ... */
SQL_API_RC SQL_API_FN
  sqlesdeg (
    sqlint32 NumAgentIds,
    sqluint32 *pAgentIds,
    sqlint32 Degree,
    struct sqlca *pSqlca);
/* ... */
```

汎用 API 構文:

```
/* File: sqlenv.h */
/* API: sqlgsdeg */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgsdeg (
    struct sqlca *pSqlca,
    sqlint32 Degree,
    sqluint32 *pAgentIds,
    sqlint32 NumAgentIds);
/* ... */
```

API パラメーター:

pSqlca
出力。*sqlca* 構造へのポインター。

Degree
入力。最大のランタイム並列処理の度合いの新規の値を指定します。値の範囲は 1 から 32767 です。

pAgentIds
入力。符号なしの長整数の配列を指すポインター。各項目は、対応するアプリケーションのエージェント ID を説明します。アクティブ・アプリケーションのエージェント ID をリストするには、db2GetSnapshot API を使用してください。

NumAgentIds
入力。新規の並列処理の度合いの値が適用されるアクティブ・アプリケーションの合計数を示す整数を指定します。この数はエージェント ID の配列のエレメント数と同じにする必要があります。

このパラメーターが SQL_ALL_USERS (sqlenv で定義されている) に設定された場合、新規の並列処理の度合いはすべてのアクティブ・アプリケーションに適用されます。このパラメーターがゼロに設定された場合、エラーが戻されます。

REXX API 構文:

この API は、SQLDB2 インターフェースを使って、REXX から呼び出すことができます。

使用上の注意:

アクティブ・アプリケーションのエージェント ID と並列処理の度合いを収集するには、データベース・システム・モニター機能を使用します。

エージェント ID の配列に関して最小限の妥当性検査が実行されます。ユーザーは、指定したエレメントの合計数を含む配列をポインターが指していることを確認する必要があります。 *NumAgentIds* が SQL_ALL_USERS に設定されている場合、その配列は無視されます。

指定されたエージェント ID の 1 つかそれ以上が見つからない場合には、認識されないエージェント ID は無視され、機能が続行されます。エラーは戻されません。エージェント ID は、たとえば、エージェント ID が収集されてから API が呼び出されるまでの間にユーザーがサインオフした場合などには、見つからないことがあります。

エージェント ID は再生され、さらに、データベース・システム・モニターによる収集のしばらく後で、アプリケーションの並列処理の度合いを変更するために使用されます。したがって、ユーザーがサインオフすると、別のユーザーがサインオンし、この再生処理を介して同じエージェント ID を獲得する可能性があります。結果として、新規の並列処理の度合いが誤ったユーザーについて変更される可能性があります。

関連タスク:

- 「*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*」の『REXX における SQLEXEC、SQLDBS、および SQLDB2 の登録』

関連資料:

- 89 ページの『db2GetSnapshot - スナップショットの入手』
- 453 ページの『SQLCA』
- 「*コマンド・リファレンス*」の『SET RUNTIME DEGREE コマンド』

関連サンプル:

- 『ininfo.c -- Set and get information at the instance level (C)』
- 『ininfo.C -- Set and get information at the instance level (C++)』

sqlesetc - クライアントの設定

アプリケーション用の接続設定を指定します。

許可:

なし

必要な接続:

なし

API 組み込みファイル:

sqlenv.h

C API 構文:

```
/* File: sqlenv.h */
/* API: sqlesetc */
/* ... */
SQL_API_RC SQL_API_FN
  sqlesetc (
    struct sqle_conn_setting *pConnectionSettings,
    unsigned short NumSettings,
    struct sqlca *pSqlca);
/* ... */
```

汎用 API 構文:

```
/* File: sqlenv.h */
/* API: sqlgsetc */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgsetc (
    struct sqle_conn_setting *pConnectionSettings,
    unsigned short NumSettings,
    struct sqlca *pSqlca);
/* ... */
```

API パラメーター:

pConnectionSettings
入力。 *sqle_conn_setting* 構造を指すポインター。接続設定のタイプおよび値を指定します。 *NumSettings* 個の *sqle_conn_setting* 構造の配列を割り振ってください。設定する接続オプションを示すために、この配列の各エレメントの *type* フィールドを設定してください。 *value* フィールドを、オプションに必要な値に設定してください。

NumSettings
入力。設定する接続オプション値の数を示す任意の整数 (0 から 7) を指定します。

pSqlca
出力。*sqlca* 構造へのポインター。

REXX API 構文:

```
SET CLIENT USING :values
```

REXX API パラメーター:

values

アプリケーション・プロセスの接続設定を含むコンパウンド REXX ホスト変数。以下の項目において、XXX はホスト変数名を表しています。

XXX.0 確立される接続設定の数。

XXX.1 CONNECTION タイプの設定方法を指定します。有効な値は以下のとおりです。

1 Type 1 CONNECT

2 Type 2 CONNECT

XXX.2 SQLRULES の設定方法を指定します。有効な値は以下のとおりです。

DB2 DB2 規則に従って type 2 CONNECT を処理します。

STD 標準規則に従って type 2 CONNECT を処理します。

XXX.3 コミット時に、データベースの切断の有効範囲を設定する方法を指定します。有効な値は以下のとおりです。

EXPLICIT SQL RELEASE ステートメントによるマークの付いたデータベース接続だけを切断します。

CONDITIONAL
オープン状態の WITH HOLD カーソルを持たないデータベース接続だけを切断します

AUTOMATIC すべてのデータベース接続を切断します。

XXX.4

コミットまたはロールバック時に、複数のデータベース接続の間で、どのような調整がなされるかを指定します。有効な値は以下のとおりです。

TWOPHASE トランザクション・マネージャー (TM) を使用して、2 フェーズ・コミットを調整します。

XXX.5

NETBIOS アダプターを使用している場合に、同時に存在できる接続の最大数を指定します。

XXX.6

PREPARE ステートメントを実行すべきときを指定します。有効な値は以下のとおりです。

NO PREPARE ステートメントは、それが発行された時点で実行されます。

YES PREPARE ステートメントは、対応する OPEN、DESCRIBE、または EXECUTE ステートメントが発行されるまで実行されません。ただし、PREPARE INTO ステートメントは据え置かれません。

ALL PREPARE INTO ステートメントも据え置かれる点を除き、YES と同じです。

使用上の注意:

この API が成功すると、それに続く作業単位内の接続では、指定された接続設定が使用されます。この API が失敗した場合、接続設定は未変更のままです。

アプリケーションの接続設定は、既存の接続がない場合 (たとえば、接続が確立される前、あるいは RELEASE ALL や COMMIT の後など) にのみ変更できます。

いったん SET CLIENT API が正常に実行されると、接続設定は固定され、再び SET CLIENT API を実行しない限り変更できません。対応するアプリケーション・モジュールのプリコンパイル・オプションはすべてオーバーライドされます。

関連資料:

- 397 ページの『sqleqryc - クライアントの照会』
- 408 ページの『sqleseti - クライアント情報の設定』
- 453 ページの『SQLCA』
- 463 ページの『SQLE-CONN-SETTING』

関連サンプル:

- 『cli_info.c -- Set and get information at the client level (C)』
- 『dbcfg.sqc -- Configure database and database manager configuration parameters (C)』
- 『dbmcon.sqc -- How to use multiple databases (C)』
- 『cli_info.C -- Set and get information at the client level (C++)』
- 『dbcfg.sqC -- Configure database and database manager configuration parameters (C++)』
- 『dbmcon.sqC -- How to use multiple databases (C++)』
- 『client.cbl -- How to set and query a client (IBM COBOL)』

sqleseti - クライアント情報の設定

接続がすでに存在する場合、アプリケーションが特定の接続と関連したクライアント情報を設定することを許可します。

TP モニターまたは 3 層のクライアント/サーバー環境では、クライアントの代わりに作動しているアプリケーション・サーバーだけでなく、クライアントについての情報も獲得する必要があります。この API を使うことにより、アプリケーション・サーバーはクライアントのユーザー ID、ワークステーション情報、プログラム情報、および他の会計情報を DB2 サーバーに渡すことができます。そうでない場合、アプリケーション・サーバーの情報だけが渡され、たいてい、その情報は同じアプリケーション・サーバーを介して行う多くのクライアント呼び出しと同じです。

アプリケーションは、クライアント情報が既存のすべての接続と、今後行われる接続に合わせて設定される場合に備え、別名を指定しないことを選択することができます。この API は作業単位の外部で変更される情報を、SQL の実行前か、コミットまたはロールバックの後のどちらかに許可するだけです。呼び出しが正常に終了した場合、接続の値は次の機会に送られ、その接続で送信される次の SQL 要求で

グループ化されます。正常な呼び出しは、値が受け入れられていること、および後続の接続にそれらの値が伝搬することを意味します。

この API は、データベースへの接続より前に値を確立するために使用するか、接続が確立されてからは値を設定または修正するために使用できます。

許可:

なし

必要な接続:

なし

API 組み込みファイル:

sqlenv.h

C API 構文:

```
/* File: sqlenv.h */
/* API: sqleseti */
/* ... */
SQL_API_RC SQL_API_FN
  sqleseti (
    unsigned short DbAliasLen,
    char *pDbAlias,
    unsigned short NumItems,
    struct sqle_client_info*pClient_Info,
    struct sqlca *pSqlca);
/* ... */
```

汎用 API 構文:

```
/* File: sqlenv.h */
/* API: sqleseti */
/* ... */
SQL_API_RC SQL_API_FN
  sqleseti (
    unsigned short DbAliasLen,
    char *pDbAlias,
    unsigned short NumItems,
    struct sqle_client_info*pClient_Info,
    struct sqlca *pSqlca);
/* ... */
```

API パラメーター:

DbAliasLen
入力。データベース別名の長さを示す 2 バイトの符号なし整数 (バイト単位) です。ゼロより大きい値が指定される場合、 *pDbAlias* は別名を指さなければならず、設定は指定された接続にのみ影響します。ゼロが指定されると、設定はすべての既存および将来の接続に影響します。

pDbAlias
入力。データベース別名を含むストリングを指すポインター。

NumItems
入力。修正される項目の数を指定します。最小値は 1 です。

pClient_Info

入力。 *NumItems* *sqle_client_info* 構造の配列を指すポインターで、それぞれは設定する値、その値の長さ、および新しい値へのポインターを示すタイプ・フィールドを含みます。

pSqlca

出力。*sqlca* 構造へのポインター。

使用上の注意:

別名が提供された場合、別名への接続がすでに存在していなければならず、その別名へのすべての接続は変更を継承します。情報は、その別名への接続が中断されるまで保存されます。別名が提供されなかった場合、すべての既存の接続の設定は変更され、将来の接続が変更を継承します。情報は、プログラムが終了するまで保存されます。

フィールド名は、提供できる情報のタイプのガイドラインを表します。たとえば、TP モニター・アプリケーションは、SQL_CLIENT_INFO_APPLNAM フィールドに、アプリケーション名と共に TP モニター・トランザクション ID を提供することができます。これにより、 DB2 トランザクション ID を TP モニター・トランザクション ID と関連付けることができるので、 DB2 サーバー上でのモニターと会計の機能が向上します。

現在、この API は DB2 (OS/390 版) バージョン 5 以降および DB2 UDB バージョン 7 以降に情報を渡します。すべての情報 (会計情報ストリングを除く) は、DISPLAY THREAD コマンドで表示され、すべて会計レコードに記録されます。

SQL 特殊レジスターもこの API によって提供されるデータ値にアクセスできます。このレジスターの値はデータベース・コード・ページに保管されます。この API によって提供されるデータ値は、特殊レジスターに保管される前にデータベース・コード・ページに変換されます。データベース・コード・ページへの変換後、サポートされる最大サイズを超えるデータ値は、サーバーに保管される前に切り捨てられます。切り捨てられた値は特殊レジスターによって戻されます。元のデータ値はサーバーに保管され、データベース・コード・ページに変換されません。変換されていない値は sqleqryi API 呼び出しによって戻すことができます。

関連資料:

- 89 ページの『db2GetSnapshot - スナップショットの入手』
- 406 ページの『sqlesetc - クライアントの設定』
- 402 ページの『sqlesact - 会計情報ストリングの設定』
- 399 ページの『sqleqryi - クライアント情報の照会』
- 453 ページの『SQLCA』
- 460 ページの『SQLE-CLIENT-INFO』

関連サンプル:

- 『cli_info.c -- Set and get information at the client level (C)』
- 『cli_info.C -- Set and get information at the client level (C++)』

sqleuncd - データベースのアンカタログ

システム・データベース・ディレクトリーから項目を削除します。

許可:

以下のいずれかです。

- *sysadm*
- *sysctrl*

必要な接続:

なし

API 組み込みファイル:

sqlenv.h

C API 構文:

```
/* File: sqlenv.h */
/* API: sqleuncd */
/* ... */
SQL_API_RC SQL_API_FN
  sqleuncd (
    _SQLOLDCHAR *pDbAlias,
    struct sqlca *pSqlca);
/* ... */
```

汎用 API 構文:

```
/* File: sqlenv.h */
/* API: sqlguncd */
/* ... */
SQL_API_RC SQL_API_FN
  sqlguncd (
    unsigned short DbAliasLen,
    struct sqlca *pSqlca,
    _SQLOLDCHAR *pDbAlias);
/* ... */
```

API パラメーター:

DbAliasLen
入力。データベース別名の長さを示す 2 バイトの符号なし整数 (バイト単位) です。

pSqlca
出力。*sqlca* 構造へのポインター。

pDbAlias
入力。アンカタログするデータベースの別名を含むストリングを指定します。

REXX API 構文:

```
UNCATALOG DATABASE dbname
```

REXX API パラメーター:

dbname
アンカタログするデータベースの別名を指定します。

使用上の注意:

システム・データベース・ディレクトリー内の項目だけがアンカタログできます。ローカル・データベース・ディレクトリー内の項目は、sqledrpd API を使用することにより削除できます。

データベースを再カタログするには、sqlecadb API を使用してください。

ノード上でカタログされているデータベースをリストするには、db2DbDirOpenScan、db2DbDirGetNextEntry、および db2DbDirCloseScan API を使用してください。

最初にデータベースをアンカタログし、次に別のタイプを指定してデータベースを再カタログすることにより、下位レベル・サーバーと通信する際に使用される、データベースの認証タイプを変更できます。

ディレクトリーのキャッシュが *dir_cache* 構成パラメーターを使用して使用可能にされている場合、データベース、ノード、および DCS ディレクトリー・ファイルはメモリーにキャッシュされます。アプリケーションのディレクトリー・キャッシュは、最初のディレクトリー検索時に作成されます。キャッシュはアプリケーションがディレクトリー・ファイルのどれかを修正したときにのみ最新にされるため、他のアプリケーションが行ったディレクトリーの変更は、アプリケーションを再始動するまで有効にならないことがあります。 DB2 の共用キャッシュを最新にするには (サーバーのみ)、データベース・マネージャーを停止させてから (**db2stop**)、再始動 (**db2start**) させてください。別のアプリケーション用のディレクトリー・キャッシュを最新にするには、そのアプリケーションを停止させてから再始動させてください。

関連資料:

- 340 ページの『sqlecadb - データベースのカタログ』
- 53 ページの『db2DbDirCloseScan - データベース・ディレクトリー・スキャンのクローズ』
- 366 ページの『sqledrpd - データベースのドロップ』
- 54 ページの『db2DbDirGetNextEntry - データベース・ディレクトリーの次項目の入手』
- 58 ページの『db2DbDirOpenScan - データベース・ディレクトリー・スキャンのオープン』
- 453 ページの『SQLCA』
- 36 ページの『db2CfgGet - 構成パラメーターの入手』

関連サンプル:

- 『dbcat.cbl -- Catalog to and uncatalog from a database (IBM COBOL)』
- 『ininfo.c -- Set and get information at the instance level (C)』
- 『ininfo.C -- Set and get information at the instance level (C++)』

sqleuncn - ノードのアンカタログ

ノード・ディレクトリーから項目を削除します。

許可:

以下のいずれかです。

- *sysadm*
- *sysctrl*

必要な接続:

なし

API 組み込みファイル:

sqlenv.h

C API 構文:

```
/* File: sqlenv.h */
/* API: sqleuncn */
/* ... */
SQL_API_RC SQL_API_FN
  sqleuncn (
    _SQLOLDCHAR *pNodeName,
    struct sqlca *pSqlca);
/* ... */
```

汎用 API 構文:

```
/* File: sqlenv.h */
/* API: sqlguncn */
/* ... */
SQL_API_RC SQL_API_FN
  sqlguncn (
    unsigned short NodeNameLen,
    struct sqlca *pSqlca,
    _SQLOLDCHAR *pNodeName);
/* ... */
```

API パラメーター:

NodeNameLen
入力。ノード名の長さを示す 2 バイトの符号なし整数 (バイト単位) です。

pSqlca
出力。*sqlca* 構造へのポインター。

pNodeName
入力。アンカタログするノードの名前を含むストリングを指定します。

REXX API 構文:

```
UNCATALOG NODE nodename
```

REXX API パラメーター:

nodename
アンカタログするノードの名前。

使用上の注意:

ノードを再カタログする場合には、sqlectnd API を使用してください。

カタログされているノードをリストするには、 db2DbDirOpenScan、db2DbDirGetNextEntry、および db2DbDirCloseScan API を使用してください。

ディレクトリーのキャッシュが *dir_cache* 構成パラメーターを使用して使用可能にされている場合、データベース、ノード、および DCS ディレクトリー・ファイルはメモリーにキャッシュされます。アプリケーションのディレクトリー・キャッシュは、最初のディレクトリー検索時に作成されます。キャッシュはアプリケーションがディレクトリー・ファイルのどれかを修正したときにのみ最新にされるため、他のアプリケーションが行ったディレクトリーの変更は、アプリケーションを再始動するまで有効にならないことがあります。 DB2 の共用キャッシュを最新にするには (サーバーのみ)、データベース・マネージャーを停止させてから (**db2stop**)、再始動 (**db2start**) させてください。別のアプリケーション用のディレクトリー・キャッシュを最新にするには、そのアプリケーションを停止させてから再始動させてください。

関連資料:

- 356 ページの『sqlectnd - ノードのカタログ』
- 392 ページの『sqlencls - ノード・ディレクトリー・スキャンのクローズ』
- 393 ページの『sqlengne - ノード・ディレクトリー次項目の入手』
- 395 ページの『sqlenops - ノード・ディレクトリー・スキャンのオープン』
- 453 ページの『SQLCA』
- 36 ページの『db2CfgGet - 構成パラメーターの入手』

関連サンプル:

- 『ininfo.c -- Set and get information at the instance level (C)』
- 『ininfo.C -- Set and get information at the instance level (C++)』
- 『nodecat.cbl -- Get node directory information (IBM COBOL)』

sqlgaddr - アドレスの入手

ある変数のアドレスを別の変数の中に入れます。 FORTRAN や COBOL など、ポインター操作を提供しないホスト言語で使用されます。

許可:

なし

必要な接続:

なし

API 組み込みファイル:

sqlutil.h

汎用 API 構文:

```
/* File: sqlutil.h */
/* API: sqlgaddr */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgaddr (
    char *pVariable,
    char **ppOutputAddress);
/* ... */
```

API パラメーター:

pVariable
入力。アドレスが戻される変数を示します。

ppOutputAddress
出力。変数アドレスが戻される 4 バイトの領域を示します。

使用上の注意:

この API を使用するのは COBOL および FORTRAN 言語だけです。

関連資料:

• 415 ページの『sqlgdref - アドレスの間接参照』

sqlgdref - アドレスの間接参照

ポインターによって定義されるバッファーからのデータを、アプリケーションが直接アクセスできる変数にコピーします。 FORTRAN や COBOL など、ポインター操作を提供しないホスト言語で使用されます。この API を使用して、必要なデータを指すポインターを戻す API からの結果を取得することができます。

許可:

なし

必要な接続:

なし

API 組み込みファイル:

sqlutil.h

汎用 API 構文:

```
/* File: sqlutil.h */
/* API: sqlgdref */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgdref (
    unsigned int NumBytes,
    char *pTargetBuffer,
    char **ppSourceBuffer);
/* ... */
```

API パラメーター:

NumBytes
入力。転送されるバイト数を示す整数です。

pTargetBuffer
出力。データの移動先の領域を示します。

ppSourceBuffer
入力。対象データを含む領域を指すポインター。

使用上の注意:

この API を使用するのは COBOL および FORTRAN 言語だけです。

関連資料:

• 414 ページの『sqlgaddr - アドレスの入手』

sqlgmcpy - メモリーのコピー

あるメモリーの領域から別のメモリーの領域へデータをコピーします。 FORTRAN や COBOL など、メモリー・ブロックのコピー機能を提供しないホスト言語で使用されます。

許可:

なし

必要な接続:

なし

API 組み込みファイル:

sqlutil.h

汎用 API 構文:

```
/* File: sqlutil.h */
/* API: sqlgmcpy */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgmcpy (
    void *pTargetBuffer,
    const void *pSource,
    sqluint32 NumBytes);
/* ... */
```

API パラメーター:

pTargetBuffer
出力。データの移動先の領域を示します。

pSource
入力。データの移動元の領域を示します。

NumBytes
入力。転送されるバイト数を示す 4 バイトの符号なし整数です。

使用上の注意:

この API を使用するのは COBOL および FORTRAN 言語だけです。

関連資料:

• 414 ページの『sqlgaddr - アドレスの入手』

sqlogstt - SQLSTATE メッセージの入手

SQLSTATE に関連したメッセージ・テキストを検索します。

許可:

なし

必要な接続:

なし

API 組み込みファイル:

sql.h

C API 構文:

```
/* File: sql.h */
/* API: sqlogstt */
/* ... */
SQL_API_RC SQL_API_FN
  sqlogstt (
    char *pBuffer,
    short BufferSize,
    short LineWidth,
    char *pSqlstate);
/* ... */
```

汎用 API 構文:

```
/* File: sql.h */
/* API: sqlggstt */
/* ... */
SQL_API_RC SQL_API_FN
  sqlggstt (
    short BufferSize,
    short LineWidth,
    char *pSqlstate,
    char *pBuffer);
/* ... */
```

API パラメーター:

BufferSize
入力。検索したメッセージ・テキストを保留するストリング・バッファーのサイズ (バイト単位) です。

LineWidth
入力。メッセージ・テキストの各行ごとの最大行幅を示します。ワード境界で改行されます。値ゼロは、メッセージ・テキストが改行されることなく戻されることを示します。

pSqlstate
入力。メッセージ・テキストが検索される SQLSTATE を含むストリングを指定します。このフィールドには英数字が入ります。 5 桁 (特定の SQLSTATE) または 2 桁 (SQLSTATE クラス、 SQLSTATE の最初の 2 桁) を入れなければなりません。 5 桁が渡される場合、このフィールドは NULL 文字で終了する必要はありません。しかし、2 桁が渡される場合は、必ず NULL 文字で終了しなければなりません。

pBuffer
出力。メッセージ・テキストが配置されるストリング・バッファーを指すポインター。メッセージをバッファーに合わせて切り捨てる必要がある場合、切り捨ては NULL ストリング終止符を見込んで行われます。

REXX API 構文:

```
GET MESSAGE FOR SQLSTATE sqlstate INTO :msg [LINEWIDTH width]
```

REXX API パラメーター:

sqlstate
メッセージ・テキストが検索される SQLSTATE。

msg メッセージが入れられる REXX 変数。

width
メッセージ・テキストの各行の最大行幅。ワード境界で改行されます。値が指定されていないか、またはこのパラメーターが 0 に設定されていると、メッセージ・テキストは改行なしで戻されます。

使用上の注意:

呼び出しごとに 1 つのメッセージが戻されます。

LF/NULL 順序列が、各メッセージの終端に置かれます。

正の行幅が指定されている場合、LF/NULL 順序列がワード間に挿入されるので、行がその行幅を超えることはありません。

あるワードが行幅よりも長い場合、その行に入るだけの文字が入ります。 LF/NULL が挿入され、入りきらなかった残りの文字は次の行に移されます。

戻りコード:

コード メッセージ

+i フォーマット設定メッセージのバイト数を示す正の整数です。呼び出し側が入力したバッファー・サイズよりもこの数の方が大きい場合、メッセージは切り捨てられます。

-1 メッセージ書式化サービスを機能させるには、利用可能なメモリーが不十分です。要求されたメッセージは、戻されません。

-2 SQLSTATE の形式が間違っています。この形式は英数字でなければならず、長さは 2 桁あるいは 5 桁のどちらかです。

-3 メッセージ・ファイルがアクセス不能または正しくありません。

-4 行幅が 0 未満です。

-5 無効 *sqlca*、不良バッファー・アドレス、または不良バッファー長を示します。

戻りコードが -1 または -3 の場合、メッセージ・バッファーには、問題に関するより詳細な情報が含まれています。

関連資料:

- 297 ページの『sqlaintp - エラー・メッセージの入手』

関連サンプル:

- 『checkerr.cbl -- Checks for and prints to the screen SQL warnings and errors (IBM COBOL)』
- 『utilapi.c -- Error-checking utility for non-embedded SQL samples in C (C)』
- 『utilapi.C -- Checks for and prints to the screen SQL warnings and errors (C++)』

sqluadau - 許可の入手

データベース・マネージャー構成ファイルおよび許可システム・カタログ・ビュー (SYSCAT.DBAUTH) の中の値から、現行ユーザーの権限を報告します。

許可:

なし

必要な接続:

データベース

API 組み込みファイル:

sqlutil.h

C API 構文:

```
/* File: sqlutil.h */
/* API: sqluadau */
/* ... */
SQL_API_RC SQL_API_FN
  sqluadau (
    struct sql_authorizations *pAuthorizations,
    struct sqlca *pSqlca);
/* ... */
```

汎用 API 構文:

```
/* File: sqlutil.h */
/* API: sqlgadau */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgadau (
    struct sql_authorizations *pAuthorizations,
    struct sqlca *pSqlca);
/* ... */
```

API パラメーター:

pAuthorizations

入力/出力。 *sql_authorizations* 構造を指すポインター。短整数のこの配列は、どの許可を現行ユーザーが保持しているかを示します。構造 *sql_authorizations_len* の最初のエレメントは、この API を呼び出す前に、渡されるバッファーのサイズに初期設定しなければなりません。

pSqlca

出力。*sqlca* 構造へのポインター。

REXX API 構文:

```
GET AUTHORIZATIONS :value
```

REXX API パラメーター:

value 許可レベルが戻されるコンパウンド REXX ホスト変数。以下の項目において、XXX はホスト変数名を表しています。「いいえ」の場合、値は 0 です。「はい」の場合は、1 です。

XXX.0	変数内のエレメントの数 (常に 18)
XXX.1	直接 SYSADM 権限
XXX.2	直接 DBADM 権限
XXX.3	直接 CREATETAB 権限
XXX.4	直接 BINDADD 権限
XXX.5	直接 CONNECT 権限
XXX.6	間接 SYSADM 権限
XXX.7	間接 DBADM 権限
XXX.8	間接 CREATETAB 権限
XXX.9	間接 BINDADD 権限
XXX.10	間接 CONNECT 権限
XXX.11	直接 SYSCTRL 権限
XXX.12	間接 SYSCTRL 権限
XXX.13	直接 SYSMAINT 権限
XXX.14	間接 SYSMAINT 権限
XXX.15	直接 CREATE_NOT_FENC 権限
XXX.16	間接 CREATE_NOT_FENC 権限
XXX.17	直接 IMPLICIT_SCHEMA 権限
XXX.18	間接 IMPLICIT_SCHEMA 権限
XXX.19	直接 LOAD 権限
XXX.20	間接 LOAD 権限

使用上の注意:

権限をユーザー ID に付与する明示的なコマンドによって獲得される権限のことを直接権限といいます。それに対し、間接権限とは、ユーザーが所属するグループによって獲得された権限を基盤としている権限のことをいいます。

注: PUBLIC は、全ユーザーが所属する特殊なグループです。

エラーがない場合、 *sql_authorizations* 構造の各エレメントには `0` または `1` が入ります。 `1` の値は、ユーザーが許可を保持していることを示します。 `0` の値は、ユーザーが許可を保持していないことを示します。

関連資料:

- 443 ページの『SQL-AUTHORIZATIONS』
- 453 ページの『SQLCA』

関連サンプル:

- 『dbauth.sqb -- How to grant and display authorities on a database (IBM COBOL)』
- 『dbauth.sqc -- How to grant, display, and revoke authorities at database level (C)』
- 『inauth.sqc -- How to display authorities at instance level (C)』
- 『dbauth.sqC -- How to grant, display, and revoke authorities at database level (C++)』
- 『inauth.sqC -- How to display authorities at instance level (C++)』

sqludrdt - データベース・パーティション・グループの再分散

データベース・パーティション・グループ内のデータベース・パーティション間でデータを再分散します。現行のデータ分散 (均等であるか、スキューであるかに関係なく) を指定できます。再分散アルゴリズムでは、現行のデータ分散に基づいて、移動させるパーティションを選択します。

この API は、カタログ・パーティションからのみ呼び出すことができます。どのデータベース・パーティション・サーバーが各データベースごとのカタログ・パーティションかを判別するには、 LIST DATABASE DIRECTORY コマンドを使用してください。

有効範囲:

この API はデータベース・パーティション・グループ内のすべてのデータベース・パーティションに影響を与えます。

許可:

以下のいずれかです。

- *sysadm*
- *sysctrl*
- *dbadm*

API 組み込みファイル:

sqlutil.h

C API 構文:

```
/* File: sqlutil.h */
/* API: sqludrdt */
/* ... */
SQL_API_RC SQL_API_FN
  sqludrdt (
    char *pNodeGroupName,
    char *pTargetPMapFileName,
    char *pDataDistFileName,
    SQL_PDB_NODE_TYPE *pAddList,
    unsigned short AddCount,
    SQL_PDB_NODE_TYPE *pDropList,
    unsigned short DropCount,
    unsigned char DataRedistOption,
    struct sqlca *pSqlca);
/* ... */
```

汎用 API 構文:

```
/* File: sqlutil.h */
/* API: sqlgdrdt */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgdrdt (
    unsigned short NodeGroupNameLen,
    unsigned short TargetPMapFileNameLen,
    unsigned short DataDistFileNameLen,
    char *pNodeGroupName,
    char *pTargetPMapFileName,
    char *pDataDistFileName,
    SQL_PDB_NODE_TYPE *pAddList,
    unsigned short AddCount,
    SQL_PDB_NODE_TYPE *pDropList,
    unsigned short DropCount,
    unsigned char DataRedistOption,
    struct sqlca *pSqlca);
/* ... */
```

API パラメーター:

NodeGroupNameLen
データベース・パーティション・グループの名前の長さ。

TargetPMapFileNameLen
ターゲット・パーティション・マップ・ファイルの名前の長さ。

DataDistFileNameLen
データ分散ファイルの名前の長さ。

pNodeGroupName
再分散が行われるデータベース・パーティション・グループ。

pTargetPMapFileName
ターゲット・パーティション・マップが入っているファイルの名前。ファイル名の一部としてディレクトリー・パスを指定しない場合には、現行ディレクトリーが使用されます。このパラメーターは、 *DataRedistOption* 値が T である場合に使用されます。ファイルは文字形式であり、 4 096 項目 (複数パーティションのデータベース・パーティション・グループの場合) または 1 項目 (単一パーティションのデータベース・パーティション・グループの

場合) を含んでいる必要があります。ファイル内の項目は、ノード番号を示します。項目は自由形式にすることができます。

pDataDistFileName
入力分散情報が入っているファイルの名前。ファイル名の一部としてディレクトリー・パスを指定しない場合には、現行ディレクトリーが使用されます。このパラメーターは、 *DataRedistOption* 値が U である場合に使用されます。ファイルは文字形式でなければならず、 4 096 個の正の整数項目を含まなければなりません。ファイル内の各項目は、対応するパーティションの重みを示します。 4 096 個の値の合計は、 4 294 967 295 以下でなければなりません。

pAddList
データの再分散中にデータベース・パーティション・グループに追加するデータベース・パーティションのリスト。リスト内の項目は、SQL_PDB_NODE_TYPE の形式でなければなりません。

AddCount
データベース・パーティション・グループに追加するデータベース・パーティションの数。

pDropList
データの再分散中にデータベース・パーティション・グループからドロップするデータベース・パーティションのリスト。リスト内の項目は、SQL_PDB_NODE_TYPE の形式でなければなりません。

DropCount
データベース・パーティション・グループからドロップするデータベース・パーティションの数。

DataRedistOption
実行されるデータ再分散のタイプを示す 1 つの文字。可能な値は以下のとおりです。

U 均等に分散するためにデータベース・パーティション・グループを再分散することを指定します。 *pDataDistFileName* が NULL である場合には、現行のデータ分散が均等である (つまり、各ハッシュ・パーティションが同じ量のデータを表している) ものと見なされます。 *pDataDistFileName* が NULL でない場合には、このファイル内の値が現行のデータ分散を表しているものと見なされます。 *DataRedistOption* が U である場合には、 *pTargetPMapFileName* が NULL でなければなりません。

追加リストに指定されたデータベース・パーティションがデータベース・パーティション・グループに追加され、ドロップ・リストに指定されたデータベース・パーティションがデータベース・パーティション・グループからドロップされます。

T *pTargetPMapFileName* を用いてデータベース・パーティション・グループを再分散することを指定します。このオプションを使用する場合には、 *pDataDistFileName*、*pAddList*、および *pDropList* が NULL でなければならず、 *AddCount* と *DropCount* の両方がゼロでなければなりません。

C 失敗した再分散操作を続行することを指定します。このオプションを使用する場合には、*pTargetPMapFileName*、*pDataDistFileName*、*pAddList*、および *pDropList* が NULL でなければならず、*AddCount* と *DropCount* の両方がゼロでなければなりません。

R 失敗した再分散操作をロールバックすることを指定します。このオプションを使用する場合には、*pTargetPMapFileName*、*pDataDistFileName*、 *pAddList*、および *pDropList* が NULL でなければならず、 *AddCount* と *DropCount* の両方がゼロでなければなりません。

pSqlca

出力。*sqlca* 構造へのポインター。

REXX API 構文:

この API は、SQLDB2 インターフェースを使って、REXX から呼び出すことができます。

使用上の注意:

再配分操作が実行されると、メッセージ・ファイルが次のディレクトリーに書き込まれます。

- サブディレクトリーとファイル名に、 *database-name.nodegroup-name.timestamp* という形式を使った、 UNIX ベースのシステムにある `$HOME/sqllib/redist` ディレクトリー。
- サブディレクトリーとファイル名に、*database-name¥first-eight-characters-of-the-nodegroup-name¥date¥time* という形式を使った、 Windows オペレーティング・システムにある `$HOME¥sqllib¥redist¥` ディレクトリー。

タイム・スタンプ値は、API が呼び出された時刻です。

このユーティリティーは、処理中に断続的な COMMIT を実行します。

ALTER DATABASE PARTITION GROUP ステートメントを使用して、データベース・パーティションをデータベース・パーティション・グループに追加します。このステートメントでは、データベース・パーティション・グループに関連付けられた表スペース用のコンテナーを定義することができます。

再分散を受けた表と従属関係があるすべてのパッケージは無効になります。データベース・パーティション・グループの再分散操作が完了した後で、そのようなパッケージを明示的に再バインドすることをお勧めします。それにより、無効なパッケージに対する最初の SQL 要求の実行での初期遅延がなくなります。再分散メッセージ・ファイルには、再分散を受けたすべての表のリストが入ります。

さらに、データベース・パーティション・グループの再分散操作が完了した後で、db2Runstats API を発行して統計を更新することをお勧めします。

DATA CAPTURE CHANGES で定義された複製サマリー表 (複数の場合もある) を含むデータベース・パーティション・グループは再分散できません。

データベース・パーティション・グループに、宣言された既存の一時表を含むユーザー TEMPORARY 表スペースがある場合、再分散を行うことはできません。

関連タスク:

- 「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」の『REXX における SQLEXEC、SQLDBS、および SQLDB2 の登録』

関連資料:

- 「*SQL リファレンス 第 2 巻*」の『ALTER DATABASE PARTITION GROUP ステートメント』
- 302 ページの『sqlarbnd - 再バインド』
- 453 ページの『SQLCA』
- 「コマンド・リファレンス」の『LIST DATABASE DIRECTORY コマンド』
- 「コマンド・リファレンス」の『REDISTRIBUTE DATABASE PARTITION GROUP コマンド』
- 267 ページの『db2Runstats - 統計の実行』

sqlugrpn - 行パーティション番号の入手

パーティション・キー値に基づいてパーティション番号およびデータベース・パーティション・サーバー番号を戻します。アプリケーションは、この情報を使用して、表の特定の行が保管されているデータベース・パーティション・サーバーを判別することができます。

パーティション・データ構造 (sqlupi) はこの API への入力となります。この構造は sqlugtpi API によって戻すことができます。別の入力としては、対応するパーティション・キー値の文字表示があります。出力は、パーティション・ストラテジーによって生成されたパーティション番号と、パーティション・マップからの対応するデータベース・パーティション・サーバー番号です。パーティション・マップ情報が提供されていない場合には、パーティション番号のみが戻されます。この API は、データ分散を分析する際に役立ちます。

この API の呼び出し時にデータベース・マネージャーが実行している必要はありません。

有効範囲:

この API は、db2nodes.cfg ファイル内の任意のデータベース・パーティション・サーバーから呼び出すことができます。

許可:

なし

API 組み込みファイル:

sqlutil.h

C API 構文:

```
/* File: sqlutil.h */
/* API: sqlugrpn */
/* ... */
SQL_API_RC SQL_API_FN
  sqlugrpn (
    unsigned short num_ptrs,
    unsigned char **ptr_array,
    unsigned short *ptr_lens,
    unsigned short ctrycode,
    unsigned short codepage,
    struct sqlupi *part_info,
    short *part_num,
    SQL_PDB_NODE_TYPE *node_num,
    unsigned short chklvl,
    struct sqlca *sqlca,
    short dataformat,
    void *pReserved1,
    void *pReserved2);
/* ... */
```

汎用 API 構文:

```
/* File: sqlutil.h */
/* API: sqlggrpn */
/* ... */
SQL_API_RC SQL_API_FN
  sqlggrpn (
    unsigned short num_ptrs,
    unsigned char **ptr_array,
    unsigned short *ptr_lens,
    unsigned short ctrycode,
    unsigned short codepage,
    struct sqlupi *part_info,
    short *part_num,
    SQL_PDB_NODE_TYPE *node_num,
    unsigned short chklvl,
    struct sqlca *sqlca,
    short dataformat,
    void *pReserved1,
    void *pReserved2);
/* ... */
```

API パラメーター:

num_ptrs
ptr_array 内のポインターの数。この値は、*part_info* に指定されるものと同じでなければなりません。つまり、*part_info ->sqld* です。

ptr_array
part_info に指定されたパーティション・キーの各パーツに対応する値の文字表示を指すポインターの配列。 NULL 値が必要とされる場合には、対応するポインターが NULL に設定されます。生成された列に関しては、この機能は行の値を生成しません。ユーザーは行を正しく分割するような値を提供する責任があります。

ptr_lens
part_info に指定されたパーティション・キーの各パーツに対応する値の文字表示の長さを含む符号なし整数の配列。

ctrycode
ターゲット・データベースの国地域コード。

この値は、GET DATABASE CONFIGURATION コマンドを使用してデータベース構成ファイルから入手することもできます。

codepage
ターゲット・データベースのコード・ページ。

この値は、GET DATABASE CONFIGURATION コマンドを使用してデータベース構成ファイルから入手することもできます。

part_info
sqlupi 構造を指すポインター。

part_num
パーティション番号の保管に使用される 2 バイトの符号付き整数を指すポインター。

node_num
ノード番号の保管に使用される SQL_PDB_NODE_TYPE フィールドを指すポインター。ポインターが NULL の場合、ノード番号は戻されません。

chklvl
入力パラメーターに対して行われる検査のレベルを指定する符号なし整数。指定された値がゼロである場合、検査は行われません。ゼロ以外の値が指定された場合には、すべての入力パラメーターがチェックされます。

sqlca
出力。 *sqlca* 構造を指すポインター。

dataformat
パーティション・キー値の表示を指定します。有効な値は以下のとおりです。

SQL_CHARSTRING_FORMAT
すべてのパーティション・キー値は文字ストリングによって表示されます。これはデフォルト値です。

SQL_IMPLIEDDECIMAL_FORMAT
暗黙指定されている小数点の位置が列定義によって決定されます。たとえば、列定義が DECIMAL(8,2) である場合、値 12345 は 123.45 として処理されます。

SQL_PACKEDDECIMAL_FORMAT
すべての 10 進数列パーティション・キー値はパック 10 進数形式になります。

SQL_BINARYNUMERICS_FORMAT
すべての数値パーティション・キー値はビッグ・エンディアン・バイナリー数形式になります。

pReserved1
将来の利用のために予約されています。

pReserved2
将来の利用のために予約されています。

使用上の注意:

オペレーティング・システムでサポートされるデータ・タイプは、パーティション・キーとして定義できるものと同じです。

CHAR、VARCHAR、GRAPHIC、および VARGRAPHIC は、この API を呼び出す前にターゲット・コード・ページに変換しなければなりません。

数値および日時データ・タイプの場合、文字表示は、 API が呼び出されるそれぞれのシステムのコード・ページで表記しなければなりません。

node_num が NULL でない場合には、パーティション・マップを提供しなければなりません。つまり、part_info->pmaplen を 2 または 8 192 のどちらかにする必要があります。そうでなければ、SQLCODE -6038 が戻されます。

パーティション・キーを定義しなければなりません。つまり、part_info->sqld をゼロよりも大きくしてください。そうでなければ、SQLCODE -2032 が戻されます。

NULL 値不可のパーティション列に NULL 値が割り当てられている場合には、SQLCODE -6039 が戻されます。

入力文字ストリングの先行ブランクと後書きブランクは、すべて除去されます。ただし、CHAR、VARCHAR、GRAPHIC、および VARGRAPHIC データ・タイプの場合は、後書きブランクのみが除去されます。

関連資料:

- 428 ページの『sqlugtpi - 表のパーティション情報の入手』
- 421 ページの『sqludrdt - データベース・パーティション・グループの再分散』
- 453 ページの『SQLCA』
- 499 ページの『SQLUPI』
- 「コマンド・リファレンス」の『GET DATABASE CONFIGURATION コマンド』
- 「*DB2 Universal Database サーバー機能 概説およびインストール*」の『サポートされる DB2 インターフェース言語』
- 36 ページの『db2CfgGet - 構成パラメーターの入手』

sqlugtpi - 表のパーティション情報の入手

アプリケーションは、表のパーティション情報を入手できます。パーティション情報には、パーティション・マップと、パーティション・キーの列定義が含まれます。この API によって戻される情報を sqlugrpn API に渡して、表の任意の行のパーティション番号とデータベース・パーティション・サーバー番号を判別することができます。

この API を使用するには、アプリケーションが、パーティション情報が要求されている表を含むデータベースに接続されていなければなりません。

有効範囲:

この API は、db2nodes.cfg ファイル内で定義されている任意のデータベース・パーティション・サーバー上で実行できます。

許可:

参照される表について、ユーザーは、少なくとも以下のいずれかを持っていなければなりません。

- *sysadm* 権限
- *dbadm* 権限
- CONTROL 特権
- SELECT 特権

必要な接続:

データベース

API 組み込みファイル:

sqlutil.h

C API 構文:

```
/* File: sqlutil.h */
/* API: sqlugtpi */
/* ... */
SQL_API_RC SQL_API_FN
  sqlugtpi (
    unsigned char *tablename,
    struct sqlupi *part_info,
    struct sqlca *sqlca);
/* ... */
```

汎用 API 構文:

```
/* File: sqlutil.h */
/* API: sqlggtpi */
/* ... */
SQL_API_RC SQL_API_FN
  sqlggtpi (
    unsigned short tn_length,
    unsigned char *tablename,
    struct sqlupi *part_info,
    struct sqlca *sqlca);
/* ... */
```

API パラメーター:

tn_length
表名の長さを示す 2 バイトの符号なし整数。

tablename
表の完全修飾名。

part_info
sqlupi 構造を指すポインター。

pSqlca
出力。*sqlca* 構造へのポインター。

関連資料:

- 425 ページの『sqlugrpn - 行パーティション番号の入手』

- 421 ページの『sqludrdt - データベース・パーティション・グループの再分散』
- 453 ページの『SQLCA』
- 499 ページの『SQLUPI』

sqlurcon - 調整

表の DATALINK データ用のファイルへの参照を妥当性検査します。ファイルへの参照を設定できない行は、例外表 (指定してある場合) へコピーされ、入力表で修正されます。

許可:

以下のいずれかです。

- *sysadm*
- *sysctrl*
- *sysmaint*
- *dbadm*
- 表に対する CONTROL 特権

必要な接続:

データベース

API 組み込みファイル:

sqlutil.h

C API 構文:

```
/* File: sqlutil.h */
/* API: sqlurcon */
/* ... */
SQL_API_RC SQL_API_FN
  sqlurcon (
    char *pTableName,
    char *pExTableName,
    char *pReportFileName,
    void *pReserved,
    struct sqlca *pSqlca);
/* ... */
```

汎用 API 構文:

```
/* File: sqlutil.h */
/* API: sqlgrcon */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgrcon (
    unsigned short TableNameLen,
    char *pTableName,
    unsigned short ExTableNameLen,
    char *pExTableName,
    unsigned short ReportFleNameLen,
    char *pReportFileName,
    void *pReserved,
    struct sqlca *pSqlca);
/* ... */
```

API パラメーター:

TableNameLen
入力。表名の長さを示す 2 バイトの符号なし整数 (バイト単位)。

pTableName
入力。調整を実行する表を指定します。別名、完全修飾、または非修飾の表名を指定することができます。修飾された表名は、*schema.tablename* の形式になります。非修飾の表名を指定すると、その表は現行許可 ID で修飾されます。

ExTableNameLen
入力。例外表の名前の長さを示す 2 バイトの符号なし整数 (バイト単位)。

pExTableName
入力。 DATALINK 値へのリンク障害が発生している行がコピーされる例外表を指定します。

ReportFileNameLen
入力。レポート・ファイル名の長さを示す 2 バイトの符号なし整数 (バイト単位)。

pReportFileName
入力。調整時にリンク解除されるファイルについての情報を含むファイルを指定します。この名前は、完全修飾名である必要があります (たとえば、`/u/johnh/report`)。調整ユーティリティーは、指定されたファイル名に `.ulk` という拡張子を追加します (たとえば、`report.ulk`)。

pReserved
将来の利用のために予約されています。

pSqlca
出力。*sqlca* 構造へのポインター。

使用上の注意:

調整時には、表データに従って存在しているファイルで、データ・リンク・ファイル・マネージャーのメタデータに従って存在しているわけではないものをリンクしようとします。ただし、このことはこれ以外に競合がない場合です。

調整は、表内のすべての DATALINK データに関して実行されます。ファイル参照を再設定できない場合は、違反の行が例外表 (指定してある場合) に挿入されます。これらの行は、入力表からは削除されません。ファイル参照の保全性を確実にするために、問題の DATALINK 値は NULL にされます。列が NULL 可能ではないものとして定義されている場合、 DATALINK 値はゼロの長さの URL に置き換えられます。

例外表を指定していない場合、ファイル参照を再設定できない DATALINK 列値は、列 ID やコメントとともに例外レポート・ファイル (*<pReportFileName* >.exp) にコピーされます。

調整処理の終了時に、この表はデータ・リンク調整ペンディング (DRP) 状態から解放されます。

関連資料:

• 453 ページの『SQLCA』

sqluvqdp - 表の表スペースの静止

特定の表の表スペースを静止させます。有効な静止モードは、共用、更新意図、および排他の 3 つです。静止関数の結果として生じる表スペースの状態は、QUIESCED SHARE、QUIESCED UPDATE、および QUIESCED EXCLUSIVE の 3 つです。

有効範囲:

単一パーティション・データベース環境では、この API はロード期間中に排他モードでロード操作に関与したすべての表スペースを静止します。パーティション・データベース環境では、この API はデータベース・パーティション上でローカルに動作します。ロードが実行されているデータベース・パーティションに属する表スペースのその部分だけを静止します。

許可:

以下のいずれかです。

- *sysadm*
- *sysctrl*
- *sysmaint*
- *dbadm*
- *load*

必要な接続:

データベース

API 組み込みファイル:

sqlutil.h

C API 構文:

```
/* File: sqlutil.h */
/* API: sqluvqdp */
/* ... */
SQL_API_RC SQL_API_FN
  sqluvqdp (
    char *pTableName,
    sqlint32 QuiesceMode,
    void *pReserved,
    struct sqlca *pSqlca);
/* ... */
```

汎用 API 構文:

```
/* File: sqlutil.h */
/* API: sqlgvqdp */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgvqdp (
    unsigned short TableNameLen,
    char *pTableName,
    sqlint32 QuiesceMode,
    void *pReserved,
    struct sqlca *pSqlca);
/* ... */
```

API パラメーター:

TableNameLen
入力。表名の長さを示す 2 バイトの符号なし整数 (バイト単位)。

pTableName
入力。システム・カタログで使用される表名を含むストリング。これは、スキーマ と、ピリオド (.) で区切られた表名の 2 部からなる名前にすることができます。スキーマ が提供されない場合は、 CURRENT SCHEMA が使用されます。 システム・カタログ表を指定することはできません。このフィールドは必須です。

QuiesceMode
入力。静止モードを指定します。有効な値 (sqlutil で定義) は、以下のとおりです。

SQLU_QUIESCEMODE_SHARE
共用モードの場合

SQLU_QUIESCEMODE_INTENT_UPDATE
更新意図モードの場合

SQLU_QUIESCEMODE_EXCLUSIVE
排他モードの場合

SQLU_QUIESCEMODE_RESET
以下のどちらかが真である場合に表スペースを正常状態にリセットします。

- 呼び出し側が静止を所有している
- 静止を設定した呼び出し側が切断し、「ファントム静止」を引き起こしている

SQLU_QUIESCEMODE_RESET_OWNED
呼び出し側が静止を所有している場合に、表スペースを正常状態にリセットします。

このフィールドは必須です。

pReserved
将来の利用のために予約されています。

pSqlca
出力。*sqlca* 構造へのポインター。

REXX API 構文:

```
QUIESCE TABLESPACES FOR TABLE table_name
{SHARE | INTENT TO UPDATE | EXCLUSIVE | RESET}
```

REXX API パラメーター:

table_name
システム・カタログで使用される表の名前。これは、スキーマ と、ピリオド (.) で区切られた表名の 2 部からなる名前にすることができます。スキーマ が提供されない場合は、 CURRENT SCHEMA が使用されます。

使用上の注意:

この API は宣言された一時表ではサポートされません。

共用静止要求を受け取ると、トランザクションは、表スペースに対する共用可能ロックと表に対する共用ロックを要求します。トランザクションがロックを獲得すると、表スペースの状態が QUIESCED SHARE に変更されます。この状態は、他のユーザーがこれと矛盾する状態を保持していない場合に限り、静止者に付与されます。表スペースの状態は、その状態が持続されるように、静止者の許可 ID およびデータベース・エージェント ID とともに、表スペース表に記録されます。

ある表の表スペースが QUIESCED SHARE 状態である間は、その表を変更できません。表および表スペースに対する他の共用モード要求は、認められます。トランザクションがコミットまたはロールバックされる際、ロックは解除されますが、その表の表スペースはその状態が明示的にリセットされるまで、 QUIESCED SHARE 状態のまま残ります。

排他静止要求が行われると、トランザクションは、表スペースに対するスーパー排他ロックと表に対するスーパー排他ロックを要求します。トランザクションがロックを獲得すると、表スペースの状態が QUIESCED EXCLUSIVE に変更されます。表スペースの状態は、静止者の許可 ID およびデータベース・エージェント ID とともに、表スペース表に記録されます。表スペースは、スーパー排他モードで保留されているため、表スペースへの他のアクセスは認められません。ただし、静止プログラム関数を呼び出したユーザー (静止者) は、表と表スペースに排他的にアクセスすることができます。

更新静止要求が行われると、表スペースは排他意図 (IX) モードでロックされ、表は更新 (U) モードでロックされます。表スペースの状態は、静止者とともに、表スペース表に記録されます。

1 つの表スペースに対する静止者の限度は、常に 5 人です。 QUIESCED EXCLUSIVE は、他の状態と非互換であり、 QUIESCED UPDATE は、別の QUIESCED UPDATE と非互換であるため、 5 という静止者の限度に達する場合は、少なくとも 4 つの QUIESCED SHARE と多くて 1 つの QUIESCED UPDATE がなければなりません。

静止者は表スペースの状態を、あまり制限的でない状態から、より制限的な状態へ (たとえば、S から U へ、または U から X へ) アップグレードさせることができます。しかし、ユーザーがすでに保持している状態より低い状態を要求しても、元の状態が戻されます。つまり、状態がダウングレードされることはありません。

表スペースの静止状態は、 SQLU_QUIESCEMODE_RESET を使用することによって明示的にリセットしなければなりません。

関連資料:

- 453 ページの『SQLCA』
- 47 ページの『db2DatabaseQuiesce - データベースの静止』
- 142 ページの『db2InstanceQuiesce - インスタンスの静止』

関連サンプル:

- 『tbmove.sqc -- How to move table data (C)』
- 『tbmove.sqC -- How to move table data (C++)』

- 『tload.sqb -- How to export and load table data (IBM COBOL)』

第 2 章 追加の REXX API

このセクションでは、REXX プログラミング言語でのみサポートされている DB2 アプリケーション・プログラミング・インターフェースについて説明します。

分離レベルの変更 (REXX)

データベースのアクセス中に、 DB2 がデータを他のプロセスから分離する方法を変更します。

許可:

なし

必要な接続:

なし

REXX API 構文:

```
CHANGE SQLISL TO {RR|CS|UR|RS|NC}
```

REXX API パラメーター:

RR 反復可能読み取り。

CS カーソル固定。これがデフォルト値です。

UR 非コミット読み取り。

RS 読み取り固定。

NC コミットなし。

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『REXX のサンプル』

第 3 章 データ構造

このセクションでは、データベース・マネージャーへのアクセスに使用されるデータ構造について説明します。

db2HistData

この構造は、db2HistoryGetEntry への呼び出し後に情報を戻すために使用されます。

表 22. db2HistData 構造のフィールド

フィールド名	データ・タイプ	説明
ioHistDataID	char(8)	記憶域ダンプ用の 8 バイトの構造 ID および「目印」。有効な値は "SQLUHINF" だけです。このストリングには、記号の定義はありません。
oObjectPart	db2Char	最初の 14 文字は、*yyyymmddhhnnss* の形式のタイム・スタンプで、操作を開始した時を示します。次の 3 文字はシーケンス番号です。バックアップ操作では、バックアップ・イメージが複数のファイルまたは複数のテープに保管されるときに、このファイルに複数の項目が入る可能性があります。シーケンス番号によって複数の位置を指定することができます。リストアおよびロード操作では、このファイルに 1 つの項目 (対応するバックアップのシーケンス番号 '001' に該当) だけが入ります。シーケンス番号と結合されるタイム・スタンプは、ユニークなものであることが必要です。
oEndTime	db2Char	操作が完了した時を示す *yyyymmddhhnnss* の形式のタイム・スタンプ。
oFirstLog	db2Char	最も古いログ・ファイル ID (範囲は S0000000 から S9999999)。 • オンライン・バックアップのロールフォワード・リカバリーを適用するために必要 • オフライン・バックアップのロールフォワード・リカバリーを適用するために必要 • ロードの開始時に現行だった全データベースまたは表スペース・レベル・バックアップのリストア後に適用
oLastLog	db2Char	最新のログ・ファイル ID (範囲は S0000000 から S9999999)。 • オンライン・バックアップのロールフォワード・リカバリーを適用するために必要 • オフライン・バックアップの現時点へのロールフォワード・リカバリーを適用するために必要 • ロード操作の終了時に現行だった全データベースまたは表スペース・レベル・バックアップのリストア後に適用 (ロールフォワード・リカバリーが適用されない場合は、*oFirstLog* と同じ)
oID	db2Char	ユニークなバックアップまたは表 ID。

表 22. *db2HistData* 構造のフィールド (続き)

フィールド名	データ・タイプ	説明
oTableQualifier	db2Char	表修飾子。
oTableName	db2Char	表名。
oLocation	db2Char	バックアップおよびロード・コピーの場合、このフィールドはデータが保管された場所を示します。ファイルに複数の項目が入る操作の場合、*oObjectPart* によって定義されるシーケンス番号は、指定された位置でバックアップのどの部分が検出されるかを識別します。リストアおよびロード操作の場合、ロケーションは、常に、リストアまたはロードされたデータの最初の部分 (複数パーツ・バックアップの順序 '001' に該当) が保管された場所を識別します。 *oLocation* のデータは、*oDeviceType* によって解釈方法が異なります。 • ディスクまたはディスケット (D または K) の場合、完全修飾ファイル名 • テープ (T) の場合、ボリューム・レベル • TSM (A) の場合、サーバー名 • ユーザー出口またはその他 (U または O) の場合、自由形式のテキスト
oComment	db2Char	自由形式のテキスト注釈。
oCommandText	db2Char	コマンド・テキストまたは DDL。
oLastLSN	SQLU_LSN	最新のログ・シーケンス番号。
oEID	構造体	ユニークな項目 ID。
poEventSQLCA	構造体	記録されたイベントの結果 *sqlca*。
poTablespace	db2Char	表スペース名のリスト。
ioNumTablespaces	db2Uint32	*poTablespace* リストの項目数。各表スペース・バックアップには 1 つ以上の表スペースが含まれます。各表スペース・リストア操作は 1 つ以上の表スペースを置換します。このフィールドがゼロでない (表スペース・レベル・バックアップまたはリストアを示している) 場合、このファイルの次の行には、18 文字のストリングで表される、バックアップまたはリストアされた表スペースの名前が含まれます。各行に 1 つの表スペース名が入ります。
oOperation	char	441 ページの表 23 を参照してください。
oObject	char	操作の細分性。D (全データベース)、P (表スペース)、および T (表)。
oOptype	char	441 ページの表 24 を参照してください。
oStatus	char	項目の状況。 A (処理)、 D (削除 (将来の利用))、E (期限切れ)、 I (非アクティブ)、 N (未コミット)、 Y (コミット済みまたはアクティブ)、 a (アクティブ・バックアップ。ただしバックアップをまだ完了していないデータ・リンク・サーバーもある)、および i (非アクティブ・バックアップ。ただしバックアップをまだ完了していないデータ・リンク・サーバーもある)。

表 22. *db2HistData* 構造のフィールド (続き)

フィールド名	データ・タイプ	説明
oDeviceType	char	装置タイプ。このフィールドは *oLocation* フィールドの解釈方法を判別します。 A (TSM)、 C (クライアント)、 D (ディスク)、 K (ディスケット)、 L (ローカル)、 O (その他 (他のベンダーの装置をサポート))、 P (パイプ)、 Q (カーソル)、 S (サーバー)、 T (テープ)、および U (ユーザー出口)。

表 23. *db2HistData* 構造の有効な *oOperation* 値

値	説明	C 定義	COBOL/FORTRAN 定義
A	表スペースの追加	DB2HISTORY_OP_ADD_ TABLESPACE	DB2HIST_OP_ADD_ TABLESPACE
B	バックアップ	DB2HISTORY_OP_BACKUP	DB2HIST_OP_BACKUP
C	ロード・コピー	DB2HISTORY_OP_LOAD_COPY	DB2HIST_OP_LOAD_COPY
D	ドロップされた表	DB2HISTORY_OP_DROPPED_ TABLE	DB2HIST_OP_DROPPED_TABLE
F	ロールフォワード	DB2HISTORY_OP_ROLLFWD	DB2HIST_OP_ROLLFWD
G	表の再編成	DB2HISTORY_OP_REORG	DB2HIST_OP_REORG
L	ロード	DB2HISTORY_OP_LOAD	DB2HIST_OP_LOAD
N	表スペースの名前変更	DB2HISTORY_OP_REN_ TABLESPACE	DB2HIST_OP_REN_ TABLESPACE
O	表スペースのドロップ	DB2HISTORY_OP_DROP_ TABLESPACE	DB2HIST_OP_DROP_ TABLESPACE
Q	静止	DB2HISTORY_OP_QUIESCE	DB2HIST_OP_QUIESCE
R	リストア	DB2HISTORY_OP_RESTORE	DB2HIST_OP_RESTORE
T	表スペースの変更	DB2HISTORY_OP_ALT_ TABLESPACE	DB2HIST_OP_ALT_TBS
U	アンロード	DB2HISTORY_OP_UNLOAD	DB2HIST_OP_UNLOAD

表 24. *db2HistData* 構造の有効な *oOptype* 値

oOperation	oOptype	説明	C/COBOL/FORTRAN 定義
B	F	オフライン	DB2HISTORY_OPTYPE_OFFLINE
	N	オンライン	DB2HISTORY_OPTYPE_ONLINE
	I	増分オフライン	DB2HISTORY_OPTYPE_INCR_ OFFLINE
	O	増分オンライン	DB2HISTORY_OPTYPE_INCR_ ONLINE
	D	差分オフライン	DB2HISTORY_OPTYPE_DELTA_ OFFLINE
	E	差分オンライン	DB2HISTORY_OPTYPE_DELTA_ ONLINE
F	E	ログの終わり	DB2HISTORY_OPTYPE_EOL
	P	時点	DB2HISTORY_OPTYPE_PIT
G	F	オフライン	DB2HISTORY_OPTYPE_OFFLINE
	N	オンライン	DB2HISTORY_OPTYPE_ONLINE
L	I	挿入	DB2HISTORY_OPTYPE_INSERT
	R	置換	DB2HISTORY_OPTYPE_REPLACE

表 24. db2HistData 構造の有効な oOptype 値 (続き)

oOperation	oOptype	説明	C/COBOL/FORTRAN 定義
Q	S	共有の静止	DB2HISTORY_OPTYPE_SHARE
	U	更新の静止	DB2HISTORY_OPTYPE_UPDATE
	X	排他の静止	DB2HISTORY_OPTYPE_EXCL
	Z	リセットの静止	DB2HISTORY_OPTYPE_RESET
R	F	オフライン	DB2HISTORY_OPTYPE_OFFLINE
	N	オンライン	DB2HISTORY_OPTYPE_ONLINE
	I	増分オフライン	DB2HISTORY_OPTYPE_INCR_ OFFLINE
	O	増分オンライン	DB2HISTORY_OPTYPE_INCR_ ONLINE
T	C	コンテナーの追加	DB2HISTORY_OPTYPE_ADD_CONT
	R	再バランス	DB2HISTORY_OPTYPE_REB

表 25. db2Char 構造のフィールド

フィールド名	データ・タイプ	説明
pioData	char	文字データ・バッファーを指すポインター。 NULL の場合、データは戻されません。
iLength	db2Uint32	入力。 *pioData* バッファーのサイズ。
oLength	db2Uint32	出力。 *pioData* バッファー内にあるデータの有効文字の数。

表 26. db2HistoryEID 構造のフィールド

フィールド名	データ・タイプ	説明
ioNode	SQL_PDB_NODE_TYPE	ノード番号。
ioHID	db2Uint32	ローカル履歴ファイルの項目 ID。

言語構文:

C 構造

```
/* File: db2ApiDf.h */
/* ... */
typedef SQL_STRUCTURE db2HistoryData
{
  char ioHistDataID[8];
  db2Char oObjectPart;
  db2Char oEndTime;
  db2Char oFirstLog;
  db2Char oLastLog;
  db2Char oID;
  db2Char oTableQualifier;
  db2Char oTableName;
  db2Char oLocation;
  db2Char oComment;
  db2Char oCommandText;
  SQLU_LSN oLastLSN;
  db2HistoryEID oEID;
  struct sqlca * poEventSQLCA;
  db2Char * poTablespace;
  db2Uint32 ioNumTablespaces;
  char oOperation;
  char oObject;
  char oOptype;
```

```
  char oStatus;
  char oDeviceType
} db2HistoryData;

typedef SQL_STRUCTURE db2Char
{
  char * pioData;
  db2Uint32 ioLength
} db2Char;

typedef SQL_STRUCTURE db2HistoryEID
{
  SQL_PDB_NODE_TYPE ioNode;
  db2Uint32 ioHID
} db2HistoryEID;
/* ... */
```

関連資料:

- 104 ページの『db2HistoryGetEntry - 履歴ファイルの次項目の入手』
- 453 ページの『SQLCA』

SQL-AUTHORIZATIONS

この構造は、sqluadau API への呼び出し後に情報を戻すために使用されます。すべてのフィールドのデータ・タイプは SMALLINT です。以下の表の上半分は、ユーザーに直接付与される権限です。下半分は、ユーザーが属するグループに付与される権限です。

表 27. SQL-AUTHORIZATIONS 構造のフィールド

フィールド名	説明
SQL_AUTHORIZATIONS_LEN	構造のサイズ。
SQL_SYSADM_AUTH	SYSADM 権限。
SQL_SYSCTRL_AUTH	SYSCTRL 権限。
SQL_SYSMAINT_AUTH	SYSMAINT 権限。
SQL_DBADM_AUTH	DBADM 権限。
SQL_CREATETAB_AUTH	CREATETAB 権限。
SQL_CREATET_NOT_FENC_AUTH	CREATE_NOT_FENCED 権限。
SQL_BINDADD_AUTH	BINDADD 権限。
SQL_CONNECT_AUTH	CONNECT 権限。
SQL_IMPLICIT_SCHEMA_AUTH	IMPLICIT_SCHEMA 権限。
SQL_LOAD_AUTH	LOAD 権限。
SQL_SYSADM_GRP_AUTH	SYSADM 権限を保持するグループに所属するユーザー。
SQL_SYSCTRL_GRP_AUTH	SYSCTRL 権限を保持するグループに所属するユーザー。
SQL_SYSMAINT_GRP_AUTH	SYSMAINT 権限を保持するグループに所属するユーザー。
SQL_DBADM_GRP_AUTH	DBADM 権限を保持するグループに所属するユーザー。
SQL_CREATETAB_GRP_AUTH	CREATETAB 権限を保持するグループに所属するユーザー。
SQL_CREATE_NON_FENC_GRP_AUTH	CREATE_NOT_FENCED 権限を保持するグループに所属するユーザー。

表 27. *SQL-AUTHORIZATIONS 構造のフィールド (続き)*

フィールド名	説明
SQL_BINDADD_GRP_AUTH	BINDADD 権限を保持するグループに所属するユーザー。
SQL_CONNECT_GRP_AUTH	CONNECT 権限を保持するグループに所属するユーザー。
SQL_IMPLICIT_SCHEMA_GRP_AUTH	IMPLICIT_SCHEMA 権限を保持するグループに所属するユーザー。
SQL_LOAD_GRP_AUTH	LOAD 権限を保持するグループに所属するユーザー。
注: SYSADM、SYSMAINT、および SYSCTRL は間接権限だけであり、ユーザーに直接付与されることはありません。これらの権限は、ユーザーが所属するグループを通じてのみ使用可能です。	

言語構文:

C 構造

```
/* File: sqlutil.h */
/* Structure: SQL-AUTHORIZATIONS */
/* ... */
SQL_STRUCTURE sql_authorizations
{
  short           sql_authorizations_len;
  short           sql_sysadm_auth;
  short           sql_dbadm_auth;
  short           sql_createtab_auth;
  short           sql_bindadd_auth;
  short           sql_connect_auth;
  short           sql_sysadm_grp_auth;
  short           sql_dbadm_grp_auth;
  short           sql_createtab_grp_auth;
  short           sql_bindadd_grp_auth;
  short           sql_connect_grp_auth;
  short           sql_sysctrl_auth;
  short           sql_sysctrl_grp_auth;
  short           sql_sysmaint_auth;
  short           sql_sysmaint_grp_auth;
  short           sql_create_not_fenc_auth;
  short           sql_create_not_fenc_grp_auth;
  short           sql_implicit_schema_auth;
  short           sql_implicit_schema_grp_auth;
  short           sql_load_auth;
  short           sql_load_grp_auth;
};
/* ... */
```

COBOL 構造

```
* File: sqlutil.cbl
01 SQL-AUTHORIZATIONS.
    05 SQL-AUTHORIZATIONS-LEN PIC S9(4) COMP-5.
    05 SQL-SYSADM-AUTH        PIC S9(4) COMP-5.
    05 SQL-DBADM-AUTH         PIC S9(4) COMP-5.
    05 SQL-CREATETAB-AUTH     PIC S9(4) COMP-5.
    05 SQL-BINDADD-AUTH       PIC S9(4) COMP-5.
    05 SQL-CONNECT-AUTH       PIC S9(4) COMP-5.
    05 SQL-SYSADM-GRP-AUTH    PIC S9(4) COMP-5.
    05 SQL-DBADM-GRP-AUTH     PIC S9(4) COMP-5.
    05 SQL-CREATETAB-GRP-AUTH PIC S9(4) COMP-5.
    05 SQL-BINDADD-GRP-AUTH   PIC S9(4) COMP-5.
    05 SQL-CONNECT-GRP-AUTH   PIC S9(4) COMP-5.
    05 SQL-SYSCTRL-AUTH       PIC S9(4) COMP-5.
```

```
    05 SQL-SYSCTRL-GRP-AUTH   PIC S9(4) COMP-5.
    05 SQL-SYSMAINT-AUTH      PIC S9(4) COMP-5.
    05 SQL-SYSMAINT-GRP-AUTH  PIC S9(4) COMP-5.
    05 SQL-CREATE-NOT-FENC-AUTH PIC S9(4) COMP-5.
    05 SQL-CREATE-NOT-FENC-GRP-AUTH PIC S9(4) COMP-5.
    05 SQL-IMPLICIT-SCHEMA-AUTH PIC S9(4) COMP-5.
    05 SQL-IMPLICIT-SCHEMA-GRP-AUTH PIC S9(4) COMP-5.
    05 SQL-LOAD-AUTH PIC S9(4) COMP-5.
    05 SQL-LOAD-GRP-AUTH PIC S9(4) COMP-5.
*
```

関連資料:

- 419 ページの『sqluadau - 許可の入手』

SQL-DIR-ENTRY

この構造は、DCS ディレクトリー API によって使用されます。

表 28. SQL-DIR-ENTRY 構造のフィールド

フィールド名	データ・タイプ	説明
STRUCT_ID	SMALLINT	構造 ID。 SQL_DCS_STR_ID (sqlenv で定義) に設定してください。
RELEASE	SMALLINT	(API が割り当てた) リリース・バージョン。
CODEPAGE	SMALLINT	コメント用コード・ページ。
COMMENT	CHAR(30)	データベースのオプション・コメント。
LDB	CHAR(8)	データベースのローカル名。システム・データベース・ディレクトリー内のデータベース別名と一致していなければなりません。
TDB	CHAR(18)	データベースの実名。
AR	CHAR(32)	アプリケーション・クライアントの名前。
PARM	CHAR(512)	トランザクション・プログラムの接頭部、トランザクション名、SQLCODE マッピング・ファイル名、および切断/セキュリティー・オプションが入ります。
注: この構造に渡された文字フィールドは、NULL 終了にするか、フィールドの長さいっぱいにまでブランクを埋め込む必要があります。		

言語構文:

C 構造

```
/* File: sqlenv.h */
/* Structure: SQL-DIR-ENTRY */
/* ... */
SQL_STRUCTURE sql_dir_entry
{
  unsigned short         struct_id;
  unsigned short         release;
  unsigned short         codepage;
  _SQLOLDCHAR            comment[SQL_CMT_SZ + 1];
  _SQLOLDCHAR            ldb[SQL_DBNAME_SZ + 1];
  _SQLOLDCHAR            tdb[SQL_LONG_NAME_SZ + 1];
  _SQLOLDCHAR            ar[SQL_AR_SZ + 1];
  _SQLOLDCHAR            parm[SQL_PARAMETER_SZ + 1];
};
/* ... */
```

COBOL 構造

```
* File: sqlenv.cbl
01 SQL-DIR-ENTRY.
    05 STRUCT-ID             PIC 9(4) COMP-5.
    05 RELEASE-LVL           PIC 9(4) COMP-5.
    05 CODEPAGE              PIC 9(4) COMP-5.
    05 COMMENT               PIC X(30).
    05 FILLER                PIC X.
    05 LDB                   PIC X(8).
    05 FILLER                PIC X.
    05 TDB                   PIC X(18).
    05 FILLER                PIC X.
    05 AR                    PIC X(32).
    05 FILLER                PIC X.
    05 PARM                  PIC X(512).
    05 FILLER                PIC X.
    05 FILLER                PIC X(1).
*
```

SQLA-FLAGINFO

この構造は、flagger 情報を保持するのに使用されます。

表 29. SQLA-FLAGINFO 構造のフィールド

フィールド名	データ・タイプ	説明
VERSION	SMALLINT	SQLA_FLAG_VERSION (sqlaprep で定義) に設定する必要のある入力フィールド。
MSGS	構造体	組み込み *sqla_flagmsgs* 構造。

表 30. SQLA-FLAGMSGS 構造のフィールド

フィールド名	データ・タイプ	説明
COUNT	SMALLINT	flagger によって戻されるメッセージの数に設定される出力フィールド。
SQLCA	配列	flagger からの情報を戻す SQLCA 構造の配列。

言語構文:

C 構造

```
/* File: sqlaprep.h */
/* Structure: SQLA-FLAGINFO */
/* ... */
SQL_STRUCTURE sqla_flaginfo
{
  short             version;
  short             padding;
  struct            sqla_flagmsgs msgs;
};
/* ... */
```

```
/* File: sqlaprep.h */
/* Structure: SQLA-FLAGMSGS */
/* ... */
SQL_STRUCTURE sqla_flagmsgs
{
  short             count;
```

```
  short             padding;
  SQL_STRUCTURE sqlca sqlca[SQLA_FLAG_MAXMSGS];
};
/* ... */
```

COBOL 構造

```
* File: sqlaprep.cbl
01 SQLA-FLAGINFO.
    05 SQLFLAG-VERSION            PIC 9(4) COMP-5.
    05 FILLER                     PIC X(2).
    05 SQLFLAG-MSGS.
        10 SQLFLAG-MSGS-COUNT      PIC 9(4) COMP-5.
        10 FILLER                  PIC X(2).
        10 SQLFLAG-MSGS-SQLCA OCCURS 10 TIMES.
*
```

SQLB-TBS-STATS

この構造は、追加の表スペース統計をアプリケーション・プログラムへ戻すのに使用されます。

表 31. SQLB-TBS-STATS 構造のフィールド

フィールド名	データ・タイプ	説明
TOTALPAGES	INTEGER	表スペースによって占有されているオペレーティング・システム・スペースの合計 (4KB ページ単位)。 DMS の場合、コンテナー・サイズ (オーバーヘッドを含む) の合計になります。 SMS の場合、この表スペースに格納された表によって使用されている全ファイル・スペースの合計になります。これは、SMS 表スペースに関して戻される唯一の情報です。他のフィールドはこの値またはゼロに設定されます。
USEABLEPAGES	INTEGER	DMS の場合、TOTALPAGES からオーバーヘッド + 部分エクステントを引いた数値になります。 SMS の場合、TOTALPAGES と同じです。
USEDPAGES	INTEGER	DMS の場合、使用中のページの合計数です。 SMS の場合、TOTALPAGES と同じです。
FREEPAGES	INTEGER	DMS の場合、USEABLEPAGES から USEDPAGES を引いた数値と同じです。 SMS には適用されません。
HIGHWATERMARK	INTEGER	DMS の場合、最高水準点は、表スペースのアドレス・スペースの現在の「終わり」です。言い換えれば、表スペースの最後に割り振られたエクステントに続く最初の空きエクステントのページ番号です。 これは、実際には「最高水準点」ではなく、「現行水準点」である (値が減少する可能性があるため) ことに注意してください。 SMS には適用されません。

表スペースの再バランス中、使用可能なページの数には、新しく追加されたコンテナーのページも含まれますが、これらの新しいページは、再バランスが完了するま

では空きページの数に反映されません。表スペースの再バランスが行われていないときは、使用中のページ数と空きページの数の合計が使用可能なページの数になります。

言語構文:

C 構造

```
/* File: sqlutil.h */
/* Structure: SQLB-TBS-STATS */
/* ... */
SQL_STRUCTURE SQLB_TBS_STATS
{
  sqluint32   totalPages;
  sqluint32   useablePages;
  sqluint32   usedPages;
  sqluint32   freePages;
  sqluint32   highWaterMark;
};
/* ... */
```

COBOL 構造

```
* File: sqlutil.cbl
01 SQLB-TBS-STATS.
    05 SQL-TOTAL-PAGES        PIC 9(9) COMP-5.
    05 SQL-USEABLE-PAGES      PIC 9(9) COMP-5.
    05 SQL-USED-PAGES         PIC 9(9) COMP-5.
    05 SQL-FREE-PAGES         PIC 9(9) COMP-5.
    05 SQL-HIGH-WATER-MARK    PIC 9(9) COMP-5.
*
```

SQLB-TBSCONTQRY-DATA

この構造は、コンテナー・データをアプリケーション・プログラムへ戻すのに使用されます。

表 32. SQLB-TBSCONTQRY-DATA 構造のフィールド

フィールド名	データ・タイプ	説明
ID	INTEGER	コンテナー ID。
NTBS	INTEGER	通常 1。
TBSID	INTEGER	表スペース ID。
NAMELEN	INTEGER	コンテナー名の長さ (C 以外の言語用)。
NAME	CHAR(256)	コンテナー名。
UNDERDBDIR	INTEGER	1 (コンテナーが DB ディレクトリーの下にある)、または 0 (コンテナーが DB ディレクトリーの下にない) のどちらか。
CONTTYPE	INTEGER	コンテナー・タイプ。
TOTALPAGES	INTEGER	表スペース・コンテナーが占有しているページの合計数。
USEABLEPAGES	INTEGER	DMS の場合、TOTALPAGES からオーバーヘッドを引いた数値になります。 SMS の場合、TOTALPAGES と同じです。
OK	INTEGER	1 (コンテナーがアクセス可能)、または 0 (コンテナーがアクセス不能) のどちらか。ゼロは異常な状態を指し示しており、大抵データベース管理者の注意を促すものです。

CONTTYPE (sqlutil で定義) に可能な値は、以下のとおりです。

SQLB_CONT_PATH
ディレクトリー・パスを指定します (SMS のみ)。

SQLB_CONT_DISK
ロー・デバイスを指定します (DMS のみ)。

SQLB_CONT_FILE
ファイルを指定します (DMS のみ)。

言語構文:

C 構造

```
/* File: sqlutil.h */
/* Structure: SQLB-TBSCONTQRY-DATA */
/* ... */
SQL_STRUCTURE SQLB_TBSCONTQRY_DATA
{
  sqluint32   id;
  sqluint32   nTbs;
  sqluint32   tbsID;
  sqluint32   nameLen;
  char        name[SQLB_MAX_CONTAIN_NAME_SZ];
  sqluint32   underDBDir;
  sqluint32   contType;
  sqluint32   totalPages;
  sqluint32   useablePages;
  sqluint32   ok;
};
/* ... */
```

COBOL 構造

```
* File: sqlutbcq.cbl
01 SQLB-TBSCONTQRY-DATA.
    05 SQL-ID                 PIC 9(9) COMP-5.
    05 SQL-N-TBS              PIC 9(9) COMP-5.
    05 SQL-TBS-ID             PIC 9(9) COMP-5.
    05 SQL-NAME-LEN           PIC 9(9) COMP-5.
    05 SQL-NAME               PIC X(256).
    05 SQL-UNDER-DBDIR        PIC 9(9) COMP-5.
    05 SQL-CONT-TYPE          PIC 9(9) COMP-5.
    05 SQL-TOTAL-PAGES        PIC 9(9) COMP-5.
    05 SQL-USEABLE-PAGES      PIC 9(9) COMP-5.
    05 SQL-OK                 PIC 9(9) COMP-5.
*
```

SQLB-TBSPQRY-DATA

この構造は、表スペース・データをアプリケーション・プログラムへ戻すのに使用されます。

表 33. SQLB-TBSPQRY-DATA 構造のフィールド

フィールド名	データ・タイプ	説明
TBSPQVER	CHAR(8)	構造のバージョン ID。
ID	INTEGER	表スペースの内部 ID。
NAMELEN	INTEGER	表スペース名の長さ。

表 33. SQLB-TBSPQRY-DATA 構造のフィールド (続き)

フィールド名	データ・タイプ	説明
NAME	CHAR(128)	表スペースの NULL 終了名。
TOTALPAGES	INTEGER	CREATE TABLESPACE で指定されたページ数 (DMS のみ)。
USEABLEPAGES	INTEGER	TOTALPAGES からオーバーヘッドを引いた数 (DMS のみ)。この値は、次の 4KB の倍数に切り捨てられます。
FLAGS	INTEGER	表スペースのビット属性。
PAGESIZE	INTEGER	表スペースのページ・サイズ (バイト単位)。現在、4KB に固定されています。
EXTSIZE	INTEGER	表スペースのエクステント・サイズ (ページ単位)。
PREFETCHSIZE	INTEGER	プリフェッチ・サイズ。
NCONTAINERS	INTEGER	表スペース内のコンテナーの数。
TBSSTATE	INTEGER	表スペースの状態。
LIFELSN	CHAR(6)	表スペースの原点を識別するタイム・スタンプ。
FLAGS2	INTEGER	表スペースのビット属性。
MINIMUMRECTIME	CHAR(27)	表スペースを特定の時点までロールフォワードする場合に指定できる最も早い時点。
STATECHNGOBJ	INTEGER	TBSSTATE が SQLB_LOAD_PENDING または SQLB_DELETE_PENDING である場合は、表スペースの状態を設定する原因となった、表スペース STATECHANGEID のオブジェクト ID。それ以外の場合は、ゼロ。
STATECHNGID	INTEGER	TBSSTATE が SQLB_LOAD_PENDING または SQLB_DELETE_PENDING である場合は、表スペースの状態を設定する原因となった、オブジェクト STATECHANGEOBJ の表スペース ID。それ以外の場合は、ゼロ。
NQUIESCERS	INTEGER	TBSSTATE が SQLB_QUIESCED_SHARE、UPDATE、または EXCLUSIVE である場合は、表スペースの静止者の数と、 QUIESCERS 内の項目の数。
QUIESCEID	INTEGER	表スペースが静止する原因となったオブジェクト QUIESCEOBJ の表スペース ID。
QUIESCEOBJ	INTEGER	表スペースが静止する原因となった表スペース QUIESCEID のオブジェクト ID。
RESERVED	CHAR(32)	将来の利用のために予約されています。

FLAGS に有効な値 (sqlutil で定義) は、以下のとおりです。

SQLB_TBS_SMS
システム管理スペース

SQLB_TBS_DMS
データベース管理スペース

SQLB_TBS_ANY
正規の内容

SQLB_TBS_LONG
長フィールド・データ

SQLB_TBS_SYSTMP
システム一時データ。

SQLB_TBS_USRTMP
ユーザー一時データ。

TBSSTATE に有効な値 (sqlutil で定義) は、以下のとおりです。

SQLB_NORMAL
通常

SQLB_QUIESCED_SHARE
静止: SHARE

SQLB_QUIESCED_UPDATE
静止: UPDATE

SQLB_QUIESCED_EXCLUSIVE
静止: EXCLUSIVE

SQLB_LOAD_PENDING
ロード・ペンディング

SQLB_DELETE_PENDING
削除ペンディング

SQLB_BACKUP_PENDING
バックアップ・ペンディング中

SQLB_ROLLFORWARD_IN_PROGRESS
ロールフォワード進行中

SQLB_ROLLFORWARD_PENDING
ロールフォワード・ペンディング

SQLB_RESTORE_PENDING
リストア・ペンディング

SQLB_DISABLE_PENDING
使用不能化ペンディング

SQLB_REORG_IN_PROGRESS
再編成進行中

SQLB_BACKUP_IN_PROGRESS
バックアップ進行中

SQLB_STORDEF_PENDING
記憶域の定義が必要

SQLB_RESTORE_IN_PROGRESS
リストア進行中

SQLB_STORDEF_ALLOWED
記憶域の定義が可能

SQLB_STORDEF_FINAL_VERSION
記憶域の定義が「最終」状態

SQLB_STORDEF_CHANGED
ロールフォワード前に記憶域定義が変更された

SQLB_REBAL_IN_PROGRESS
DMS リバランサーがアクティブ

SQLB_PSTAT_DELETION
表スペース削除の進行中

SQLB_PSTAT_CREATION
表スペース作成の進行中

FLAGS2 に有効な値 (sqlutil で定義) は、以下のとおりです。

SQLB_STATE_SET
サービスの目的でのみ使用されます。

言語構文:

C 構造

```
/* File: sqlutil.h */
/* ... */
SQL_STRUCTURE SQLB_TBSPQRY_DATA
{
   char                        tbspqver[SQLB_SVERSION_SIZE];
   sqluint32                   id;
   sqluint32                   nameLen;
   char                        name[SQLB_MAX_TBS_NAME_SZ];
   sqluint32                   totalPages;
   sqluint32                   useablePages;
   sqluint32                   flags;
   sqluint32                   pageSize;
   sqluint32                   extSize;
   sqluint32                   prefetchSize;
   sqluint32                   nContainers;
   sqluint32                   tbsState;
   char                        lifeLSN[6];
   char                        pad[2];
   sqluint32                   flags2;
   char                        minimumRecTime[SQL_STAMP_STRLEN+1];
   char                        pad1[1];
   sqluint32                   StateChngObj;
   sqluint32                   StateChngID;
   sqluint32                   nQuiescers;
   struct SQLB_QUIESCER_DATA   quiescer[SQLB_MAX_QUIESCERS];
   char                        reserved[32];
};
/* ... */
```

```
/* File: sqlutil.h */
/* ... */
SQL_STRUCTURE SQLB_QUIESCER_DATA
{
   sqluint32   quiesceId;
   sqluint32   quiesceObject;
};
/* ... */
```

COBOL 構造

```
* File: sqlutbsp.cbl
01 SQLB-TBSPQRY-DATA.
    05 SQL-TBSPQVER          PIC X(8).
    05 SQL-ID                PIC 9(9) COMP-5.
    05 SQL-NAME-LEN          PIC 9(9) COMP-5.
    05 SQL-NAME              PIC X(128).
    05 SQL-TOTAL-PAGES       PIC 9(9) COMP-5.
```

```
    05 SQL-USEABLE-PAGES      PIC 9(9) COMP-5.
    05 SQL-FLAGS              PIC 9(9) COMP-5.
    05 SQL-PAGE-SIZE          PIC 9(9) COMP-5.
    05 SQL-EXT-SIZE           PIC 9(9) COMP-5.
    05 SQL-PREFETCH-SIZE      PIC 9(9) COMP-5.
    05 SQL-N-CONTAINERS       PIC 9(9) COMP-5.
    05 SQL-TBS-STATE          PIC 9(9) COMP-5.
    05 SQL-LIFE-LSN           PIC X(6).
    05 SQL-PAD                PIC X(2).
    05 SQL-FLAGS2             PIC 9(9) COMP-5.
    05 SQL-MINIMUM-REC-TIME   PIC X(26).
    05 FILLER                 PIC X.
    05 SQL-PAD1               PIC X(1).
    05 SQL-STATE-CHNG-OBJ     PIC 9(9) COMP-5.
    05 SQL-STATE-CHNG-ID      PIC 9(9) COMP-5.
    05 SQL-N-QUIESCERS        PIC 9(9) COMP-5.
    05 SQL-QUIESCER OCCURS 5 TIMES.
        10 SQL-QUIESCE-ID     PIC 9(9) COMP-5.
        10 SQL-QUIESCE-OBJECT PIC 9(9) COMP-5.
    05 SQL-RESERVED           PIC X(32).
*
```

SQLCA

SQL 連絡域 (SQLCA) 構造は、データベース・マネージャーがエラー情報をアプリケーション・プログラムへ戻すために使用します。この構造は、API 呼び出しおよび SQL ステートメントが発行されるたびに、更新されます。

言語構文:

C 構造

```
/* File: sqlca.h */
/* Structure: SQLCA */
/* ... */
SQL_STRUCTURE  sqlca
{
  _SQLOLDCHAR    sqlcaid[8];
  sqlint32       sqlcabc;
  #ifdef DB2_SQL92E
  sqlint32       sqlcade;
  #else
  sqlint32       sqlcode;
  #endif
  short          sqlerrml;
  _SQLOLDCHAR    sqlerrmc[70];
  _SQLOLDCHAR    sqlerrp[8];
  sqlint32       sqlerrd[6];
  _SQLOLDCHAR    sqlwarn[11];
  #ifdef DB2_SQL92E
  _SQLOLDCHAR    sqlstat[5];
  #else
  _SQLOLDCHAR    sqlstate[5];
  #endif
};
/* ... */
```

COBOL 構造

```
* File: sqlca.cbl
01 SQLCA SYNC.
    05 SQLCAID PIC X(8) VALUE "SQLCA   ".
    05 SQLCABC PIC S9(9) COMP-5 VALUE 136.
    05 SQLCODE PIC S9(9) COMP-5.
```

```
    05 SQLERRM.
    05 SQLERRP PIC X(8).
    05 SQLERRD OCCURS 6 TIMES PIC S9(9) COMP-5.
    05 SQLWARN.
        10 SQLWARN0 PIC X.
        10 SQLWARN1 PIC X.
        10 SQLWARN2 PIC X.
        10 SQLWARN3 PIC X.
        10 SQLWARN4 PIC X.
        10 SQLWARN5 PIC X.
        10 SQLWARN6 PIC X.
        10 SQLWARN7 PIC X.
        10 SQLWARN8 PIC X.
        10 SQLWARN9 PIC X.
        10 SQLWARNA PIC X.
    05 SQLSTATE PIC X(5).
*
```

関連資料:

- 「*SQL リファレンス 第 1 巻*」の『SQLCA (SQL 連絡域)』

SQLCHAR

この構造は、可変長データをデータベース・マネージャーへ渡すのに使用されます。

表 34. SQLCHAR 構造のフィールド

フィールド名	データ・タイプ	説明
LENGTH	SMALLINT	*DATA* が指す文字ストリングの長さ。
DATA	CHAR(n)	長さ *LENGTH* の文字の配列。

言語構文:

C 構造

```
/* File: sql.h */
/* Structure: SQLCHAR */
/* ... */
SQL_STRUCTURE sqlchar
{
  short                 length;
  _SQLOLDCHAR           data[1];
};
/* ... */
```

COBOL 構造

この構造は、ヘッダー・ファイルでは定義されません。以下に示すのは、この構造をどのように定義できるかを示した一例です。

```
* Replace maxlen with the appropriate value:
01 SQLCHAR.
49 SQLCHAR-LEN  PIC S9(4) COMP-5.
49 SQLCHAR-DATA PIC X(maxlen).
```

SQLDA

SQL 記述子域 (SQLDA) 構造は、 SQL DESCRIBE ステートメントを実行するために必要な変数の集合です。 SQLDA 変数は、 PREPARE、 OPEN、 FETCH、 EXECUTE、および CALL ステートメントと一緒に使用できるオプションです。

SQLDA は、動的 SQL と通信します。これを DESCRIBE ステートメントで使用し、ホスト変数のアドレスを用いて修正し、FETCH ステートメントで再利用することができます。

SQLDA は、すべての言語についてサポートされます。ただし、事前定義宣言は、C、REXX、FORTRAN、および COBOL 専用に用意されています。

SQLDA 内の情報の意味は、その用途によって異なります。 PREPARE および DESCRIBE では、SQLDA はアプリケーション・プログラムに準備済みステートメントに関する情報を提供します。 OPEN、EXECUTE、FETCH、および CALL では、SQLDA はホスト変数を記述します。

言語構文:

C 構造

```
/* File: sqlda.h */
/* Structure: SQLDA */
/* ... */
SQL_STRUCTURE  sqlda
{
  _SQLOLDCHAR    sqldaid[8];
  long           sqldabc;
  short          sqln;
  short          sqld;
  struct sqlvar  sqlvar[1];
};
/* ... */

/* File: sqlda.h */
/* Structure: SQLVAR */
/* ... */
SQL_STRUCTURE  sqlvar
{
  short          sqltype;
  short          sqllen;
  _SQLOLDCHAR   *SQL_POINTER sqldata;
  short         *SQL_POINTER sqlind;
  struct sqlname sqlname;
};
/* ... */

/* File: sqlda.h */
/* Structure: SQLNAME */
/* ... */
SQL_STRUCTURE  sqlname
{
  short          length;
  _SQLOLDCHAR    data[30];
};
/* ... */

/* File: sqlda.h */
/* Structure: SQLVAR2 */
/* ... */
SQL_STRUCTURE  sqlvar2
```

```
{
  union sql8bytelen len;
  char *SQL_POINTER sqldatalen;
  struct sqldistinct_type sqldatatype_name;
};
/* ... */
```

```
/* File: sqlda.h */
/* Structure: SQL8BYTELEN */
/* ... */
union sql8bytelen
{
  long        reserve1[2];
  long        sqllonglen;
};
/* ... */
```

```
/* File: sqlda.h */
/* Structure: SQLDISTINCT-TYPE */
/* ... */
SQL_STRUCTURE  sqldistinct_type
{
  short          length;
  char           data[27];
  char           reserved1[3];
};
/* ... */
```

COBOL 構造

```
* File: sqlda.cbl
01 SQLDA SYNC.
    05 SQLDAID PIC X(8) VALUE "SQLDA  ".
    05 SQLDABC PIC S9(9) COMP-5.
    05 SQLN PIC S9(4) COMP-5.
    05 SQLD PIC S9(4) COMP-5.
    05 SQLVAR-ENTRIES OCCURS 0 TO 1489 TIMES
        10 SQLVAR.
        10 SQLVAR2 REDEFINES SQLVAR.
*
```

関連資料:

- 「*SQL リファレンス 第 1 巻*」の『SQLDA (SQL 記述子域)』

SQLDCOL

この構造は、変数列情報を db2Export、db2Import、および db2Load API へ渡すために使用されます。

表 35. SQLDCOL 構造のフィールド

フィールド名	データ・タイプ	説明
DCOLMETH	SMALLINT	データ・ファイル内の列の選択に使用する方式を示す文字。
DCOLNUM	SMALLINT	*DCOLNAME* 配列で指定された列の数。
DCOLNAME	配列	*DCOLNUM* 個の *sqldcoln* 構造の配列。

表 36. SQLDCOLN 構造のフィールド

フィールド名	データ・タイプ	説明
DCOLNLEN	SMALLINT	*DCOLNPTR* が指すデータの長さ。
DCOLNPTR	ポインター	*DCOLMETH* により判別されたデータ・エレメントを指すポインター。
注: *DCOLNLEN* および *DCOLNPTR* フィールドは、指定された列ごとに繰り返されます。		

表 37. SQLLOCTAB 構造のフィールド

フィールド名	データ・タイプ	説明
LOCPAIR	配列	*sqllocpair* 構造の配列。

表 38. SQLLOCPAIR 構造のフィールド

フィールド名	データ・タイプ	説明
BEGIN_LOC	SMALLINT	外部ファイル内の列データの開始位置。
END_LOC	SMALLINT	外部ファイル内の列データの終了位置。

DCOLMETH に有効な値 (sqlutil で定義) は、以下のとおりです。

SQL_METH_N
名前。インポートまたはロード時には、この構造により提供された列名が、外部ファイルからインポートまたはロードされるデータの識別に使用されます。これらの列名の大文字小文字の区別は、システム・カタログ内の対応する名前の大文字小文字の区別と一致しなければなりません。エクスポート時は、この構造により提供された列名が出力ファイル内の列名として使用されます。

dcolname 配列の各エレメントの *dcolnptr* ポインターは、インポートまたはロードされる列の名前を構成する、長さ *dcolnlen* バイトの文字の配列を指します。 *dcolnum* フィールド (正でなければならない) は、 *dcolname* 配列内のエレメント数を示します。

外部ファイルが列名を含んでいない場合 (たとえば、DEL や ASC 形式ファイルの場合)、この方式は無効です。

SQL_METH_P
位置。インポートまたはロード時には、この構造により提供された開始列位置が、外部ファイルからインポートまたはロードされるデータの識別に使用されます。データのエクスポート時には、この方式は無効です。

dcolnlen フィールドに外部ファイルの列位置が入っていると、 *dcolname* 配列の各エレメントの *dcolnptr* ポインターは無視されます。 *dcolnum* フィールド (正でなければならない) は、 *dcolname* 配列内のエレメント数を示します。

最も低い有効な列位置の値は、1 (最初の列を示す) です。最も高い有効な値は、外部ファイルのタイプによって異なります。位置による選択は、ASC ファイルのインポートでは無効です。

SQL_METH_L

ロケーション。インポートまたはロード時には、この構造により提供された開始および終了列位置が、外部ファイルからインポートまたはロードされるデータの識別に使用されます。データのエクスポート時には、この方式は無効です。

dcolname 配列の最初のエレメントの *dcolnptr* フィールドは、 *sqlloctab* 構造 (*sqllocpair* 構造の配列を構成) を指します。この構造内のエレメント数は、*sqldcol* 構造の *dcolnum* フィールド (正でなければならない) によって判別されます。配列内の各エレメントは、列の始まりと終わりの位置を示す 2 バイトの整数の対になっています。ロケーションの対の最初のエレメントは、列が始まるファイル中のバイトです。 2 番目のエレメントは、列が終了するバイトです。ファイル中の特定の行にある最初のバイト位置は、バイト位置 1 と見なされます。列は、オーバーラップさせることができます。

SQL_METH_D

デフォルト。 DEL ファイルおよび IXF ファイルのインポートまたはロード時には、ファイルの最初の列は表の最初の列にロードまたはインポートされ、以下同様に続きます。 ASC ファイルのインポートまたはロード時には、ファイルの名前がファイル・タイプの修飾子 POSITIONSFILE に含まれているファイルの中で列が選択されます。エクスポート時には、デフォルト名が外部ファイル内の列に使用されます。

sqldcol 構造の *dcolnum* と *dcolname* フィールドはどちらも無視され、外部ファイルの列はそのままの順序で取り出されます。

外部ファイルからの列は、配列の中で複数回使用することができます。外部ファイルからの列をすべて使用する必要はありません。

言語構文:

C 構造

```
/* File: sqlutil.h */
/* Structure: SQLDCOL */
/* ... */
SQL_STRUCTURE sqldcol
{
  short           dcolmeth;
  short           dcolnum;
  struct sqldcoln dcolname[1];
};
/* ... */

/* File: sqlutil.h */
/* Structure: SQLDCOLN */
/* ... */
SQL_STRUCTURE sqldcoln
{
  short           dcolnlen;
  char            *dcolnptr;
};
/* ... */

/* File: sqlutil.h */
/* Structure: SQLLOCTAB */
/* ... */
SQL_STRUCTURE sqlloctab
```

```
{
  struct sqllocpair locpair[1];
};
/* ... */
```

```
/* File: sqlutil.h */
/* Structure: SQLLOCPAIR */
/* ... */
SQL_STRUCTURE sqllocpair
{
  short           begin_loc;
  short           end_loc;
};
/* ... */
```

COBOL 構造

```
* File: sqlutil.cbl
01 SQL-DCOLDATA.
    05 SQL-DCOLMETH           PIC S9(4) COMP-5.
    05 SQL-DCOLNUM            PIC S9(4) COMP-5.
    05 SQLDCOLN OCCURS 0 TO 255 TIMES DEPENDING ON SQL-DCOLNUM.
        10 SQL-DCOLNLEN       PIC S9(4) COMP-5.
        10 FILLER             PIC X(2).
        10 SQL-DCOLN-PTR      USAGE IS POINTER.
*
```

```
* File: sqlutil.cbl
01 SQL-LOCTAB.
    05 SQL-LOC-PAIR OCCURS 1 TIMES.
        10 SQL-BEGIN-LOC      PIC S9(4) COMP-5.
        10 SQL-END-LOC        PIC S9(4) COMP-5.
*
```

関連資料:

- 62 ページの『db2Export - エクスポート』
- 114 ページの『db2Import - インポート』
- 168 ページの『db2Load - ロード』

SQLE-ADDN-OPTIONS

この構造は、sqleaddn API に情報を渡すために使用されます。

表 39. SQLE-NODE-APPN 構造のフィールド

フィールド名	データ・タイプ	説明
SQLADDID	CHAR	SQLE_ADDOPTID_V51 に設定しなければならない「目印」値。
TBLSPACE_TYPE	sqluint32	追加されるノードのために使用される SYSTEM TEMPORARY 表スペース定義のタイプを指定します。値については、以下を参照してください。
TBLSPACE_NODE	SQL_PDB_NODE_TYPE	SYSTEM TEMPORARY 表スペース定義を取得するノード番号を指定します。このノード番号は、db2nodes.cfg ファイルに存在しなければならず、*tblspace_type* フィールドが SQLE_TABLESPACES_LIKE_NODE に設定される場合にのみ使用されます。

TBLSPACE_TYPE に有効な値 (sqlenv で定義) は、以下のとおりです。

SQLE_TABLESPACES_NONE
システム TEMPORARY 表スペースを作成しません。

SQLE_TABLESPACES_LIKE_NODE
システム TEMPORARY 表スペース用のコンテナーは、指定されたノード用のものと同じでなければなりません。

SQLE_TABLESPACES_LIKE_CATALOG
システム TEMPORARY 表スペース用のコンテナーは、各データベースのカタログ・ノード用のものと同じでなければなりません。

言語構文:

C 構造

```
/* File: sqlenv.h */
/* Structure: SQLE-ADDN-OPTIONS */
/* ... */
SQL_STRUCTURE sqle_addn_options
{
  char                  sqladdid[8];
  sqluint32             tblspace_type;
  SQL_PDB_NODE_TYPE     tblspace_node;
};
/* ... */
```

COBOL 構造

```
* File: sqlenv.cbl
01 SQLE-ADDN-OPTIONS.
    05 SQLADDID               PIC X(8).
    05 SQL-TBLSPACE-TYPE      PIC 9(9) COMP-5.
    05 SQL-TBLSPACE-NODE      PIC S9(4) COMP-5.
    05 FILLER                 PIC X(2).
*
```

関連資料:

- 332 ページの『sqleaddn - ノードの追加』

SQLE-CLIENT-INFO

この構造は、sqleseti API および sqleqryi API に情報を渡すために使用されます。

この構造では以下のものを指定します。

- 設定または照会する情報のタイプ
- 設定または照会するデータの長さ
- 以下のいずれかへのポインター
  - 設定中のデータを含む領域
  - 照会中のデータを含めるのに十分な長さの領域

アプリケーションは、以下のタイプの情報を指定できます。

- 設定または照会するクライアント・ユーザー ID。最大で 255 文字を設定することができますが、サーバーによってプラットフォーム固有の値に切り捨てられることがあります。

注: このユーザー ID は識別目的でのみ使用され、許可のために使われることはありません。

- 設定または照会するクライアント・ワークステーション名。最大で 255 文字を設定することができますが、サーバーによってプラットフォーム固有の値に切り捨てられることがあります。
- 設定または照会するクライアント・アプリケーション名。最大で 255 文字を設定することができますが、サーバーによってプラットフォーム固有の値に切り捨てられることがあります。
- 設定または照会するクライアント現行パッケージ・パス。最大で 255 文字を設定することができますが、サーバーによってプラットフォーム固有の値に切り捨てられることがあります。
- 設定または照会するプログラム ID。最大で 80 文字を設定できますが、サーバーによってプラットフォーム固有の値に切り捨てられることがあります。
- 設定または照会するクライアント会計情報ストリング。最大で 200 文字を設定できますが、サーバーによってプラットフォーム固有の値に切り捨てられることがあります。

注: 情報の設定には、sqlesact API を使用することもできます。ただし、いったん接続が確立されると、sqlesact では会計情報ストリングを変更できませんが、sqleseti では接続が確立された後でも、将来に会計情報を変更できます。

表 40. SQLE-CLIENT-INFO 構造のフィールド

フィールド名	データ・タイプ	説明
TYPE	sqlint32	設定のタイプ。
LENGTH	sqlint32	値の長さ。 sqleseti 呼び出しでは、長さの範囲はゼロからそのタイプに定義された最大長までになります。長さがゼロの場合、NULL 値になります。 sqleqryi 呼び出しでは長さが戻されますが、 *pValue* で示される領域はそのタイプの最大長を含めるのに十分な大きさでなければなりません。長さがゼロの場合、NULL 値になります。
PVALUE	ポインター	指定した値を含む、アプリケーション割り振りバッファーへのポインター。この値のデータ・タイプは、そのタイプ・フィールドによって異なります。

SQLE-CLIENT-INFO TYPE エレメントに有効な項目および各項目の説明が、以下の表に示されています。

表 41. 接続設定

種類	データ・タイプ	説明
SQLE_CLIENT_INFO_USERID	CHAR(255)	クライアントのユーザー ID。サーバーによってはこの値を切り捨てるものもあります。たとえば、DB2 for z/OS サーバーがサポートしている長さは最大 16 文字です。このユーザー ID は識別目的でのみ使用され、許可のために使われることはありません。
SQLE_CLIENT_INFO_WRKSTNNAME	CHAR(255)	クライアントのワークステーション名。サーバーによってはこの値を切り捨てるものもあります。たとえば、DB2 for z/OS サーバーがサポートしている長さは最大 18 文字です。
SQLE_CLIENT_INFO_APPLNAME	CHAR(255)	クライアントのアプリケーション名。サーバーによってはこの値を切り捨てるものもあります。たとえば、DB2 for z/OS サーバーがサポートしている長さは最大 32 文字です。
SQLE_CLIENT_INFO_PACKAGEPATH	CHAR(255)	クライアントの現行パッケージ・パス。サーバーによってはこの値を切り捨てるものもあります。たとえば、DB2 for z/OS V8 サーバーがサポートしている長さは最大 80 文字です。
SQLE_CLIENT_INFO_PROGRAMID	CHAR(80)	クライアントのプログラム ID。このエレメントを設定すると、DB2 UDB for z/OS バージョン 8 は、この ID を動的 SQL ステートメント・キャッシュに挿入されている任意のステートメントと関連付けます。このエレメントは、DB2 UDB for z/OS バージョン 8 にアクセスするアプリケーションでのみサポートされます。
SQLE_CLIENT_INFO_ACCTSTR	CHAR(200)	クライアントの会計情報ストリング。サーバーによってはこの値を切り捨てるものもあります。たとえば、DB2 for z/OS サーバーがサポートしている長さは最大 200 文字です。
SQLE_CLIENT_INFO_AUTOCOMMIT	CHAR(1)	クライアントの autocommit 設定。SQLE_CLIENT_AUTOCOMMIT_ON または SQLE_CLIENT_AUTOCOMMIT_OFF に設定できます。
注: これらのフィールド名は、C プログラミング言語用に定義されています。 FORTRAN および COBOL には、同じセマンティクスの類似した名前があります。		

言語構文:

C 構造

```
/* File: sqlenv.h */
/* Structure: SQLE-CLIENT-INFO */
/* ... */
SQL_STRUCTURE sqle_client_info
{
  unsigned short         type;
```

```
  unsigned short         length;
  char                   *pValue;
};
/* ... */
```

COBOL 構造

```
* File: sqlenv.cbl
01 SQLE-CLIENT-INFO.
    05 SQLE-CLIENT-INFO-ITEM OCCURS 4 TIMES.
        10 SQLE-CLIENT-INFO-TYPE   PIC S9(4) COMP-5.
        10 SQLE-CLIENT-INFO-LENGTH PIC S9(4) COMP-5.
        10 SQLE-CLIENT-INFO-VALUE  USAGE IS POINTER.
*
```

関連資料:

- 402 ページの『sqlesact - 会計情報ストリングの設定』
- 408 ページの『sqleseti - クライアント情報の設定』
- 399 ページの『sqleqryi - クライアント情報の照会』

SQLE-CONN-SETTING

この構造は、sqleqryc および sqlesetc API の接続設定のタイプおよび値を指定するために使用します。

表 42. SQLE-CONN-SETTING 構造のフィールド

フィールド名	データ・タイプ	説明
TYPE	SMALLINT	設定のタイプ。
VALUE	SMALLINT	設置値。

SQLE-CONN-SETTING TYPE エレメントに有効な項目および各項目の説明が、以下の表に示されています (sqlenv および sql で定義)。

表 43. 接続設定

種類	値	説明
SQL_CONNECT_TYPE	SQL_CONNECT_1	Type 1 CONNECT は、旧リリースの作業単位の意味 (リモート作業単位 (RUOW) に関する規則とも呼ばれる) ごとに 1 つのデータベースを適用します。
	SQL_CONNECT_2	Type 2 CONNECT は、DUOW の作業単位のセマンティクスごとに複数のデータベースをサポートします。
SQL_RULES	SQL_RULES_DB2	現行接続を、確立済みの (休止) 接続に切り替える SQL CONNECT ステートメントを使用可能にします。
	SQL_RULES_STD	SQL CONNECT ステートメントによる新規接続の確立だけを許可します。現行の接続から休止接続へ切り替えるには、 SQL SET CONNECTION ステートメントを使用する必要があります。

表 43. 接続設定 (続き)

種類	値	説明
SQL_DISCONNECT	SQL_DISCONNECT_EXPL	コミット時に SQL RELEASE ステートメントにより解放されるように明示的にマークされた接続を除去します。
	SQL_DISCONNECT_COND	コミット時にオープン状態の WITH HOLD カーソルを持たない接続、および SQL RELEASE ステートメントにより解放されるようにマークされた接続を切断します。
	SQL_DISCONNECT_AUTO	コミット時にすべての接続を切断します。
SQL_SYNCPOINT	SQL_SYNC_TWOPHASE	トランザクション・マネージャー (TM) に、このプロトコルをサポートするデータベース間で 2 フェーズ・コミットを調整するよう要求します。
	SQL_SYNC_ONEPHASE	1 フェーズ・コミットを使用して、複数のデータベース・トランザクション内のそれぞれのデータベースにより行われた作業をコミットします。単一の更新プログラムに、複数の読み取り動作という形を適用します。
	SQL_SYNC_NONE	1 フェーズ・コミットを使用して、行われた処理をコミットしますが、単一の更新プログラムに、複数の読み取り動作という形を適用しません。
SQL_MAX_NETBIOS_ CONNECTIONS	1 から 254	特定のアプリケーションで、NETBIOS アダプターを使用して確立できる同時接続の最大数を指定します。
SQL_DEFERRED_PREPARE	SQL_DEFERRED_PREPARE_NO	PREPARE ステートメントは、それが発行されたときに実行されます。
	SQL_DEFERRED_PREPARE_ YES	PREPARE ステートメントの実行は、対応する OPEN、DESCRIBE、または EXECUTE ステートメントが発行されるまで据え置かれます。 PREPARE ステートメントは、INTO 文節 (SQLDA がすぐに戻されることを必要とする) を使用する場合には据え置かれません。ただし、パラメーター・マーカーを使用しないカーソルについて PREPARE INTO ステートメントが発行された場合には、 PREPARE の実行時にカーソルを事前オープンすることによって、処理が最適化されます。

表 43. 接続設定 (続き)

種類	値	説明
	SQL_DEFERRED_PREPARE_ALL	パラメーター・マーカーを含む PREPARE INTO ステートメントが据え置かれる 点を除き、YES と同じです。 PREPARE INTO ステートメントにパラメーター・マーカーが含まれていない場合には、カーソルの事前オープンが依然として実行されます。 PREPARE ステートメントが SQLDA を戻すために INTO 文節を使用する場合、アプリケーションでは、 OPEN、 DESCRIBE、または EXECUTE ステートメントが発行され、戻されるまで、この SQLDA の内容を参照してはなりません。
SQL_CONNECT_NODE	0 から 999 の範囲、あるいはキーワード SQL_CONN_CATALOG_NODE。	接続が確立される先のノードを指定します。環境変数 **DB2NODE** の値をオーバーライドします。 たとえば、ノード 1、2、および 3 が定義されている場合、クライアントはこれらのノードの 1 つにアクセスできれば十分です。データベースを含むノード 1 だけがカタログされており、このパラメーターが 3 に設定されると、次の接続の試みは、ノード 1 での初期接続の後、ノード 3 での接続になります。
SQL_ATTACH_NODE	0 から 999 の範囲。	アタッチが確立される先のノードを指定します。環境変数 **DB2NODE** の値をオーバーライドします。 たとえば、ノード 1、2、および 3 が定義されている場合、クライアントはこれらのノードの 1 つにアクセスできれば十分です。データベースを含むノード 1 だけがカタログされており、このパラメーターが 3 に設定されると、次のアタッチの試みは、ノード 1 での初期アタッチの後、ノード 3 でのアタッチになります。
注: これらのフィールド名は、C プログラミング言語用に定義されています。 FORTRAN および COBOL には、同じセマンティクスの類似した名前があります。		

言語構文:

C 構造

```
/* File: sqlenv.h */
/* Structure: SQLE-CONN-SETTING */
/* ... */
```

```
SQL_STRUCTURE sqle_conn_setting
{
  unsigned short         type;
  unsigned short         value;
};
/* ... */
```

COBOL 構造

```
* File: sqlenv.cbl
01 SQLE-CONN-SETTING.
    05 SQLE-CONN-SETTING-ITEM OCCURS 7 TIMES.
        10 SQLE-CONN-TYPE  PIC S9(4) COMP-5.
        10 SQLE-CONN-VALUE PIC S9(4) COMP-5.
*
```

関連資料:

- 406 ページの『sqlesetc - クライアントの設定』
- 397 ページの『sqleqryc - クライアントの照会』

SQLE-NODE-APPC

この構造は、sqlectnd API の APPC ノードをカタログするために使用されます。

表 44. SQLE-NODE-APPC 構造のフィールド

フィールド名	データ・タイプ	説明
LOCAL_LU	CHAR(8)	ローカル LU 名。
PARTNER_LU	CHAR(8)	パートナー LU 名の別名。
MODE	CHAR(8)	モード。
注: この構造に渡された文字フィールドは、NULL 終了にするか、フィールドの長さいっぱいにまでブランクを埋め込む必要があります。		

言語構文:

C 構造

```
/* File: sqlenv.h */
/* Structure: SQLE-NODE-APPC */
/* ... */
SQL_STRUCTURE sqle_node_appc
{
  _SQLOLDCHAR    local_lu[SQL_LOCLU_SZ + 1];
  _SQLOLDCHAR    partner_lu[SQL_RMTLU_SZ + 1];
  _SQLOLDCHAR    mode[SQL_MODE_SZ + 1];
};
/* ... */
```

COBOL 構造

```
* File: sqlenv.cbl
01 SQL-NODE-APPC.
    05 LOCAL-LU              PIC X(8).
    05 FILLER                PIC X.
    05 PARTNER-LU            PIC X(8).
    05 FILLER                PIC X.
    05 TRANS-MODE            PIC X(8).
    05 FILLER                PIC X.
*
```

関連資料:

• 356 ページの『sqlectnd - ノードのカタログ』

SQLE-NODE-APPN

この構造は、sqlectnd API の APPN ノードをカタログするために使用されます。

表 45. SQLE-NODE-APPN 構造のフィールド

フィールド名	データ・タイプ	説明
NETWORKID	CHAR(8)	ネットワーク ID。
REMOTE_LU	CHAR(8)	リモート LU 名の別名。
LOCAL_LU	CHAR(8)	ローカル LU 名の別名。
MODE	CHAR(8)	モード。
注: この構造に渡された文字フィールドは、NULL 終了にするか、フィールドの長さいっぱいにまでブランクを埋め込む必要があります。		

言語構文:

C 構造

```
/* File: sqlenv.h */
/* Structure: SQLE-NODE-APPN */
/* ... */
SQL_STRUCTURE sqle_node_appn
{
  _SQLOLDCHAR    networkid[SQL_NETID_SZ + 1];
  _SQLOLDCHAR    remote_lu[SQL_RMTLU_SZ + 1];
  _SQLOLDCHAR    local_lu[SQL_LOCLU_SZ + 1];
  _SQLOLDCHAR    mode[SQL_MODE_SZ + 1];
};
/* ... */
```

COBOL 構造

```
* File: sqlenv.cbl
01 SQL-NODE-APPN.
    05 NETWORKID              PIC X(8).
    05 FILLER                 PIC X.
    05 REMOTE-LU              PIC X(8).
    05 FILLER                 PIC X.
    05 LOCAL-LU               PIC X(8).
    05 FILLER                 PIC X.
    05 TRANS-MODE             PIC X(8).
    05 FILLER                 PIC X.
*
```

関連資料:

• 356 ページの『sqlectnd - ノードのカタログ』

SQLE-NODE-CPIC

この構造は、sqlectnd API の CPIC ノードをカタログするために使用されます。

表 46. SQLE-NODE-CPIC 構造のフィールド

フィールド名	データ・タイプ	説明
SYM_DEST_NAME	CHAR(8)	リモート・パートナーのシンボリック宛先名。
SECURITY_TYPE	SMALLINT	セキュリティー・タイプ。
注: この構造に渡された文字フィールドは、NULL 終了にするか、フィールドの長さいっぱいにまでブランクを埋め込む必要があります。		

SECURITY_TYPE に有効な値 (sqlenv で定義) は、以下のとおりです。

SQL_CPIC_SECURITY_NONE

SQL_CPIC_SECURITY_SAME

SQL_CPIC_SECURITY_PROGRAM

言語構文:

C 構造

```
/* File: sqlenv.h */
/* Structure: SQLE-NODE-CPIC */
/* ... */
SQL_STRUCTURE sqle_node_cpic
{
  _SQLOLDCHAR    sym_dest_name[SQL_SYM_DEST_NAME_SZ+1];
  unsigned short security_type;
};
/* ... */
```

COBOL 構造

```
* File: sqlenv.cbl
01 SQL-NODE-CPIC.
    05 SYM-DEST-NAME          PIC X(8).
    05 FILLER                 PIC X.
    05 FILLER                 PIC X(1).
    05 SECURITY-TYPE          PIC 9(4) COMP-5.
*
```

関連資料:

- 356 ページの『sqlectnd - ノードのカタログ』

SQLE-NODE-IPXSPX

この構造は、sqlectnd API の IPX/SPX ノードをカタログするために使用されます。

表 47. SQLE-NODE-IPXSPX 構造のフィールド

フィールド名	データ・タイプ	説明
FILESERVER	CHAR(48)	DB2 サーバー・インスタンスが登録されている NetWare ファイル・サーバーの名前。

表 47. SQLE-NODE-IPXSPX 構造のフィールド (続き)

フィールド名	データ・タイプ	説明
OBJECTNAME	CHAR(48)	データベース・マネージャー・サーバー・インスタンスは、 NetWare ファイル・サーバー上ではオブジェクト *objectname* と表されます。サーバーの IPX/SPX インターネットワーク・アドレスはこのオブジェクトに保管され、このオブジェクトから検索されます。
注: この構造に渡された文字フィールドは、NULL 終了にするか、フィールドの長さいっぱいにまでブランクを埋め込む必要があります。		

言語構文:

C 構造

```
/* File: sqlenv.h */
/* Structure: SQLE-NODE-IPXSPX */
/* ... */
SQL_STRUCTURE sqle_node_ipxspx
{
  char           fileserver[SQL_FILESERVER_SZ+1];
  char           objectname[SQL_OBJECTNAME_SZ+1];
};
/* ... */
```

COBOL 構造

```
* File: sqlenv.cbl
01 SQL-NODE-IPXSPX.
    05 SQL-FILESERVER         PIC X(48).
    05 FILLER                 PIC X.
    05 SQL-OBJECTNAME         PIC X(48).
    05 FILLER                 PIC X.
*
```

関連資料:

• 356 ページの『sqlectnd - ノードのカタログ』

SQLE-NODE-LOCAL

この構造は、sqlectnd API のローカル・ノードをカタログするために使用されます。

表 48. SQLE-NODE-LOCAL 構造のフィールド

フィールド名	データ・タイプ	説明
INSTANCE_NAME	CHAR(8)	インスタンスの名前。
注: この構造に渡された文字フィールドは、NULL 終了にするか、フィールドの長さいっぱいにまでブランクを埋め込む必要があります。		

言語構文:

C 構造

```
/* File: sqlenv.h */
/* Structure: SQLE-NODE-LOCAL */
/* ... */
SQL_STRUCTURE sqle_node_local
```

```
{
  char           instance_name[SQL_INSTNAME_SZ+1];
};
/* ... */
```

COBOL 構造

```
* File: sqlenv.cbl
01 SQL-NODE-LOCAL.
    05 SQL-INSTANCE-NAME      PIC X(8).
    05 FILLER                 PIC X.
*
```

関連資料:

- 356 ページの『sqlectnd - ノードのカタログ』

SQLE-NODE-NETB

この構造は、sqlectnd API の NetBIOS ノードをカタログするために使用されます。

表 49. SQLE-NODE-NETB 構造のフィールド

フィールド名	データ・タイプ	説明
ADAPTER	SMALLINT	ローカル LAN アダプター。
REMOTE_NNAME	CHAR(8)	サーバー・インスタンス上のデータベース・マネージャー構成ファイルに保管されている、リモート・ワークステーションの *Nname*。
注: この構造に渡された文字フィールドは、NULL 終了にするか、フィールドの長さいっぱいにまでブランクを埋め込む必要があります。		

言語構文:

C 構造

```
/* File: sqlenv.h */
/* Structure: SQLE-NODE-NETB */
/* ... */
SQL_STRUCTURE sqle_node_netb
{
  unsigned short adapter;
  _SQLOLDCHAR    remote_nname[SQL_RMTLU_SZ + 1];
};
/* ... */
```

COBOL 構造

```
* File: sqlenv.cbl
01 SQL-NODE-NETB.
    05 ADAPTER                PIC 9(4) COMP-5.
    05 REMOTE-NNAME           PIC X(8).
    05 FILLER                 PIC X.
    05 FILLER                 PIC X(1).
*
```

関連資料:

- 356 ページの『sqlectnd - ノードのカタログ』

SQLE-NODE-NPIPE

この構造は、sqlectnd API の名前付きパイプ・ノードをカタログするために使用されます。

表 50. SQLE-NODE-NPIPE 構造のフィールド

フィールド名	データ・タイプ	説明
COMPUTERNAME	CHAR(15)	コンピューター名。
INSTANCE_NAME	CHAR(8)	インスタンスの名前。
注: この構造に渡された文字フィールドは、NULL 終了にするか、フィールドの長さいっぱいにまでブランクを埋め込む必要があります。		

言語構文:

C 構造

```
/* File: sqlenv.h */
/* Structure: SQLE-NODE-NPIPE */
/* ... */
SQL_STRUCTURE sqle_node_npipe
{
  char           computername[SQL_COMPUTERNAME_SZ+1];
  char           instance_name[SQL_INSTNAME_SZ+1];
};
/* ... */
```

COBOL 構造

```
* File: sqlenv.cbl
01 SQL-NODE-NPIPE.
    05 COMPUTERNAME           PIC X(15).
    05 FILLER                 PIC X.
    05 INSTANCE-NAME          PIC X(8).
    05 FILLER                 PIC X.
*
```

関連資料:

- 356 ページの『sqlectnd - ノードのカタログ』

SQLE-NODE-STRUCT

この構造は、sqlectnd API のノードをカタログするために使用されます。

表 51. SQLE-NODE-STRUCT 構造のフィールド

フィールド名	データ・タイプ	説明
STRUCT_ID	SMALLINT	構造 ID。
CODEPAGE	SMALLINT	コメント用コード・ページ。
COMMENT	CHAR(30)	ノードのオプション・コメント。
NODENAME	CHAR(8)	データベースが存在するノードのローカル名。
PROTOCOL	CHAR(1)	通信プロトコル・タイプ。
注: この構造に渡された文字フィールドは、NULL 終了にするか、フィールドの長さいっぱいにまでブランクを埋め込む必要があります。		

PROTOCOL に有効な値 (sqlenv で定義) は、以下のとおりです。

SQL_PROTOCOL_APPC

SQL_PROTOCOL_APPN

SQL_PROTOCOL_CPIC

SQL_PROTOCOL_IPXSPX

SQL_PROTOCOL_LOCAL

SQL_PROTOCOL_NETB

SQL_PROTOCOL_NPIPE

SQL_PROTOCOL_SOCKS

SQL_PROTOCOL_TCPIP

言語構文:

C 構造

```
/* File: sqlenv.h */
/* Structure: SQLE-NODE-STRUCT */
/* ... */
SQL_STRUCTURE sqle_node_struct
{
  unsigned short struct_id;
  unsigned short codepage;
  _SQLOLDCHAR    comment[SQL_CMT_SZ + 1];
  _SQLOLDCHAR    nodename[SQL_NNAME_SZ + 1];
  unsigned char  protocol;
};
/* ... */
```

COBOL 構造

```
* File: sqlenv.cbl
01 SQL-NODE-STRUCT.
    05 STRUCT-ID             PIC 9(4) COMP-5.
    05 CODEPAGE              PIC 9(4) COMP-5.
    05 COMMENT               PIC X(30).
    05 FILLER                PIC X.
    05 NODENAME              PIC X(8).
    05 FILLER                PIC X.
    05 PROTOCOL              PIC X.
    05 FILLER                PIC X(1).
*
```

関連資料:

- 356 ページの『sqlectnd - ノードのカタログ』

SQLE-NODE-TCPIP

この構造は、sqlectnd API の TCP/IP ノードをカタログするために使用されます。

注: TCP/IP SOCKS ノードをカタログするには、sqlectnd API を呼び出す前に、ノード・ディレクトリー構造の PROTOCOL タイプを *SQLE-NODE-STRUCT* 構造の SQL_PROTOCOL_SOCKS に設定してください。

表 52. SQLE-NODE-TCPIP 構造のフィールド

フィールド名	データ・タイプ	説明
HOSTNAME	CHAR(255)	DB2 サーバー・インスタンスが存在する TCP/IP ホストの名前。
SERVICE_NAME	CHAR(14)	DB2 サーバー・インスタンスの TCP/IP サービス名または関連するポート番号。
注: この構造に渡された文字フィールドは、NULL 終了にするか、フィールドの長さいっぱいにまでブランクを埋め込む必要があります。		

言語構文:

C 構造

```
/* File: sqlenv.h */
/* Structure: SQLE-NODE-TCPIP */
/* ... */
SQL_STRUCTURE sqle_node_tcpip
{
  _SQLOLDCHAR    hostname[SQL_HOSTNAME_SZ+1];
  _SQLOLDCHAR    service_name[SQL_SERVICE_NAME_SZ+1];
};
/* ... */
```

COBOL 構造

```
* File: sqlenv.cbl
01 SQL-NODE-TCPIP.
    05 HOSTNAME               PIC X(255).
    05 FILLER                 PIC X.
    05 SERVICE-NAME           PIC X(14).
    05 FILLER                 PIC X.
*
```

関連資料:

- 356 ページの『sqlectnd - ノードのカタログ』
- 471 ページの『SQLE-NODE-STRUCT』

SQLE-REG-NWBINDERY

この構造は NetWare ファイル・サーバーのバインダリー上で、 DB2 サーバーを登録 (sqleregs API を使用) または登録解除 (sqledreg API を使用) するために使用します。

表 53. SQLE-REG-NWBINDERY 構造のフィールド

フィールド名	データ・タイプ	説明
UID	CHAR(48)	NetWare ファイル・サーバーへのログインに使用されるユーザー ID。
PSWD	CHAR(128)	ユーザー ID の妥当性検査に使用されるパスワード。

言語構文:

C 構造

SQLE-REG-NWBINDERY

```
/* File: sqlenv.h */
/* Structure: SQLE-REG-NWBINDERY */
/* ... */
SQL_STRUCTURE sqle_reg_nwbindery
{
  char                  uid[SQL_NW_UID_SZ+1];
  unsigned short        reserved_len_1;
  char                  pswd[SQL_NW_PSWD_SZ+1];
  unsigned short        reserved_len_2;
};
/* ... */
```

COBOL 構造

```
* File: sqlenv.cbl
01 SQLE-REG-NWBINDERY.
    05 SQL-UID               PIC X(48).
    05 FILLER                PIC X.
    05 FILLER                PIC X(1).
    05 SQL-UID-LEN           PIC 9(4) COMP-5.
    05 SQL-PSWD              PIC X(128).
    05 FILLER                PIC X.
    05 FILLER                PIC X(1).
    05 SQL-PSWD-LEN          PIC 9(4) COMP-5.
*
```

関連資料:

- 400 ページの『sqleregs - 登録』
- 364 ページの『sqledreg - 登録解除』

SQLEDBTERRITORYINFO

この構造は、コード・セットおよび地域オプションを sqlecrea API に渡すために使用されます。

表 54. SQLEDBTERRITORYINFO 構造のフィールド

フィールド名	データ・タイプ	説明
SQLDBCODESET	CHAR(9)	データベース・コード・セット。
SQLDBLOCALE	CHAR(5)	データベース・テリトリー。

言語構文:

C 構造

```
/* File: sqlenv.h */
/* Structure: SQLEDBTERRITORYINFO */
/* ... */
SQL_STRUCTURE sqledbterritoryinfo
{
  char                  sqldbcodeset[SQL_CODESET_LEN + 1];
  char                  sqldblocale[SQL_LOCALE_LEN + 1];
};
/* ... */
```

COBOL 構造

```
* File: sqlenv.cbl
01 SQLEDBTERRITORYINFO.
    05 SQLDBCODESET          PIC X(9).
```

```
    05 FILLER                PIC X.
    05 SQLDBLOCALE           PIC X(5).
    05 FILLER                PIC X.
*
```

関連資料:

- 348 ページの『sqlecrea - データベースの作成』

SQLEDBDESC

データベース記述ブロック (SQLEDBDESC) 構造は、 sqlecrea API への呼び出し中に、データベース属性の永続値を指定するために使用できます。これらの属性には、データベース・コメント、照合シーケンス、および表スペース定義が含まれます。

表 55. SQLEDBDESC 構造のフィールド

フィールド名	データ・タイプ	説明
SQLDBDID	CHAR(8)	記憶域ダンプ用構造 ID および「目印」。値 SQLE_DBDESC_2 (sqlenv で定義) で初期設定されなければならない 8 バイトのストリングです。このフィールドの内容に対して、バージョン制御を目的とした妥当性の検査がなされます。
SQLDBCCP	INTEGER	データベース・コメントのコード・ページ。この値は、データベース・マネージャーによって使用されなくなりました。
SQLDBCSS	INTEGER	データベース照合シーケンスのソースを示す値。値については、以下を参照してください。 **注:** データベースの照合シーケンスが (バイナリー照合シーケンスをインプリメントする) IDENTITY で あることを指定するには、 SQL_CS_NONE を指定します。 SQL_CS_NONE はデフォルトです。
SQLDBUDC	CHAR(256)	このフィールドの *n* 番目のバイトには、基礎となる 10 進表記がデータベースのコード・ページ中で *n* であるコード・ポイントのソートの重みが含まれます。 SQLDBCSS が SQL_CS_USER と等しくない場合、このフィールドは無視されます。
SQLDBCMT	CHAR(30)	データベースのコメント。
SQLDBSGP	INTEGER	予約フィールド。使用されなくなりました。
SQLDBNSG	SHORT	データベース内で作成されるファイル・セグメントの数を示す値。このフィールドの最小値は 1 であり、最大値は 256 です。 -1 の値が提供されると、このフィールドはデフォルトで 1 になります。 **注:** ゼロに設定された SQLDBNSG は、バージョン 1 との互換性のためのデフォルトを生成します。
SQLTSEXT	INTEGER	データベース中にある各表スペースのデフォルトのエクステント・サイズを示す値 (4KB ページ単位)。このフィールドの最小値は 2 であり、最大値は 256 です。 -1 の値が提供されると、このフィールドはデフォルトで 32 になります。
SQLCATTS	ポインター	カタログ表スペースを定義する表スペース記述制御ブロック SQLETSDESC を指すポインター。 NULL である場合、 SQLTSEXT および SQLDBNSG の値に基づくデフォルトのカタログ表スペースが作成されます。

表 55. SQLEDBDESC 構造のフィールド (続き)

フィールド名	データ・タイプ	説明
SQLUSRTS	ポインター	ユーザー表スペースを定義する表スペース記述制御ブロック SQLETSDESC を指すポインター。 NULL である場合、SQLTSEXT および SQLDBNSG の値に基づくデフォルトのユーザー表スペースが作成されます。
SQLTMPTS	ポインター	SYSTEM TEMPORARY 表スペースを定義する表スペース記述制御ブロック SQLETSDESC を指すポインター。 NULL である場合、SQLTSEXT および SQLDBNSG の値に基づくデフォルトの SYSTEM TEMPORARY 表スペースが作成されます。

表スペース記述ブロック構造 (SQLETSDESC) は、3 つの初期表スペースのいずれかの属性を指定するのに使用されます。

表 56. SQLETSDESC 構造のフィールド

フィールド名	データ・タイプ	説明
SQLTSDID	CHAR(8)	記憶域ダンプ用構造 ID および「目印」。値 SQLE_DBTSDESC_2 (sqlenv で定義) で初期設定されなければならない 8 バイトのストリングです。このフィールドの内容に対して、バージョン制御を目的とした妥当性の検査がなされます。
SQLEXTNT	INTEGER	表スペースのエクステント・サイズ (4KB ページ単位)。 -1 の値が提供されると、このフィールドはデフォルトで *dft_extent_sz* 構成パラメーターの現行値になります。
SQLPRFTC	INTEGER	表スペースのプリフェッチ・サイズ (4KB ページ単位)。 -1 の値が提供されると、このフィールドはデフォルトで *dft_prefetch_sz* 構成パラメーターの現行値になります。
SQLPOVHD	DOUBLE	表スペース I/O オーバーヘッド (ミリ秒)。 -1 の値が提供されると、このフィールドは、デフォルトで、将来のリリースで変更される可能性がある内部データベース・マネージャー値 (現在 24.1 ミリ秒) になります。
SQLTRFRT	DOUBLE	表スペースの I/O 転送速度 (ミリ秒)。 -1 の値が提供されると、このフィールドは、デフォルトで、将来のリリースで変更される可能性がある内部データベース・マネージャー値 (現在 0.9 ミリ秒) になります。
SQLTSTYP	CHAR(1)	表スペースがシステムによって管理されるか、データベースによって管理されるかを示します。値については、以下を参照してください。
SQLCCNT	SMALLINT	表スペースに割り当てられるコンテナーの数。後続の SQLCTYPE/SQLCSIZE/SQLCLEN/SQLCONTR 値の数を示します。
CONTAINR	配列	*sqlccnt* 個の *SQLETSCDESC* 構造の配列。

表 57. SQLETSCDESC 構造のフィールド

フィールド名	データ・タイプ	説明
SQLCTYPE	CHAR(1)	このコンテナーのタイプを識別します。値については、以下を参照してください。
SQLCSIZE	INTEGER	*SQLCONTR* で識別されたコンテナーのサイズ (4KB ページ単位)。 *SQLTSTYP* が SQL_TBS_TYP_DMS に設定されたときのみ有効です。
SQLCLEN	SMALLINT	後続の *SQLCONTR* 値の長さ。

表 57. SQLETSCDESC 構造のフィールド (続き)

フィールド名	データ・タイプ	説明
SQLCONTR	CHAR(256)	コンテナー・ストリング。

SQLDBCSS に有効な値 (sqlenv で定義) は、以下のとおりです。

SQL_CS_SYSTEM
データベース・テリトリーに基づいた照合シーケンス。

SQL_CS_USER
照合シーケンスは、ユーザーが提供する 256 バイト重み表によって指定されます。表内のそれぞれの重みの長さは 1 バイトです。

SQL_CS_NONE
照合シーケンスは IDENTITY、つまりバイナリー・コード・ポイント順です。

SQLE_CS_COMPATABILITY
バージョン 5 以前の照合シーケンスを使用。

SQL_CS_SYSTEM_NLSCHAR
文字タイプの比較ルーチンに NLS バージョンを使用したシステムの照合シーケンス Thai タイ語 TIS620-1 データベースを作成する場合に限り、この値を指定できます。

SQL_CS_USER_NLSCHAR
照合シーケンスは、ユーザーが提供する 256 バイト重み表によって指定されます。表内のそれぞれの重みの長さは 1 バイトです。 Thai タイ語 TIS620-1 データベースを作成する場合に限り、この値を指定できます。

SQL_CS_IDENTITY_16BIT
(Unicode Consortium の Web サイト www.unicode.org から入手可能な) Unicode Technical Report #26 によって指定された、 CESU-8 (Compatibility Encoding Scheme for UTF-16: 8-Bit) 照合シーケンス。 Unicode データベースを作成する場合に限り、この値を指定できます。

SQL_CS_UCA400_NO
Unicode 標準バージョン 4.00 に基づく UCA (Unicode Collation Algorithm) 照合シーケンス (正規化を暗黙的にオン に設定)。 UCA についての詳細は、 (Unicode Consortium の Web サイト www.unicode.org から入手可能な) Unicode Technical Standard #10 に記載されています。 Unicode データベースを作成する場合に限り、この値を指定できます。

SQL_CS_UCA400_LTH
Unicode 標準バージョン 4.00 に基づく UCA (Unicode Collation Algorithm) 照合シーケンス (Royal Thai Dictionary 順に従って、すべてのタイ語文字をソート)。 UCA についての詳細は、 (Unicode Consortium の Web サイト www.unicode.org から入手可能な) Unicode Technical Standard #10 に記載されています。 Unicode データベースを作成する場合に限り、この値を指定できます。

SQLTSTYP に有効な値 (sqlenv で定義) は、以下のとおりです。

SQL_TBS_TYP_SMS
システムにより管理

SQL_TBS_TYP_DMS
データベースにより管理

SQLCTYPE に有効な値 (sqlenv で定義) は、以下のとおりです。

SQL_TBSC_TYP_DEV
装置。 *SQLTSTYP* = SQL_TBS_TYP_DMS の場合のみ有効。

SQL_TBSC_TYP_FILE
ファイル。 *SQLTSTYP* = SQL_TBS_TYP_DMS の場合のみ有効。

SQL_TBSC_TYP_PATH
パス (ディレクトリー)。 *SQLTSTYP* = SQL_TBS_TYP_SMS の場合のみ有効。

言語構文:

C 構造

```
/* File: sqlenv.h */
/* Structure: SQLEDBDESC */
/* ... */
SQL_STRUCTURE sqledbdesc
{
  _SQLOLDCHAR    sqldbdid[8];
  sqlint32       sqldbccp;
  sqlint32       sqldbcss;
  unsigned char  sqldbudc[SQL_CS_SZ];
  _SQLOLDCHAR    sqldbcmt[SQL_CMT_SZ+1];
  _SQLOLDCHAR    pad[1];
  sqluint32      sqldbsgp;
  short          sqldbnsg;
  char           pad2[2];
  sqlint32       sqltsext;
  struct SQLETSDESC *sqlcatts;
  struct SQLETSDESC *sqlusrts;
  struct SQLETSDESC *sqltmpts;
};
/* ... */
```

```
/* File: sqlenv.h */
/* Structure: SQLETSDESC */
/* ... */
SQL_STRUCTURE SQLETSDESC
{
  char           sqltsdid[8];
  sqlint32       sqlextnt;
  sqlint32       sqlprftc;
  double         sqlpovhd;
  double         sqltrfrt;
  char           sqltstyp;
  char           pad1;
  short          sqlccnt;
  struct SQLETSCDESC containr[1];
};
/* ... */
```

```
/* File: sqlenv.h */
/* Structure: SQLETSCDESC */
/* ... */
SQL_STRUCTURE SQLETSCDESC
{
  char           sqlctype;
```

```
  char           pad1[3];
  sqlint32       sqlcsize;
  short          sqlclen;
  char           sqlcontr[SQLB_MAX_CONTAIN_NAME_SZ];
  char           pad2[2];
};
/* ... */
```

COBOL 構造

```
* File: sqlenv.cbl
01 SQLEDBDESC.
    05 SQLDBDID              PIC X(8).
    05 SQLDBCCP              PIC S9(9) COMP-5.
    05 SQLDBCSS              PIC S9(9) COMP-5.
    05 SQLDBUDC              PIC X(256).
    05 SQLDBCMT              PIC X(30).
    05 FILLER                PIC X.
    05 SQL-PAD               PIC X(1).
    05 SQLDBSGP              PIC 9(9) COMP-5.
    05 SQLDBNSG              PIC S9(4) COMP-5.
    05 SQL-PAD2              PIC X(2).
    05 SQLTSEXT              PIC S9(9) COMP-5.
    05 SQLCATTS              USAGE IS POINTER.
    05 SQLUSRTS              USAGE IS POINTER.
    05 SQLTMPTS              USAGE IS POINTER.
*

* File: sqletsd.cbl
01 SQLETSDESC.
    05 SQLTSDID              PIC X(8).
    05 SQLEXTNT              PIC S9(9) COMP-5.
    05 SQLPRFTC              PIC S9(9) COMP-5.
    05 SQLPOVHD              USAGE COMP-2.
    05 SQLTRFRT              USAGE COMP-2.
    05 SQLTSTYP              PIC X.
    05 SQL-PAD1              PIC X.
    05 SQLCCNT               PIC S9(4) COMP-5.
    05 SQL-CONTAINR OCCURS 001 TIMES.
        10 SQLCTYPE          PIC X.
        10 SQL-PAD1          PIC X(3).
        10 SQLCSIZE          PIC S9(9) COMP-5.
        10 SQLCLEN           PIC S9(4) COMP-5.
        10 SQLCONTR          PIC X(256).
        10 SQL-PAD2          PIC X(2).
*

* File: sqlenv.cbl
01 SQLETSCDESC.
    05 SQLCTYPE              PIC X.
    05 SQL-PAD1              PIC X(3).
    05 SQLCSIZE              PIC S9(9) COMP-5.
    05 SQLCLEN               PIC S9(4) COMP-5.
    05 SQLCONTR              PIC X(256).
    05 SQL-PAD2              PIC X(2).
*
```

関連概念:

- 「*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』

関連資料:

- 348 ページの『sqlecrea - データベースの作成』

SQLENINFO

この構造は、sqlengne API への呼び出し後に情報を戻します。

表 58. SQLENINFO 構造のフィールド

フィールド名	データ・タイプ	説明
NODENAME	CHAR(8)	NetBIOS プロトコルに使用されます。データベースが存在するノードの *nname* (システム・ディレクトリー内でのみ有効)。
LOCAL_LU	CHAR(8)	APPN プロトコルに使用されます。ローカル LU。
PARTNER_LU	CHAR(8)	APPN プロトコルに使用されます。パートナー LU。
MODE	CHAR(8)	APPN プロトコルに使用されます。伝送サービス・モード。
COMMENT	CHAR(30)	ノードに関するコメント。
COM_CODEPAGE	SMALLINT	コメントのコード・ページ。このフィールドは、データベース・マネージャーによって使用されなくなりました。
ADAPTER	SMALLINT	NetBIOS プロトコルに使用されます。ローカル・ネットワーク・アダプター。
NETWORKID	CHAR(8)	APPN プロトコルに使用されます。ネットワーク ID。
PROTOCOL	CHAR(1)	通信プロトコル。
SYM_DEST_NAME	CHAR(8)	APPC プロトコルに使用されます。シンボリック宛先名。
SECURITY_TYPE	SMALLINT	APPC プロトコルに使用されます。セキュリティー・タイプ。値については、以下を参照してください。
HOSTNAME	CHAR(255)	TCP/IP プロトコル用に使用されます。 DB2 サーバー・インスタンスが存在する TCP/IP ホストの名前。
SERVICE_NAME	CHAR(14)	TCP/IP プロトコルに使用されます。 DB2 サーバー・インスタンスの TCP/IP サービス名または関連するポート番号。
FILESERVER	CHAR(48)	IPX/SPX プロトコル用に使用されます。 DB2 サーバー・インスタンスが登録されている NetWare ファイル・サーバーの名前。
OBJECTNAME	CHAR(48)	データベース・マネージャー・サーバー・インスタンスは、 NetWare ファイル・サーバー上ではオブジェクト *objectname* と表されます。サーバーの IPX/SPX インターネットワーク・アドレスはこのオブジェクトに保管され、このオブジェクトから検索されます。
INSTANCE_NAME	CHAR(8)	ローカルおよび NPIPE プロトコルに使用されます。サーバー・インスタンスの名前。
COMPUTERNAME	CHAR(15)	NPIPE プロトコルに使用されます。サーバー・ノードのコンピューター名。
SYSTEM_NAME	CHAR(21)	リモート・サーバーの DB2 システム名。
REMOTE_INSTNAME	CHAR(8)	DB2 サーバー・インスタンスの名前。
CATALOG_NODE_TYPE	CHAR	カタログ・ノード・タイプ。

表 58. SQLENINFO 構造のフィールド (続き)

フィールド名	データ・タイプ	説明
OS_TYPE	UNSIGNED SHORT	サーバーのオペレーティング・システムを識別します。
注: 戻される各文字フィールドは、フィールドの長さに達するまでブランクが埋め込まれます。		

SECURITY_TYPE に有効な値 (sqlenv で定義) は、以下のとおりです。

SQL_CPIC_SECURITY_NONE

SQL_CPIC_SECURITY_SAME

SQL_CPIC_SECURITY_PROGRAM

言語構文:

C 構造

```
/* File: sqlenv.h */
/* Structure: SQLENINFO */
/* ... */
SQL_STRUCTURE sqleninfo
{
  _SQLOLDCHAR    nodename[SQL_NNAME_SZ];
  _SQLOLDCHAR    local_lu[SQL_LOCLU_SZ];
  _SQLOLDCHAR    partner_lu[SQL_RMTLU_SZ];
  _SQLOLDCHAR    mode[SQL_MODE_SZ];
  _SQLOLDCHAR    comment[SQL_CMT_SZ];
  unsigned short com_codepage;
  unsigned short adapter;
  _SQLOLDCHAR    networkid[SQL_NETID_SZ];
  _SQLOLDCHAR    protocol;
  _SQLOLDCHAR    sym_dest_name[SQL_SYM_DEST_NAME_SZ];
  unsigned short security_type;
  _SQLOLDCHAR    hostname[SQL_HOSTNAME_SZ];
  _SQLOLDCHAR    service_name[SQL_SERVICE_NAME_SZ];
  char           fileserver[SQL_FILESERVER_SZ];
  char           objectname[SQL_OBJECTNAME_SZ];
  char           instance_name[SQL_INSTNAME_SZ];
  char           computername[SQL_COMPUTERNAME_SZ];
  char           system_name[SQL_SYSTEM_NAME_SZ];
  char           remote_instname[SQL_REMOTE_INSTNAME_SZ];
  _SQLOLDCHAR    catalog_node_type;
  unsigned short os_type;
};
/* ... */
```

COBOL 構造

```
* File: sqlenv.cbl
01 SQLENINFO.
    05 SQL-NODE-NAME          PIC X(8).
    05 SQL-LOCAL-LU           PIC X(8).
    05 SQL-PARTNER-LU         PIC X(8).
    05 SQL-MODE               PIC X(8).
    05 SQL-COMMENT            PIC X(30).
    05 SQL-COM-CODEPAGE       PIC 9(4) COMP-5.
    05 SQL-ADAPTER            PIC 9(4) COMP-5.
    05 SQL-NETWORKID          PIC X(8).
    05 SQL-PROTOCOL           PIC X.
    05 SQL-SYM-DEST-NAME      PIC X(8).
    05 FILLER                 PIC X(1).
    05 SQL-SECURITY-TYPE      PIC 9(4) COMP-5.
    05 SQL-HOSTNAME           PIC X(255).
```

```
    05 SQL-SERVICE-NAME       PIC X(14).
    05 SQL-FILESERVER         PIC X(48).
    05 SQL-OBJECTNAME         PIC X(48).
    05 SQL-INSTANCE-NAME      PIC X(8).
    05 SQL-COMPUTERNAME       PIC X(15).
    05 SQL-SYSTEM-NAME        PIC X(21).
    05 SQL-REMOTE-INSTNAME    PIC X(8).
    05 SQL-CATALOG-NODE-TYPE  PIC X.
    05 SQL-OS-TYPE            PIC 9(4) COMP-5.
*
```

関連資料:

• 393 ページの『sqlengne - ノード・ディレクトリー次項目の入手』

SQLFUPD

この構造は、データベース構成ファイルおよびデータベース・マネージャー構成ファイルについての情報を渡します。

表 59. SQLFUPD 構造のフィールド

フィールド名	データ・タイプ	説明
TOKEN	UINT16	戻すかまたは更新するための構成値を指定します。
PTRVALUE	ポインター	*TOKEN* によって指定されたデータを保持する、アプリケーション割り振りバッファーへのポインター。

トークン ・エレメントについての有効なデータ・タイプは、以下のとおりです。

Uint16 符号なし、2 バイトの整数

Sint16 符号付き、2 バイトの整数

Uint32 符号なし、4 バイトの整数

Sint32 符号付き、4 バイトの整数

Uint64 符号なし、8 バイトの整数

float 4 バイトの浮動小数点数

char(*n*) 長さ *n* のストリング (NULL 終了文字は含みません)。

SQLFUPD トークン・エレメントの有効な項目を以下に示します。

表 60. 更新可能なデータベース構成パラメーター

パラメーター名	トークン	トークン値	データ・タイプ
app_ctl_heap_sz	SQLF_DBTN_APP_CTL_HEAP_SZ	500	Uint16
applheapsz	SQLF_DBTN_APPLHEAPSZ	51	Uint16
appgroup_mem_sz	SQLF_DBTN_APPGROUP_MEM_SZ	800	Uint32
audit_buf_sz	SQLF_KTN_AUDIT_BUF_SZ	312	Sint32
autorestart	SQLF_DBTN_AUTO_RESTART	25	Uint16
avg_appls	SQLF_DBTN_AVG_APPLS	47	Uint16
blk_log_dsk_ful	SQLF_DBTN_BLK_LOG_DSK_FUL	804	Uint16
catalogcache_sz	SQLF_DBTN_CATALOGCACHE_SZ	56	Sint32
chngpgs_thresh	SQLF_DBTN_CHNGPGS_THRESH	38	Uint16

表 60. 更新可能なデータベース構成パラメーター (続き)

パラメーター名	トークン	トークン値	データ・タイプ
database_memory	SQLF_DBTN_DATABASE_MEMORY	803	Uint64
dbheap	SQLF_DBTN_DB_HEAP	58	Uint64
dft_degree	SQLF_DBTN_DFT_DEGREE	301	Sint32
dft_extent_sz	SQLF_DBTN_DFT_EXTENT_SZ	54	Uint32
dft_loadrec_ses	SQLF_DBTN_DFT_LOADREC_SES	42	Sint16
dft_prefetch_sz	SQLF_DBTN_DFT_PREFETCH_SZ	40	Sint16
dft_queryopt	SQLF_DBTN_DFT_QUERYOPT	57	Sint32
dft_refresh_age	SQLF_DBTN_DFT_REFRESH_AGE	702	char(22)
dft_sqlmathwarn	SQLF_DBTN_DFT_SQLMATHWARN	309	Sint16
dir_obj_name	SQLF_DBTN_DIR_OBJ_NAME	46	char(255)
discover	SQLF_DBTN_DISCOVER	308	Uint16
dl_expint	SQLF_DBTN_DL_EXPINT	350	Sint32
dl_num_copies	SQLF_DBTN_DL_NUM_COPIES	351	Uint16
dl_time_drop	SQLF_DBTN_DL_TIME_DROP	353	Uint16
dl_token	SQLF_DBTN_DL_TOKEN	602	char(10)
dl_upper	SQLF_DBTN_DL_UPPER	603	Sint16
dl_w__expint	SQLF_DBTN_DL_WT_IEXPINT	354	Sint32
dlchktime	SQLF_DBTN_DLCHKTIME	9	Uint32
dyn_query_mgmt	SQLF_DBTN_DYN_QUERY_MGMT	604	Uint16
estore_seg_sz	SQLF_DBTN_ESTORE_SEG_SZ	303	Sint32
groupheap_ratio	SQLF_DBTN_GROUPHEAP_RATIO	801	Uint16
indexrec[a]	SQLF_DBTN_INDEXREC	30	Uint16
indexsort	SQLF_DBTN_INDEXSORT	35	Uint16
locklist	SQLF_DBTN_LOCK_LIST	704	Uint64
locktimeout	SQLF_DBTN_LOCKTIMEOUT	34	Sint16
logbufsz	SQLF_DBTN_LOGBUFSZ	33	Uint16
logfilsiz	SQLF_DBTN_LOGFIL_SIZ	92	Uint32
logprimary	SQLF_DBTN_LOGPRIMARY	16	Uint16
logretain[b]	SQLF_DBTN_LOG_RETAIN	23	Uint16
logsecond	SQLF_DBTN_LOGSECOND	17	Uint16
maxappls	SQLF_DBTN_MAXAPPLS	6	Uint16
maxfilop	SQLF_DBTN_MAXFILOP	3	Uint16
maxlocks	SQLF_DBTN_MAXLOCKS	15	Uint16
maxlog	SQLF_DBTN_MAX_LOG	807	Uint16
mincommit	SQLF_DBTN_MINCOMMIT	32	Uint16
mirrorlogpath	SQLF_DBTN_MIRRORLOGPATH	806	char(242)
newlogpath	SQLF_DBTN_NEWLOGPATH	20	char(242)
num_db_backups	SQLF_DBTN_NUM_DB_BACKUPS	601	Uint16
num_estore_segs	SQLF_DBTN_NUM_ESTORE_SEGS	304	Sint32
num_freqvalues	SQLF_DBTN_NUM_FREQVALUES	36	Uint16
num_iocleaners	SQLF_DBTN_NUM_IOCLEANERS	37	Uint16
num_ioservers	SQLF_DBTN_NUM_IOSERVERS	39	Uint16
numlogspan	SQLF_DBTN_NUM_LOG_SPAN	808	Uint16
num_quantiles	SQLF_DBTN_NUM_QUANTILES	48	Uint16
overflowlogpath	SQLF_DBTN_OVERFLOWLOGPATH	805	char(242)

表 60. 更新可能なデータベース構成パラメーター (続き)

パラメーター名	トークン	トークン値	データ・タイプ
pckcachesz	SQLF_DBTN_PCKCACHE_SZ	505	Uint32
rec_his_retentn	SQLF_DBTN_REC_HIS_RETENTN	43	Sint16
seqdetect	SQLF_DBTN_SEQDETECT	41	Uint16
sheapthres_shr	SQLF_DBTN_SHEAPTHRES_SHR	802	Uint32
softmax	SQLF_DBTN_SOFTMAX	5	Uint16
sortheap	SQLF_DBTN_SORT_HEAP	52	Uint32
stat_heap_sz	SQLF_DBTN_STAT_HEAP_SZ	45	Uint32
stmtheap	SQLF_DBTN_STMTHEAP	53	Uint16
trackmod	SQLF_DBTN_TRACKMOD	703	Uint16
tsm_mgmtclass	SQLF_DBTN_TSM_MGMTCLASS	307	char(30)
tsm_nodename	SQLF_DBTN_TSM_NODENAME	306	char(64)
tsm_owner	SQLF_DBTN_TSM_OWNER	305	char(64)
tsm_password	SQLF_DBTN_TSM_PASSWORD	501	char(64)
userexit	SQLF_DBTN_USER_EXIT	24	Uint16
util_heap_sz	SQLF_DBTN_UTIL_HEAP_SZ	55	Uint32

[a] 有効な値 (`sqlutil.h` で定義) は、以下のとおりです。

```
SQLF_INX_REC_SYSTEM (0)
SQLF_INX_REC_REFERENCE (1)
SQLF_INX_REC_RESTART (2)
```

[b] 有効な値 (`sqlutil.h` で定義) は、以下のとおりです。

```
SQLF_LOGRETAIN_NO (0)
SQLF_LOGRETAIN_RECOVERY (1)
SQLF_LOGRETAIN_CAPTURE (2)
```

表 61. 更新不能のデータベース構成パラメーター

パラメーター名	トークン	トークン値	データ・タイプ
backup_pending	SQLF_DBTN_BACKUP_PENDING	112	Uint16
codepage	SQLF_DBTN_CODEPAGE	101	Uint16
codeset	SQLF_DBTN_CODESET	120	char(9)[a]
collate_info	SQLF_DBTN_COLLATE_INFO	44	char(260)
country	SQLF_DBTN_COUNTRY	100	Uint16
database_consistent	SQLF_DBTN_CONSISTENT	111	Uint16
database_level	SQLF_DBTN_DATABASE_LEVEL	124	Uint16
log_retain_status	SQLF_DBTN_LOG_RETAIN_STATUS	114	Uint16
loghead	SQLF_DBTN_LOGHEAD	105	char(12)
logpath	SQLF_DBTN_LOGPATH	103	char(242)
multipage_alloc	SQLF_DBTN_MULTIPAGE_ALLOC	506	Uint16
numsegs	SQLF_DBTN_NUMSEGS	122	Uint16
release	SQLF_DBTN_RELEASE	102	Uint16
restore_pending	SQLF_DBTN_RESTORE_PENDING	503	Uint16
rollfwd_pending	SQLF_DBTN_ROLLFWD_PENDING	113	Uint16
territory	SQLF_DBTN_TERRITORY	121	char(5)[b]
user_exit_status	SQLF_DBTN_USER_EXIT_STATUS	115	Uint16

表 61. 更新不能のデータベース構成パラメーター (続き)

パラメーター名	トークン	トークン値	データ・タイプ
[a] HP-UX および Solaris オペレーティング環境 上では char(17)。 [b] HP-UX および Solaris オペレーティング環境 上では char(33)。			

SQLFUPD トークン・エレメントの有効な項目を以下に示します。

表 62. 更新可能な データベース・マネージャー 構成パラメーター

パラメーター名	トークン	トークン値	データ・タイプ
agent_stack_sz	SQLF_KTN_AGENT_STACK_SZ	61	Uint16
agentpri	SQLF_KTN_AGENTPRI	26	Sint16
aslheapsz	SQLF_KTN_ASLHEAPSZ	15	Uint32
audit_buf_sz	SQLF_KTN_AUDIT_BUF_SZ	312	Sint32
authentication[a]	SQLF_KTN_AUTHENTICATION	78	Uint16
backbufsz	SQLF_KTN_BACKBUFSZ	18	Uint32
catalog_noauth	SQLF_KTN_CATALOG_NOAUTH	314	Uint16
comm_bandwidth	SQLF_KTN_COMM_BANDWIDTH	307	float
conn_elapse	SQLF_KTN_CONN_ELAPSE	508	Uint16
cpuspeed	SQLF_KTN_CPUSPEED	42	float
datalinks	SQLF_KTN_DATALINKS	603	Sint16
dft_account_str	SQLF_KTN_DFT_ACCOUNT_STR	28	char(25)
dft_client_adpt	SQLF_KTN_DFT_CLIENT_ADPT	82	Uint16
dft_client_comm	SQLF_KTN_DFT_CLIENT_COMM	77	char(31)
dft_monswitches	SQLF_KTN_DFT_MONSWITCHES[b]	29	Uint16
dft_mon_bufpool	SQLF_KTN_DFT_MON_BUFPOOL	33	Uint16
dft_mon_lock	SQLF_KTN_DFT_MON_LOCK	34	Uint16
dft_mon_sort	SQLF_KTN_DFT_MON_SORT	35	Uint16
dft_mon_stmt	SQLF_KTN_DFT_MON_STMT	31	Uint16
dft_mon_table	SQLF_KTN_DFT_MON_TABLE	32	Uint16
dft_mon_uow	SQLF_KTN_DFT_MON_UOW	30	Uint16
dftdbpath	SQLF_KTN_DFTDBPATH	27	char(215)
diaglevel	SQLF_KTN_DIAGLEVEL	64	Uint16
diagpath	SQLF_KTN_DIAGPATH	65	char(215)
dir_cache	SQLF_KTN_DIR_CACHE	40	Uint16
discover[c]	SQLF_KTN_DISCOVER	304	Uint16
discover_comm	SQLF_KTN_DISCOVER_COMM	305	char(35)
discover_inst	SQLF_KTN_DISCOVER_INST	308	Uint16
dos_rqrioblk	SQLF_KTN_DOS_RQRIOBLK	72	Uint16
fcm_num_buffers	SQLF_KTN_FCM_NUM_BUFFERS	503	Uint32
fed_noauth	SQLF_KTN_FED_NOAUTH	806	Uint16
federated	SQLF_KTN_FEDERATED	604	Sint16
fileserver	SQLF_KTN_FILESERVER	47	char(48)
health_mon	SQLF_KTN_HEALTH_MON	804	Uint16
indexrec[d]	SQLF_KTN_INDEXREC	20	Uint16
initdari_jvm	SQLF_KTN_INITDARI_JVM	602	Sint16
instance_memory	SQLF_KTN_INSTANCE_MEMORY	803	Uint64

表 62. 更新可能な データベース・マネージャー 構成パラメーター (続き)

パラメーター名	トークン	トークン値	データ・タイプ
intra_parallel	SQLF_KTN_INTRA_PARALLEL	306	Sint16
ipx_socket	SQLF_KTN_IPX_SOCKET	71	char(4)
java_heap_sz	SQLF_KTN_JAVA_HEAP_SZ	310	Sint32
jdk11_path	SQLF_KTN_JDK11_PATH	311	char(255)
keepfenced	SQLF_KTN_KEEPFENCED	81	Uint16
max_connections	SQLF_DBTN_MAX_CONNECTIONS	802	Sint32
max_connretries	SQLF_KTN_MAX_CONNRETRIES	509	Uint16
max_coordagents	SQLF_KTN_MAX_COORDAGENTS	501	Sint32
max_querydegree	SQLF_KTN_MAX_QUERYDEGREE	303	Sint32
max_time_diff	SQLF_KTN_MAX_TIME_DIFF	510	Uint16
maxagents	SQLF_KTN_MAXAGENTS	12	Uint32
maxcagents	SQLF_KTN_MAXCAGENTS	13	Sint32
maxdari	SQLF_KTN_MAXDARI	80	Sint32
maxtotfilop	SQLF_KTN_MAXTOTFILOP	45	Uint16
min_priv_mem	SQLF_KTN_MIN_PRIV_MEM	43	Uint32
mon_heap_sz	SQLF_KTN_MON_HEAP_SZ	79	Uint16
nname	SQLF_KTN_NNAME	7	char(8)
notifylevel	SQLF_KTN_NOTIFYLEVEL	605	Sint16
num_initagents	SQLF_KTN_NUM_INITAGENTS	500	Uint32
num_initdaris	SQLF_KTN_NUM_INITDARIS	601	Sint32
num_poolagents	SQLF_KTN_NUM_POOLAGENTS	502	Sint32
numdb	SQLF_KTN_NUMDB	6	Uint16
objectname	SQLF_KTN_OBJECTNAME	48	char(48)
priv_mem_thresh	SQLF_KTN_PRIV_MEM_THRESH	44	Sint32
query_heap_sz	SQLF_KTN_QUERY_HEAP_SZ	49	Sint32
restbufsz	SQLF_KTN_RESTBUFSZ	19	Uint32
resync_interval	SQLF_KTN_RESYNC_INTERVAL	68	Uint16
rqrioblk	SQLF_KTN_RQRIOBLK	1	Uint16
sheapthres	SQLF_KTN_SHEAPTHRES	21	Uint32
spm_log_file_sz	SQLF_KTN_SPM_LOG_FILE_SZ	90	Sint32
spm_max_resync	SQLF_KTN_SPM_MAX_RESYNC	91	Sint32
spm_name	SQLF_KTN_SPM_NAME	92	char(8)
start_stop_time	SQLF_KTN_START_STOP_TIME	511	Uint16
svcename	SQLF_KTN_SVCENAME	24	char(14)
sysadm_group	SQLF_KTN_SYSADM_GROUP	39	char(16)
sysctrl_group	SQLF_KTN_SYSCTRL_GROUP	63	char(16)
sysmaint_group	SQLF_KTN_SYSMAINT_GROUP	62	char(16)
tm_database	SQLF_KTN_TM_DATABASE	67	char(8)
tp_mon_name	SQLF_KTN_TP_MON_NAME	66	char(19)
tpname	SQLF_KTN_TPNAME	25	char(64)
trust_allclnts[e]	SQLF_KTN_TRUST_ALLCLNTS	301	Uint16
trust_clntauth	SQLF_KTN_TRUST_CLNTAUTH	302	Uint16
udf_mem_sz	SQLF_KTN_UDF_MEM_SZ	69	Uint16
use_sna_auth	SQLF_KTN_USE_SNA_AUTH	805	Uint16

表 62. 更新可能な データベース・マネージャー 構成パラメーター (続き)

パラメーター名	トークン	トークン値	データ・タイプ

[a] 有効な値 (sqlenv.h で定義) は、以下のとおりです。

```
SQL_AUTHENTICATION_SERVER (0)
SQL_AUTHENTICATION_CLIENT (1)
SQL_AUTHENTICATION_DCS (2)
SQL_AUTHENTICATION_DCE (3)
SQL_AUTHENTICATION_SVR_ENCRYPT (4)
SQL_AUTHENTICATION_DCS_ENCRYPT (5)
SQL_AUTHENTICATION_DCE_SVR_ENC (6)
SQL_AUTHENTICATION_KERBEROS (7)
SQL_AUTHENTICATION_KRB_SVR_ENC (8)
SQL_AUTHENTICATION_NOT_SPEC (255)
```

[b] SQLF_KTN_DFT_MONSWITCHES は Uint16 パラメーターで、このビットはデフォルト・モニター・スイッチ設定値を示します。これにより、一度にいくつかのパラメーターを指定できるようになります。この複合パラメーターを構成する個々のビットは、以下のとおりです。

```
Bit 1 (xxxx xxx1): dft_mon_uow
Bit 2 (xxxx xx1x): dft_mon_stmt
Bit 3 (xxxx x1xx): dft_mon_table
Bit 4 (xxxx 1xxx): dft_mon_buffpool
Bit 5 (xxx1 xxxx): dft_mon_lock
Bit 6 (xx1x xxxx): dft_mon_sort
```

[c] 有効な値 (sqlutil.h で定義) は、以下のとおりです。

```
SQLF_DSCVR_KNOWN (1)
SQLF_DSCVR_SEARCH (2)
```

[d] 有効な値 (sqlutil.h で定義) は、以下のとおりです。

```
SQLF_INX_REC_SYSTEM (0)
SQLF_INX_REC_REFERENCE (1)
```

[e] 有効な値 (sqlutil.h で定義) は、以下のとおりです。

```
SQLF_TRUST_ALLCLNTS_NO (0)
SQLF_TRUST_ALLCLNTS_YES (1)
SQLF_TRUST_ALLCLNTS_DRDAONLY (2)
```

表 63. 更新不可の データベース・マネージャー 構成パラメーター

パラメーター名	トークン	トークン値	データ・タイプ
nodetype[a]	SQLF_KTN_NODETYPE	100	Uint16
release	SQLF_KTN_RELEASE	101	Uint16

[a] 有効な値 (sqlutil.h で定義) は、以下のとおりです。

```
SQLF_NT_STANDALONE (0)
SQLF_NT_SERVER (1)
SQLF_NT_REQUESTOR (2)
SQLF_NT_STAND_REQ (3)
SQLF_NT_MPP (4)
SQLF_NT_SATELLITE (5)
```

言語構文:

C 構造

```
/* File: sqlutil.h */
/* Structure: SQLFUPD */
/* ... */
SQL_STRUCTURE sqlfupd
{
```

```
  unsigned short  token;
  char            *ptrvalue;
};
/* ... */
```

COBOL 構造

```
* File: sqlutil.cbl
01 SQL-FUPD.
    05 SQL-TOKEN              PIC 9(4) COMP-5.
    05 FILLER                 PIC X(2).
    05 SQL-VALUE-PTR          USAGE IS POINTER.
*
```

SQLM-COLLECTED

この構造は、データベース・システム・モニター API への呼び出しの後に情報を戻すために使用されます。これは、SQLM_DBMON_VERSION5_2 レベル以下で実行されるスナップショット要求のためだけに指定されます。

表 64. SQLM-COLLECTED 構造のフィールド

フィールド名	データ・タイプ	説明
SIZE	sqluint32	構造のサイズ。
DB2	sqluint32	廃止。
DATABASES	sqluint32	廃止。
TABLE_DATABASES	sqluint32	廃止。
LOCK_DATABASES	sqluint32	廃止。
APPLICATIONS	sqluint32	廃止。
APPLINFOS	sqluint32	廃止。
DCS_APPLINFOS	sqluint32	廃止。
SERVER_DB2_TYPE	sqluint32	データベース・マネージャー・サーバー・タイプ (sqlutil.h で定義)。
TIME_STAMP	TIMESTAMP	スナップショットがとられた時刻。
GROUP_STATES	OBJECT SQLM_ RECORDING_ GROUP	モニター・スイッチの現在の状態。
SERVER_PRDID	CHAR(20)	サーバー上のデータベース・マネージャーの製品名およびバージョン番号。
SERVER_NNAME	CHAR(20)	サーバーの構成ノード名。
SERVER_ INSTANCE_NAME	CHAR(20)	データベース・マネージャーのインスタンス名。
RESERVED	CHAR(22)	将来の利用のために予約されています。
NODE_NUMBER	UNSIGNED SHORT	データを送信しているノードの番号。
TIME_ZONE_DISP	sqlint32	GMT とローカル時刻の差 (秒単位)。
NUM_TOP_LEVEL_ STRUCTS	sqluint32	スナップショット出力バッファーに戻されたハイレベル構造の合計数。ハイレベル構造は、いくつかの下位レベルのデータ構造から構成される可能性があります。このカウンターは、現在は廃止された、各ハイレベル構造についての個々のカウンター (*table_databases* など) に代わるものです。
TABLESPACE_ DATABASES	sqluint32	廃止。
SERVER_VERSION	sqluint32	データを戻しているサーバーのバージョン。

言語構文:

C 構造

```
/* File: sqlmon.h */
/* Structure: SQLM-COLLECTED */
/* ... */
typedef struct sqlm_collected
{
  sqluint32      size;
  sqluint32      db2;
  sqluint32      databases;
  sqluint32      table_databases;
  sqluint32      lock_databases;
  sqluint32      applications;
  sqluint32      applinfos;
  sqluint32      dcs_applinfos;
  sqluint32      server_db2_type;
  sqlm_timestamp time_stamp;
  sqlm_recording_group group_states[SQLM_NUM_GROUPS];
  _SQLOLDCHAR    server_prdid[SQLM_IDENT_SZ];
  _SQLOLDCHAR    server_nname[SQLM_IDENT_SZ];
  _SQLOLDCHAR    server_instance_name[SQLM_IDENT_SZ];
  _SQLOLDCHAR    reserved[22];
  unsigned short node_number;
  long           time_zone_disp;
  sqluint32      num_top_level_structs;
  sqluint32      tablespace_databases;
  sqluint32      server_version;
}sqlm_collected;
/* ... */
```

COBOL 構造

```
* File: sqlmonct.cbl
01 SQLM-COLLECTED.
    05 SQLM-SIZE               PIC 9(9) COMP-5.
    05 DB2                     PIC 9(9) COMP-5.
    05 DATABASES               PIC 9(9) COMP-5.
    05 TABLE-DATABASES         PIC 9(9) COMP-5.
    05 LOCK-DATABASES          PIC 9(9) COMP-5.
    05 APPLICATIONS            PIC 9(9) COMP-5.
    05 APPLINFOS               PIC 9(9) COMP-5.
    05 DCS-APPLINFOS           PIC 9(9) COMP-5.
    05 SERVER-DB2-TYPE         PIC 9(9) COMP-5.
    05 TIME-STAMP.
        10 SECONDS              PIC 9(9) COMP-5.
        10 MICROSEC             PIC 9(9) COMP-5.
    05 GROUP-STATES OCCURS 6.
        10 INPUT-STATE          PIC 9(9) COMP-5.
        10 OUTPUT-STATE         PIC 9(9) COMP-5.
        10 START-TIME.
    05 SERVER-PRDID            PIC X(20).
    05 SERVER-NNAME            PIC X(20).
    05 SERVER-INSTANCE-NAME    PIC X(20).
    05 RESERVED                PIC X(32).
    05 TABLESPACE-DATABASES    PIC 9(9) COMP-5.
    05 SERVER-VERSION          PIC 9(9) COMP-5.
*
```

SQLM-RECORDING-GROUP

この構造は、データベース・システム・モニター API への呼び出しの後に情報を戻すために使用されます。

表 65. SQLM-RECORDING-GROUP 構造のフィールド

フィールド名	データ・タイプ	説明
INPUT_STATE	INTEGER	特定のモニター・グループに必要な状態。
OUTPUT_STATE	INTEGER	特定のモニター・スイッチの状態に関する戻り情報。
START_TIME	構造体	モニター・グループ・スイッチがオンになったときのタイム・スタンプ。

表 66. SQLM-TIMESTAMP 構造のフィールド

フィールド名	データ・タイプ	説明
SECONDS	INTEGER	1970 年 1 月 1 日 (GMT) から経過した秒数として表現される日時。
MICROSEC	INTEGER	現在の秒で、経過したマイクロ秒数。

input_state および *output_state* の両方について、特定のモニター・スイッチは、db2MonitorSwitches API に渡される配列中の索引によって識別されます。索引をスイッチにマップする定数は、 SQLM_*XXXX* _SW と呼ばれます。ここで、*XXXX* はモニター・グループの名前です。これらの定数は、sqlmon.h で定義されます。

言語構文:

C 構造

```
/* File: sqlmon.h */
/* Structure: SQLM-RECORDING-GROUP */
/* ... */
typedef struct sqlm_recording_group
{
  sqluint32      input_state;
  sqluint32      output_state;
  sqlm_timestamp start_time;
}sqlm_recording_group;
/* ... */
```

```
/* File: sqlmon.h */
/* Structure: SQLM-TIMESTAMP */
/* ... */
typedef struct sqlm_timestamp
{
  sqluint32  seconds;
  sqluint32  microsec;
}sqlm_timestamp;
/* ... */
```

COBOL 構造

```
* File: sqlmonct.cbl
01 SQLM-RECORDING-GROUP OCCURS 6 TIMES.
    05 INPUT-STATE            PIC 9(9) COMP-5.
    05 OUTPUT-STATE           PIC 9(9) COMP-5.
```

```
    05 START-TIME.
        10 SECONDS              PIC 9(9) COMP-5.
        10 MICROSEC            PIC 9(9) COMP-5.
*
* File: sqlmonct.cbl
01 SQLM-TIMESTAMP.
    05 SECONDS                 PIC 9(9) COMP-5.
    05 MICROSEC                PIC 9(9) COMP-5.
*
```

関連資料:

• 211 ページの『db2MonitorSwitches - モニター・スイッチの入手/更新』

SQLMA

SQL モニター・エリア (SQLMA) 構造は、データベース・モニター・スナップショット要求をデータベース・マネージャーに送信するために使用されます。また、スナップショット出力のサイズ (バイト単位) を見積もるためにも使用されます。

表 67. SQLMA 構造のフィールド

フィールド名	データ・タイプ	説明
OBJ_NUM	INTEGER	モニターされるオブジェクトの数。
OBJ_VAR	配列	モニターされるオブジェクトの記述を含む *sqlm_obj_struct* 構造の配列。配列の長さは、*OBJ_NUM* によって判別されます。

表 68. SQLM-OBJ-STRUCT 構造のフィールド

フィールド名	データ・タイプ	説明
AGENT_ID	INTEGER	モニターされるアプリケーションのアプリケーション・ハンドル。 *OBJ_TYPE* に *agent_id* (アプリケーション・ハンドル) が必要な場合にのみ指定します。すべての情報を収集するヘルス・スナップショットを検索するには、このフィールドに SQLM_HMON_OPT_COLL_FULL を指定します。
OBJ_TYPE	INTEGER	モニターされるオブジェクトのタイプ。
OBJECT	CHAR(36)	モニターされるオブジェクトの名前。 *OBJ_TYPE* に名前 (*appl_id* など) またはデータベース別名が必要な場合にのみ指定します。

OBJ_TYPE に有効な値 (`sqlmon` で定義) は、以下のとおりです。

SQLMA_DB2
DB2 関連情報

SQLMA_DBASE
データベース関連情報

SQLMA_APPL
アプリケーション ID により編成されるアプリケーション情報

SQLMA_AGENT_ID
エージェント ID により編成されるアプリケーション情報

SQLMA_DBASE_TABLES
データベースの表情報

SQLMA_DBASE_APPLS
データベースのアプリケーション情報

SQLMA_DBASE_APPLINFO
データベースのサマリー・アプリケーション情報

SQLMA_DBASE_LOCKS
データベースのロック情報

SQLMA_DBASE_ALL
データベース・マネージャー内の全アクティブ・データベースに関するデータベース情報

SQLMA_APPL_ALL
データベース・マネージャー内の全アクティブ・アプリケーションに関するアプリケーション情報

SQLMA_APPLINFO_ALL
データベース・マネージャー内の全アクティブ・アプリケーションに関するサマリー・アプリケーション情報

SQLMA_DCS_APPLINFO_ALL
データベース・マネージャー内の全アクティブ・アプリケーションに関するデータベース接続サービス・アプリケーション情報のサマリー

SQLMA_DYNAMIC_SQL
動的 SQL のスナップショットの入手

SQLMA_DCS_DBASE
データベース接続サービスのデータベース・レベル情報

SQLMA_DCS_DBASE_ALL
すべてのアクティブ・データベースに関する、データベース接続サービスのデータベース情報

SQLMA_DCS_APPL_ALL
すべての接続に関する、データベース接続サービスのアプリケーション情報

SQLMA_DCS_APPL
アプリケーション ID によって識別される、データベース接続サービスのアプリケーション情報

SQLMA_DCS_APPL_HANDLE
アプリケーション・ハンドルによって識別される、データベース接続サービスのアプリケーション情報

SQLMA_DCS_DBASE_APPLS
データベースへのアクティブな接続すべてに関する、データベース接続サービスのアプリケーション情報

SQLMA_DBASE_TABLESPACES
データベースの表スペース情報

SQLMA_DBASE_REMOTE
DataJoiner データベースの情報。

SQLMA_DBASE_REMOTE_ALL
すべての DataJoiner データベースの情報。

SQLMA_DBASE_APPLS_REMOTE
特定の DataJoiner データベースのアプリケーション情報。

SQLMA_APPLS_REMOTE_ALL
すべての DataJoiner データベースのアプリケーション情報。

言語構文:

C 構造

```
/* File: sqlmon.h */
/* Structure: SQLMA */
/* ... */
typedef struct sqlma
{
  sqluint32 obj_num;
  sqlm_obj_struct obj_var[1];
}sqlma;
/* ... */
```

```
/* File: sqlmon.h */
/* Structure: SQLM-OBJ-STRUCT */
/* ... */
typedef struct sqlm_obj_struct
{
  sqluint32     agent_id;
  sqluint32     obj_type;
  _SQLOLDCHAR   object[SQLM_OBJECT_SZ];
}sqlm_obj_struct;
/* ... */
```

COBOL 構造

```
* File: sqlmonct.cbl
01 SQLMA.
    05 OBJ-NUM                 PIC 9(9) COMP-5.
    05 OBJ-VAR OCCURS 0 TO 100 TIMES DEPENDING ON OBJ-NUM.
        10 AGENT-ID             PIC 9(9) COMP-5.
        10 OBJ-TYPE             PIC 9(9) COMP-5.
        10 OBJECT               PIC X(36).
*
```

SQLOPT

この構造は、sqlabndx API に BIND オプションを渡し、 sqlaprep API にプリコンパイル・オプションを渡し、 sqlarbnd API に再 BIND オプションを渡すために使用されます。

表 69. SQLOPT 構造のフィールド

フィールド名	データ・タイプ	説明
HEADER	構造体	*sqloptheader* 構造。
OPTION	配列	*sqloptions* 構造の配列。この配列中のエレメントの数は、ヘッダー の *allocated* フィールドの値によって判別されます。

表 70. SQLOPTHEADER 構造のフィールド

フィールド名	データ・タイプ	説明
ALLOCATED	INTEGER	*sqlopt* 構造の *option* 配列にあるエレメントの数。
USED	INTEGER	*sqlopt* 構造の *option* 配列内の実際に使用されるエレメントの数。これは、提供されたオプションの対 (*TYPE* および *VAL*) の数です。

表 71. SQLOPTIONS 構造のフィールド

フィールド名	データ・タイプ	説明
TYPE	INTEGER	バインド/プリコンパイル/再 BIND オプション・タイプ。
VAL	INTEGER	バインド/プリコンパイル/再 BIND オプション値。
注: *TYPE* および *VAL* フィールドは、指定されるそれぞれのバインド/プリコンパイル/再 BIND オプションについて繰り返されます。		

言語構文:

C 構造

```
/* File: sql.h */
/* Structure: SQLOPT */
/* ... */
SQL_STRUCTURE sqlopt
{
  SQL_STRUCTURE sqloptheader header;
  SQL_STRUCTURE sqloptions   option[1];
};
/* ... */
```

```
/* File: sql.h */
/* Structure: SQLOPTHEADER */
/* ... */
SQL_STRUCTURE sqloptheader
{
  sqluint32 allocated;
  sqluint32 used;
};
/* ... */
```

```
/* File: sql.h */
/* Structure: SQLOPTIONS */
/* ... */
SQL_STRUCTURE sqloptions
{
  sqluint32 type;
  sqluint32 val;
};
/* ... */
```

COBOL 構造

```
* File: sql.cbl
01 SQLOPT.
    05 SQLOPTHEADER.
        10 ALLOCATED   PIC 9(9) COMP-5.
        10 USED        PIC 9(9) COMP-5.
    05 SQLOPTIONS OCCURS 1 TO 50 DEPENDING ON ALLOCATED.
        10 SQLOPT-TYPE        PIC 9(9) COMP-5.
        10 SQLOPT-VAL         PIC 9(9) COMP-5.
        10 SQLOPT-VAL-PTR     REDEFINES SQLOPT-VAL
*
```

関連資料:

- 294 ページの『sqlabndx - バインド』
- 300 ページの『sqlaprep - プログラムのプリコンパイル』
- 302 ページの『sqlarbnd - 再バインド』

SQLU-LSN

db2ReadLog API によって使用されるこの共用体には、ログ・シーケンス番号の定義が含まれます。ログ・シーケンス番号 (LSN) は、データベース・ログ内の相対バイト・アドレスを示します。すべてのログ・レコードは、この番号によって識別されます。この番号は、ログ・レコードのデータベース・ログの始まりからのバイト・オフセットを示します。

表 72. SQLU-LSN 共用体のフィールド

フィールド名	データ・タイプ	説明
lsnChar	UNSIGNED CHAR の配列	6 メンバー文字配列のログ・シーケンス番号を指定します。
lsnWord	UNSIGNED SHORT の配列	3 のショート型配列のログ・シーケンス番号を指定します。

言語構文:

C 構造

```
typedef union SQLU_LSN
{
unsigned char  lsnChar  [6] ;
unsigned short lsnWord  [3] ;
} SQLU_LSN;
```

関連資料:

- 219 ページの『db2ReadLog - ログの非同期読み取り』

SQLU-MEDIA-LIST

この構造は、db2Load API に情報を渡すために使用されます。

表 73. SQLU-MEDIA-LIST 構造のフィールド

フィールド名	データ・タイプ	説明
MEDIA_TYPE	CHAR(1)	メディア・タイプを示す文字。
SESSIONS	INTEGER	この構造の *target* フィールドにより示される配列中のエレメントの数を示します。
TARGET	Union	このフィールドは、4 つの構造タイプのうちの 1 つを指すポインターです。示される構造のタイプは、*media_type* フィールドの値によって判別されます。このフィールドに提供する値の詳細については、適切な API を参照してください。

表 74. SQLU-MEDIA-LIST-TARGETS 構造のフィールド

フィールド名	データ・タイプ	説明
MEDIA	ポインター	*sqlu_media_entry* 構造を指すポインター。
VENDOR	ポインター	*sqlu_vendor* 構造を指すポインター。
LOCATION	ポインター	*sqlu_location_entry* 構造を指すポインター。
PSTATEMENT	ポインター	*sqlu_statement_entry* 構造を指すポインター。

表 75. SQLU-MEDIA-ENTRY 構造のフィールド

フィールド名	データ・タイプ	説明
RESERVE_LEN	INTEGER	*media_entry* フィールドの長さ。 C 以外の言語用。
MEDIA_ENTRY	CHAR(215)	バックアップおよびリストア・ユーティリティーが使用するバックアップ・イメージのパス。

表 76. SQLU-VENDOR 構造のフィールド

フィールド名	データ・タイプ	説明
RESERVE_LEN1	INTEGER	*shr_lib* フィールドの長さ。 C 以外の言語用。
SHR_LIB	CHAR(255)	ベンダーにより提供される、データの保管または検索用の共用ライブラリーの名前。
RESERVE_LEN2	INTEGER	*filename* フィールドの長さ。 C 以外の言語用。
FILENAME	CHAR(255)	共用ライブラリーの使用時にロード入力ソースを識別するファイル名。

表 77. SQLU-LOCATION-ENTRY 構造のフィールド

フィールド名	データ・タイプ	説明
RESERVE_LEN	INTEGER	*location_entry* フィールドの長さ。 C 以外の言語用。
LOCATION_ENTRY	CHAR(256)	ロード・ユーティリティー用の入力データ・ファイルの名前。

表 78. SQLU-STATEMENT-ENTRY 構造のフィールド

フィールド名	データ・タイプ	説明
LENGTH	INTEGER	*data* フィールドの長さ。
PDATA	ポインター	SQL 照会へのポインター。

MEDIA_TYPE に有効な値 (`sqlutil` で定義) は、以下のとおりです。

SQLU_LOCAL_MEDIA
ローカル装置 (テープ、ディスク、またはディスケット)

SQLU_SERVER_LOCATION
サーバー装置 (テープ、ディスク、またはディスケット。ロード専用)。
piSourceList パラメーターにのみ指定できます。

SQLU_CLIENT_LOCATION
クライアント装置 (ファイルまたは名前付きパイプ。ロードのみ。)
piSourceList パラメーターにのみ指定できます。

SQLU_SQL_STMT
SQL 照会 (ロードのみ) *piSourceList* パラメーターにのみ指定できます。

SQLU_TSM_MEDIA
TSM

SQLU_OTHER_MEDIA
ベンダー・ライブラリー

SQLU_USER_EXIT
ユーザー出口 (OS/2 のみ)

SQLU_PIPE_MEDIA
名前付きパイプ (ベンダー API のみ)

SQLU_DISK_MEDIA
ディスク (ベンダー API のみ)

SQLU_DISKETTE_MEDIA
ディスケット (ベンダー API のみ)

SQLU_TAPE_MEDIA
テープ (ベンダー API のみ)

言語構文:

C 構造

```
/* File: sqlutil.h */
/* Structure: SQLU-MEDIA-LIST */
/* ... */
typedef SQL_STRUCTURE sqlu_media_list
{
  char            media_type;
  char            filler[3];
  sqlint32            sessions;
  union sqlu_media_list_targets target;
} sqlu_media_list;
/* ... */
```

```
/* File: sqlutil.h */
/* Structure: SQLU-MEDIA-LIST-TARGETS */
/* ... */
union sqlu_media_list_targets
{
  struct sqlu_media_entry        *media;
  struct sqlu_vendor             *vendor;
  struct sqlu_location_entry     *location;
  struct sqlu_statement_entry    *pStatement;
};
/* ... */
```

```
/* File: sqlutil.h */
/* Structure: SQLU-MEDIA-ENTRY */
/* ... */
typedef SQL_STRUCTURE sqlu_media_entry
{
  sqluint32       reserve_len;
  char            media_entry[SQLU_DB_DIR_LEN+1];
} sqlu_media_entry;
/* ... */
```

```
/* File: sqlutil.h */
/* Structure: SQLU-VENDOR */
/* ... */
typedef SQL_STRUCTURE sqlu_vendor
{
  sqluint32       reserve_len1;
  char            shr_lib[SQLU_SHR_LIB_LEN+1];
```

```
  sqluint32       reserve_len2;
  char            filename[SQLU_SHR_LIB_LEN+1];
} sqlu_vendor;
/* ... */

/* File: sqlutil.h */
/* Structure: SQLU-LOCATION-ENTRY */
/* ... */
typedef SQL_STRUCTURE sqlu_location_entry
{
  sqluint32       reserve_len;
  char            location_entry[SQLU_MEDIA_LOCATION_LEN+1];
} sqlu_location_entry;
/* ... */

/* File: sqlutil.h */
/* Structure: SQLU-STATEMENT-ENTRY */
/* ... */
SQL_STRUCTURE sqlu_statement_entry
{
  sqluint32       length;
  char            *pEntry;
};
/* ... */
```

COBOL 構造

```
* File: sqlutil.cbl
01 SQLU-MEDIA-LIST.
    05 SQL-MEDIA-TYPE         PIC X.
    05 SQL-FILLER             PIC X(3).
    05 SQL-SESSIONS           PIC S9(9) COMP-5.
    05 SQL-TARGET.
        10 SQL-MEDIA          USAGE IS POINTER.
        10 SQL-VENDOR         REDEFINES SQL-MEDIA
        10 SQL-LOCATION       REDEFINES SQL-MEDIA
        10 SQL-STATEMENT      REDEFINES SQL-MEDIA
        10 FILLER             REDEFINES SQL-MEDIA
*

* File: sqlutil.cbl
01 SQLU-MEDIA-ENTRY.
    05 SQL-MEDENT-LEN         PIC 9(9) COMP-5.
    05 SQL-MEDIA-ENTRY        PIC X(215).
    05 FILLER                 PIC X.
*

* File: sqlutil.cbl
01 SQLU-VENDOR.
    05 SQL-SHRLIB-LEN         PIC 9(9) COMP-5.
    05 SQL-SHR-LIB            PIC X(255).
    05 FILLER                 PIC X.
    05 SQL-FILENAME-LEN       PIC 9(9) COMP-5.
    05 SQL-FILENAME           PIC X(255).
    05 FILLER                 PIC X.
*

* File: sqlutil.cbl
01 SQLU-LOCATION-ENTRY.
    05 SQL-LOCATION-LEN       PIC 9(9) COMP-5.
    05 SQL-LOCATION-ENTRY     PIC X(255).
    05 FILLER                 PIC X.
*

* File: sqlutil.cbl
01 SQLU-STATEMENT-ENTRY.
    05 SQL-STATEMENT-LEN       PIC 9(9) COMP-5.
    05 SQL-STATEMENT-ENTRY     USAGE IS POINTER.
*
```

SQLU-RLOG-INFO

この構造は、db2ReadLog およびデータベース・ログへの呼び出し状況に関する情報を含んでいます。

表 79. SQLU-RLOG-INFO 構造のフィールド

フィールド名	データ・タイプ	説明
initialLSN	SQLU_LSN	最初のデータベース接続ステートメントが発行された後で、書き込まれた最初のログ・レコードの LSN 値を指定します。詳細については、SQLU-LSN を参照してください。
firstReadLSN	SQLU_LSN	最初に読み取るログ・レコードの LSN 値を指定します。
lastReadLSN	SQLU_LSN	最後に読み取るログ・レコードの LSN 値を指定します。
curActiveLSN	SQLU_LSN	現行の (アクティブ) ログの LSN 値を指定します。
logRecsWritten	sqluint32	バッファーに書き込まれるログ・レコードの数を指定します。
logBytesWritten	sqluint32	バッファーに書き込まれるバイト数を指定します。

言語構文:

C 構造

```
typedef SQL_STRUCTURE SQLU_RLOG_INFO
{
SQLU_LSN       initialLSN ;
SQLU_LSN       firstReadLSN ;
SQLU_LSN       lastReadLSN ;
SQLU_LSN       curActiveLSN ;
sqluint32      logRecsWritten ;
sqluint32      logBytesWritten ;
} SQLU_RLOG_INFO;
```

関連資料:

- 219 ページの『db2ReadLog - ログの非同期読み取り』
- 495 ページの『SQLU-LSN』

SQLUPI

この構造は、表のパーティション・マップやパーティション・キーなどのパーティション情報を保管するのに使用されます。

表 80. SQLUPI 構造のフィールド

フィールド名	データ・タイプ	説明
PMAPLEN	INTEGER	パーティション・マップの長さ (バイト単位)。単一ノードの表の場合、値は `sizeof(SQL_PDB_NODE_TYPE)` です。複数ノードの表の場合、値は `SQL_PDB_MAP_SIZE * sizeof (SQL_PDB_NODE_TYPE)` です。
PMAP	SQL_PDB_NODE_TYPE	パーティション・マップ。

表 80. SQLUPI 構造のフィールド (続き)

フィールド名	データ・タイプ	説明
SQLD	INTEGER	使用された SQLPARTKEY エレメントの数。つまり、パーティション・キー内のキー・パーツの数。
SQLPARTKEY	構造体	パーティション・キー内のパーティション列の記述。パーティション列の最大数は SQL_MAX_NUM_PART_KEYS です。

表 81 に、SQLUPI データ構造の SQL データ・タイプおよび長さを示します。SQLTYPE 欄は、項目のデータ・タイプを表す数値を指定します。

表 81. SQLUPI 構造の SQL データ・タイプおよび長さ

データ・タイプ	SQLTYPE (NULL 使用不可)	SQLTYPE (NULL 使用可能)	SQLLEN	AIX
日付	384	385	無視	はい
時刻	388	389	無視	はい
タイム・スタンプ	392	393	無視	はい
可変長文字ストリング	448	449	ストリングの長さ	はい
固定長文字ストリング	452	453	ストリングの長さ	はい
長文字ストリング	456	457	無視	いいえ
NULL 終了文字ストリング	460	461	ストリングの長さ	はい
浮動小数点	480	481	無視	はい
10 進	484	485	バイト 1 = 精度バイト 2 = 位取り	はい
長精度整数	496	497	無視	はい
短精度整数	500	501	無視	はい
可変長 GRAPHIC ストリング	464	465	2 バイト文字の長さ	はい
固定長 GRAPHIC ストリング	468	469	2 バイト文字の長さ	はい
LONG GRAPHIC ストリング	472	473	無視	いいえ

言語構文:

C 構造

```
/* File: sqlutil.h */
/* Structure: SQLUPI */
/* ... */
SQL_STRUCTURE sqlupi
{
  unsigned short  pmaplen;
  SQL_PDB_NODE_TYPE pmap[SQL_PDB_MAP_SIZE];
  unsigned short  sqld;
  struct sqlpartkey sqlpartkey[SQL_MAX_NUM_PART_KEYS];
};
/* ... */
```

```
/* File: sqlutil.h */
/* Structure: SQLPARTKEY */
/* ... */
SQL_STRUCTURE sqlpartkey
{
  unsigned short  sqltype;
  unsigned short  sqllen;
};
/* ... */
```

SQLXA-XID

トランザクション API によって使用され、XA トランザクションを識別します。

表 82. SQLXA-XID 構造のフィールド

フィールド名	データ・タイプ	説明
FORMATID	INTEGER	XA 形式 ID。
GTRID_LENGTH	INTEGER	グローバル・トランザクション ID の長さ。
BQUAL_LENGTH	INTEGER	ブランチ ID の長さ。
DATA	CHAR[128]	BQUAL および後書きブランクが続く、合計 128 バイトの GTRID。
注: GTRID および BQUAL の最大サイズは、それぞれ 64 バイトです。		

言語構文:

C 構造

```
/* File: sqlxa.h */
/* Structure: SQLXA-XID */
/* ... */
typedef struct sqlxa_xid_t SQLXA_XID;
/* ... */
```

```
/* File: sqlxa.h */
/* Structure: SQLXA-XID-T */
/* ... */
struct sqlxa_xid_t
{
  sqlint32 formatID;
  sqlint32 gtrid_length;
  sqlint32 bqual_length;
  char data[SQLXA_XIDDATASIZE];
};
/* ... */
```

付録 A. 命名規則

データベースや表、認証 ID などのデータベース・マネージャー・オブジェクトの命名の際に適用される規則について説明します。

- データベース・マネージャー・オブジェクトの名前を表示する文字ストリングには、次のいずれかを含めることができます。 a から z、A から Z、0 から 9、@、#、および $。
- さらに、セキュリティー・プラグインでサポートされる場合、ユーザー ID およびグループに次の文字を含めることもできます。 _、 !、 %、 (、)、 {、 }、 -、 .、 ^。
- 次の文字を含むユーザー ID およびグループをコマンド行プロセッサーで入力するときには、引用符で区切る必要があります。 !、%、(、)、{、 }、-、.、^。
- ストリングの最初の文字は、アルファベット文字、@、 #、または $ にする必要があります。先頭文字に数字または SYS、DBM、または IBM のストリングを使用することはできません。
- 特に注記がなければ、名前は小文字で入力して構いません。ただし、データベース・マネージャーはそれを大文字と見なして処理します。

  ただし、システム・ネットワーク体系 (SNA) 下の名前を表す文字ストリングは例外です。論理単位名 (partner_lu および local_lu) など、多くの値では大文字小文字が区別されます。こうした名前は、それらの用語に対応する SNA 定義に出ているとおりに入力してください。
- データベース名やデータベース別名は、前に説明した集合内の 1 つから 8 つの文字、数字、キーボード文字を含むユニークな文字ストリングです。

  データベースはシステム内にカタログされており、ローカル・データベース・ディレクトリーの別名が 1 つのフィールドに、元の名前が別のフィールドに入っています。ほとんどの機能の場合、データベース・マネージャーは、データベース・ディレクトリーの別名フィールドに入力された名前を使用します。 (ただし、CHANGE DATABASE COMMENT および CREATE DATABASE は例外です。この場合は、ディレクトリー・パスを指定しなければなりません。)
- 表またはビューの名前や別名は、SQL ID です。これは、長さが 1 から 128 文字のユニークな文字ストリングです。列名の長さは、1 から 30 文字です。

  完全修飾表名は、*schema.tablename* で構成されます。スキーマはユニークなユーザー ID で、その下に表が作成されます。宣言された一時表のスキーマ名は SESSION でなければなりません。
- 認証 ID の長さは、Windows 32 ビット・オペレーティング・システムで 30 文字以内に、他のすべてのオペレーティング・システムでは 8 文字以内に制限されています。
- グループ ID の長さは、30 文字以内に制限されています。
- ノード・ディレクトリー内でカタログ化するリモート・ノードのローカル別名の長さは、 8 文字より長くはできません。

付録 B. ヒューリスティック API

ヒューリスティック API

データベースは、分散トランザクション処理 (DTP) 環境で使用することができます。

リソース所有者 (たとえばデータベース管理者) がトランザクション・マネージャー (TM) による*再同期* アクションの実行を待てないときに、ツール作成者が未確定トランザクションに対してヒューリスティック関数を実行するための一連の API が用意されています。この状態は、たとえば通信回線が故障し、未確定トランザクションが必要なリソースを拘束している場合などに発生する可能性があります。データベース・マネージャーの場合、これらのリソースには、トランザクションによって使用されている表と索引、ログ・スペース、および記憶域に対するロックが含まれます。さらに、未確定トランザクションが 1 つ増えるたびに、データベース・マネージャーが処理できる並行トランザクションの最大数が 1 つずつ減少します。

ヒューリスティック API では、未確定トランザクションを照会、コミット、およびロールバックすることができます。さらにログ・レコードを除去し、ログ・ページを解放することにより、手動操作によりコミットあるいはロールバックされたトランザクションを取り消すことができます。

重要: ヒューリスティック API は、慎重に、そして最終手段としてのみ使用してください。 TM は、再同期イベントを開始します。 TM に再同期化アクションを開始するためのオペレーター・コマンドがある場合は、それが使用されます。ユーザーが TM によって開始される再同期を待てない場合は、ヒューリスティック・アクションが必要となります。

これらのアクションを実行する絶対確実な方法というものは存在しませんが、以下の指針を参考にしてください。

- 未確定トランザクションを表示するには、db2XaListIndTrans 関数を使用してください。それらの未確定トランザクションは、状態 = 「P」 (準備済み) であり、接続されていません。 *xid* の *gtrid* 部分は、グローバル・トランザクションに参加する、他のリソース・マネージャー (RM) と同一のグローバル・トランザクション ID です。
- アプリケーションとオペレーティング環境の知識を利用して、参加している他の RM を識別してください。
- トランザクション・マネージャーが CICS® で、 RM だけが CICS リソースである場合は、ヒューリスティック・ロールバックを実行してください。
- トランザクション・マネージャーが CICS でない場合は、それを使用して、未確定トランザクションと同じ *gtrid* を持つトランザクションの状況を判別してください。
- 1 つでも RM がコミットあるいはロールバックされた場合は、ヒューリスティック・コミットまたはロールバックを実行してください。

- それらすべてが準備済み状態にある場合は、ヒューリスティック・ロールバックを実行してください。
- 1 つでも使用不可である RM がある場合は、ヒューリスティック・ロールバックを実行してください。

トランザクション・マネージャーが利用可能であり、未確定トランザクションが、第 2 フェーズまたは以前の再同期において RM が使用可能でなかったことが原因で発生している場合、 DBA は、TM のログから、他の RM に対して行われたアクションを判別し、同じことを行わなければなりません。 *gtrid* は、TM と RM 間の突き合わせキーです。

ヒューリスティックにコミットまたはロールバックされたトランザクションが、ログ満杯の状態を引き起こさない限り、sqlxhfrg を実行しないでください。この forget 関数は、この未確定トランザクションにより占有されているログ・スペースを解放します。トランザクション・マネージャーがこの未確定トランザクションに対して再同期アクションを実行する場合、 TM は、この RM 内にレコードが検索されないために、間違った判断を下し、他の RM をコミットまたはロールバックしてしまう可能性があります。一般に、レコードの脱落は、RM がロールバックされたことを示唆します。

db2XaGetInfo - リソース・マネージャー情報の入手

xa_open 呼び出しが行われた場合、特定のリソース・マネージャーの情報を抽出します。

許可:

なし

必要な接続:

データベース

API 組み込みファイル:

sqlxa.h

C API 構文:

```
/* File: sqlxa.h */
/* API: Get Information for Resource Manager */
/* ... */
SQL_API_RC SQL_API_FN
db2XaGetInfo (
  db2Uint32 versionNumber,
  void * pParmStruct,
  struct sqlca *pSqlca);

typedef SQL_STRUCTURE db2XaGetInfoStruct
{
  db2int32 iRmid;
  struct sqlca oLastSqlca;
} db2XaGetInfoStruct;
/* ... */
```

API パラメーター:

versionNumber
入力。 2 番目のパラメーター *pParmStruct* として渡される構造のバージョンとリリースのレベルを指定します。

pParmStruct
入力。 *db2XaGetInfoStruct* 構造を指すポインター。

pSqlca
出力。 *sqlca* 構造を指すポインター。

iRmid 入力。情報が必要となるリソース・マネージャーを指定します。

oLastSqlca
出力。最後の XA API 呼び出しの *sqlca* を含みます。

注: 最後に失敗した XA API の結果の *sqlca* が検索されます。

関連資料:

• 453 ページの『SQLCA』

db2XaListIndTrans - 未確定トランザクションのリスト

現在データベースに接続されている、すべての未確定トランザクションのリストを提供します。

有効範囲:

この API は、それを発行したデータベース・パーティションにのみ影響を与えます。

許可:

以下のいずれかです。

- *sysadm*
- *dbadm*

必要な接続:

データベース

API 組み込みファイル:

db2ApiDf.h

C API 構文:

```
/* File: db2ApiDf.h */
/* API: List Indoubt Transactions */
/* ... */
SQL_API_RC SQL_API_FN
db2XaListIndTrans (
  db2Uint32      versionNumber,
  void * pParmStruct,
  struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2XaListIndTransStruct
{
```

```
db2XaRecoverStruct * piIndoubtData;
db2Uint32            iIndoubtDataLen;
db2Uint32            oNumIndoubtsReturned;
db2Uint32            oNumIndoubtsTotal;
db2Uint32            oReqBufferLen;
} db2XaListIndTransStruct;

typedef SQL_STRUCTURE db2XaRecoverStruct
{
sqluint32      timestamp;
SQLXA_XID   xid;
char           dbalias[SQLXA_DBNAME_SZ];
char           applid[SQLXA_APPLID_SZ];
char           sequence_no[SQLXA_SEQ_SZ];
char           auth_id[SQL_USERID_SZ];
char           log_full;
char           indoubt_status;
char           originator;
char           reserved[8];
} db2XaRecoverStruct;
```

API パラメーター:

versionNumber
入力。 2 番目のパラメーター *pParmStruct* として渡される構造のバージョンとリリースのレベルを指定します。

pParmStruct
入力。 *db2XaListIndTransStruct* 構造を指すポインター。

pSqlca
出力。*sqlca* 構造へのポインター。

piIndoubtData
入力。未確定データが戻されるアプリケーション提供バッファーを指すポインター。未確定データは *db2XaRecoverStruct* の形式です。アプリケーションは *db2XaRecoverStruct* 構造のサイズを使用して、未確定トランザクションのリストを、このパラメーターが提供するアドレスから始めて全探索することができます。

値が NULL の場合、DB2 は必要なバッファー・サイズを計算し、その値を *oReqBufferLen* で戻します。*oNumIndoubtsTotal* には未確定トランザクションの総数が入ります。アプリケーションは必要なバッファー・サイズを割り振り、API を再発行します。

oNumIndoubtsReturned
出力。 *pIndoubtData* によって指定されたバッファーに戻される未確定トランザクション・レコードの数。

oNumIndoubtsTotal
出力。 API 呼び出し時に使用可能な未確定トランザクション・レコードの総数。 *piIndoubtData* バッファーが小さすぎてすべてのレコードを入れることができない場合、 *oNumIndoubtsTotal* は *oNumIndoubtsReturned* の総数より大きくなります。アプリケーションは、すべてのレコードを入手するために、API を再発行することができます。

注: この数は、自動またはヒューリスティック未確定トランザクションの再同期の結果として、または未確定状態を入力する他のトランザクションの結果として、 API 呼び出しの合間に変更される可能性があります。

oReqBufferLen
出力。 API 呼び出し時にすべての未確定トランザクション・レコードを保持するために必要なバッファー長。アプリケーションは、この値を使用して、 *pIndoubtData* を NULL に設定して API を呼び出すことにより必要なバッファー・サイズを判別できます。また、この値は必要なバッファーを割り振るために使用できます。この API は、割り振られたバッファーのアドレスに設定された *pIndoubtData* が指定されて発行されます。

注: 必要なバッファー・サイズは、自動またはヒューリスティック未確定トランザクションの再同期の結果として、または未確定状態を入力する他のトランザクションの結果として、 API 呼び出しの合間に変更される可能性があります。このためアプリケーションは、さらに大きなバッファーを割り振る場合があります。

timestamp
出力。トランザクションが未確定状態を入力した時間を指定します。

xid 出力。グローバル・トランザクションを固有に識別する、トランザクション・マネージャーによって割り当てられた XA ID を指定します。

dbalias
出力。未確定トランザクションが検索されるデータベースの別名を指定します。

applid 出力。データベース・マネージャーがこのトランザクションに割り当てたアプリケーション ID を指定します。

sequence_no
出力。データベース・マネージャーが *applid* の拡張として割り当てたシーケンス番号を指定します。

auth_id
出力。トランザクションを実行したユーザーの許可 ID を指定します。

log_full
出力。このトランザクションによってログが満杯状態になったかどうかを示します。有効な値は以下のとおりです。

SQLXA_TRUE
この未確定トランザクションによってログが満杯状態になりました。

SQLXA_FALSE
この未確定トランザクションではログは満杯状態にはなりませんでした。

indoubt_status
出力。この未確定トランザクションの状態を示します。有効な値は以下のとおりです。

SQLXA_TS_PREP
トランザクションの準備が完了しました。接続されたパラメーターを使用して、トランザクションが通常のコミット処理の第 2 フェー

ズを待っているのか、またはエラーが発生し、トランザクション・マネージャーでの再同期を必要としているかどうかを判別できます。

SQLXA_TS_HCOM
トランザクションがヒューリスティックにコミットされました。

SQLXA_TS_HROL
トランザクションがヒューリスティックにロールバックされました。

SQLXA_TS_MACK
トランザクションはパーティション・データベース内のノードからのコミット確認通知を消失しています。

SQLXA_TS_END
このデータベースでトランザクションが終了しました。このトランザクションは後で、再活動化、コミット、またはロールバックする場合があります。トランザクション・マネージャーがエラーを検出し、トランザクションが完了していない可能性もあります。この場合には、このトランザクションは、ロックされていて他のアプリケーションがデータにアクセスできない可能性があるため、ヒューリスティックな処置が必要です。

使用上の注意:

一般的なアプリケーションは、現行の接続をデータベースに設定するか、またはパーティション・データベース・コーディネーター・ノードに設定した後で、以下のステップを実行します。

1. *piIndoubtData* を NULL に設定して、**db2XaListIndTrans** を呼び出します。これによって *oReqBufferLen* および *oNumIndoubtsTotal* に値が戻されます。
2. *oReqBufferLen* に戻された値を使用して、バッファーを割り振ります。*oReqBufferLen* を入手するためのこの API の最初に呼び出しが原因で、このバッファーは、追加の未確定トランザクションがある場合は、十分な大きさがない可能性があります。アプリケーションは *oReqBufferLen* より大きいバッファーを提供することもできます。
3. すべての未確定トランザクション・レコードが入手されたかどうか判別します。これは、*oNumIndoubtsReturned* を *oNumIndoubtTotal* と比較することにより実行できます。 *oNumIndoubtsTotal* が *oNumIndoubtsReturned* より大きい場合は、アプリケーションは上記のステップを繰り返すことができます。

関連資料:

- 453 ページの『SQLCA』
- 511 ページの『sqlxphcm - 未確定トランザクションのコミット』
- 513 ページの『sqlxphrl - 未確定トランザクションのロールバック』

sqlxhfrg - トランザクション状況の忘却

RM に、ヒューリスティックに完了した (ヒューリスティックにコミットあるいはロールバックされた) トランザクションの知識を消去させます。

許可:

以下のいずれかです。

- *sysadm*
- *dbadm*

必要な接続:

データベース

API 組み込みファイル:

sqlxa.h

C API 構文:

```
/* File: sqlxa.h */
/* API: Forget Transaction Status */
/* ... */
extern int SQL_API_FN sqlxhfrg(
   SQLXA_XID            *pTransId,
   struct sqlca         *pSqlca
   );
/* ... */
```

API パラメーター:

pTransId
入力。ヒューリスティックに忘却される、またはデータベース・ログから除去されるトランザクションの XA ID です。

pSqlca
出力。*sqlca* 構造へのポインター。

使用上の注意:

FORGET 操作は、*ヒューリスティック・コミット済み* あるいは*ロールバック済み* 状態のトランザクションにのみ適用できます。

関連資料:

- 453 ページの『SQLCA』
- 501 ページの『SQLXA-XID』

sqlxphcm - 未確定トランザクションのコミット

未確定トランザクション (つまり、コミットされる準備ができているトランザクション) をコミットします。操作が成功した場合、トランザクションはヒューリスティック・コミット済み 状態になります。

有効範囲:

この API は、それが発行されたノードにのみ影響を与えます。

許可:

以下のいずれかです。

- *sysadm*
- *dbadm*

必要な接続:

データベース

API 組み込みファイル:

sqlxa.h

C API 構文:

```
/* File: sqlxa.h */
/* API: Commit an Indoubt Transaction */
/* ... */
extern int SQL_API_FN sqlxphcm(
   int                  exe_type,
   SQLXA_XID            *pTransId,
   struct sqlca         *pSqlca
   );
/* ... */
```

API パラメーター:

exe_type
入力。 EXE_THIS_NODE が指定されると、操作はこのノードでのみ実行されます。

pTransId
入力。ヒューリスティックにコミットされるトランザクションの XA ID です。

pSqlca
出力。*sqlca* 構造へのポインター。

使用上の注意:

準備済み 状態のトランザクションのみがコミットできます。トランザクションがヒューリスティックにコミットされると、データベース・マネージャー は、sqlxhfrg が発行されるまで、トランザクションの状態を記憶します。

関連資料:

- 453 ページの『SQLCA』
- 501 ページの『SQLXA-XID』
- 511 ページの『sqlxhfrg - トランザクション状況の忘却』

sqlxphrl - 未確定トランザクションのロールバック

未確定トランザクション (つまり、準備済みのトランザクション) をロールバックします。操作が成功した場合、トランザクションはヒューリスティック・ロールバック済み 状態になります。

有効範囲:

この API は、それが発行されたノードにのみ影響を与えます。

許可:

以下のいずれかです。

- *sysadm*
- *dbadm*

必要な接続:

データベース

API 組み込みファイル:

sqlxa.h

C API 構文:

```
/* File: sqlxa.h */
/* API: Roll Back an Indoubt Transaction */
/* ... */
extern int SQL_API_FN sqlxphrl(
   int                  exe_type,
   SQLXA_XID            *pTransId,
   struct sqlca         *pSqlca
   );
/* ... */
```

API パラメーター:

exe_type
入力。 EXE_THIS_NODE が指定されると、操作はこのノードでのみ実行されます。

pTransId
入力。ヒューリスティックにロールバックされる、トランザクションの XA ID です。

pSqlca
出力。*sqlca* 構造へのポインター。

使用上の注意:

準備済み またはアイドル 状態のトランザクションのみがロールバックできます。トランザクションがヒューリスティックにロールバックされると、データベース・マネージャーは、sqlxhfrg が発行されるまで、トランザクションの状態を記憶します。

関連資料:

- 453 ページの『SQLCA』

- 501 ページの『SQLXA-XID』
- 511 ページの『sqlxhfrg - トランザクション状況の忘却』

付録 C. プリコンパイラーのカスタマイズ API

他のアプリケーション開発ツールの製品が、それらの製品内で DB2 用のプリコンパイラー・サポートを直接実現するための文書化された API のセット。たとえば AIX 上の IBM COBOL は、このインターフェースを使用します。プリコンパイラー・サービス API のセットに関する情報は、以下の DB2 アプリケーション開発 Web サイトにある PDF ファイル prepapi.pdf に記載されています。

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

関連資料:

- 633 ページの『付録 J. IBM と連絡をとる』

付録 D. ベンダー製品用のバックアップおよびリストア API

Storage Manager に対するバックアップおよびリストアの API

DB2 では、サード・パーティーのメディア管理製品が、バックアップおよびリストア操作およびログ・ファイルのデータを保管および検索するために使用できるインターフェースが用意されています。この機能は、DB2 の標準部分としてサポートされている、ディスケット、ディスク、テープ、および Tivoli Storage Manager のバックアップ、リストア、およびログ・アーカイブ・データのターゲットを拡大できるように設計されています。

このようなサード・パーティーのメディア管理製品は、この付録ではベンダー製品と呼ばれています。

DB2 では、多くのベンダーが使用できる汎用データ・インターフェースを提供する関数プロトタイプのセットが定義されており、これらを用いてバックアップ、リストア、ログ・アーカイブを行います。これらの関数は、共用ライブラリー (UNIX ベースのシステムの場合) または DLL (Windows オペレーティング・システムの場合) において、ベンダーにより提供されます。関数が DB2 によって呼び出されると、バックアップ、リストア、またはログ・アーカイブ呼び出しルーチンによって指定された共用ライブラリーまたは DLL がロードされ、必要なタスクを実行するためにベンダー提供の関数が呼び出されます。

DB2 ベンダー機能を実演するサンプル・ファイルは、UNIX プラットフォームでは `sqllib/samples/BARVendor` ディレクトリーに、 Windows では `sqllib¥samples¥BARVendor` ディレクトリーにあります。

操作概要

DB2 とベンダー製品の間のインターフェースのために、 7 つの関数が定義されています。

- sqluvint - 初期設定と装置へのリンク
- sqluvget - 装置からのデータの読み取り
- sqluvput - 装置へのデータの書き込み
- sqluvend - 装置へのリンク解除
- sqluvdel - コミット済みセッションの削除
- db2VendorQueryApiVersion - 装置でサポートされる API レベルの照会
- db2VendorGetNextObj - 装置での次のオブジェクトの入手

DB2 がこれらの関数を呼び出したら、これらの関数は共用ライブラリー (UNIX ベースのシステムの場合) または DLL (Windows オペレーティング・システムの場合) において、ベンダー製品から提供されなければなりません。

注: 共用ライブラリーまたは DLL コードは、データベース・エンジン・コードの一部として実行されます。したがって、再入可能でなければならず、徹底的に

デバッグされなければなりません。関数に誤りがあると、データベースのデータ保全性を危険にさらす可能性があります。

特定のバックアップまたはリストア操作時に DB2 が呼び出す関数の順序は、以下の要因によって異なります。

- 使用されるセッションの数。
- 操作の内容 (バックアップ、リストア、ログ・アーカイブ、またはログ検索)。
- バックアップまたはリストア操作で指定された PROMPTING モード。
- データが格納されている装置の特性。
- 操作中に生じたエラー。

セッションの数

DB2 は、1 つまたは複数のデータ・ストリームまたはセッションを使用したデータベース・オブジェクトのバックアップおよびリストアをサポートしています。バックアップまたはリストアに 3 つのセッションを使用しているときには、 3 つの物理装置または LU が使用できなければなりません。ベンダー装置サポートが使用されているときには、それぞれの物理装置または LU へのインターフェースを管理するのはベンダーの関数です。 DB2 の側では、ベンダー提供の関数との間でデータ・バッファーの送受信を行うだけです。

使用されるセッションの数は、データベースのバックアップまたはリストア関数を呼び出すアプリケーションによってパラメーターとして指定されます。この値は、sqluvint によって使用される INIT-INPUT 構造で提供されます。

DB2 は、指定された数値に達するか、または sqluvint 呼び出しから SQLUV_MAX_LINK_GRANT 警告戻りコードを受信するまで、セッションの初期設定を継続します。サポートされているセッションの最大数に達したことを DB2 へ警告するために、ベンダー製品にはアクティブ・セッションの数を追跡するためのコードが必要です。 DB2 への警告が失敗すると、DB2 のセッションの初期設定要求が失敗し、結果としてすべてのセッションが終了し、バックアップまたはリストア操作全体が失敗に終わる可能性があります。

この操作がバックアップであれば、DB2 は各セッションの開始時にメディア・ヘッダー・レコードを書き込みます。このレコードには、DB2 がリストア操作時にセッションを識別するために使用する情報が入ります。 DB2 は、バックアップ・イメージの名前にシーケンス番号を付加することによって、各セッションを固有に識別します。この番号は、最初のセッションを 1 として始まり、バックアップまたはリストア操作の sqluvint 呼び出しで別のセッションが開始されるたびに 1 ずつ増加します。

バックアップ操作が正常に完了すると、DB2 はクローズする最後のセッションにメディア・トレーラーを書き込みます。このトレーラーには、バックアップ操作を実行するために使用したセッションの数を DB2 に通知する情報が組み込まれています。この情報は、リストア操作時に、すべてのセッションまたはデータ・ストリームがリストアされたことを確認するために使用されます。

エラー、警告、またはプロンプトなしの操作

バックアップの場合、DB2 は、*それぞれの* セッションごとに次の順序の呼び出しを発行します。

```
sqluvint, action = SQLUV_WRITE
```

続いて、1 個 から n 個の

```
sqluvput
```

続いて、1 個の

```
sqluvend, action = SQLUV_COMMIT
```

DB2 が sqluvend 呼び出し (アクション SQLUV_COMMIT) を発行するときには、ベンダー製品が出力データを適切に保管することを予期します。 DB2 へ SQLUV_OK の戻りコードが戻されれば、成功です。

sqluvint 呼び出しで使用される DB2-INFO 構造には、バックアップを識別するのに必要な情報が含まれます。シーケンス番号が提供されます。ベンダー製品は、この情報を保管することを選択する場合があります。 DB2 は、この情報をリストア時に使用し、リストアされるバックアップを識別します。

リストアの場合、セッションごとの呼び出しの順序は次のとおりです。

```
sqluvint, action = SQLUV_READ
```

続いて、1 個 から n 個の

```
sqluvget
```

続いて、1 個の

```
sqluvend, action = SQLUV_COMMIT
```

sqluvint 呼び出しで使用される DB2-INFO 構造内の情報には、バックアップを識別するのに必要な情報が含まれます。シーケンス番号は提供されません。 DB2 はすべてのバックアップ・オブジェクト (バックアップ時にコミットされたセッション出力) が戻されることを予期します。最初に戻されるバックアップ・オブジェクトは、シーケンス番号 1 で生成されたオブジェクトです。他のすべてのオブジェクトは順番に関係なくリストアされます。 DB2 は、すべてのオブジェクトが処理されたかどうか確認するために、メディアの末尾をチェックします。

注: すべてのベンダー製品が、バックアップ・オブジェクトの名前のレコードを保持するわけではありません。これは、テープや、容量が限られているその他のメディアにバックアップが行われる場合に特に当てはまります。リストア・セッションの初期設定時に、識別情報を利用して、必要なバックアップ・オブジェクトを、それらが必要とされるときに使用可能になるように計画することができます。これは、バックアップの保管にジュークボックスまたはロボット・システムを使用する場合に最も役立ちます。 DB2 は、必ず正しいデータがリストアされるようにするために、必ずメディア・ヘッダー (各セッションの出力の先頭レコード) をチェックします。

プロンプト・モード

バックアップまたはリストア操作の開始時には、次の 2 つのプロンプト・モードが使用可能です。

- ベンダー製品がユーザーに対してメッセージを書き込んだり、ユーザーがそれらに応答する機会のない WITHOUT PROMPTING または NOINTERRUPT。
- ユーザーがベンダー製品からメッセージを受け取り、それに応答することができる PROMPTING または INTERRUPT。

PROMPTING モードの場合、バックアップおよびリストアについて次の 3 つの応答が可能です。

- 継続

  装置へのデータの読み取りまたは書き込みの操作が再開されます。
- 装置終了

  装置が追加のデータを受信せず、セッションが終了します。
- 終了

  バックアップまたはリストア操作全体が終了します。

PROMPTING および WITHOUT PROMPTING モードの使用法については、以下のセクションで説明されています。

装置の特性

ベンダー装置サポート API のために、 2 つの一般タイプの装置が定義されています。

- メディアを交換するためのユーザー・アクションが必要とされる、容量が限られている装置。たとえばテープ・ドライブ、ディスケット、または CD-ROM ドライブ。
- 通常の操作ではユーザーがメディアを扱う必要のない大容量の装置。たとえば、ジュークボックスや高機能のロボット・メディア処理装置。

容量が限られている装置では、バックアップまたはリストア操作時に、追加メディアをロードするようユーザーに指示することが必要になる可能性があります。通常、DB2 では、バックアップまたはリストア操作のいずれにおいても、メディアがロードされる順序は重要ではありません。また、DB2 では、ベンダー・メディア処理メッセージをユーザーに渡すための機能も用意されています。このプロンプトでは、PROMPTING をオンにしてバックアップまたはリストア操作を開始することが必要です。メディア処理メッセージのテキストは、戻りコード構造の記述フィールドで指定されます。

PROMPTING がオンになっており、DB2 が sqluvput (書き込み) または sqluvget (読み取り) 呼び出しから SQLUV_ENDOFMEDIA または SQLUV_ENDOFMEDIA_NO_DATA の戻りコードを受信すると、 DB2 は次のことを行います。

- 呼び出しが sqluvput であれば、セッションに送信された最後のバッファーが再送されるようにマークする。これは、後でセッションに置かれます。

- sqluvend (アクション = SQLUV_COMMIT) を用いてセッションを呼び出す。成功すると (SQLUV_OK 戻りコード)、DB2 は次のことを行います。
  - メディア終端条件を示している戻りコード構造からのベンダー・メディア処理メッセージをユーザーに送信する。
  - 継続、装置終了、または終了の応答をするようユーザーに指示する。
- 応答が*継続* の場合、 DB2 は sqluvint 呼び出しを使用して別のセッションを初期設定します。成功すると、セッションへのデータの書き込みまたはセッションからのデータの読み取りを開始します。書き込み時にセッションを固有に識別するために、 DB2 はシーケンス番号を増加させます。シーケンス番号は、sqluvint で使用される DB2-INFO 構造内で使用できます。この番号は、セッションへ送信される最初のデータ・レコードであるメディア・ヘッダー・レコード内にあります。

  DB2 は、バックアップまたはリストア操作の開始時に要求された数、または sqluvint 呼び出し時に SQLUV_MAX_LINK_GRANT 警告を出してベンダー製品が表示した数よりも多くのセッションを開始しません。
- 応答が装置終了 の場合、 DB2 は別のセッションを初期設定せず、アクティブ・セッションの数が 1 減ります。 DB2 は、装置終了の応答によってすべてのセッションを終了させることはありません。バックアップまたはリストア操作が完了するまで、少なくとも 1 つのセッションがアクティブなままでなければなりません。
- 応答が終了 の場合は、 DB2 はバックアップまたはリストア操作を終了します。セッションを終了させるときの DB2 の処理については、 522 ページの『エラー条件が DB2 に戻される場合』を参照してください。

バックアップまたはリストア時のパフォーマンスは、使用している装置の数によって異なってくるため、並列処理を保持しておくことが重要になります。バックアップ操作の場合、残りのアクティブ・セッションで、書き込まれるデータが保持されることが分かっていない限り、継続の応答を行うようお勧めします。リストア操作の場合も、すべてのメディアが処理されるまで継続の応答を行うようお勧めします。

バックアップまたはリストア・モードが WITHOUT PROMPTING であり、 DB2 がセッションから SQLUV_ENDOFMEDIA または SQLUV_ENDOFMEDIA_NO_DATA の戻りコードを受信すると、 DB2 はセッションを終了し、別のセッションをオープンしません。バックアップまたはリストア操作が完了する前に、すべてのセッションが DB2 へメディア終端を戻すと、操作が失敗します。このため、容量が限られた装置で WITHOUT PROMPTING を使用する際には注意が必要です。ただし、大容量の装置にとっては、このモードで稼働することは有意義です。

ベンダー製品では、装置に限りなく容量があるように見えるように、メディアの装てんおよび交換アクションを DB2 から隠すことができます。一部の大容量装置は、このモードで稼働します。そのような場合は、バックアップされたすべてのデータがリストア操作の進行中に同じ順序で DB2 に戻されることが重要です。そうでなければ、データが消失する可能性がありますが、 DB2 には消失したデータを検出する方法がないため、リストア操作が成功したものと見なします。

DB2 は、各バッファーが 1 つのメディア (たとえば、テープ) に入れられるということを前提にして、データをベンダー製品へ書き込みます。ベンダー製品は、DB2 側には知らせずに、バッファーを複数のメディアに分割することができます。このような場合、複数のメディアから再構成したバッファーを DB2 に戻すのはベンダー製品の責任であるため、リストア操作中にメディアが処理される順序は重要になります。順序が正しくなければ、リストア操作は失敗します。

エラー条件が DB2 に戻される場合

バックアップまたはリストア操作時に、DB2 は、すべてのセッションが正常に完了するか、そうでなければバックアップまたはリストア操作全体が失敗することを予期します。セッションは、sqluvend (アクション = SQLUV_COMMIT) の呼び出し時に、 SQLUV_OK 戻りコードで DB2 に正常終了を知らせます。

リカバリー不能エラーが検出されると、セッションは DB2 によって終了されます。これらのエラーは DB2 エラーとなるか、ベンダー製品から DB2 へ戻されるエラーとなります。バックアップまたはリストア操作が完了するにはすべてのセッションが正常にコミットされなければならないため、 1 つが失敗すると、DB2 はその操作と関連した他のセッションも終了させてしまいます。

ベンダー製品が DB2 からの呼び出しに対してリカバリー不能を示す戻りコードで応答する場合、ベンダー製品は、オプションで、RETURN-CODE 構造の記述フィールドに置かれたメッセージ・テキストを使用して、追加情報を提供することができます。このメッセージ・テキストは、訂正の処置をとれるように DB2 情報と共に表示されます。

バックアップのシナリオとして、あるセッションが正常にコミットされたものの、バックアップ操作に関連した他のセッションでリカバリー不能エラーが発生したというケースが考えられます。バックアップ操作が成功するためにはすべてのセッションが正常に完了しなければならないため、 DB2 はコミットされたセッション内の出力データを削除する必要があります。 DB2 は sqluvdel 呼び出しを発行して、オブジェクトの削除を要求します。この呼び出しは入出力セッションとは見なされず、バックアップ・オブジェクトの削除に必要な接続を初期化したり終了したりする機能を果たします。

DB2-INFO 構造にはシーケンス番号は含まれていません。 sqluvdel は、 DB2-INFO 構造内の残りのパラメーターと一致するバックアップ・オブジェクトをすべて削除します。

警告条件

DB2 は、ベンダー製品から警告戻りコードを受信する可能性があります。たとえば、装置が作動不能である場合や、その他の訂正可能条件が生じた場合などです。これは、読み取りと書き込みの両方の操作に当てはまります。

sqluvput および sqluvget 呼び出し時に、ベンダーは戻りコードを SQLUV_WARNING に設定することができます。さらに、オプションで RETURN-CODE 構造の記述フィールドに置かれたメッセージ・テキストを使用して、追加情報を提供することができます。このメッセージ・テキストはユーザーに表示され、訂正の処置をとることができます。ユーザーは、3 つの方法 (継続、装置終了、または終了) の 1 つで応答することができます。

- 応答が継続 の場合は、DB2 はバックアップ操作中に sqluvput を使用してバッファーへの再書き込みを試みます。リストア操作時に、DB2 は sqluvget 呼び出しを発行して、次のバッファーを読み取ります。
- 応答が装置終了 または終了 の場合、DB2 はリカバリー不能エラー発生後に応答するときと同じ方法で、バックアップまたはリストア操作全体を終了させます (たとえば、アクティブ・セッションを終了し、コミット済みセッションを削除する)。

操作上のヒント

このセクションでは、ベンダー製品を作成するためのヒントをいくつか提供します。

履歴ファイル

履歴ファイルは、データベース・リカバリー操作に役立てることができます。このファイルは、各データベースに関連しており、バックアップまたはリストア操作が行われるたびに自動的に更新されます。ファイル内の情報は、以下の機能を使用して表示、更新、または除去することができます。

- コントロール・センター
- コマンド行プロセッサー (CLP)
  - LIST HISTORY コマンド
  - UPDATE HISTORY FILE コマンド
  - PRUNE HISTORY コマンド
- API
  - db2HistoryOpenScan
  - db2HistoryGetEntry
  - db2HistoryCloseScan
  - db2HistoryUpdate
  - db2Prune

ファイルのレイアウトについては、db2HistData を参照してください。

バックアップ操作が完了すると、1 つまたは複数のレコードがファイルへ書き込まれます。バックアップ操作の出力がベンダー装置に送られた場合で、LOAD キーワードが使用された場合、履歴レコードの DEVICE フィールドには 0 が入ります。バックアップ操作が TSM に送られた場合、DEVICE フィールドには A が入ります。 LOCATION フィールドには、以下のいずれかが入ります。

- バックアップ操作が呼び出されたときに指定されたベンダー・ファイル名。
- ベンダー・ファイル名が指定されていない場合は、共用ライブラリーの名前。

このオプションの指定についての詳細は、524 ページの『ベンダー製品を使用してバックアップまたはリストア操作を呼び出す』を参照してください。

LOCATION フィールドはコントロール・センター、CLP、または API を使用して更新できます。バックアップ・イメージを保存するために容量の限られた装置 (たとえば、取り外し可能メディア) が使用されており、そのメディアが異なる記憶場所 (たとえばオフ・サイト) へ物理的に移動された場合、バックアップ情報のロケー

ションを更新することができます。この場合に、リストア操作が必要になったら、履歴ファイルを使用してバックアップ・イメージを見つけることができます。

ベンダー製品を使用してバックアップまたはリストア操作を呼び出す

DB2 バックアップまたは DB2 リストア・ユーティリティーを以下の方法で呼び出す場合に、ベンダー製品を指定できます。

- コントロール・センター
- コマンド行プロセッサー (CLP)
- アプリケーション・プログラミング・インターフェース (API)。

コントロール・センター

コントロール・センターは、 DB2 と共に出荷されるデータベース管理用のグラフィカル・ユーザー・インターフェースです。

指定内容	バックアップまたはリストア操作用のコントロール・センター入力変数
ベンダー装置を使用することと、ライブラリー名	*Use Library*。ライブラリー名 (UNIX ベースのシステムの場合) または DLL 名 (Windows オペレーティング・システムの場合) を指定します。
セッションの数	*Sessions*
ベンダー・オプション	サポートされていません
ベンダー・ファイル名	サポートされていません
転送バッファー・サイズ	バックアップの場合: *Size of each Buffer*。リストアの場合: 適用されません。

コマンド行プロセッサー (CLP)

コマンド行プロセッサー (CLP) を使用して DB2 BACKUP DATABASE または RESTORE DATABASE コマンドを呼び出すことができます。

指定内容	コマンド行プロセッサー・パラメーター	
	バックアップ用	リストア用
ベンダー装置を使用することと、ライブラリー名	*library-name*	*shared-library*
セッションの数	*num-sessions*	*num-sessions*
ベンダー・オプション	サポートされていません	サポートされていません
ベンダー・ファイル名	サポートされていません	サポートされていません
転送バッファー・サイズ	*buffer-size*	*buffer-size*

バックアップおよびリストア API 関数呼び出し

2 つの API 関数呼び出しがバックアップおよびリストア操作をサポートしています。バックアップ用には db2Backup、リストア用には db2Restore です。

指定内容	API パラメーター (db2Backup および db2Restore)
ベンダー装置を使用することと、ライブラリー名	構造 *sqlu_media_list* にはメディア・タイプ SQLU_OTHER_MEDIA を指定し、構造 *sqlu_vendor* には *shr_lib* に共用ライブラリーまたは DLL を指定します。
セッションの数	構造 *sqlu_media_list* に *sessions* を指定します。
ベンダー・オプション	*PVendorOptions*
ベンダー・ファイル名	構造 *sqlu_media_list* にはメディア・タイプ SQLU_OTHER_MEDIA を指定し、構造 *sqlu_vendor* には *filename* にファイル名を指定します。
転送バッファー・サイズ	*BufferSize*

関連資料:

- 525 ページの『sqluvint - 初期設定と装置へのリンク』
- 528 ページの『sqluvget - 装置からのデータの読み取り』
- 530 ページの『sqluvput - 装置へのデータの書き込み』
- 532 ページの『sqluvend - 装置のリンク解除およびリソースの解放』
- 534 ページの『sqluvdel - コミット済みセッションの削除』
- 538 ページの『DB2-INFO』
- 540 ページの『VENDOR-INFO』
- 541 ページの『INIT-INPUT』
- 542 ページの『INIT-OUTPUT』
- 542 ページの『DATA』
- 543 ページの『RETURN-CODE』
- 535 ページの『db2VendorQueryApiVersion - 装置でサポートされる API レベルの照会』
- 536 ページの『db2VendorGetNextObj - 装置での次のオブジェクトの入手』

sqluvint - 初期設定と装置へのリンク

この関数は、 DB2 とベンダー製品との間の論理リンクの初期設定および確立に関する情報を提供するために呼び出されます。

許可:

以下のいずれかが必要です。

- *sysadm*
- *dbadm*

必要な接続:

データベース

API 組み込みファイル:

sql.h

C API 構文:

```
/* File: sqluvend.h */
/* API: Initialize and Link to Device */
/* ... */
int sqluvint (
  struct Init_input   *,
  struct Init_output  *,
  struct Return_code  *);
/* ... */
```

API パラメーター:

Init_input
入力。ベンダー装置との論理リンクを確立するための、 DB2 が提供する情報を含む情報。

Init_output
出力。ベンダー装置が戻した出力を含む構造。

Return_code
出力。 DB2 へ渡される戻りコードと短いテキスト記述を含む構造。

使用上の注意:

それぞれのメディア入出力セッションごとに、 DB2 はこの関数を呼び出して装置ハンドルを取得します。何かの理由で初期設定時にベンダー関数にエラーが発生すると、戻りコードを通してそのことを知らせます。エラーを示す戻りコードであれば、DB2 は、 **sqluvend** 関数を呼び出して操作を終了させることがあります。可能な戻りコードと、これらのそれぞれに対する DB2 の反応については、戻りコード表 (527 ページの表 83 を参照) に記載されています。

INIT-INPUT 構造には、ベンダー製品がバックアップまたはリストアを続行できるかどうかを判別するために使用できるエレメントが含まれます。

- size_HI_order および size_LOW_order

  バックアップの見積サイズです。これらを使用すると、ベンダー装置がバックアップ・イメージのサイズを処理できるかどうかを判別することができます。また、バックアップを保存するために必要な取り外し可能メディアの容量を見積もることができます。問題が予期される場合は、最初の **sqluvint** 呼び出しで失敗した方が有益である可能性があります。

- req_sessions

  要求したセッションの数を見積サイズやプロンプト・レベルと関連付けて使用して、バックアップまたはリストア操作が可能かどうかを判別することができます。

- prompt_lvl

プロンプト・レベルは、取り外し可能メディアの変更 (たとえば、テープ・ドライブへ他のテープを入れる) などのアクションを指示できるかどうかをベンダーに示します。これは、ユーザーへの指示方法がないために操作が続行できないことを示唆する場合があります。

プロンプト・レベルが WITHOUT PROMPTING であり、取り外し可能メディアの容量が要求されたセッション数よりも大きい場合、 DB2 は操作を正常に完了することはできません。

DB2 は、DB2-INFO 構造内のフィールドを使用して、書き込まれているバックアップまたは読み取られるリストアを指定します。アクション = SQLUV_READ の場合、ベンダー製品は指定されたオブジェクトの存在を調べなければなりません。見つからなければ、DB2 が適切な処置をとれるように戻りコードが SQLUV_OBJ_NOT_FOUND に設定される必要があります。

初期設定が正常に完了した後、DB2 は他のデータ転送機能を発行して継続しますが、 **sqluvend** 呼び出しでいつでもセッションを終了させることができます。

戻りコード:

表 83. sqluvint についての有効な戻りコードと結果の DB2 アクション

ヘッダー・ファイルのリテラル	説明	予想される次の呼び出し	その他のコメント
SQLUV_OK	操作が成功した。	sqluvput、sqluvget (コメント参照)	アクション = SQLUV_WRITE であれば、次の呼び出しは sqluvput (データのバックアップ) です。アクション = SQLUV_READ であれば、SQLUV_OK を戻す前に、指定されたオブジェクトの存在を確認します。次の呼び出しは、sqluvget (データのリストア) になります。
SQLUV_LINK_EXIST	セッションがすでに活動化されている。	ありません	セッションの開始に失敗しました。セッションに割り振られていたメモリーを解放し、終了します。セッションが確立されなかったため、sqluvend 呼び出しは受信されません。
SQLUV_COMM_ERROR	装置との通信エラー	ありません	セッションの開始に失敗しました。セッションに割り振られていたメモリーを解放し、終了します。セッションが確立されなかったため、sqluvend 呼び出しは受信されません。
SQLUV_INV_VERSION	DB2 とベンダー製品に互換性がない。	ありません	セッションの開始に失敗しました。セッションに割り振られていたメモリーを解放し、終了します。セッションが確立されなかったため、sqluvend 呼び出しは受信されません。
SQLUV_INV_ACTION	無効なアクションが要求された。これは、パラメーターの組み合わせにより不可能な操作が生じたことを示すためにも使用される。	ありません	セッションの開始に失敗しました。セッションに割り振られていたメモリーを解放し、終了します。セッションが確立されなかったため、sqluvend 呼び出しは受信されません。
SQLUV_NO_DEV_AVAIL	現在使用できる装置がない。	ありません	セッションの開始に失敗しました。セッションに割り振られていたメモリーを解放し、終了します。セッションが確立されなかったため、sqluvend 呼び出しは受信されません。
SQLUV_OBJ_NOT_FOUND	指定されたオブジェクトが見つからない。これは、sqluvint 呼び出しでのアクションが 'R' (読み取り) であり、 DB2-INFO 構造で指定された基準に基づいて要求されたオブジェクトが見つからないときに使用される。	ありません	セッションの開始に失敗しました。セッションに割り振られていたメモリーを解放し、終了します。セッションが確立されなかったため、sqluvend 呼び出しは受信されません。

表 83. sqluvint についての有効な戻りコードと結果の DB2 アクション (続き)

ヘッダー・ファイルのリテラル	説明	予想される次の呼び出し	その他のコメント
SQLUV_OBJS_FOUND	2 つ以上のオブジェクトが、指定された基準に一致する。これは、sqluvint 呼び出しでのアクションが 'R' (読み取り) であり、 DB2-INFO 構造内の基準と一致するオブジェクトが 2 つ以上あるときに発生する。	ありません	セッションの開始に失敗しました。セッションに割り振られていたメモリーを解放し、終了します。セッションが確立されなかったため、sqluvend 呼び出しは受信されません。
SQLUV_INV_USERID	指定されたユーザー ID が無効である。	ありません	セッションの開始に失敗しました。セッションに割り振られていたメモリーを解放し、終了します。セッションが確立されなかったため、sqluvend 呼び出しは受信されません。
SQLUV_INV_PASSWORD	提供されたパスワードが無効である。	ありません	セッションの開始に失敗しました。セッションに割り振られていたメモリーを解放し、終了します。セッションが確立されなかったため、sqluvend 呼び出しは受信されません。
SQLUV_INV_OPTIONS	ベンダー・オプション・フィールド内で無効なオプションが検出された。	ありません	セッションの開始に失敗しました。セッションに割り振られていたメモリーを解放し、終了します。セッションが確立されなかったため、sqluvend 呼び出しは受信されません。
SQLUV_INIT_FAILED	初期設定が失敗し、セッションが終了される。	ありません	セッションの開始に失敗しました。セッションに割り振られていたメモリーを解放し、終了します。セッションが確立されなかったため、sqluvend 呼び出しは受信されません。
SQLUV_DEV_ERROR	装置エラー	ありません	セッションの開始に失敗しました。セッションに割り振られていたメモリーを解放し、終了します。セッションが確立されなかったため、sqluvend 呼び出しは受信されません。
SQLUV_MAX_LINK_GRANT	最大数のリンクが確立された。	sqluvput、sqluvget (コメント参照)	これは、DB2 からの警告として扱われます。この警告は、サポート可能なセッションの最大数に達しているので、これ以上ベンダー製品とのセッションをオープンしないよう DB2 に知らせるものです (注意: これは、装置の可用性が原因となっている可能性もあります)。アクション = SQLUV_WRITE (BACKUP) であれば、次の呼び出しは sqluvput です。アクション = SQLUV_READ であれば、SQLUV_MAX_LINK_GRANT を戻す前に、指定されたオブジェクトの存在を確認します。次の呼び出しは、sqluvget (データのリストア) になります。
SQLUV_IO_ERROR	入出力エラー	ありません	セッションの開始に失敗しました。セッションに割り振られていたメモリーを解放し、終了します。セッションが確立されなかったため、sqluvend 呼び出しは受信されません。
SQLUV_NOT_ENOUGH_SPACE	バックアップ・イメージ全体を格納するための十分なスペースがない (サイズの見積もりは 64 ビットのバイト数で提供される)。	ありません	セッションの開始に失敗しました。セッションに割り振られていたメモリーを解放し、終了します。セッションが確立されなかったため、sqluvend 呼び出しは受信されません。

sqluvget - 装置からのデータの読み取り

初期設定の後、この関数を呼び出して装置からデータを読み取ることができます。

許可:

以下のいずれかが必要です。

- *sysadm*

- *dbadm*

必要な接続:

データベース

API 組み込みファイル:

sqluvend.h

C API 構文:

```
/* File: sqluvend.h */
/* API: Reading Data from Device */
/* ... */
int sqluvget (
  void * pVendorCB,
  struct Data  *,
  struct Return_code  *);
/* ... */

typedef struct Data
}
   sqlint32  obj_num;
   sqlint32  buff_size;
   sqlint32  actual_buff_size;
   void      *dataptr;
   void      *reserve;
{ Data;
```

API パラメーター:

pVendorCB
入力。 DATA 構造 (データ・バッファーを含む) および Return_code 用に割り振られたスペースを指すポインター。

Data 入出力。データ 構造を指すポインター。

Return_code
出力。 API 呼び出しからの戻りコード。

obj_num
検索するバックアップ・オブジェクトを指定します。

buff_size
使用するバッファー・サイズを指定します。

actual_buff_size
読み取られる、または書き込まれる実際のバイト数を指定します。この値は、実際に読み取られたデータのバイト数を示す出力に設定してください。

dataptr
データ・バッファーを指すポインター。

reserve
将来の利用のために予約されています。

使用上の注意:

この関数はリストア・ユーティリティーで使用されます。

戻りコード:

表 84. sqluvget についての有効な戻りコードと結果の DB2 アクション

ヘッダー・ファイルのリテラル	説明	予想される次の呼び出し	その他のコメント
SQLUV_OK	操作が成功した。	sqluvget	DB2 はデータを処理します。
SQLUV_COMM_ERROR	装置との通信エラー	sqluvend、アクション = SQLU_ABORT[a]	セッションは終了します。
SQLUV_INV_ACTION	無効なアクションが要求された。	sqluvend、アクション = SQLU_ABORT[a]	セッションは終了します。
SQLUV_INV_DEV_HANDLE	無効な装置ハンドル	sqluvend、アクション = SQLU_ABORT[a]	セッションは終了します。
SQLUV_INV_BUFF_SIZE	無効なバッファー・サイズが指定された。	sqluvend、アクション = SQLU_ABORT[a]	セッションは終了します。
SQLUV_DEV_ERROR	装置エラー	sqluvend、アクション = SQLU_ABORT[a]	セッションは終了します。
SQLUV_WARNING	警告。これは、DB2 にメディアの終わりを示すためには使用されない (その目的では、SQLUV_ENDOFMEDIA または SQLUV_ENDOFMEDIA_NO_DATA が使用される)。ただし、装置の作動不能条件がこの戻りコードによって示される可能性がある。	sqluvget、または sqluvend、アクション =SQLU_ABORT	
SQLUV_LINK_NOT_EXIST	リンクが現在存在していない。	sqluvend、アクション = SQLU_ABORT[a]	セッションは終了します。
SQLUV_MORE_DATA	操作が成功し、さらにデータが使用可能である。	sqluvget	
SQLUV_ENDOFMEDIA_NO_DATA	メディアが終了し、0 バイトが読み取られた (たとえば、テープの終わり)。	sqluvend	
SQLUV_ENDOFMEDIA	メディアが終了し、> 0 バイトが読み取られた (たとえば、テープの終わり)。	sqluvend	DB2 はデータを処理し、メディア終端条件を処理します。
SQLUV_IO_ERROR	入出力エラー	sqluvend、アクション = SQLU_ABORT[a]	セッションは終了します。
次の呼び出し: [a] 次の呼び出しが sqluvend、action = SQLU_ABORT である場合には、このセッションおよび他のすべてのセッションは終了します。			

sqluvput - 装置へのデータの書き込み

初期設定の後、この関数を使用して装置へデータを書き込むことができます。

許可:

以下のいずれかが必要です。

- *sysadm*
- *dbadm*

必要な接続:

データベース

API 組み込みファイル:

sqluvend.h

C API 構文:

```
/* File: sqluvend.h */
/* API: Writing Data to Device */
/* ... */
int sqluvput (
  void * pVendorCB,
  struct Data  *,
  struct Return_code  *);
/* ... */

typedef struct Data
}
   sqlint32  obj_num;
   sqlint32  buff_size;
   sqlint32  actual_buff_size;
   void      *dataptr;
   void      *reserve;
{ Data;
```

API パラメーター:

pVendorCB
入力。 DATA 構造 (データ・バッファーを含む) および Return_code 用に割り振られたスペースを指すポインター。

Data 出力。書き込まれるデータが入れられたデータ・バッファー。

Return_code
出力。 API 呼び出しからの戻りコード。

obj_num
検索するバックアップ・オブジェクトを指定します。

buff_size
使用するバッファー・サイズを指定します。

actual_buff_size
読み取られる、または書き込まれる実際のバイト数を指定します。この値は、実際に読み取られたデータのバイト数を示すように設定してください。

dataptr
データ・バッファーを指すポインター。

reserve
将来の利用のために予約されています。

使用上の注意:

この関数はバックアップ・ユーティリティーで使用されます。

戻りコード:

表 85. sqluvput についての有効な戻りコードと結果の DB2 アクション

ヘッダー・ファイルのリテラル	説明	予想される次の呼び出し	その他のコメント
SQLUV_OK	操作が成功した。	完了 (たとえば、DB2 にこれ以上データがなくなった) 後、 sqluvput または sqluvend	他のプロセスに操作の成功を通知します。
SQLUV_COMM_ERROR	装置との通信エラー	sqluvend、アクション = SQLU_ABORT[a]	セッションは終了します。
SQLUV_INV_ACTION	無効なアクションが要求された。	sqluvend、アクション = SQLU_ABORT[a]	セッションは終了します。

表 85. sqluvput についての有効な戻りコードと結果の DB2 アクション (続き)

ヘッダー・ファイルのリテラル	説明	予想される次の呼び出し	その他のコメント
SQLUV_INV_DEV_HANDLE	無効な装置ハンドル	sqluvend、アクション = SQLU_ABORT[a]	セッションは終了します。
SQLUV_INV_BUFF_SIZE	無効なバッファー・サイズが指定された。	sqluvend、アクション = SQLU_ABORT[a]	セッションは終了します。
SQLUV_ENDOFMEDIA	メディアが終了した (たとえば、テープの終了)。	sqluvend	
SQLUV_DATA_RESEND	装置が再びバッファーを送信するよう要求した。	sqluvput	DB2 は最新のバッファーを再度送信します。これは一度だけ行われます。
SQLUV_DEV_ERROR	装置エラー	sqluvend、アクション = SQLU_ABORT[a]	セッションは終了します。
SQLUV_WARNING	警告。これは、DB2 にメディアの終わりを示すためには使用されない (その目的には SQLUV_ENDOFMEDIA が使用される)。ただし、装置の作動不能条件がこの戻りコードによって示される可能性がある。	sqluvput	
SQLUV_LINK_NOT_EXIST	リンクが現在存在していない。	sqluvend、アクション = SQLU_ABORT[a]	セッションは終了します。
SQLUV_IO_ERROR	入出力エラー	sqluvend、アクション = SQLU_ABORT[a]	セッションは終了します。

次の呼び出し:

[a] 次の呼び出しが sqluvend、action = SQLU_ABORT である場合には、このセッションおよび他のすべてのセッションは終了します。コミット済みセッションは、sqluvint、sqluvdel、 sqluvend の順序の呼び出しで削除されます。

sqluvend - 装置のリンク解除およびリソースの解放

装置を終了またはリンク解除し、関連したリソースをすべて解放します。ベンダーは DB2 へ戻る前に、使用していないリソース (たとえば、割り振られたスペースやファイル・ハンドル) を解放しなければなりません。

許可:

以下のいずれかが必要です。

- *sysadm*
- *dbadm*

必要な接続:

データベース

API 組み込みファイル:

sql.h

C API 構文:

```
/* File: sqluvend.h */
/* API: Unlink the Device and Release its Resources */
/* ... */
int sqluvend (
  sqlint32 action,
```

```
  void * pVendorCB,
  struct Init_output  *,
  struct Return_code  *);
/* ... */
```

API パラメーター:

action 入力。セッションのコミットまたは打ち切りに使用されます。

- SQLUV_COMMIT (0 = コミット)
- SQLUV_ABORT (1 = 打ち切り)

pVendorCB
入力。 Init_output 構造を指すポインター。

Init_output
出力。割り振り解除された Init_output 用のスペース。アクションがコミットであれば、データはバックアップのために保管用のストレージへコミットされています。アクションが打ち切りであれば、データはバックアップのために除去されます。

Return code
出力。 API 呼び出しからの戻りコード。

使用上の注意:

この関数はオープンされたセッションごとに呼び出されます。可能なアクション・コードは、次の 2 つがあります。

- コミット

  このセッションへのデータの出力またはセッションからのデータの読み取りが完了します。

  書き込み (バックアップ) セッションの場合、ベンダーが SQLUV_OK の戻りコードと共に DB2 へ戻ると、 DB2 は、出力データがベンダー製品によって適切に保管されており、後の **sqluvint** 呼び出しで参照されたときにアクセスできるものと判断します。

  読み取り (リストア) セッションで、ベンダーが SQLUV_OK の戻りコードと共に DB2 へ戻った場合、データは再び必要になる可能性があるため、削除すべきではありません。

  ベンダーが SQLUV_COMMIT_FAILED を戻す場合、 DB2 はバックアップ操作またはリストア操作全体に問題があると判断します。すべてのアクティブ・セッションは、アクション = SQLUV_ABORT の **sqluvend** 呼び出しで終了されます。バックアップ操作の場合、コミット済みセッションは **sqluvint**、**sqluvdel**、および **sqluvend** の順序の呼び出しを受信します。

- 打ち切り

  DB2 によって問題が生じているときには、セッションではデータの読み取りまたは書き込みは行われません。

  書き込み (バックアップ) セッションの場合、ベンダーは部分的な出力データ・セットを削除しなければなりません。削除されていれば、SQLUV_OK 戻りコード

を使用します。 DB2 はバックアップ全体に問題があると判断します。すべてのアクティブ・セッションは、action = SQLUV_ABORT の **sqluvend** 呼び出しで終了され、コミット済みセッションは、**sqluvint**、**sqluvdel**、および **sqluvend** の順序の呼び出しを受信します。

読み取り (リストア) セッションの場合、ベンダーはデータを削除してはなりません (再び必要になる可能性があるからです)。ただし、終結処理を行い、SQLUV_OK 戻りコードを出して DB2 に戻ってください。 DB2 は action = SQLUV_ABORT の **sqluvend** 呼び出しですべてのリストア・セッションを終了させます。ベンダーが SQLUV_ABORT_FAILED を DB2 へ戻す場合は、呼び出し側にこのエラーは通知されません。これは DB2 が最初の致命的な障害は戻し、その後に続く障害は無視するためです。この場合、action = SQLUV_ABORT の **sqluvend** を呼び出した DB2 に関して、最初の致命的なエラーが発生しています。

戻りコード:

表 86. sqluvend についての有効な戻りコードと結果の DB2 アクション

ヘッダー・ファイルのリテラル	説明	予想される次の呼び出し	その他のコメント
SQLUV_OK	操作が成功した。	ありません	このセッションに割り振られていたメモリーをすべて解放し、終了します。
SQLUV_COMMIT_FAILED	コミット要求が失敗した。	ありません	このセッションに割り振られていたメモリーをすべて解放し、終了します。
SQLUV_ABORT_FAILED	打ち切り要求が失敗した。	ありません	

sqluvdel - コミット済みセッションの削除

コミット済みセッションを削除します。

許可:

以下のいずれかが必要です。

- *sysadm*
- *dbadm*

必要な接続:

データベース

API 組み込みファイル:

sqluvend.h

C API 構文:

```
/* File: sqluvend.h */
/* API: Delete Committed Session */
/* ... */
int sqluvdel (
  struct Init_input   *,
  struct Init_output  *,
  struct Return_code  *);
/* ... */
```

API パラメーター:

Init_input
入力。 Init_input および Return_code 用に割り振られたスペース。

Return_code
出力。 API 呼び出しからの戻りコード。 Init_input 構造によって指示されたオブジェクトは削除されます。

使用上の注意:

複数のセッションがオープンしていて、いくつかのセッションはコミットされたが 1 つが失敗したというような場合、この関数が呼び出されて、コミット済みセッションを削除します。シーケンス番号は指定されません。特定のバックアップ操作時に作成されたすべてのオブジェクトを検出して削除するのは、 **sqluvdel** の責任です。 INIT-INPUT 構造の情報が、削除する出力データを識別するために使用されます。ベンダー装置からバックアップ・オブジェクトを削除するのに必要な接続またはセッションを確立するのは、 **sqluvdel** の責任です。この呼び出しからの戻りコードが SQLUV_DELETE_FAILED であれば、DB2 は呼び出し側へ通知しません。DB2 は最初の致命的な障害は戻し、その後に続く障害は無視するという方式なので、このように行います。この場合、**sqluvdel** を呼び出した DB2 に関して、最初の致命的エラーが発生しています。

戻りコード:

表 87. sqluvdel についての有効な戻りコードと結果の DB2 アクション

ヘッダー・ファイルのリテラル	説明	予想される次の呼び出し	その他のコメント
SQLUV_OK	操作が成功した。	ありません	
SQLUV_DELETE_FAILED	削除要求が失敗した。	ありません	

db2VendorQueryApiVersion - 装置でサポートされる API レベルの照会

この関数は、ベンダー・ライブラリーによってサポートされるベンダー API のレベルを判別するために呼び出されます。ベンダー・ライブラリーが DB2 と互換性がない場合、そのベンダー・ライブラリーは使用されません。

ベンダー・ライブラリーに、ログ用にこの API がインプリメントされていない場合、ベンダー・ライブラリーは使用できず、DB2 はエラーをレポートします。このことは、既存のベンダー・ライブラリーを処理しているイメージには影響しません。

許可:

以下のいずれかが必要です。

- *sysadm*
- *dbadm*

必要な接続:

データベース。

API 組み込みファイル:

db2VendorApi.h

C API 構文:

```
void db2VendorQueryApiVersion(db2Uint32 *supportedVersion);
```

API パラメーター:

supportedVersion
出力。ベンダー・ライブラリーでサポートされるベンダー API のバージョンを戻します。

使用上の注意:

この関数は、他のベンダー API の呼び出し前に呼び出されます。

db2VendorGetNextObj - 装置での次のオブジェクトの入手

この関数は、検索条件に一致する次のオブジェクトを入手するよう (sqluvint を使用して) 照会がセットアップされた後に呼び出されます。一度にセットアップできるのは、イメージかログ・ファイルに対する 1 つの検索だけです。

許可:

以下のいずれかが必要です。

- *sysadm*
- *dbadm*

必要な接続:

データベース。

API 組み込みファイル:

db2VendorApi.h

C API 構文:

```
int db2VendorGetNextObj(void *vendorCB,
  struct db2VendorQueryInfo *queryInfo,
  struct Return_code *returnCode);

typedef struct db2VendorQueryInfo
{
   char               db2Instance[SQL_INSTNAME_SZ + 1];
   char               dbname[SQL_DBNAME_SZ + 1];
   char               dbalias[SQL_ALIAS_SZ + 1];
   char               timestamp[SQLU_TIME_STAMP_LEN + 1];
   char               filename[DB2VENDOR_MAX_FILENAME_SZ + 1];
   char               owner[DB2VENDOR_MAX_OWNER_SZ + 1];
   char               mgmtClass[DB2VENDOR_MAX_MGMTCLASS_SZ + 1];
   char               oldestLogfile[DB2_LOGFILE_NAME_LEN + 1];
   db2Uint16          sequenceNum
   SQL_PDB_NODE_TYPE  dbPartitionNum;
   db2Uint32          type;
   db2Uint64          sizeEstimate;
} db2VendorQueryInfo;
```

API パラメーター:

vendorCB
入力。ベンダー・ライブラリーによって割り振られたスペースへのポインター。

queryInfo
出力。ベンダー・ライブラリーによって記入される *db2VendorQueryInfo* 構造へのポインター。

returnCode
出力。 API 呼び出しからの戻りコード。

db2Instance
オブジェクトが属するインスタンスの名前を指定します。

dbname
オブジェクトが属するデータベースの名前を指定します。

dbalias
オブジェクトが属するデータベースの別名を指定します。

timestamp
バックアップ・イメージを識別するのに使用されるタイム・スタンプを指定します。オブジェクトがバックアップ・イメージである場合にのみ有効です。

filename
オブジェクトがロード・コピー・イメージかアーカイブ・ログ・ファイルの場合に、オブジェクトの名前を指定します。

owner オブジェクトの所有者を指定します。

mgmtClass
オブジェクトが保管される管理クラスを指定します (TSM によって使用される)。

oldestLogfile
バックアップ・イメージに保管された一番古いログ・ファイルを指定します。

sequenceNum
バックアップ・イメージのファイル拡張子を指定します。オブジェクトがバックアップである場合にのみ有効です。

dbPartitionNum
オブジェクトが属するデータベース・パーティションの数を指定します。

type オブジェクトがバックアップ・イメージである場合に、イメージ・タイプを指定します。

sizeEstimate
オブジェクトの見積もりサイズを指定します。

使用上の注意:

すべてのフィールドが、各オブジェクトまたは各ベンダーに関係するわけではありません。記入する必要のある必須フィールドは、db2Instance、 dbname、 dbalias、 timestamp (イメージ用)、 filename (ログおよびロード・コピー・イメージ用)、 owner、 sequenceNum (イメージ用)、および dbPartitionNum です。残りのフィール

ドは、定義する特定のベンダー用に残されます。特定のフィールドが関係しない場合、ストリングの場合には "" に、そして数値タイプの場合には 0 に初期設定する必要があります。

DB2-INFO

この構造には、ベンダー装置に DB2 を識別させる情報が含まれます。

表 88. DB2-INFO 構造のフィールド： フィールドはすべて、NULL 終了ストリングです。

フィールド名	データ・タイプ	説明
DB2_id	char	DB2 製品の ID。指すストリングの最大長は 8 文字です。
version	char	DB2 製品の現行バージョン。指すストリングの最大長は 8 文字です。
release	char	DB2 製品の現行リリース。重要性がなければ NULL に設定します。指すストリングの最大長は 8 文字です。
level	char	DB2 製品の現行レベル。重要性がなければ NULL に設定します。指すストリングの最大長は 8 文字です。
action	char	実行するアクションを指定します。指すストリングの最大長は 1 文字です。
filename	char	バックアップ・イメージの識別に使用されるファイル名。 NULL であれば、*server_id*、*db2instance*、 *dbname* および *timestamp* によってバックアップ・イメージが固有に識別されます。指すストリングの最大長は 255 文字です。
server_id	char	データベースが存在するサーバーを識別するユニーク名。指すストリングの最大長は 8 文字です。
db2instance	char	db2instance ID。これはコマンドを呼び出すユーザー ID です。指すストリングの最大長は 8 文字です。
type	char	作成するバックアップのタイプ、または実行するリストアのタイプを指定します。以下の値を指定することができます。 アクションが SQLUV_WRITE の場合: 0 - データベースの全バックアップ 3 - 表スペース・レベルのバックアップ アクションが SQLUV_READ の場合: 0 - 全リストア 3 - オンラインの表スペース・リストア 4 - 表スペース・リストア 5 - 履歴ファイルのリストア
dbname	char	バックアップまたはリストアするデータベースの名前。指すストリングの最大長は 8 文字です。
alias	char	バックアップまたはリストアするデータベースの別名。指すストリングの最大長は 8 文字です。
timestamp	char	バックアップ・イメージを識別するのに使用されるタイム・スタンプ。指すストリングの最大長は 26 文字です。

表 88. DB2-INFO 構造のフィールド (続き): フィールドはすべて、NULL 終了ストリングです。

フィールド名	データ・タイプ	説明
sequence	char	バックアップ・イメージのファイル拡張子を指定します。書き込み操作の場合、最初のセッションの値は 1 で、 sqluvint 呼び出しで他のセッションが開始されるたびに、値が 1 ずつ増加します。読み取り操作の場合、値は常にゼロです。指すストリングの最大長は 3 文字です。
obj_list	struct sqlu_gen_list	将来の利用のために予約されています。
max_bytes_per_txn	sqlint32	ユーザーによって指定された転送バッファー・サイズをバイト単位でベンダーに指定します。
image_filename	char	将来の利用のために予約されています。
reserve	void	将来の利用のために予約されています。
nodename	char	バックアップが生成されたノードの名前。
password	char	バックアップが生成されたノードのパスワード。
owner	char	バックアップの開始元の ID。
mcNameP	char	管理クラス。
nodeNum	SQL_PDB_NODE_TYPE	ノード番号。ベンダー・インターフェースでは、255 より大きい番号がサポートされます。

バックアップ・イメージは、*filename* か、または *server_id*、*db2instance*、*type*、*dbname*、および *timestamp* によって固有に識別されます。 *sequence* によって指定されるシーケンス番号はファイル拡張子を識別します。バックアップ・イメージをリストアするときには、同じ値を使用してバックアップ・イメージを検索しなければなりません。ベンダー製品によっては、*filename* が使用された場合に他のパラメーターが NULL に設定されたり、その逆が起こることがあります。

言語構文:

C 構造

```
/* File: sqluvend.h */
/* ... */
typedef struct DB2_info
{
  char                  *DB2_id;
  char                  *version;
  char                  *release;
  char                  *level;
  char                  *action;
  char                  *filename;
  char                  *server_id;
  char                  *db2instance;
  char                  *type;
  char                  *dbname;
  char                  *alias;
  char                  *timestamp;
  char                  *sequence;
  struct sqlu_gen_list  *obj_list;
  long                  max_bytes_per_txn;
  char                  *image_filename;
  void                  *reserve;
  char                  *nodename;
  char                  *password;
  char                  *owner;
```

```
  char                  *mcNameP;
  SQL_PDB_NODE_TYPE     nodeNum;
} DB2_info;
/* ... */
```

VENDOR-INFO

この構造には、装置のベンダーとバージョンを識別するための情報が含まれています。

表 89. VENDOR-INFO 構造のフィールド： フィールドはすべて、NULL 終了ストリングです。

フィールド名	データ・タイプ	説明
vendor_id	char	ベンダーを表す ID。指すストリングの最大長は 64 文字です。
version	char	ベンダー製品の現行バージョン。指すストリングの最大長は 8 文字です。
release	char	ベンダー製品の現行リリース。重要性がなければ NULL に設定します。指すストリングの最大長は 8 文字です。
level	char	ベンダー製品の現行レベルです。重要性がなければ NULL に設定します。指すストリングの最大長は 8 文字です。
server_id	char	データベースが存在するサーバーを識別するユニーク名。指すストリングの最大長は 8 文字です。
max_bytes_per_txn	sqlint32	サポートされる最大の転送バッファー・サイズ。ベンダーが指定します (バイト単位)。これは、ベンダーの初期設定関数からの戻りコードが SQLUV_BUFF_SIZE (無効なバッファー・サイズが指定されたことを示す) である場合にのみ使用されます。
num_objects_in_backup	sqlint32	あるバックアップを完了するために使用されたセッションの数。これは、リストア操作時に、すべてのバックアップ・イメージがいつ処理されたのかを判別するために使用されます。
reserve	void	将来の利用のために予約されています。

言語構文:

C 構造

```
typedef struct Vendor_info
{
  char       *vendor_id;
  char       *version;
  char       *release;
  char       *level;
  char       *server_id;
  sqlint32   max_bytes_per_txn;
  sqlint32   num_objects_in_backup;
  void       *reserve;
} Vendor_info;
```

INIT-INPUT

この構造には、ベンダー装置との論理リンクを設定し、確立するために DB2 が提供する情報が含まれます。

表 90. INIT-INPUT 構造のフィールド： フィールドはすべて、NULL 終了ストリングです。

フィールド名	データ・タイプ	説明
DB2_session	struct DB2_info	DB2 側から見たセッションの説明。
size_options	unsigned short	オプション・フィールドの長さ。 DB2 バックアップ関数またはリストア関数を使用している場合、このフィールドのデータは *VendorOptionsSize* パラメーターから直接渡されます。
size_HI_order	sqluint32	バイト単位で見積もられた DB サイズの高位 32 ビット。合計サイズは 64 ビットです。
size_LOW_order	sqluint32	バイト単位で見積もられた DB サイズの低位 32 ビット。合計サイズは 64 ビットです。
options	void	この情報は、バックアップまたはリストア関数の呼び出し時にアプリケーションから渡されます。このデータ構造はフラットでなければなりません。つまり、間接のレベルはサポートされません。このデータについてはバイト反転は行われず、コード・ページがチェックされません。 DB2 バックアップ関数、またはリストア関数を使用している場合、このフィールドのデータは *pVendorOptions* パラメーターから直接渡されます。
reserve	void	将来の利用のために予約されています。
prompt_lvl	char	バックアップ操作またはリストア操作を呼び出したときにユーザーが要求したプロンプト・レベル。指すストリングの最大長は 1 文字です。
num_sessions	unsigned short	バックアップ操作またはリストア操作を呼び出したときにユーザーが要求したセッションの数。

言語構文:

C 構造

```
typedef struct Init_input
{
  struct DB2_info  *DB2_session;
  unsigned short   size_options;
  sqluint32        size_HI_order;
  sqluint32        size_LOW_order;
  void             *options;
  void             *reserve;
  char             *prompt_lvl;
  unsigned short   num_sessions;
} Init_input;
```

INIT-OUTPUT

この構造には、ベンダー装置が戻した出力が含まれます。

表 91. INIT-OUTPUT 構造のフィールド

フィールド名	データ・タイプ	説明
vendor_session	struct Vendor_info	ベンダーを DB2 に識別させるための情報が含まれます。
pVendorCB	void	ベンダーの制御ブロック。
reserve	void	将来の利用のために予約されています。

言語構文:

C 構造

```
typedef struct Init_output
{
  struct Vendor_info  *vendor_session;
  void                *pVendorCB;
  void                *reserve;
} Init_output;
```

DATA

この構造には、DB2 とベンダー装置との間で転送されるデータが含まれます。

表 92. DATA 構造のフィールド

フィールド名	データ・タイプ	説明
obj_num	sqlint32	バックアップ操作時に DB2 によって割り当てられるシーケンス番号。
buff_size	sqlint32	バッファーのサイズ。
actual_buf_size	sqlint32	送受信された実際のバイト数。これは *buff_size* を超えてはなりません。
dataptr	void	データ・バッファーを指すポインター。 DB2 はこのバッファーにスペースを割り振ります。
reserve	void	将来の利用のために予約されています。

言語構文:

C 構造

```
typedef struct Data
{
  sqlint32  obj_num;
  sqlint32  buff_size;
  sqlint32  actual_buff_size;
  void      *dataptr;
  void      *reserve;
} Data;
```

RETURN-CODE

この構造には、DB2 に戻される戻りコードとエラーの短い説明が含まれます。

表 93. RETURN-CODE 構造のフィールド

フィールド名	データ・タイプ	説明
return_code[a]	sqlint32	ベンダー関数からの戻りコード。
description	char	戻りコードの短い記述。
reserve	void	将来の利用のために予約されています。
[a] これはベンダー特有の戻りコードで、さまざまな DB2 API によって戻される値とは異なります。ベンダー製品から受け入れられる戻りコードについては、個々の API の説明を参照してください。		

言語構文:

C 構造

```
typedef struct Return_code
{
  sqlint32  return_code,
  char      description[30],
  void      *reserve,
} Return_code;
```

圧縮バックアップ用の API

圧縮プラグイン・インターフェース

DB2 には、COMPR_DB2INFO 構造の定義が用意されています。ベンダーは、以下の構造および API 以外のそれぞれの定義を用意しています。以下の構造、プロトタイプ、および定数は、DB2 に付属するファイル sqlucompr.h で定義されています。

DB2 環境の記述 - COMPR_DB2INFO:

```
struct COMPR_DB2INFO {
   char    tag[16];
   db2Uint32    version;
   db2Uint32    size;
   char         dbalias[SQLU_ALIAS_SZ+1];
   char         instance[SQL_INSTNAME_SZ+1];
   SQL_PDB_NODE_TYPE node;
   SQL_PDB_NODE_TYPE catnode;
   char         timestamp[SQLU_TIME_STAMP_LEN+1];
   db2Uint32    bufferSize;
   db2Uint32    options;
   db2Uint32    bkOptions;

   db2Uint32    db2Version;
   db2Uint32    platform;
   db2int32     comprOptionsByteOrder;
   db2Uint32    comprOptionsSize;
   void        *comprOptions;
   db2Uint32    savedBlockSize;
   void        *savedBlock;
};
```

COMPR_DB2INFO

DB2 は、この構造を割り振って定義し、InitCompression および InitDecompression API に対して、パラメーターとして渡します。この構造は、バックアップまたはリストアされるデータベースについて記述し、操作が行われる DB2 環境についての詳細を示しています。構造のフィールドは、以下のとおりです。

tag[16]
構造の目印として使用。これは、必ず ″COMPR_DB2INFO ¥0″ ストリングに設定されます。

version
使用される構造のバージョンを示します。これにより、 API は追加のフィールドが存在することを認識できます。現在は、バージョンは 1 です。将来は、この構造にさらにフィールドが追加される可能性があります。

size COMPR_DB2INFO 構造のサイズを指定します (バイト単位)。

dbalias[SQLU_ALIAS_SZ+1]

instance[SQL_INSTNAME_SZ+1]

node

catnode

timestamp[SQLU_TIME_STAMP_LEN+1]
バックアップまたはリストアされるデータベースについて記述します。これらは、バックアップ・イメージに命名する際に使用されるフィールドです。リストア操作の場合、dbalias はソース・データベースの別名を示します。

bufferSize
転送バッファーのサイズを指定します (4 K ページ単位)。

options
db2Backup API または db2Restore API で指定される iOptions フィールド。

bkOptions
リストア操作の場合、バックアップが作成されたときに db2Backup API で使用された iOptions フィールドを指定します。バックアップ操作の場合、ゼロに設定されます。

db2Version
DB2 エンジンのバージョンを指定します。

platform
DB2 エンジンが実行されているプラットフォームを指定します。この値は、<sqlmon.h> にリストされているいずれかの値になります。

comprOptionsByteOrder
API を実行するクライアントで使用するバイト・オーダーを指定します。 DB2 は、*comprOptions* として渡されたデータの解釈または変換を行わないため、このフィールドを使用して、データの使用前

にデータのバイトを反転する必要があるかどうかを判別しなければなりません。プラグイン・ライブラリー自体によって、何らかの変換を実行する必要があります。

comprOptionsSize
db2Backup および db2Restore API の *piComprOptionsSize* パラメーターの値を指定します。

***comprOptions**
db2Backup および db2Restore API の *piComprOptions* フィールドの値を指定します。

savedBlockSize

***savedBlock**
DB2 では、プラグイン・ライブラリーによって、任意のデータ・ブロックをバックアップ・イメージに保管できます。そのようなデータ・ブロックが特定のバックアップで保管された場合、リストア操作時にこれらのフィールドに戻されます。バックアップ操作の場合、これらのフィールドはゼロに設定されます。

プラグインの記述 - COMPR_PIINFO:

```
struct COMPR_PIINFO {
   char    tag[16];
   db2Uint32    version;
   db2Uint32    size;
   db2Uint32    useCRC;
   db2Uint32    useGran;
   db2Uint32    useAllBlocks;
   db2Uint32    savedBlockSize;
};
```

COMPR_PIINFO
この構造は、DB2 に対して記述するために、プラグイン・ライブラリーによって使用されます。この構造は、DB2 によって割り振られて初期設定され、主なフィールドは、 InitCompression 呼び出し時に、プラグイン・ライブラリーによって記入されます。

tag[16]
構造の目印として使用。 (DB2 側で設定します。) これは、必ず ″COMPR_PIINFO ¥0″ ストリングに設定されます。

version
使用される構造のバージョンを示します。これにより、 API は追加のフィールドが存在することを認識できます。現在は、バージョンは 1 です。 (DB2 側で設定します。) 将来は、この構造にさらにフィールドが追加される可能性があります。

size COMPR_PIINFO 構造のサイズを示します (バイト単位)。 (DB2 側で設定します。)

useCRC
DB2 では、圧縮プラグインは、32 ビット CRC またはチェックサム値を使用して、圧縮および解凍されるデータの保全性を検査でき

ます。ライブラリーがそのような検査を使用する場合、このフィールドは 1 に設定されます。それ以外の場合、フィールドは 0 に設定されます。

useGran
圧縮ルーチンでデータを任意のサイズ単位で圧縮できる場合、ライブラリーは、このフィールドを 1 に設定します。圧縮圧縮ルーチンが、データをバイト・サイズ単位でのみ圧縮する場合、ライブラリーは、このフィールドを 0 に設定します。この指標の設定に関係する詳細は、Compress の useGran パラメーターの説明を参照してください。リストア操作の場合、このフィールドは無視されます。

useAllBlocks
DB2 が、圧縮されていない元のブロックよりも大きい圧縮済みデータ・ブロックをバックアップするかどうかを指定します。デフォルトでは、DB2 は、圧縮されたバージョンのほうが大きい場合はデータを圧縮しないで保管しますが、特定の環境では、プラグイン・ライブラリーは、どの場合でも圧縮されたデータ・バックアップを保管します。 DB2 が、圧縮済みバージョンの全データ・ブロックを保管する予定である場合、ライブラリーは、この値を 1 に設定します。 DB2 が、元のデータよりも小さい場合にだけ、圧縮済みバージョンのデータを保管するのであれば、ライブラリーは、この値を 0 に設定します。リストア操作の場合、このフィールドは無視されます。

savedBlockSize
DB2 では、プラグイン・ライブラリーによって、任意のデータ・ブロックをバックアップ・イメージに保管できます。そのようなデータ・ブロックを特定のバックアップで保管するのであれば、ライブラリーは、このフィールドを、このデータに割り振られたブロックのサイズに設定します。 (実際のデータは、後続の API 呼び出しで DB2 に渡されます。) データを保管しない場合、プラグイン・ライブラリーは、このフィールドをゼロに設定します。リストア操作の場合、このフィールドは無視されます。

制御ブロックの記述 - COMPR_CB:

```
struct COMPR_CB;

extern "C" {

int InitCompression(
      const COMPR_DB2INFO *db2Info,
      COMPR_PIINFO  *piInfo,
      COMPR_CB  **pCB);

int GetSavedBlock(
      COMPR_CB   *pCB,
      db2Uint32   blockSize,
      void       *data);

int Compress(
      COMPR_CB   *pCB,
      const char *src,
      db2int32    srcLen,
      db2Uint32   srcGran,
      char       *tgt,
```

```
      db2int32    tgtSize,
      db2int32   *srcAct,
      db2int32   *tgtAct,
      db2Uint32  *tgtCRC);

int GetMaxCompressedSize(
      COMPR_CB   *pCB,
      db2Uint32   srcLen);

int TermCompression(
      COMPR_CB   *pCB);

int InitDecompression(
      const COMPR_DB2INFO *db2Info,
      COMPR_CB  **pCB);

int Decompress(
      COMPR_CB   *pCB,
      const char *src,
      db2int32    srcLen,
      char       *tgt,
      db2int32    tgtSize,
      db2int32   *tgtAct,
      db2Uint32  *tgtCRC);

int TermDecompression(
      COMPR_CB   *pCB);
}
```

COMPR_CB

これは、プラグイン・ライブラリーによって内部的に使用される構造です。ここには、圧縮および解凍ルーチンによって内部的に使用されるデータが含まれます。 DB2 は、この構造をプラグイン・ライブラリーに対して行うそれぞれの呼び出しに渡しますが、構造の性質はすべて、構造のフィールドの定義や構造のメモリー管理を含め、ライブラリーに残されます。

```
int InitCompression(
      const COMPR_DB2INFO    *db2Info,
      COMPR_PIINFO           *piInfo,
      COMPR_CB              **pCB);
```

圧縮ライブラリーを初期設定します。 DB2 は、db2Info および piInfo 構造を渡します。ライブラリーは、piInfo の適切なフィールドを記入し、pCB を割り当てて、割り振られたメモリーへのポインターを戻します。

```
int GetSavedBlock(
      COMPR_CB              *pCB,
      db2Uint32              blockSize,
      void                  *data);
```

バックアップ・イメージに保管するベンダー固有のデータ・ブロックを入手します。ライブラリーが、piInfo->savedBlockSize にゼロ以外の値を戻した場合、DB2 は、その値を *blockSize* として使用して、GetSavedBlock を呼び出します。プラグイン・ライブラリーは、指定されたサイズのデータを、データによって参照されるメモリーに書き込みます。この機能は、バックアップの場合にのみ、BM1 での初期データ処理中に呼び出されます。 db2Backup API で > 1 の並列処理が指定される場合でも、この機能は、バックアップごとに一回だけ呼び出されます。

```
int Compress(
      COMPR_CB              *pCB,
      const char            *src,
      db2int32               srcLen,
      db2int32               srcGran,
      char                  *tgt,
      db2int32               tgtSize,
      db2int32              *srcAct,
      db2int32              *tgtAct,
      db2Uint32             *tgtCRC);
```

データ・ブロックを圧縮します。 *src* は、サイズが *srcLen* バイトのデータ・ブロックを示します。 *tgt* は、サイズが *tgtSize* バイトのバッファーを示します。プラグイン・ライブラリーは、アドレス *src* でデータを圧縮し、アドレス *tgt* で圧縮済みデータをバッファーに書き込みます。圧縮したデータの中で、圧縮されていないデータの実際量が、*srcAct* に保管されます。圧縮済みデータの実サイズは、*tgtAct* として戻されます。

ライブラリーが piInfo->useCRC に 1 の値を戻した場合、圧縮されていないブロックの CRC 値は *tgtCRC* として戻されます。ライブラリーが piInfo->useCRC に 0 の値を戻した場合、 *tgtCRC* は NULL ポインターになります。

ライブラリーが piInfo->useGran に 1 の値を戻した場合、 *srcGran* は、データのページ・サイズの log2 を指定します。 (たとえば、データのページ・サイズが 4096 バイトである場合、*srcGran* は 12 になります。) ライブラリーは、実際の圧縮されたデータの量 (*srcAct*) が、このページ・サイズの厳密な倍数になるようにします。ライブラリーが useGran フラグを設定する場合、DB2 では、圧縮済みデータをバックアップ・イメージに合わせるため、より効率的なアルゴリズムを使用することができます。そのようにすると、プラグインのパフォーマンスが改善されると同時に、圧縮済みバックアップ・イメージが小さくなります。ライブラリーが for piInfo->srcGran に 0 の値を戻した場合、細分度は 1 バイトです。

```
int GetMaxCompressedSize(
      COMPR_CB              *pCB,
      db2Uint32              srcLen,
      db2Uint32             *tgtLen);
```

データ・ブロックを圧縮するのに必要な最大可能バッファー・サイズを見積もります。 *srcLen* は、圧縮する予定のデータ・ブロックのサイズを示します。ライブラリーは、*tgtLen* として圧縮した後で、理論上の最大バッファー・サイズを戻します。

DB2 では、*tgtLen* で戻された値を使用して、メモリー使用状況を内部的に最適化します。値を計算しなかったり誤った値を計算するときのペナルティーは、DB2 で、 1 つのデータ・ブロックで圧縮 API を複数回呼び出さなければならないか、ユーティリティー・ヒープのメモリーを浪費するということです。バックアップは、戻された値に関係なく正しく作成されます。

```
int TermCompression(
      COMPR_CB   *pCB);
```

圧縮ライブラリーを終了します。ライブラリーは、pCB に使用したメモリーを解放します。

```
int InitDecompression(
      const COMPR_DB2INFO    *db2Info,
      COMPR_CB              **pCB);
```

解凍ライブラリーを初期設定します。 DB2 は、db2Info 構造を渡します。ライブラリーは、*pCB* を割り振り、割り振られたメモリーへのポインターを戻します。

```
int Decompress(
      COMPR_CB              *pCB,
      const char            *src,
      db2int32               srcLen,
      char                  *tgt,
      db2int32               tgtSize,
      db2int32              *tgtLen,
      db2Uint32             *tgtCRC);
```

データ・ブロックを解凍します。 *src* は、サイズが *srcLen* バイトのデータ・ブロックを示します。 *tgt* は、サイズが *tgtSize* バイトのバッファーを示します。プラグイン・ライブラリーは、アドレス *src* でデータを解凍し、アドレス *tgt* で解凍済みデータをバッファーに書き込みます。解凍済みデータの実サイズは、*tgtLen* として戻されます。ライブラリーが piInfo->useCRC に 1 の値を戻した場合、圧縮されていないブロックの CRC は *tgtCRC* として戻されます。ライブラリーが piInfo->useCRC に 0 の値を戻した場合、 *tgtLen* は NULL ポインターになります。

```
int TermDecompression(
      COMPR_CB   *pCB);
```

解凍ライブラリーを終了します。ライブラリーは、*pCB* に使用したメモリーを解放します。これらの API によって内部的に使用されたメモリーはすべて、ベンダーによって管理されます。プラグイン・ライブラリーは、COMPR_CB 構造によって使用されるメモリーを管理します。 DB2 は、データ・バッファー (API の *src* および *tgt* パラメーター) に使用したメモリーを管理します。

プラグイン・インターフェース戻りコード:

これらは、API によって戻される可能性のある戻りコードです。指定されている箇所を除き、DB2 は、ゼロ以外の戻りコードが戻される場合に、バックアップまたはリストアを終了します。

SQLUV_OK	0	操作は成功しました。
SQLUV_BUFFER_TOO_SMALL	100	ターゲット・バッファーが小さすぎます。バックアップ時に示される場合、tgtAct フィールドには、オブジェクトを圧縮するのに必要な見積もりサイズが示されます。 DB2 は、少なくとも指定された大きさのバッファーで操作を再試行します。リストア時に示される場合、操作は失敗します。

SQLUV_PARTIAL_BUFFER	101	バッファーは部分的に圧縮されました。バックアップ時に示される場合、srcAct フィールドには、実際に圧縮されたデータの実際の量が示され、 tgtAct フィールドには、圧縮されたデータの実際のサイズが示されます。リストア時に示される場合、操作は失敗します。
SQLUV_NO_MEMORY	102	メモリー不足
SQLUV_EXCEPTION	103	コードでシグナルまたは例外が生じました。
SQLUV_INTERNAL_ERROR	104	内部エラーが検出されました。

SQLUV_BUFFER_TOO_SMALL と SQLUV_PARTIAL_BUFFER の違いは、SQLUV_PARTIAL_BUFFER が戻されると、 DB2 は出力バッファー内のデータを有効であると見なす点です。

関連資料:

- 28 ページの『db2Backup - データベースのバックアップ』
- 244 ページの『db2Restore - データベースのリストア』

付録 E. 並行アクセスを伴うスレッド化アプリケーション

並行アクセスを伴うスレッド化アプリケーション

DB2 データベースに対するスレッド化アプリケーションのデフォルトのインプリメンテーションでは、データベースへのアクセスのシリアライズはデータベース API によって実施されます。あるスレッドがデータベース呼び出しを実行する場合、最初の呼び出しが完了するまでは、続く呼び出しが最初の呼び出しに関連しないデータベース・オブジェクトにアクセスするとしても、他のスレッドによって行われた呼び出しはブロックされます。さらに、プロセス内のすべてのスレッドがコミット有効範囲を共用します。本当の意味でのデータベースへの並行アクセスは、独立したプロセスによって、あるいはこのセクションで説明されている API を用いることによってのみ達成できます。

このセクションでは、データベース API および組み込み SQL を使用するための独立した環境 (コンテキスト) を割り振り、操作するのに使用できる API について説明します。各コンテキストは独立したエンティティーであり、あるコンテキストを使用した接続は、それ以外のコンテキスト (したがって、プロセス内の他のすべての接続) から独立します。あるコンテキストで作業が行われるためには、まず、そのコンテキストがスレッドに関連付けられなければなりません。スレッドは、データベース API 呼び出しを行うとき、または組み込み SQL を使用するときには、必ずコンテキストを所有していなければなりません。

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、複数コンテキストの使用が可能です。 (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。

- sqleAttachToCtx - コンテキストへのアタッチ
- 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
- sqleDetachFromCtx - コンテキストからの切り離し
- 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
- sqleGetCurrentCtx - 現行コンテキストの入手
- sqleInterruptCtx - コンテキストへの割り込み

接続またはアタッチの期間中、コンテキストはある特定のスレッドに関連付けられていなくても構いません。あるスレッドがコンテキストにアタッチし、データベースに接続し、そのコンテキストからデタッチされたら、2 番目のスレッドがそのコンテキストにアタッチし、既存のデータベース接続を用いて作業を継続することが可能です。コンテキストは 1 つのプロセス内のスレッド間で受け渡しすることはできますが、複数のプロセス間で受け渡しすることはできません。

新しい API を使用する場合でも、以下の API は引き続きシリアライズされます。

- sqlabndx - バインド
- sqlaprep - プログラムのプリコンパイル
- sqluexpr - エクスポート
- db2Import and sqluimpr - インポート

以下の API は、アプリケーションのスレッド化をサポートしないプラットフォームでは無効です (つまり、操作を行いません)。

注:

1. DB2 CLI は自動的に複数のコンテキストを使用して、マルチスレッドをサポートするプラットフォームで、スレッド・セーフの並行データベース・アクセスを実現します。 DB2 では推奨されていませんが、必要な場合には、この機能を明示的に使用不可にできます。
2. デフォルトでは、AIX では 32 ビットのアプリケーションがプロセスあたり 11 を超える共有メモリー・セグメントにアタッチすることは許可されていません。その内、最大で 10 の共有メモリー・セグメントが DB2 接続に使用できます。しかし、Java アプリケーションでは、多くのアプリケーションはプロセスごとに 1 つの共有メモリー・セグメントに制限されています。

   この制限に到達すると、DB2 は SQL CONNECT で SQLCODE -1224 を戻します。 DB2 Connect では、ローカル・ユーザーが SNA を介して 2 フェーズ・コミットを実行している場合や、 TP モニター (SNA または TCP/IP) との 2 フェーズ・コミットを実行している場合にも、接続数は 10 に制限されます。

   AIX 環境変数 **EXTSHM** を使用すれば、プロセスをアタッチする共有メモリー・セグメントの最大数を増やすことができます。

   EXTSHM を DB2 で使用するには、以下を実行してください。

   クライアント・セッションの場合:

   ```
   export EXTSHM=ON
   ```

   DB2 サーバーを始動する場合:

   ```
   export EXTSHM=ON
   db2set DB2ENVLIST=EXTSHM
   db2start
   ```

   ESE では、userprofile または usercshrc ファイル に以下の行を追加します。

   ```
   EXTSHM=ON
   export EXTSHM
   ```

   ローカル・データベースまたは DB2 Connect を別のマシンに移動してそれにリモートからアクセスするか、 TCP/IP ループバックによってローカル・データベースまたは DB2 Connect データベースにアクセスできるようにするため、ローカル・マシンの TCP/IP アドレスを持つリモート・ノードとしてそのデータベースをカタログするのが代替の手段です。

関連資料:

- 593 ページの『管理 API および アプリケーションの移行』

- 594 ページの『変更された API およびデータ構造』

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++)』

sqleAttachToCtx - コンテキストへのアタッチ

現行のスレッドが、指定されたコンテキストを使用するようにします。このスレッドで行われる後続のすべてのデータベース呼び出しでは、このコンテキストが使用されます。特定のコンテキストに複数のスレッドがアタッチされると、これらのスレッドについてはアクセスがシリアライズされ、これらのスレッドはコミット有効範囲を共用します。

有効範囲:

この API の有効範囲は、即時プロセスに限定されます。

許可:

なし

必要な接続:

なし

API 組み込みファイル:

sql.h

C API 構文:

```
int sqleAttachToCtx (
void           *pCtx,
void           *reserved,
struct sqlca   *pstSqlca);
```

API パラメーター:

pCtx 入力。 sqleBeginCtx によってあらかじめ割り振られた有効なコンテキスト。

reserved
将来の利用のために予約されています。 NULL に設定しなければなりません。

pstSqlca
出力。 *sqlca* 構造を指すポインター。

関連資料:

- 453 ページの『SQLCA』
- 554 ページの『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』

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++)』

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

アプリケーション・コンテキストを作成するか、あるいはアプリケーション・コンテキストを作成した後、それにアタッチします。複数のアプリケーション・コンテキストを作成することができます。各コンテキストは、独自のコミット有効範囲を持ちます。コンテキストが異なれば、別個のスレッドをアタッチできます (『sqleAttachToCtx』を参照してください)。そのようなスレッドによって行われるデータベース API 呼び出しは、相互にシリアライズされません。

有効範囲:

この API の有効範囲は、即時プロセスに限定されます。

許可:

なし

必要な接続:

なし

API 組み込みファイル:

sql.h

C API 構文:

```
int sqleBeginCtx (
void           **ppCtx,
sqlint32       lOptions,
void           *reserved,
struct sqlca   *pstSqlca);
```

API パラメーター:

ppCtx 出力。コンテキスト情報を格納するために専用メモリーの中から割り振られるデータ域。

lOptions
入力。有効な値は以下のとおりです。

SQL_CTX_CREATE_ONLY
コンテキスト・メモリーが割り振られますが、アタッチは行われません。

SQL_CTX_BEGIN_ALL
コンテキスト・メモリーが割り振られた後、現行のスレッド用に sqleAttachToCtx の呼び出しが行われます。このオプションを使用する場合には、 *ppCtx* パラメーターを NULL にすることができます。スレッドがすでにコンテキストにアタッチされている場合には、呼び出しは失敗します。

reserved
将来の利用のために予約されています。 NULL に設定しなければなりません。

pstSqlca
出力。 *sqlca* 構造を指すポインター。

関連資料:

- 453 ページの『SQLCA』
- 553 ページの『sqleAttachToCtx - コンテキストへのアタッチ』

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++)』

sqleDetachFromCtx - コンテキストからの切り離し

現行のスレッドによって使用されているコンテキストを切り離します。コンテキストが切り離されるのは、そのコンテキストに以前にアタッチしていた場合に限ります。

有効範囲:

この API の有効範囲は、即時プロセスに限定されます。

許可:

なし

必要な接続:

なし

API 組み込みファイル:

sql.h

C API 構文:

```
int sqleDetachFromCtx (
void           *pCtx,
void           *reserved,
struct sqlca   *pstSqlca);
```

API パラメーター:

pCtx 入力。 sqleBeginCtx によってあらかじめ割り振られた有効なコンテキスト。

reserved
将来の利用のために予約されています。 NULL に設定しなければなりません。

pstSqlca
出力。 *sqlca* 構造を指すポインター。

関連資料:

- 453 ページの『SQLCA』
- 554 ページの『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』

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++)』

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

特定のコンテキストに関連付けられているすべてのメモリーを解放します。

有効範囲:

この API の有効範囲は、即時プロセスに限定されます。

許可:

なし

必要な接続:

なし

API 組み込みファイル:

sql.h

C API 構文:

```
int sqleEndCtx (
void           **ppCtx,
sqlint32       lOptions,
void           *reserved,
struct sqlca   *pstSqlca);
```

API パラメーター:

ppCtx 出力。専用メモリー (コンテキスト情報を格納するために使用されている) 内で解放されるデータ域。

lOptions
入力。有効な値は以下のとおりです。

SQL_CTX_FREE_ONLY
コンテキスト・メモリーは、以前に切り離しが行われている場合にのみ解放されます。

注: *pCtx* は sqleBeginCtx によってあらかじめ割り振られた有効なコンテキストでなければなりません。

SQL_CTX_END_ALL
必要であれば、メモリーが解放される前に sqleDetachFromCtx の呼び出しが行われます。

注: 切り離しは、コンテキストがまだ使用中である場合でも行われます。このオプションを使用する場合には、*ppCtx* パラメーターを NULL にすることができます (ただし、このパラメーターを渡す場合には、sqleBeginCtx によってあらかじめ割り振られた有効なコンテキストでなければなりません)。
sqleGetCurrentCtx への呼び出しが行われ、そこから現行のコンテキストが解放されます。

reserved
将来の利用のために予約されています。 NULL に設定しなければなりません。

pstSqlca
出力。 *sqlca* 構造を指すポインター。

使用上の注意:

データベース接続が存在するか、またはコンテキストが別のスレッドに接続されている場合には、この呼び出しは失敗します。

注: あるコンテキストで、インスタンス・アタッチを確立する API (たとえば、db2CfgGet) を呼び出す場合には、 sqleEndCtx を呼び出す前に、sqledtin を用いてコンテキストをインスタンスから切り離すことが必要です。

関連資料:

- 370 ページの『sqledtin - 切り離し』
- 453 ページの『SQLCA』
- 554 ページの『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』
- 555 ページの『sqleDetachFromCtx - コンテキストからの切り離し』
- 557 ページの『sqleGetCurrentCtx - 現行コンテキストの入手』
- 36 ページの『db2CfgGet - 構成パラメーターの入手』

sqleGetCurrentCtx - 現行コンテキストの入手

スレッドに関連付けられている現行のコンテキストを戻します。

有効範囲:

この API の有効範囲は、即時プロセスに限定されます。

許可:

なし

必要な接続:

なし

API 組み込みファイル:

sql.h

C API 構文:

```
int sqleGetCurrentCtx (
void           **ppCtx,
void           *reserved,
struct sqlca   *pstSqlca);
```

API パラメーター:

ppCtx 出力。コンテキスト情報を格納するために専用メモリーの中から割り振られるデータ域。

h reserved
将来の利用のために予約されています。 NULL に設定しなければなりません。

pstSqlca
出力。 *sqlca* 構造を指すポインター。

関連資料:

• 453 ページの『SQLCA』

sqleInterruptCtx - コンテキストへの割り込み

指定されたコンテキストに割り込みます。

有効範囲:

この API の有効範囲は、即時プロセスに限定されます。

許可:

なし

必要な接続:

データベース

API 組み込みファイル:

sql.h

C API 構文:

```
int sqleInterruptCtx (
void           *pCtx,
void           *reserved,
struct sqlca   *pstSqlca);
```

API パラメーター:

pCtx 入力。 sqleBeginCtx によってあらかじめ割り振られた有効なコンテキスト。

reserved
将来の利用のために予約されています。 NULL に設定しなければなりません。

pstSqlca
出力。 *sqlca* 構造を指すポインター。

使用上の注意:

処理中に、この API は以下のことを行います。

- 渡されたコンテキストへの切り替え
- 割り込みの送信
- 元のコンテキストへの切り替え
- 終了

関連資料:

- 453 ページの『SQLCA』
- 554 ページの『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』

sqleSetTypeCtx - アプリケーション・コンテキスト・タイプの設定

アプリケーション・コンテキストのタイプを設定します。この API は、アプリケーション内で呼び出される最初のデータベース API でなければなりません。

有効範囲:

この API の有効範囲は、即時プロセスに限定されます。

許可:

なし

必要な接続:

なし

API 組み込みファイル:

sql.h

C API 構文:

```
int sqleSetTypeCtx (
sqlint32   lOptions);
```

API パラメーター:

lOptions
入力。有効な値は以下のとおりです。

SQL_CTX_ORIGINAL
すべてのスレッドが同じコンテキストを使用し、並行アクセスはブロックされます。これは、どの API も呼び出されない場合のデフォルトです。

SQL_CTX_MULTI_MANUAL
すべてのスレッドが独立したコンテキストを使用します。各スレッドごとにコンテキストを管理するのは、アプリケーション側の責任です。以下を参照してください。

- sqleBeginCtx
- sqleAttachToCtx
- sqleDetachFromCtx
- sqleEndCtx

このオプションが使用されるときには、以下の制限/変更が適用されます。

- 正常終了の場合、プロセス終了時の自動 COMMIT は使用不可になります。未解決トランザクションはすべてロールバックされます。すべての COMMIT を明示的に行わなければなりません。
- sqleintr は、すべてのコンテキストに割り込みます。特定のコンテキストに割り込むには、sqleInterruptCtx を使用してください。

使用上の注意:

この API は、他のデータベース呼び出しの*前* に呼び出されなければならず、最初の呼び出しだけが有効です。

関連資料:

- 386 ページの『sqleintr - 割り込み』
- 553 ページの『sqleAttachToCtx - コンテキストへのアタッチ』
- 554 ページの『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』
- 555 ページの『sqleDetachFromCtx - コンテキストからの切り離し』
- 556 ページの『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』
- 558 ページの『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++)』

付録 F. DB2 UDB ログ・レコード

このセクションでは、db2ReadLog API によって戻される DB2 UDB ログ・レコードの構造を記述します。

すべての DB2 UDB ログ・レコードは、ログ・マネージャー・ヘッダーで始まります。このヘッダーには、ログ・レコード・サイズの合計、ログ・レコードのタイプ、およびトランザクション固有の情報が入ります。会計、統計、トレース、またはパフォーマンス評価に関する情報は含まれません。詳細については、563 ページの『ログ・マネージャー・ヘッダー』を参照してください。

ログ・レコードは、ログ・シーケンス番号 (LSN) によって固有に識別されます。LSN は、データベース・ログにおける、ログ・レコードの最初のバイトに対応する相対バイト・アドレスを表します。データベース・ログの始まりからのログ・レコードのオフセットをマークします。

単一のトランザクションで書き込まれたログ・レコードは、ログ・レコード・ヘッダーにあるフィールドによって固有に識別することができます。ユニークなトランザクション ID は、新しいトランザクションが開始されるたびに 1 ずつ増加する 6 バイト・フィールドです。単一のトランザクションによって書き込まれたすべてのログ・レコードには、同じ ID が含まれます。

トランザクションが、DATA CAPTURE CHANGES がオンの状態で表に対する書き込み可能作業を実行するか、またはログ書き込みユーティリティーを呼び出すときには、トランザクションは伝搬可能としてマークされます。伝搬可能なトランザクションだけが、トランザクション・マネージャーのログ・レコードを伝搬可能としてマークします。

表 94. DB2 UDB ログ・レコード

データ・マネージャー	
567 ページの『表の初期設定』	新しい永続表の作成。
569 ページの『インポート置換 (切り捨て)』	インポート置換活動。
569 ページの『挿入のロールバック』	ロールバック行の挿入。
570 ページの『表の REORG』	REORG のコミット。
570 ページの『索引の作成、索引のドロップ』	索引活動。
571 ページの『表の作成、表のドロップ、表作成のロールバック、表ドロップのロールバック』	表活動。
571 ページの『表変更属性』	伝搬、チェック・ペンディング、および付加モード活動。
572 ページの『表の変更による列の追加、列追加のロールバック』	既存の表への列の追加。
572 ページの『レコードの挿入、レコードの削除、レコード削除のロールバック、レコード更新のロールバック』	表レコード活動。

表 94. DB2 UDB ログ・レコード (続き)

578 ページの『レコードの更新』	記憶場所が変更されない場合の行更新。
長フィールド・マネージャー	
580 ページの『長フィールド・レコードの追加/削除/非更新』	長フィールド・レコード活動。
トランザクション・マネージャー	
581 ページの『通常コミット』	トランザクションのコミット。
581 ページの『ヒューリスティック・コミット』	未確定トランザクションのコミット。
581 ページの『MPP コーディネーター・ノード・コミット』	トランザクションのコミット。これは、少なくとも 1 つの従属ノードで更新を実行するアプリケーションのコーディネーター・ノードに書き込まれます。
582 ページの『MPP 従属ノード・コミット』	トランザクションのコミット。これは、従属ノードに書き込まれます。
582 ページの『通常打ち切り』	トランザクションの打ち切り。
582 ページの『ヒューリスティック打ち切り』	未確定トランザクションの打ち切り。
583 ページの『ローカル・ペンディング・リスト』	ペンディング・リストが存在する場合のトランザクションのコミット。
583 ページの『グローバル・ペンディング・リスト』	ペンディング・リストが存在するトランザクションのコミット (2 フェーズ)。
584 ページの『XA 準備』	2 フェース・コミット環境での XA トランザクションの準備。
584 ページの『MPP 従属ノード準備』	2 フェース・コミット環境での MPP トランザクションの準備。このログ・レコードは、従属ノードにのみ存在します。
585 ページの『バックアウト解放』	バックアウト解放インターバル終了のマーク。バックアウト解放インターバルは、トランザクションが打ち切られたときに補正されていないログ・レコードのセットです。
ユーティリティー・マネージャー	
586 ページの『移行の開始』	カタログ移行の開始。
586 ページの『移行の終了』	カタログ移行の完了。
586 ページの『ロードの開始』	表ロードの開始。
586 ページの『表ロードの削除開始』	ロード削除フェーズの開始。
586 ページの『ロード削除開始の補正』	ロード削除フェーズの終了。
587 ページの『ロード・ペンディング・リスト』	表ロードの完了。
587 ページの『バックアップの終了』	バックアップ活動の完了。
587 ページの『表スペースのロールフォワード』	表スペース・ロールフォワードの完了。
587 ページの『PIT への表スペース・ロールフォワードの開始』	特定の時点への表スペース・ロールフォワードの開始。

表 94. DB2 UDB ログ・レコード (続き)

588 ページの『PIT への表スペース・ロールフォワードの終了』	特定の時点への表スペース・ロールフォワードの終了。
Data Links Manager	
589 ページの『ファイルのリンク』	DATALINK 列を持つ表に対して挿入または更新が行われた結果、ファイルへのリンクが作成されたときに書き込まれます。
590 ページの『ファイルのリンク解除』	DATALINK 列を持つ表に対して削除または更新が行われた結果、ファイルへのリンクがドロップされたときに書き込まれます。
591 ページの『グループの削除』	DATALINK 列を持つ表 (ファイルのリンク制御属性を持つ) がドロップされたときに書き込まれます。
591 ページの『p グループの削除』	表スペースがドロップされたときに書き込まれます。
592 ページの『DLFM 準備』	DB2 Data Links Manager に関係したトランザクションに 2 フェーズ・コミットが使用されている場合の準備段階で書き込まれます。

ログ・マネージャー・ヘッダー

すべての DB2 UDB ログ・レコードは、ログ・マネージャー・ヘッダーで始まります。このヘッダーには、ログ・レコードの詳細な情報とログ・レコード書き込み機能のトランザクション情報が含まれます。

注: タイプ「i」のログ・レコードは単に通知目的のログ・レコードです。このタイプのログ・レコードは、ロールフォワード時、ロールバック時、およびクラッシュ・リカバリー時に DB2 に無視されます。

表 95. ログ・マネージャーのログ・レコード・ヘッダー

説明	種類	オフセット (バイト)
ログ・レコード全体の長さ	int	0(4)
ログ・レコードのタイプ (564 ページの表 96 を参照。)	short	4(2)
ログ・レコードの汎用フラグ [1]	short	6(2)
このトランザクションによって書き込まれた前のログ・レコードのログ・シーケンス番号。これは、ログ・レコードをつなぐためにトランザクションが使用します。値が 0000 0000 0000 である場合、これがトランザクションによって書き込まれた最初のログ・レコードです。	SQLU_LSN[2]	8(6)
ユニークなトランザクション ID	SQLU_TID[3]	14(6)

表 95. ログ・マネージャーのログ・レコード・ヘッダー (続き)

説明	種類	オフセット (バイト)
このトランザクションの、補正されているログ・レコードの前のログ・レコードのログ・シーケンス番号。 (注: 補正ログ・レコードおよびバックアウト解放ログ・レコードの場合のみ。)	SQLU_LSN	20(6)
このトランザクションの、補正されているログ・レコードのログ・シーケンス番号。 (注: 伝搬可能補正ログ・レコードの場合のみ。)	SQLU_LSN	26(6)
ログ・マネージャーのログ・レコード・ヘッダーの全長: • *補正なし: 20 バイト* • *補正: 26 バイト* • *伝搬可能補正: 32 バイト*		

注:

1. ログ・レコードの汎用フラグ定数

   ```
   常時再実行                  0x0001
   伝搬可能                    0x0002
   Single record UOW           0x0010
   条件付きリカバリー可能      0x0080
   ```

   0x0010 フラグを持つログ・レコードは、単一ログ・レコードからなる作業単位と見なされます。このトランザクションに関しては、コミットまたは打ち切りのログ・レコードが存在しません。

2. ログ・シーケンス番号 (LSN)

   ログ・レコードのデータベース・ログにおける相対バイト・アドレスを示す固有なログ・レコード ID。

   ```
   SQLU_LSN: union {  char  [6] ;
                      short [3] ;
                    }
   ```

3. トランザクション ID (TID)

   トランザクションを示すユニークなログ・レコード ID。

   ```
   SQLU_TID: union {  char  [6] ;
                      short [3] ;
                    }
   ```

表 96. ログ・マネージャーのログ・レコード・ヘッダーのログ・タイプ値および定義

値	定義
0x0061	Data Links Manager ログ・レコード
0x006F	バックアップ開始
0x0041	通常打ち切り
0x004F	バックアップの終了
0x0042	バックアウト解放
0x0089	PIT への表スペース・ロールフォワードの開始

表 96. ログ・マネージャーのログ・レコード・ヘッダーのログ・タイプ値および定義 (続き)

値	定義
0x0063	MPP コーディネーター・ノード・コミット
0x0050	表の静止
0x0043	適合
0x0071	PIT への表スペース・ロールフォワードの終了
0x0044	表スペースのロールフォワード
0x0051	グローバル・ペンディング・リスト
0x0045	ローカル・ペンディング・リスト
0x0052	再実行
0x0088	トランザクションの忘却
0x0085	MPP 従属ノード・コミット
0x0080	MPP ログ同期
0x0053	適合が必要
G	ロード・ペンディング・リスト
0x0054	一部打ち切り
0x0048	表ロードの削除開始
0x0055	取り消し
0x0069	伝搬のみ
V	移行の開始
0x0049	ヒューリスティック打ち切り
0x0056	移行の終了
0x004A	ロードの開始
0x0083	TM の準備
0x004B	ロード削除開始の補正
0x0087	ヒューリスティック・コミット
L	ロックの説明
0x0081	MPP の準備
0x0084	通常コミット
0x0082	XA 準備
0x004E	通常

データ・マネージャーのログ・レコード

データ・マネージャーのログ・レコードは、DDL、DML、またはユーティリティー活動の結果です。

データ・マネージャーのログ・レコードには、次の 2 つのタイプがあります。

- データ管理システム (DMS) ログ。ヘッダーのコンポーネント ID が 1 になります。
- データ・オブジェクト・マネージャー (DOM) ログ。ヘッダーのコンポーネント ID が 4 になります。

表 97. DMS ログ・レコード・ヘッダーの構造 (DMSLogRecordHeader)

説明	種類	オフセット (バイト)
コンポーネント ID (=1)	unsigned char	0(1)
ファンクション ID (表 98 を参照。)	unsigned char	1(1)
表 ID		
表スペース ID	unsigned short	2(2)
表 ID	unsigned short	4(2)
全長: 6 バイト		

表 98. DMS ログ・レコード・ヘッダー構造のファンクション ID 値および定義

値	定義
102	表に列を追加する
104	列の追加を取り消す
106	レコードの削除
110	レコードの挿入を取り消す
111	レコードの削除を取り消す
112	レコードの更新を取り消す
113	表に列を追加する
104	列の長さを変更する
115	列の長さ変更を取り消す
118	レコードの挿入
120	レコードの更新
124	表属性を変更する
128	表の初期設定
129	空ページからレコードを削除
130	空ページへレコードを挿入
131	空のページへのレコードの挿入を取り消す
132	空ページからのレコードの削除の取り消し

表 99. DOM ログ・レコード・ヘッダーの構造 (DOMLogRecordHeader)

説明	種類	オフセット (バイト)
コンポーネント ID (=4)	unsigned char	0(1)
ファンクション ID (567 ページの表 100 を参照。)	unsigned char	1(1)
オブジェクト ID		
表スペース ID	unsigned short	2(2)
オブジェクト ID	unsigned short	4(2)

表 99. DOM ログ・レコード・ヘッダーの構造 (DOMLogRecordHeader) (続き)

説明	種類	オフセット (バイト)
表 ID		
表スペース ID	unsigned short	6(2)
表 ID	unsigned short	8(2)
オブジェクト・タイプ	unsigned char	10(1)
フラグ	unsigned char	11(1)
全長: 12 バイト		

表 100. DOM ログ・レコード・ヘッダー構造のファンクション ID 値および定義

値	定義
2	索引の作成
3	索引のドロップ
4	表のドロップ
11	表を切り捨てる (インポート置換)
35	表の REORG
101	表の作成
130	表の作成を取り消す

注: データ・マネージャーのすべてのログ・レコード・オフセットは、ログ・マネージャーのレコード・ヘッダーの終わりからのものです。

ファンクション ID の省略名が UNDO で始まるすべてのログ・レコードは、当該アクションの UNDO または ROLLBACK 中に書き込まれたログ・レコードです。

ROLLBACK は、次の結果として起こる可能性があります。

- ユーザーが発行したトランザクションの ROLLBACK ステートメント
- 選択されたトランザクションの ROLLBACK を引き起こすデッドロック
- クラッシュ・リカバリーに続く、コミットされていないトランザクションの ROLLBACK
- ログの RESTORE および ROLLFORWARD に続く、コミットされていないトランザクションの ROLLBACK

表の初期設定

表の初期設定ログ・レコードは、新しい永続表が作成されるときに書き込まれ、表の初期設定を示します。このレコードは、DATA 記憶オブジェクト作成ログ・レコードの後、 LF および LOB 記憶オブジェクト作成ログ・レコードの前に入れられます。これは再実行ログ・レコードです。

表 101. 表の初期設定ログ・レコードの構造

説明	種類	オフセット (バイト)
ログ・ヘッダー	DMSLogRecordHeader	0(6)
ファイル作成 LSN	SQLU_LSN	6(6)

表 101. 表の初期設定ログ・レコードの構造 (続き)

説明	種類	オフセット (バイト)
表ディレクトリー・レコード	variable	12(72)
レコード・タイプ	unsigned char	12(1)
予約済み	char	13(1)
索引フラグ	unsigned short	14(2)
索引ルート・ページ	sqluint32	16(4)
TDESC レコード ID	sqluint32	20(4)
予約済み	char	24(56)
フラグ [1]	sqluint32	80(4)
表記述の長さ	sqluint32	84(4)
表記述レコード	variable	88 (可変)
レコード・タイプ	unsigned char	88(1)
予約済み	char	89(1)
列の数	unsigned short	90(2)
配列	variable long	92 (可変)
全長: 88 バイト + 表記述レコードの長さ		

注:

1. ビット 0x00000010 は、表が VALUE COMPRESSION オプションを使用して作成されたことを示します。ビット 0x00000020 は、表が NOT LOGGED INITIALLY オプションを指定して作成されたことと、表を作成したトランザクションがコミットされるまではこの表に関する DML 活動が記録されないことを示します。ビット 0x00000800 は、表が ORGANIZE BY 文節によって作成された、マルチディメンション・クラスタリング (MDC) された表であったことを示します。

表記述レコード: 列記述子配列:

(列の数) * 8、配列の各エレメントの内容は次のとおりです。

- フィールド・タイプ (unsigned short、2 バイト)

```
SMALLINT     0x0000
INTEGER      0x0001
DECIMAL      0x0002
DOUBLE       0x0003
REAL         0x0004
BIGINT       0x0005
CHAR         0x0100
VARCHAR      0x0101
LONG VARCHAR 0x0104
DATE         0x0105
TIME         0x0106
TIMESTAMP    0x0107
BLOB         0x0108
CLOB         0x0109
DATALINK     0x010E
GRAPHIC      0x0200
VARGRAPH     0x0201
LONG VARG    0x0202
DBCLOB       0x0203
```

- 長さ (2 バイト)

  - BLOB、CLOB、または DBCLOB の場合、このフィールドは使用されません。このフィールドの最大長については、列記述子配列に続く配列を参照してください。
  - DECIMAL でない場合、長さはフィールドの最大長 (short) になります。
  - PACKED DECIMAL の場合、バイト 1 は unsigned char、精度 (全長)、バイト 2 は unsigned char、位取り (小数部の桁数) になります。
- NULL フラグ (unsigned short、2 バイト)
  - 相互に排他的: NULL を許可するか、あるいは NULL を許可しない
  - 有効なオプション : デフォルトなし、デフォルト入力、ユーザー・デフォルト、またはデフォルト入力の短縮

```
ISNULL                  0x01
NONULLS                 0x02
TYPE_DEFAULT            0x04
USER_DEFAULT            0x08
COMPRESS_SYSTEM_DEFAULT 0x80
```

- フィールド・オフセット (unsigned short、2 バイト)。これは、定様式レコードの始まりからフィールドの固定値が見つかる位置までのオフセットです。

表記述レコード: LOB 記述子配列:

(LOB、CLOB、および DBCLOB フィールドの数) * 12、配列の各エレメントの内容は次のとおりです。

- 長さ (MAX LENGTH OF FIELD、sqluint32、4 バイト)
- 予約済み (内部、sqluint32、4 バイト)
- ログ・フラグ (IS COLUMN LOGGED、sqluint32、4 バイト)

列記述子配列内の最初の LOB、CLOB、または DBCLOB は、 LOB 記述子配列内の最初のエレメントを使用します。列記述子配列内の 2 番目の LOB、CLOB、または DBCLOB は、 LOB 記述子配列内の 2 番目のエレメントを使用し、以下同様に続きます。

インポート置換 (切り捨て)

インポート置換 (切り捨て) ログ・レコードは、 IMPORT REPLACE アクションが実行されるときに書き込まれます。このレコードは、表の再初期設定を示します (ユーザー・レコードを伴わず、新しい LSN を持ちます)。ログのヘッダーにある表スペースおよびオブジェクト ID の 2 番目のセットにより、切り捨て (IMPORT REPLACE) の対象である表が識別されます。これは再実行ログ・レコードです。

表 102. インポート置換 (切り捨て) ログ・レコードの構造

説明	種類	オフセット (バイト)
ログ・ヘッダー	DOMLogRecordHeader	0(12)
内部	variable	12 (可変)
全長: 12 バイト + 可変長部分		

挿入のロールバック

挿入のロールバック・ログ・レコードは、行の挿入アクション (INSERT RECORD) がロールバックされるときに書き込まれます。これは補正ログ・レコードです。

表 103. 挿入のロールバック・ログ・レコードの構造

説明	種類	オフセット (バイト)
ログ・ヘッダー	DMSLogRecordHeader	0(6)
埋め込み	char[ ]	6(2)
RID	sqluint32	8(4)
レコード長	unsigned short	12(2)
フリー・スペース	unsigned short	14(2)
全長: 16 バイト		

表の REORG

表の再編成ログ・レコードは、表の再編成を完了させるために REORG ユーティリティーがコミットされるときに書き込まれます。これは通常ログ・レコードです。

表 104. 表の REORG ログ・レコードの構造

説明	種類	オフセット (バイト)
ログ・ヘッダー	DOMLogRecordHeader	0(12)
内部	variable	12(392)
索引トークン [1]	unsigned short	2(404)
TEMPORARY 表スペース ID[2]	unsigned short	2(406)
全長: 408 バイト		

注:

1. 0 でない場合、これは REORG のクラスタリングに使用された索引 (クラスタリング索引) です。
2. 0 でない場合、これは REORG の組み立てに使用された SYSTEM TEMPORARY 表スペースです。

索引の作成、索引のドロップ

これらのログ・レコードは、索引が作成またはドロップされるときに書き込まれます。このログ・レコードには、次の 2 つのエレメントがあります。

- 索引ルート・ページ。これは内部 ID です。
- 索引トークン。これは、SYSIBM.SYSINDEXES 内の IID 列と同じです。このエレメントの値が 0 でない場合、ログ・レコードは内部索引に対するアクションを表しており、ユーザー索引とは関連していません。

これは正常なログ・レコードです。

表 105. 索引の作成、索引のドロップ・ログ・レコードの構造

説明	種類	オフセット (バイト)
ログ・ヘッダー	DOMLogRecordHeader	0(12)
埋め込み	char[ ]	12(2)
索引トークン	unsigned short	14(2)
索引ルート・ページ	sqluint32	16(4)

表 105. 索引の作成、索引のドロップ・ログ・レコードの構造 (続き)

説明	種類	オフセット (バイト)
全長: 20 バイト		

表の作成、表のドロップ、表作成のロールバック、表ドロップのロールバック

これらのログ・レコードは、永続表の DATA オブジェクトが作成またはドロップされるときに作成されます。 DATA オブジェクトは、CREATE TABLE 中に、表の初期設定 (Initialize Table) に先立って作成されます。表の作成および表のドロップは、通常ログ・レコードです。表作成のロールバックおよび表ドロップのロールバックは、補正ログ・レコードです。

表 106. 表の作成、表のドロップ、表作成のロールバック、表ドロップのロールバック・ログ・レコードの構造

説明	種類	オフセット (バイト)
ログ・ヘッダー	DOMLogRecordHeader	0(12)
内部	variable	12(72)
全長: 84 バイト		

表変更属性

表属性変更ログ・レコードは、表の状態が ALTER TABLE ステートメントを介して変更されたときや、制約の追加または妥当性検査の結果として変更されたときに書き込まれます。

表 107. 表変更属性、表変更の取り消し属性

説明	種類	オフセット (バイト)
ログ・ヘッダー	DMSLogRecordHeader	0(6)
埋め込み	char[ ]	6(2)
ビット変更 (属性) マスク	sqluint32	8(4)
ビット変更 (属性) 値	sqluint32	12(4)
全長: 16 バイト		

属性ビット:

```
0x00000001   Propagation
0x00000002   Check Pending
0x00000010   Value Compression
0x00010000   Append Mode
0x00200000   LF Propagation
```

上記のビットのいずれかがビット変更マスクに存在する場合、該当する表の属性が変更されています。表属性の新しい値 (0 = OFF および 1 = ON) を判別するには、ビット変更値にある対応ビットをチェックしてください。

表の変更による列の追加、列追加のロールバック

表の変更による列の追加ログ・レコードは、ユーザーが ALTER TABLE ステートメントを用いて既存の表に列を追加するときに書き込まれます。以前の列と新しい列についての完全な情報が記録されます。

- 列カウント・エレメントは、列の古い数と、列の新しい合計数を示します。
- 平行配列には、表で定義されている列に関する情報が入ります。古い平行配列では、ALTER TABLE ステートメントの前の表が定義され、新しい平行配列では、ALTER TABLE ステートメントの結果の表が定義されます。
- 各平行配列は、次のものから構成されます。
  - 表記述レコード内の列記述子配列と等しい配列 (567 ページの『表の初期設定』 を参照)。
  - 表記述レコード内の LOB 記述子配列と等しい 2 番目の配列。ただし、この配列は最初の配列に対する平行配列であるため、使用されるエレメントは、最初の配列内の対応するエレメントがタイプ BLOB、CLOB、または DBCLOB であるものだけです。

表の変更による列の追加は通常ログ・レコードです。列追加のロールバックは補正ログ・レコードです。

表 108. 表の変更による列の追加、列追加のロールバック・ログ・レコードの構造

説明	種類	オフセット (バイト)
ログ・ヘッダー	DMSLogRecordheader	0(6)
埋め込み	char[ ]	6(2)
古い列カウント	sqluint32	8(4)
新しい列カウント	sqluint32	12(4)
古い平行配列 [1]	variable	16 (可変)
新しい平行配列 [2]	variable	可変
*全長: 40 バイト + 2 セットの平行配列。配列のサイズは、(古い/新しい列カウント) * 20 です。*		

配列エレメント:

1. この配列内の各エレメントは 8 バイトです。
2. この配列内の各エレメントは 12 バイトです。

列記述子配列または LOB 記述子配列の詳細については、 567 ページの表 101 を参照してください。

レコードの挿入、レコードの削除、レコード削除のロールバック、レコード更新のロールバック

これらのログ・レコードは、表の行が挿入または削除されるときに書き込まれます。レコードの挿入およびレコードの削除ログ・レコードは、更新中に、修正されたレコード・データに合わせてレコードの位置を変更しなければならない場合に生成されます。レコードの挿入およびレコードの削除は、通常ログ・レコードです。レコード削除のロールバックおよびレコード更新のロールバックは、補正ログ・レコードです。

表 109. レコードの挿入、レコードの削除、レコード削除のロールバック、レコード更新のロールバック・ログ・レコードの構造

説明	種類	オフセット (バイト)
ログ・ヘッダー	DMSLogRecordHeader	0(6)
埋め込み	char[ ]	6(2)
RID	sqluint32	8(4)
レコード長	unsigned short	12(2)
フリー・スペース	unsigned short	14(2)
レコード・オフセット	unsigned short	16(2)
レコードのヘッダーとデータ	variable	18 (可変)
全長: 18 バイト + レコード長		

レコード・ヘッダーおよびデータの詳細:

レコード・ヘッダー

- 4 バイト
- レコード・タイプ [a] (unsigned char、1 バイト)。
  - ビット値は異なるクラスおよびクラス内の可能なタイプを表します。レコードには、次の 2 つのクラスがあります。
    - 更新可能
    - 特殊制御
  - 各クラスには、次の 3 つのタイプを含めることができます。
    - 通常
    - ポインター
    - オーバーフロー
  - レコードは以下の場合にユーザー・データを含みます。
    - レコード・タイプが 0x00 または 0x10 の場合
    - ビット 0x04 が設定されている場合
- 予約済み (char、1 バイト)
- レコード長 (unsigned short、2 バイト)

レコード

- 可変長
- レコード・タイプ (unsigned char、1 バイト)。更新可能レコードには、次の 3 つのタイプがあります。
  - 0 - 内部制御
  - 1 - VALUE COMPRESSION オプションを使用しない定様式ユーザー・データ
  - 2 - VALUE COMPRESSION オプションを使用した定様式ユーザー・データ
- 予約済み (char、1 バイト)

- レコードの残りの部分は、レコード・タイプと、表に関して定義されている表記述子レコードによって異なります。レコード・タイプが内部制御である場合には、データは表示できません。
- レコード・タイプ値が 1 であるユーザー・データ・レコードには、次のフィールドが適用されます。
  - 固定長 (unsigned short、2 バイト)。これは、データ行の固定長セクションの長さです。
  - 定様式レコード (固定長および可変長)。
- レコード・タイプ値が 2 であるユーザー・データ・レコードには、次のフィールドが適用されます。
  - 列の数 (unsigned short、2 バイト)。これは、データ行のデータ部分の列の数です。
  - 定様式レコード (オフセット配列およびデータ部分)。

[a] レコード・データは、レコード・タイプ (レコード・ヘッダー内で指定される) が更新可能 (つまり、特殊制御ではない) 場合にのみ表示できます。

複数レコードの挿入、複数レコードの挿入のロールバック

これらのログ・レコードは、表の同じページに複数の行が挿入されるときに書き込まれます。複数レコードの挿入のロールバックは、補正ログ・レコードです。

表 110. 複数レコードの挿入

説明	種類	オフセット (バイト)
ログ・ヘッダー	DMSLogRecordHeader	0(6)
埋め込み	char[ ]	6(2)
レコードの数	unsigned short	8(2)
フリー・スペース	unsigned short	10(2)
レコード長の合計	unsigned short	12(2)
可変部の長さ	unsigned short	14(2)
プール・ページ番号	sqluint32	16(4)
レコード記述またはロールバック記述 表 111 および 575 ページの表 112 を参照。	variable	20(可変)
全長: 20 バイト + レコード長		

表 111. レコード記述 (各レコードごとに 1 つ)

説明	種類	オフセット (バイト)
RID	sqluint32	0(4)
レコード・オフセット	unsigned short	4(2)
レコードのヘッダーとデータ	variable	6 (可変)
全長: 6 バイト + レコード長		

表 112. ロールバック記述 (各レコードごとに 1 つ)

説明	種類	オフセット (バイト)
RID	sqluint32	0(4)
レコード・オフセット	unsigned short	4(2)
全長: 6 バイト		

レコード・ヘッダーとデータの詳細については、 573 ページの『レコード・ヘッダーおよびデータの詳細』 を参照してください。

VALUE COMPRESSION を使用しない表の定様式ユーザー・データ・レコード

VALUE COMPRESSION を使用しないで作成/変更された表の定様式レコードは、固定長データと可変長データの組み合わせにすることができます。すべてのフィールドには、固定長部分が含まれます。さらに、可変長部分を持つ 8 つのフィールド・タイプがあります。

- VARCHAR
- LONG VARCHAR
- DATALINK
- BLOB
- CLOB
- VARGRAPHIC
- LONG VARG
- DBCLOB

各フィールド・タイプの固定長部分の長さは、次のように判別できます。

- DECIMAL

  このフィールドは、*nnnnnn...s* の形式の標準のパック 10 進数です。フィールドの長さは、(精度 + 2)/2 です。符号ニブルは、正 (+) を表す xC と、負 (-) を表す xD または xB です。

- SMALLINT INTEGER BIGINT DOUBLE REAL CHAR GRAPHIC

  表記述子レコード内のこの列のエレメントにある長さフィールドは、フィールドの固定長サイズを含んでいます。

- DATE

  このフィールドは、*yyyymmdd* の形式の 4 バイトのパック 10 進数です。たとえば、1996 年 4 月 3 日は x'19960403' として表されます。

- TIME

  このフィールドは、*hhmmss* の形式の 3 バイトのパック 10 進数です。たとえば、1:32PM は x'133200' として表されます。

- TIMESTAMP

このフィールドは、*yyyymmddhhmmssuuuuuu* (日付 | 時刻 | ミリ秒) の形式の 10 バイトのパック 10 進数です。

- VARCHAR LONG VARCHAR DATALINK BLOB CLOB VARGRAPHIC LONG VARG DBCLOB

すべての可変長フィールドの固定長部分の長さは 4 です。

注: エレメント・アドレスについては、 567 ページの表 101 を参照してください。

以下のセクションでは、定様式レコード内の各フィールドの固定長部分の位置を説明します。

表記述子レコードは、表の列形式を記述します。列構造の配列を含み、そのエレメントはフィールド・タイプ、フィールド長、 NULL フラグ、およびフィールド・オフセットを示します。フィールド・オフセットは、定様式レコードの始まりからのオフセットであり、ここにフィールドの固定長部分があります。

表 113. 表記述子レコードの構造

レコード・タイプ	列の数	列の構造 • フィールド・タイプ • 長さ • NULL フラグ • フィールド・オフセット	LOB 情報

注: 詳細については、 567 ページの表 101 を参照してください。

NULL 可能 (NULL フラグによって指定されている) の列の場合、フィールドの固定長部分の後に追加のバイトがあります。このバイトには、次の 2 つの値のうち 1 つが含まれます。

- NOT NULL (0x00)
- NULL (0x01)

NULL 可能の列について定様式レコード内の NULL フラグが 0x00 に設定された場合には、レコードの固定長データ部分に有効な値があります。 NULL フラグ値が 0x01 である場合、データ・フィールド値は NULL です。

定様式ユーザー・データ・レコードには、ユーザーが見ることのできる表データが含まれます。これは固定長レコードとして形式化され、その後に可変長セクションが続きます。

表 114. VALUE COMPRESSION を使用しない表の定様式ユーザー・データ・レコードの構造

レコード・タイプ	固定長セクションの長さ	固定長セクション	可変長データ・セクション

注: 詳細については、 573 ページの表 109 を参照してください。

すべての可変長フィールド・タイプには、固定長セクションに 4 バイトの固定長データ部分 (さらに、列が NULL 可能であれば NULL フラグ) があります。最初の 2 バイト (short) は、固定長セクションの始まりからのオフセット (可変長データが存在する場所) を示します。次の 2 バイト (short) は、オフセット値によって参照されている可変長データの長さを指定します。

VALUE COMPRESSION を使用した表の定様式ユーザー・データ・レコード

VALUE COMPRESSION を使用して作成または変更された表の定様式レコードはオフセット配列とデータ部分から構成されています。配列内の各項目は、データ部分の対応する列データへの 2 バイト・オフセットです。データ部分の列データの数は、レコード・ヘッダーから知ることができます。オフセット配列の項目の数は、データ部分に存在する列データの数に 1 を加えたものです。

1. 圧縮された列の値は、属性バイトに使用されるディスク・スペースの中で 1 バイトのみ使用します。属性バイトは、列データが圧縮されていることを示します (たとえば、データ値は分かっているが、ディスクに保管されていないこと)。オフセットの高ビット (0x80) は、アクセスされているデータが属性バイトであることを示すために使用されます。 (そのため、対応する列データのオフセットを表すために使用されるのは 15 ビットのみです。)
2. 通常の列データの場合、列データが後に続きます。属性バイト、およびいかなる長さの標識も存在しません。
3. アクセスされているデータが属性バイトである場合、以下の 2 つの異なる値をとることができます。
   - `NULL  0x01  (値は NULL)`
   - `COMPRESSED SYSTEM DEFAULT  0x80  (値はシステム・デフォルトと等しい)`
4. 列データの長さは、現行オフセットと次の列のオフセットとの差です。

表 115. VALUE COMPRESSION を使用した表の定様式ユーザー・データ・レコードの構造

レコード・タイプ	データ部分の列の数	オフセット配列	データ部分

注: 詳細については、 573 ページの表 109 を参照してください。

空ページへのレコードの挿入、空ページからのレコードの削除、空ページからのレコードの削除のロールバック、空ページへのレコードの挿入のロールバック

これらのログ・レコードは、表がマルチディメンション・クラスタリング (MDC) 表の場合に、書き込まれます。空ページへの挿入ログ・レコードでは、レコードが挿入され、それがページの最初のレコードである場合に書き込まれます。この場合、ページはブロックの先頭ページではありません。このログ・レコードは、ブロックの先頭ページでのビットの更新に加え、ページへの挿入も記録します。これは、そのページが空ではないことを示しています。空ページからの削除ログ・レコードには、最後のレコードがページから削除されたときに書き込まれます。この場合、ページはブロックの先頭ページではありません。このログ・レコードは、ブロックの先頭ページでのビットの更新に加え、ページからの削除も記録します。これは、そのページが空であることを示しています。レコードの挿入およびレコードの

削除空ページは、通常のログ・レコードです。削除レコードのロールバックおよび挿入レコードのロールバックは、補正ログ・レコードです。

表 116. 空ページへのレコードの挿入、空ページからのレコードの削除、空ページからのレコードの削除のロールバック、空ページへのレコードの挿入のロールバック

説明	種類	オフセット (バイト)
ログ・ヘッダー	DMSLogRecordHeader	0(6)
埋め込み	char[]	6(2)
RID	sqluint32	8(4)
レコード長	unsigned short	12(2)
フリー・スペース	unsigned short	14(2)
ブロックの先頭ページ	sqluint32	16(4)
レコード・オフセット	unsigned short	20(2)
レコードのヘッダーとデータ	variable	22(可変)
全長: 22 バイト + レコード長		

注: レコードのヘッダーとデータの詳細については、 573 ページの表 109 を参照してください。

レコードの更新

レコードの更新ログ・レコードは、行が更新され、その記憶場所が変わらない場合に書き込まれます。ログ・レコードの形式は 2 つあり、それらはレコードの挿入およびレコードの削除ログ・レコードと同じです (572 ページの『レコードの挿入、レコードの削除、レコード削除のロールバック、レコード更新のロールバック』 を参照)。一方は、更新される行の更新*前* のイメージを含み、もう一方は更新される行の更新*後* のイメージを含みます。これは通常ログ・レコードです。

表 117. レコードの更新ログ・レコードの構造

説明	種類	オフセット (バイト)
ログ・ヘッダー	DMSLogRecordHeader	0(6)
埋め込み	char[ ]	6(2)
RID	sqluint32	8(4)
新しいレコード長	unsigned short	12(2)
フリー・スペース	unsigned short	14(2)
レコード・オフセット	unsigned short	16(2)
古いレコードのヘッダーとデータ	variable	18 (可変)
ログ・ヘッダー	DMSLogRecordHeader	可変 (6)
埋め込み	char[ ]	可変 (2)
RID	sqluint32	可変 (4)
古いレコード長	unsigned short	可変 (2)
フリー・スペース	unsigned short	可変 (2)
レコード・オフセット	unsigned short	可変 (2)
新しいレコードのヘッダーとデータ	variable	可変 (可変)

表 117. レコードの更新ログ・レコードの構造 (続き)

説明	種類	オフセット (バイト)
全長: 36 バイト + 2 つのレコード長		

長フィールド・マネージャーのログ・レコード

長フィールド・マネージャーのログ・レコードが書き込まれるのは、 LOG RETAIN がオンであるかまたは USEREXITS が使用可能の状態でデータベースが構成されている場合のみです。これらのログ・レコードは、長フィールド・データが挿入、削除、または更新されるたびに書き込まれます。

ログ・スペースを節約するために、データベースが循環ロギング用に構成されている場合には、表に挿入された長フィールド・データは記録されません。さらに、長フィールドの値が更新されると、前のイメージはシャドー化されて記録されません。

長フィールド・マネージャーのすべてのログ・レコードは、ヘッダーで始まります。

長フィールド・マネージャーのすべてのログ・レコード・オフセットは、ログ・マネージャーのログ・レコード・ヘッダーの終わりからのものです。

LONG VARCHAR OR LONG VARGRAPHIC 列をキャプチャーするように表が変更された (ALTER TABLE で INCLUDE LONGVAR COLUMNS が指定された) 場合、

- 長フィールド・マネージャーは適切な長フィールド・ログ・レコードを書き込みます。
- 長フィールド・データが更新されるときには、その更新は、古い長フィールド値の削除と、新しい値の挿入として扱われます。削除/追加長フィールド・レコードが表の更新操作に関連付けられるかどうかを判別するために、元の操作の値が長フィールド・マネージャーのログ・レコードに記録されます。
- 長フィールド列が存在する表が更新されたが、長フィールド列は更新されなかったときには、長フィールド・レコードの非更新が書き込まれます。
- 長フィールド・レコードの削除および長フィールド・レコードの非更新は、通知専用のログ・レコードです。

表 118. 長フィールド・マネージャーのログ・レコード・ヘッダー (LongFieldLogRecordHeader)

説明	種類	オフセット (バイト)
発信元コード (コンポーネント ID = 3)	unsigned char	0(1)
操作タイプ (580 ページの表 119 を参照。)	unsigned char	1(1)
表スペース ID	unsigned short	2(2)
オブジェクト ID	unsigned short	4(2)
親表スペース ID [1]	unsigned short	6(2)
親オブジェクト ID [2]	unsigned short	8(2)

表 118. 長フィールド・マネージャーのログ・レコード・ヘッダー (LongFieldLogRecordHeader) (続き)

説明	種類	オフセット (バイト)
全長: 10 バイト		

注:

1. データ・オブジェクトの表スペース ID
2. データ・オブジェクトのオブジェクト ID

表 119. 長フィールド・マネージャーのログ・レコード・ヘッダーの操作タイプ値および定義

値	定義
113	長フィールド・レコードの追加
114	長フィールド・レコードの削除
115	長フィールド・レコードの非更新

長フィールド・レコードの追加/削除/非更新

これらのログ・レコードは、長フィールド・データが挿入、削除、または更新されるたびに書き込まれます。データの長さは、次の 512 バイト境界に切り上げられます。

表 120. 長フィールド・レコードの追加/削除/非更新ログ・レコードの構造

説明	種類	オフセット (バイト)
ログ・ヘッダー	LongFieldLogRecordHeader	0(10)
予約済み	char	10(1)
元の操作タイプ [1]	char	11(1)
列 ID [2]	char	12(2)
長フィールドの長さ [3]	unsigned short	14(2)
ファイル・オフセット [4]	sqluint32	16(4)
長フィールド・データ	char[ ]	20(可変)

注:

1. 元の操作タイプ

   ```
   1     Insert
   2     Delete
   4     Update
   ```

2. ログ・レコードが適用される列番号。列番号は 0 から始まります。
3. 長フィールド・データの長さは、 512 バイト単位です (実際のデータ長は記録されません)。このフィールドは常に正の値です。ゼロの長さの長フィールドが挿入、削除、または更新された場合は、長フィールド・マネージャーはログ・レコードを作成しません。
4. データが挿入されている長フィールド・オブジェクトへの 512 バイト単位のオフセット。

トランザクション・マネージャーのログ・レコード

トランザクション・マネージャーは、トランザクション・イベント (たとえば、コミットまたはロールバック) の完了を示すログ・レコードを生成します。ログ・レコード内のタイム・スタンプは、協定世界時 (CUT) であり、 1970 年 1 月 1 日から経過した時間 (秒単位) を示します。

通常コミット

このログ・レコードは、トランザクションのコミット時に書き込まれます。これは、単一パーティション・データベース環境のトランザクションに関して、またはコーディネーター・データベース・パーティションのデータだけを変更したパーティション・データベース環境のトランザクションに関して使用されます。

表 121. 通常コミット・ログ・レコードの構造

説明	種類	オフセット (バイト)
ログ・ヘッダー	LogManagerLogRecordHeader	0(20)
トランザクションがコミットされた時刻	sqluint64	20(8)
アプリケーションの許可 ID (ログ・レコードが伝搬可能とマークされている場合)	char[ ]	28(可変)
全長: 28 バイト + 伝搬可能な変数 (伝搬不可能な 28 バイト)		

ヒューリスティック・コミット

このログ・レコードは、未確定トランザクションがコミットされるときに書き込まれます。

表 122. ヒューリスティック・コミット・ログ・レコードの構造

説明	種類	オフセット (バイト)
ログ・ヘッダー	LogManagerLogRecordHeader	0(20)
トランザクションがコミットされた時刻	sqluint64	20(8)
アプリケーションの許可 ID (ログ・レコードが伝搬可能とマークされている場合)	char[ ]	28(可変)
全長: 28 バイト + 伝搬可能な変数 (伝搬不可能な 28 バイト)		

MPP コーディネーター・ノード・コミット

このログ・レコードは、少なくとも 1 つの従属ノードで更新を実行するアプリケーションについてコーディネーター・ノードに書き込まれます。

表 123. MPP コーディネーター・ノード・コミット・ログ・レコードの構造

説明	種類	オフセット (バイト)
ログ・ヘッダー	LogManagerLogRecordHeader	0(20)
トランザクションがコミットされた時刻	sqluint64	20(8)
トランザクションの MPP ID	SQLP_GXID	28(20)
最大ノード番号	unsigned short	48(2)

表 123. MPP コーディネーター・ノード・コミット・ログ・レコードの構造 (続き)

説明	種類	オフセット (バイト)
TNL	unsigned char [ ]	50(最大ノード番号/8 + 1)
アプリケーションの許可 ID (ログ・レコードが伝搬可能とマークされている場合)	char[ ]	可変 (可変)
全長: 変数		

MPP 従属ノード・コミット

このログ・レコードは、MPP 内の従属ノードで書き込まれます。

表 124. MPP 従属ノード・コミット・ログ・レコードの構造

説明	種類	オフセット (バイト)
ログ・ヘッダー	LogManagerLogRecordHeader	0(20)
トランザクションがコミットされた時刻	sqluint64	20(8)
トランザクションの MPP ID	SQLP_GXID	28(20)
予約済み	unsigned short	48(可変 + 2)
許可 ID (ログ・レコードが伝搬可能とマークされている場合)	char[ ]	可変 (可変)
全長: 48 バイト + 伝搬可能な変数 (伝搬不可能な 48 バイト)		

通常打ち切り

このログ・レコードは、以下のいずれかのイベントの後でトランザクションが打ち切られるときに書き込まれます。

- ユーザーが ROLLBACK を発行した
- デッドロックが発生した
- クラッシュ・リカバリー中に暗黙的なロールバックが起こった
- ROLLFORWARD リカバリー中に暗黙的なロールバックが起こった

表 125. 通常打ち切りログ・レコードの構造

説明	種類	オフセット (バイト)
ログ・ヘッダー	LogManagerLogRecordHeader	0(20)
アプリケーションの許可 ID (ログ・レコードが伝搬可能とマークされている場合)	char[ ]	20(可変)
全長: 20 バイト + 変数 (伝搬不可能な 20 バイト)		

ヒューリスティック打ち切り

このログ・レコードは、未確定トランザクションが打ち切られるときに書き込まれます。

表 126. ヒューリスティック打ち切りログ・レコードの構造

説明	種類	オフセット (バイト)
ログ・ヘッダー	LogManagerLogRecordHeader	0(20)

表 126. ヒューリスティック打ち切りログ・レコードの構造 (続き)

説明	種類	オフセット (バイト)
アプリケーションの許可 ID (ログ・レコードが伝搬可能とマークされている場合)	char[ ]	20(可変)
全長: 20 バイト + 変数 (伝搬不可能な 20 バイト)		

ローカル・ペンディング・リスト

このログ・レコードは、トランザクションがコミットされるときに、ペンディング・リストが存在する場合に書き込まれます。ペンディング・リストは、ユーザー/アプリケーションが COMMIT を発行するときにのみ実行できるリカバリー不能操作 (ファイルの削除など) のリンク・リストです。この可変長構造には、ペンディング・リストの項目が含まれます。

表 127. ローカル・ペンディング・リスト・ログ・レコードの構造

説明	種類	オフセット (バイト)
ログ・ヘッダー	LogManagerLogRecordHeader	0(20)
トランザクションがコミットされた時刻	sqluint64	20(8)
許可 ID の長さ [1]	unsigned short	28(2)
アプリケーションの許可 ID[1]	char[ ]	30(可変)[2]
ペンディング・リストの項目	variable	可変 (可変)
全長: 30 バイト + 伝搬可能な変数 (28 バイト + 伝搬不可能なペンディング・リスト項目)		

注:

1. ログ・レコードが伝搬可能としてマークされている場合
2. 許可 ID の長さに基づく変数

グローバル・ペンディング・リスト

このログ・レコードは、2 フェーズ・コミットに必要なトランザクションがコミットされるときに、ペンディング・リストが存在する場合に書き込まれます。ペンディング・リストには、ユーザー/アプリケーションが COMMIT を発行するときにのみ実行できるリカバリー不能操作 (ファイルの削除など) が含まれます。この可変長構造には、ペンディング・リストの項目が含まれます。

表 128. グローバル・ペンディング・リスト・ログ・レコードの構造

説明	種類	オフセット (バイト)
ログ・ヘッダー	LogManagerLogRecordHeader	0(20)
許可 ID の長さ [1]	unsigned short	20(2)
アプリケーションの許可 ID[1]	char[ ]	22(可変)[2]
グローバル・ペンディング・リストの項目	variable	可変 (可変)
全長: 22 バイト + 伝搬可能な変数 (20 バイト + 伝搬不可能なペンディング・リスト項目)		

注:

1. ログ・レコードが伝搬可能としてマークされている場合
2. 許可 ID の長さに基づく変数

XA 準備

このログ・レコードは、単一ノード環境にある XA トランザクションについて、あるいは MPP にあるコーディネーター・ノードに書き込まれます。これは、XA アプリケーション専用です。このログ・レコードは、トランザクションの準備を 2 フェーズ・コミットの一部としてマークするために書き込まれます。 XA 準備ログ・レコードは、トランザクションを開始したアプリケーションを記述し、未確定トランザクションの再作成に使用されます。

表 129. XA 準備ログ・レコードの構造

説明	種類	オフセット (バイト)
ログ・ヘッダー	LogManagerLogRecordHeader	0(20)
トランザクションが準備された時刻	sqluint64	20(8)
トランザクションによって使用されたログ・スペース	sqluint64	28(8)
トランザクション・ノード・リストのサイズ	sqluint32	36(4)
トランザクション・ノード・リスト	unsigned char [ ]	40(可変)
トランザクションの XA ID	SQLXA_XID	可変 (140)
アプリケーション情報の長さ	sqluint32	可変 (4)
コード・ページ ID	sqluint32	可変 (4)
トランザクションの開始時刻	sqluint32	可変 (4)
アプリケーション名	char[ ]	可変 (20)
アプリケーション ID	char[ ]	可変 (32)
シーケンス番号	char[ ]	可変 (4)
クライアントによって使用されたデータベース別名	char[ ]	240(20)
許可 ID	char[ ]	可変 (可変)
同期ログ情報	variable	可変 (可変)
全長: 268 バイト + 変数		

MPP 従属ノード準備

このログ・レコードは、従属ノード上の MPP トランザクションについて書き込まれます。このログ・レコードは、トランザクションの準備を 2 フェーズ・コミットの一部としてマークするために書き込まれます。 MPP 従属ノード準備ログ・レコードは、トランザクションを開始したアプリケーションを記述し、未確定トランザクションの再作成に使用されます。

表 130. MPP 従属ノード準備ログ・レコードの構造

説明	種類	オフセット (バイト)
ログ・ヘッダー	LogManagerLogRecordHeader	0(20)
トランザクションが準備された時刻	sqluint64	20(8)
トランザクションによって使用されたログ・スペース	sqluint64	28(8)
コーディネーター・ノード LSN	SQLP_LSN	36(6)
埋め込み	char[ ]	42(2)

表 130. MPP 従属ノード準備ログ・レコードの構造 (続き)

説明	種類	オフセット (バイト)
トランザクションの MPP ID	SQLP_GXID	44(20)
アプリケーション情報の長さ	sqluint32	64(4)
コード・ページ	sqluint32	68(4)
トランザクションの開始時刻	sqluint32	72(4)
アプリケーション名	char[ ]	76(20)
アプリケーション ID	char[ ]	96(32)
シーケンス番号	char[ ]	128(4)
クライアントによって使用されたデータベース別名	char[ ]	132(20)
許可 ID	char[ ]	152(可変)
全長 : 152 バイト + 変数		

バックアウト解放

このログ・レコードは、バックアウト解放インターバルの終了をマークするために使用されます。バックアウト解放インターバルは、トランザクションが打ち切られたときに補正されていないログ・レコードのセットです。このログ・レコードには、6 バイトのログ・シーケンス番号 (*complsn*、オフセット 20 から始まるログ・レコード・ヘッダーに格納される) だけが含まれます。このログ・レコードを (打ち切られたトランザクションの後の) ロールバック時に読み取ると、 *complsn* が次に補正すべきログ・レコードをマークします。

表 131. 移行の開始ログ・レコードの構造

説明	種類	オフセット (バイト)
ログ・ヘッダー	LogManagerLogRecordHeader	0(20)
Complsn	SQLP_LSN	20(6)
全長: 26 バイト		

ユーティリティー・マネージャーのログ・レコード

ユーティリティー・マネージャーは、次の DB2 UDB ユーティリティーに関連するログ・レコードを生成します。

- 移行
- ロード
- バックアップ
- 表スペースのロールフォワード

ログ・レコードは、要求された活動の始まりと終わりを示します。ユーティリティー・マネージャーのすべてのログ・レコードは、それらが影響を与える表に関係なく、伝搬可能としてマークされます。

移行の開始

このログ・レコードは、カタログ移行の開始と関連しています。

表 132. 移行の開始ログ・レコードの構造

説明	種類	オフセット (バイト)
ログ・ヘッダー	LogManagerLogRecordHeader	0(20)
移行の開始時刻	char[ ]	20(10)
移行元のリリース	unsigned short	30(2)
移行先のリリース	unsigned short	32(2)
全長: 34 バイト		

移行の終了

このログ・レコードは、カタログ移行の正常終了と関連しています。

表 133. 移行の終了ログ・レコードの構造

説明	種類	オフセット (バイト)
ログ・ヘッダー	LogManagerLogRecordHeader	0(20)
移行の終了時刻	char[ ]	20(10)
移行先のリリース	unsigned short	30(2)
全長: 32 バイト		

ロードの開始

このログ・レコードは、ロードの開始と関連しています。

表 134. ロードの開始ログ・レコードの構造

説明	種類	オフセット (バイト)
ログ・ヘッダー	LogManagerLogRecordHeader	0(20)
ログ・レコード ID	sqluint32	20(4)
プール ID	unsigned short	24(2)
オブジェクト ID	unsigned short	26(2)
フラグ	unsigned char	28(1)
オブジェクト・プールのリスト	variable	29 (可変)
全長: 29 バイト + 可変長部分		

表ロードの削除開始

このログ・レコードは、ロード操作における削除フェーズの開始と関連しています。削除フェーズが開始されるのは、主キー値が重複して存在する場合だけです。

表 135. 表ロードの削除開始ログ・レコードの構造

説明	種類	オフセット (バイト)
ログ・ヘッダー	LogManagerLogRecordHeader	0(20)
全長: 20 バイト		

ロード削除開始の補正

このログ・レコードは、ロード操作における削除フェーズの終了と関連しています。

表 136. ロード削除開始の補正ログ・レコードの構造

説明	種類	オフセット (バイト)
ログ・ヘッダー	LogManagerLogRecordHeader	0(20)
全長: 20 バイト		

ロード・ペンディング・リスト

このログ・レコードは、ロード・トランザクションのコミット時に書き込まれます。ペンディング・リストは、トランザクションがコミットするまで据え置かれるリカバリー不能操作のリンク・リストです。このトランザクションの後には、コミット・ログ・レコードは書き込まれません。

表 137. ロード・ペンディング・リスト・ログ・レコードの構造

説明	種類	オフセット (バイト)
ログ・ヘッダー	LogManagerLogRecordHeader	0(20)
トランザクションがコミットされた時刻	sqluint64	20(8)
アプリケーションの許可 ID (ログ・レコードが伝搬可能とマークされている場合)	char[ ]	28(9)
ペンディング・リストの項目	variable	37(可変)
全長: 37 バイト + 伝搬可能なペンディング・リスト項目 (28 バイト + 伝搬不可能なペンディング・リスト項目)		

バックアップの終了

このログ・レコードは、バックアップの正常終了と関連しています。

表 138. バックアップの終了ログ・レコードの構造

説明	種類	オフセット (バイト)
ログ・ヘッダー	LogManagerLogRecordHeader	0(20)
バックアップの終了時刻	sqluint64	20(8)
全長: 28 バイト		

表スペースのロールフォワード

このログ・レコードは、表スペースの ROLLFORWARD リカバリーと関連しています。これは、正常にロールフォワードされたそれぞれの表スペースについて書き込まれます。

表 139. 表スペースのロールフォワード・ログ・レコードの構造

説明	種類	オフセット (バイト)
ログ・ヘッダー	LogManagerLogRecordHeader	0(20)
表スペース ID	unsigned short	20(2)
全長: 22 バイト		

PIT への表スペース・ロールフォワードの開始

このログ・レコードは、表スペースの ROLLFORWARD リカバリーと関連しています。これは、特定の時点への表スペース・ロールフォワードの始まりをマークします。

表 140. PIT への表スペース・ロールフォワードの開始ログ・レコードの構造

説明	種類	オフセット (バイト)
このログ・レコードのタイム・スタンプ	sqluint64	0(8)
表スペースがロールフォワードされる先のタイム・スタンプ	sqluint64	8(8)
ロールフォワードされるプールの数	unsigned short	16(2)
ロールフォワードされるプール ID の整数リスト	int*numpools	18 (可変)
全長: 10 バイト + 可変長部分		

PIT への表スペース・ロールフォワードの終了

このログ・レコードは、表スペースの ROLLFORWARD リカバリーと関連しています。これは、特定の時点への表スペース・ロールフォワードの終わりをマークします。

表 141. PIT への表スペース・ロールフォワードの終了ログ・レコードの構造

説明	種類	オフセット (バイト)
このログ・レコードのタイム・スタンプ	sqluint64	0(8)
表スペースがロールフォワードされた先のタイム・スタンプ	sqluint32	8(8)
値が TRUE (ロールフォワードが成功した場合) または FALSE (ロールフォワードが取り消された場合) になるフラグ	int	16(4)
全長: 24 バイト		

Data Links Manager のログ・レコード

Data Links Manager のログ・レコードは、DDL、DML、または DATALINK 列に関係したトランザクション・イベント完了の結果です。これらのログ・レコードが書き込まれるのは、 DDL または DML がファイルのリンク制御属性を持つ DATALINK 列に関連している場合だけです。

表 142. Data Links Manager のログ・レコード・ヘッダーの構造 (DLMLogRecordHeader)

説明	種類	オフセット (バイト)
コンポーネント ID (=8)	unsigned char	0(1)
ファンクション ID (589 ページの表 143 を参照してください。)	unsigned char	1(1)
埋め込み	char[ ]	2(2)
全長: 6 バイト		

表 143. Data Links Manager のログ・レコード・ヘッダーのファンクション ID および値

ID	値
LINK_FILE	33
UNLINK_FILE	34
DELETE_GROUP	35
DELETE_PGROUP	36
DLFM_PREPARE	37

ファイルのリンク

ファイルのリンク・ログ・レコードは、DATALINK 列を持つ表に対して挿入または更新が行われた結果、ファイルへのリンクが作成されたときに書き込まれます。新しいリンクが 1 つ作成されるごとに、ログ・レコードが 1 つずつ書き込まれます。このログ・レコードは、取り消しのためにのみ使用されます。

表 144. ファイルのリンク・ログ・レコードの構造

説明	種類	オフセット (バイト)
ログ・ヘッダー	DLMLogRecordHeader	0(4)
サーバー ID	sqlint32	4(4)
読み取り専用	int	8(4)
許可 ID	char[ ]	12(8)
グループ ID	char[ ]	20(17)
操作タイプ (表 145 を参照。)	char[ ]	37(1)
アクセス・コントロール	unsigned short	38(2)
接頭部 ID	char[ ]	40(9)
埋め込み	char[ ]	49(3)
リカバリー ID	char[ ]	52(7)
埋め込み	char[ ]	59(1)
タイム・スタンプ	sqluint32	60(4)
語幹名の長さ	sqluint32	64(4)
語幹名	variable	68(可変)
ServerNameLen[1]	sqluint32	可変 (4)
PrefixNameLen[1]	sqluint32	可変 (4)
ServeNamePrefixName[1]	variable	可変 (可変)
全長: 伝搬不可能の場合、 68 + 語幹名の長さ (伝搬可能の場合、 76 + 語幹名の長さ + サーバー名の長さ + 接頭部名の長さ)		

注:

1. ログ・レコードが伝搬可能な場合。

表 145. リンク・ファイルのログ・レコード構造の操作タイプおよび値

ID	値
LINK_FILE_ONLY (DLVALUE によって構成された値)	0

表 145. リンク・ファイルのログ・レコード構造の操作タイプおよび値 (続き)

ID	値
LINK_NEW_VERSION (DLNEWCOPY によって構成された値)	10
LINK_PREVIOUS_VERSION (DLPREVIOUSCOPY によって構成された値)	20
LINK_REPLACE_CONTENT (DLREPLACECONTENT によって構成された値)	30

ファイルのリンク解除

ファイルのリンク解除ログ・レコードは、DATALINK 列を持つ表に対して削除または更新が行われた結果、ファイルへのリンクがドロップされたときに書き込まれます。リンクが 1 つドロップされるごとに、ログ・レコードが 1 つずつ書き込まれます。このログ・レコードは、取り消しのためにのみ使用されます。

表 146. ファイルのリンク解除ログ・レコードの構造

説明	種類	オフセット (バイト)
ログ・ヘッダー	DLMLogRecordHeader	0(4)
サーバー ID	sqlint32	4(4)
接頭部 ID	char[ ]	8(9)
操作タイプ (591 ページの表 147 を参照。)	char[ ]	17(1)
埋め込み	char[ ]	18(2)
リカバリー ID	char[ ]	20(7)
埋め込み	char[ ]	27(1)
タイム・スタンプ	sqluint32	28(4)
語幹名の長さ	sqluint32	32(4)
語幹名	variable	36(可変)
poolID[1]	unsigned short	可変 (2)
objectID[1]	unsigned short	可変 (2)
colNum[1]	unsigned short	可変 (2)
padding[1]	char[ ]	可変 (2)
ServerNameLen[1]	sqluint32	可変 (4)
PrefixNameLen[1]	sqluint32	可変 (4)
ServerNamePrefixName[1]	variable	可変 (可変)
全長: 伝搬不可能の場合、 36 + 語幹名の長さ (伝搬可能の場合、 52 + 語幹名の長さ + サーバー名の長さ + 接頭部名の長さ)		

注:

1. ログ・レコードが伝搬可能な場合。

表 147. リンク・ファイルのログ・レコード構造の操作タイプおよび値

ID	値
UNLINK_REGULAR_FILE	0
UNLINK_UPDATE_IN_PLACE_FILE	10

グループの削除

グループの削除ログ・レコードは、DATALINK 列を持つ表 (ファイルのリンク制御属性を持つ) がドロップされたときに書き込まれます。データベースに DB2 Data Links Manager が構成されるごとに、そのような DATALINK 列についてログ・レコードが 1 つずつ書き込まれます。ある DB2 Data Links Manager に関係してログ・レコードが書き込まれるのは、表がドロップされたときに、その DB2 Data Links Manager にグループが定義されていた場合です。このログ・レコードは、取り消しのためにのみ使用されます。

表 148. グループの削除ログ・レコードの構造

説明	種類	オフセット (バイト)
ログ・ヘッダー	DLMLogRecordHeader	0(4)
サーバー ID	sqlint32	4(4)
リカバリー ID	char[ ]	8(7)
埋め込み	char[ ]	15(1)
グループ ID	char[ ]	16(17)
埋め込み	char[ ]	33(3)
全長: 36 バイト		

p グループの削除

p グループの削除ログ・レコードは、表スペースがドロップされるときに書き込まれます。データベースに DB2 Data Links Manager が 1 つ構成されるごとに、ログ・レコードが 1 つずつ書き込まれます。ある DB2 Data Links Manager に関係してログ・レコードが書き込まれるのは、表スペースがドロップされたときに、その DB2 Data Links Manager に p グループが定義されていた場合です。このログ・レコードは、取り消しのためにのみ使用されます。

表 149. p グループの削除ログ・レコードの構造

説明	種類	オフセット (バイト)
ログ・ヘッダー	DLMLogRecordHeader	0(4)
サーバー ID	sqlint32	4(4)
poolLifeLSN	SQLU_LSN	8(6)
プール ID	unsigned short	14(2)
リカバリー ID	char[ ]	16(7)
埋め込み	char[ ]	23(1)
全長: 24 バイト		

DLFM 準備

DLFM 準備ログ・レコードは、DB2 Data Links Manager に関係したトランザクションに 2 フェーズ・コミットが使用されている場合の準備段階で書き込まれます。DLFM 準備ログ・レコードは、未確定な DB2 Data Links Manager に関係したトランザクションを再作成するために使用されます。

表 150. DLFM 準備ログ・レコードの構造

説明	種類	オフセット (バイト)
ログ・ヘッダー	DLMLogRecordHeader	0(4)
DLFM の数	unsigned short	4(4)
サーバー ID	variable	8(可変)
*全長: 8 バイト + (DLFM の数 * 4)*		

関連資料:

• 219 ページの『db2ReadLog - ログの非同期読み取り』

付録 G. アプリケーションの移行

管理 API および アプリケーションの移行

このセクションでは、アプリケーションをバージョン 8 に移行する際に考慮すべき問題について説明します。

可能な操作のシナリオが 4 つあります。

1. 移行されていないデータベースに対してバージョン 8 以前のアプリケーションを実行する
2. 移行されたデータベースに対してバージョン 8 以前のアプリケーションを実行する
3. バージョン 8 の API でアプリケーションを更新する
4. 移行されたデータベースに対してバージョン 8 のアプリケーションを実行する

1 番目と 4 番目は、修飾を必要としない、一貫したオペレーティング環境です。

データベースのみが移行されている 2 番目の操作環境は、アプリケーションに変更を加えなくても機能するはずです (バックレベルのアプリケーションがサポートされているため)。ただし、新しいバージョンについては、いくらかの非互換性が生じる可能性があります。

アプリケーションがバージョン 8 の API で更新される 3 番目のシナリオでは、以下の点を考慮する必要があります。

- バージョン 8 で廃止されている、バージョン 8 以前のすべての API は、バージョン 8 のヘッダー・ファイルで引き続き定義されています。これにより、古いアプリケーションがバージョン 8 のヘッダーを用いてコンパイルおよびリンクすることができます。
- 廃止された API はできるだけ早くアプリケーションから除去して、アプリケーションがバージョン 8 で利用可能な新しい関数を十分に活用できるようにし、将来の拡張に備える必要があります。
- 以下にリストされている API の名前は、バージョン 8 の新しい関数用に変更されています。ユーザーはアプリケーションのソース・コード内でこれらの名前をスキャンして、アプリケーションのバージョン 8 への移行に続いて必要になる変更を識別することが必要です。

  ここにリストされていない API には、アプリケーションの移行に続く変更は必要ありません。

  アプリケーションには、使用中のアプリケーション・プログラミング言語によって、 API 呼び出しの汎用バージョンが含まれることがあります。すべてのケースで、汎用バージョンの API 名は、 4 番目の文字が常に **g** である点を除き、 C バージョンの名前と同一です。

関連資料:

• 594 ページの『変更された API およびデータ構造』

変更された API およびデータ構造

表 151. バックレベルがサポートされた API およびデータ構造

API またはデータ構造 (バージョン)	記述名	新しい API または データ構造 (バージョン)
sqlbftsq (V2)	表スペース照会の取り出し	sqlbftpq (V5)
sqlbstsq (V2)	単一の表スペース照会	sqlbstpq (V5)
sqlbtsq (V2)	表スペース照会	sqlbmtsq (V5)
sqlectdd (V2)	データベースのカタログ	sqlecadb (V5)
sqledosd (V8.1)	データベース・ディレクトリー・スキャンのオープン	db2DbDirOpenScan (V8.2)
sqledgne (V8.1)	データベース・ディレクトリーの次項目の入手	db2DbDirGetNextEntry (V8.2)
sqledcls (V8.1)	データベース・ディレクトリー・スキャンのクローズ	db2DbDirCloseScan (V8.2)
sqlepstart (V5)	データベース・マネージャーの始動	db2InstanceStart (V8)
sqlepstp (V5)	データベース・マネージャーの開始	db2InstanceStop (V8)
sqlepstr (V2)	データベース・マネージャーの開始 (DB2 パラレル・エディション バージョン 1.2)	db2InstanceStart (V8)
sqlestar (V2)	データベース・マネージャーの開始 (DB2 バージョン 2)	db2InstanceStart (V8)
sqlestop (V2)	データベース・マネージャーの開始	db2InstanceStop (V8)
sqlerstd (V5)	データベースの再始動	db2DatabaseRestart (V6)
sqlfddb (V7)	データベース構成のデフォルトの入手	db2CfgGet (V8)
sqlfdsys (V7)	データベース・マネージャー構成のデフォルトの入手	db2CfgGet (V8)
sqlfrdb (V7)	データベース構成のリセット	db2CfgSet (V8)
sqlfrsys (V7)	データベース・マネージャー構成のリセット	db2CfgSet (V8)
sqlfudb (V7)	データベース構成の更新	db2CfgSet (V8)
sqlfusys (V7)	データベース・マネージャー構成の更新	db2CfgSet (V8)
sqlfxdb (V7)	データベース構成の入手	db2CfgGet (V8)
sqlfxsys (V7)	データベース構成の入手	db2CfgGet (V8)
sqlmon (V6)	モニター・スイッチの入手/更新	db2MonitorSwitches (V7)
sqlmonss (V5)	スナップショットの入手	db2GetSnapshot (V6)
sqlmonsz (V6)	sqlmonss() 出力バッファーに必要なサイズの見積もり	db2GetSnapshotSize (V7)
sqlmrset (V6)	モニターのリセット	db2ResetMonitor (V7)
sqlubkp (V5)	データベースのバックアップ	db2Backup (V8)
sqlubkup (V2)	データベースのバックアップ	db2Backup (V8)
sqluexpr	エクスポート	db2Export (V8)
sqlugrpi (V2)	行のパーティション情報の入手 (DB2 パラレル・エディション バージョン 1.x)	sqlugrpn (V5)
sqluhcls (V5)	リカバリー履歴ファイルのスキャンのクローズ	db2HistoryCloseScan (V6)
sqluhget (V5)	履歴ファイルからの DDL 情報の検索	db2HistoryGetEntry (V6)
sqluhgne (V5)	リカバリー履歴ファイルの次項目の入手	db2HistoryGetEntry (V6)

表 151. バックレベルがサポートされた API およびデータ構造 (続き)

API またはデータ構造 (バージョン)	記述名	新しい API またはデータ構造 (バージョン)
sqluhops (V5)	リカバリー履歴ファイルのスキャンのオープン	db2HistoryOpenScan (V6)
sqluhprn (V5)	リカバリー履歴ファイルの整理	db2Prune (V6)
sqluhupd (V5)	リカバリー履歴ファイルの更新	db2HistoryUpdate (V6)
sqluimpr	インポート	db2Import (V8)
sqluload (V7)	ロード	db2Load (V8)
sqluqry (V5)	ロードの照会	db2LoadQuery (V6)
sqlureot (V7)	表の再編成	db2Reorg (V8)
sqlurestore (V7)	RESTORE DATABASE	db2Restore (V8)
sqlurlog (V7)	ログの非同期読み取り	db2ReadLog (V8)
sqluroll (V7)	データベースのロールフォワード	db2Rollforward (V8)
sqlursto (V2)	RESTORE DATABASE	sqlurst (V5)
sqlustat (V7)	統計の実行	db2Runstats (V8)
sqlxhcom (V2)	未確定トランザクションのコミット	sqlxphcm (V5)
sqlxhqry (V2)	未確定トランザクションのリスト	sqlxphqr (V5)
sqlxhrol (V2)	未確定トランザクションのロールバック	sqlxphrl (V5)
sqlxphqr (V7)	未確定トランザクションのリスト	db2XaListIndTrans (V8)
SQLB-TBSQRY-DATA (V2)	表スペース・データ構造	SQLB-TBSPQRY-DATA (V5)
SQLE-START-OPTIONS (V7)	データベース・マネージャーのデータ構成の始動	db2StartOptionsStruct (V8)
SQLEDBSTOPOPT (V7)	データベース・マネージャーのデータ構成の始動	db2StopOptionsStruct (V8)
SQLEDBSTRTOPT (V2)	データベース・マネージャーの開始データ構造 (DB2 パラレル・エディション バージョン 1.2)	db2StartOptionsStruct (V8)
SQLEDINFO (v8.1)	データベース・ディレクトリーの次項目の入手データ構造	db2DbDirInfo (V8.2)
SQLUEXPT-OUT	出力構成のエクスポート	db2ExportOut (V8.2)
SQLUHINFO and SQLUHADM (V5)	履歴ファイルのデータ構造	db2HistData (V6)
SQLUIMPT-IN	入力構成のインポート	db2ImportIn (V8.2)
SQLUIMPT-OUT	出力構成のインポート	db2ImportOut (V8.2)
SQLULOAD-IN (V7)	入力構成のロード	db2LoadIn (V8)
SQLULOAD-OUT (V7)	出力構成のロード	db2LoadOut (V8)
SQLXA-RECOVER (V7)	トランザクション API 構成	db2XaRecoverStruct

表 152. バックレベル・サポートが存在しない API

名前	記述名	V8 でサポートされている API
sqlufrol/sqlgfrol	データベースのロールフォワード (DB2 バージョン 1.1)	db2Rollforward
sqluprfw	データベースのロールフォワード (DB2 パラレル・エディション バージョン 1.x)	db2Rollforward
sqlurfwd/sqlgrfwd	データベースのロールフォワード (DB2 バージョン 1.2)	db2Rollforward

表 152. バックレベル・サポートが存在しない API (続き)

名前	記述名	V8 でサポートされている API
sqlurllf/sqlgrfwd	データベースのロールフォワード (DB2 バージョン 2)	db2Rollforward

関連資料:

• 593 ページの『管理 API および アプリケーションの移行』

付録 H. DB2 Universal Database の技術情報

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』
- 598 ページの『DB2 インフォメーション・センター』

関連タスク:

- 619 ページの『DB2 ツールからコンテキスト・ヘルプを呼び出す』
- 609 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』
- 620 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』
- 620 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』
- 621 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:

- 611 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:

- 600 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:

- 609 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』
- 610 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』
- 608 ページの『DB2 インフォメーション・センターの呼び出し』
- 602 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)』
- 605 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:

- 598 ページの『DB2 インフォメーション・センター』

関連タスク:

- 609 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』
- 602 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)』
- 605 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)』

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:

- 598 ページの『DB2 インフォメーション・センター』
- 600 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)』
- 609 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』
- 610 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』
- 608 ページの『DB2 インフォメーション・センターの呼び出し』
- 605 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:

- 598 ページの『DB2 インフォメーション・センター』
- 600 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)』

- 609 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』
- 610 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』
- 608 ページの『DB2 インフォメーション・センターの呼び出し』
- 602 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:

- 598 ページの『DB2 インフォメーション・センター』

関連タスク:

- 610 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』
- 619 ページの『DB2 ツールからコンテキスト・ヘルプを呼び出す』
- 609 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』
- 620 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』
- 620 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』
- 621 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』

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:

- 600 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:

- 608 ページの『DB2 インフォメーション・センターの呼び出し』
- 602 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)』
- 605 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:

- 598 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 153. 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 154. 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 154. 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 155. 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 156. 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 157. 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 158. 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 159. 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 160. 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 161. 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:

- 597 ページの『DB2 資料とヘルプ』

関連タスク:

- 617 ページの『PDF ファイルからの DB2 資料の印刷方法』
- 618 ページの『DB2 の印刷資料の注文方法』
- 619 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:

- 598 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)』
- 618 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:

- 611 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:

- 617 ページの『PDF ファイルからの DB2 資料の印刷方法』

関連資料:

- 611 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:

- 608 ページの『DB2 インフォメーション・センターの呼び出し』
- 620 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』

- 620 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』
- 621 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 UDB ヘルプの使用法: 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:

- 619 ページの『DB2 ツールからコンテキスト・ヘルプを呼び出す』
- 608 ページの『DB2 インフォメーション・センターの呼び出し』
- 620 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』

- 621 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:

- 608 ページの『DB2 インフォメーション・センターの呼び出し』
- 620 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』
- 620 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:

- 598 ページの『DB2 インフォメーション・センター』
- 「*Troubleshooting Guide*」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、 624 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、 624 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、 624 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:

• 624 ページの『ドット 10 進シンタックス・ダイアグラム』

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:

- 623 ページの『アクセス支援』

関連タスク:

- 『目次』

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 I. 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。

© (お客様の会社名) (西暦年). このコードの一部は、IBM Corp. のサンプル・プログラムから取られています。 © Copyright IBM Corp. _年を入れる_. All rights reserved.

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。

UNIX は、The Open Group の米国およびその他の国における登録商標です。

他の会社名、製品名およびサービス名等はそれぞれ各社の商標です。

付録 J. 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。

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]

[カ行]

[サ行]

[タ行]

[ナ行]

[ハ行]

[マ行]

[ヤ行]

[ラ行]

[ワ行]

[数字]

A

B

C

D

F

G

H

I

K

L

M

N

P

R

S

T

U

V

X

Z

Printed in Japan

SC88-9136-01

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