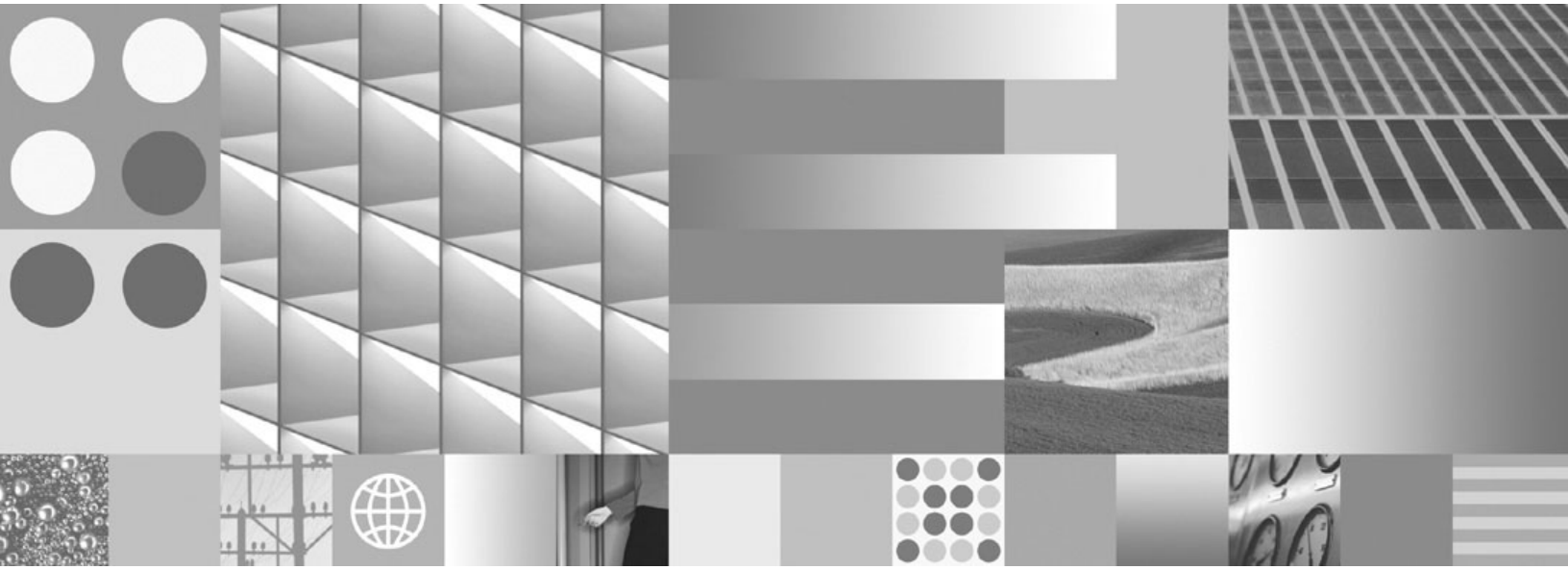
DB2 Version 9.5
for Linux, UNIX, and Windows

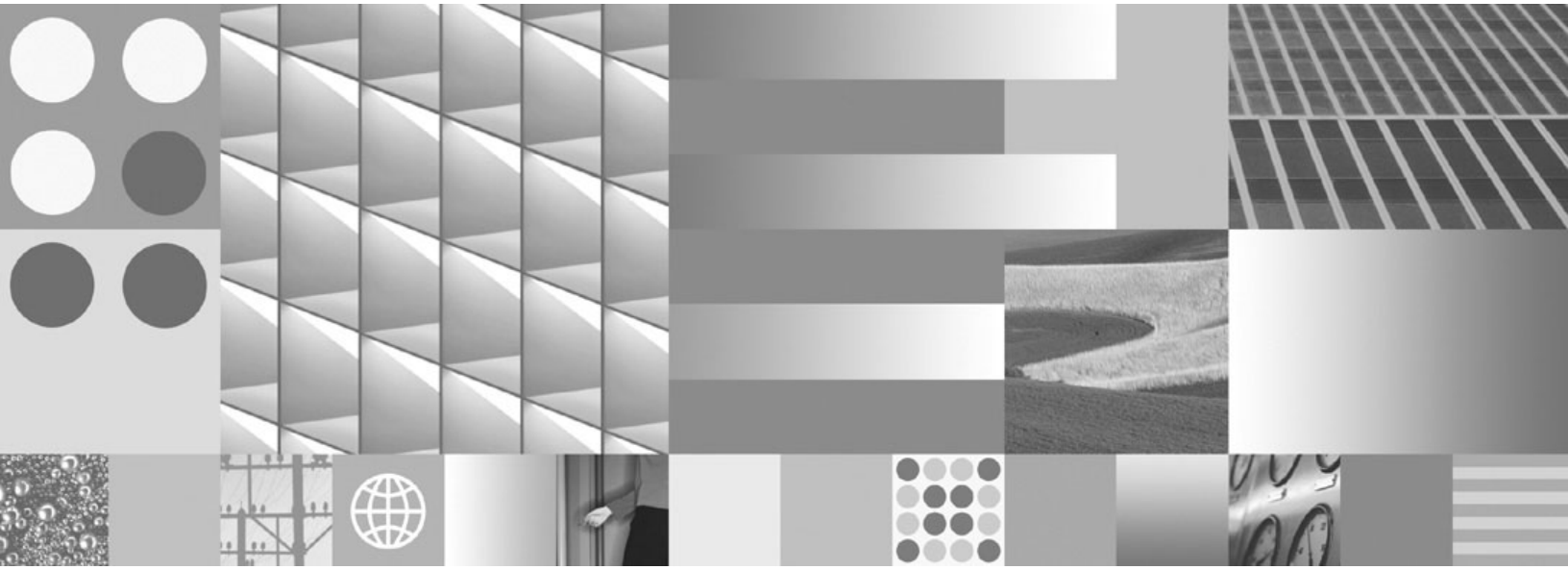


管理 API リファレンス

最終更新: 2009 年 4 月

SC88-4431-02
(英文原典：SC23-5842-02)

DB2 Version 9.5
for Linux, UNIX, and Windows



管理 API リファレンス

最終更新: 2009 年 4 月

SC88-4431-02
(英文原典：SC23-5842-02)

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、655 ページの『付録 D. 特記事項』に記載されている情報をお読みください。

IBM 資料は、オンラインでご注文いただくことも、ご自分の国または地域の IBM 担当員を通してお求めいただくこともできます。

- オンラインで資料を注文するには、www.ibm.com/shop/publications/order にある IBM Publications Center をご利用ください。
- ご自分の国または地域の IBM 担当員を見つけるには、www.ibm.com/planetwide にある IBM Directory of Worldwide Contacts をお調べください。

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。

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： SC23-5872-02
DB2 Version 9.5 for Linux, UNIX, and Windows
Administrative API Reference
Updated April, 2009

発行： 日本アイ・ビー・エム株式会社

担当： トランスレーション・サービス・センター

第1刷 2009.3

目次

本書について

本書には、データベース管理機能を実行するためのアプリケーション・プログラミング・インターフェース (API) の使用に関する情報を記載しています。以下のプログラミング言語で作成されたアプリケーションでのデータベース・マネージャー API 呼び出しの使用について詳しく説明しています。

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

コンパイルされる言語には、ステートメントを処理できる適切なプリコンパイラーが必要です。サポートされるすべての言語についてプリコンパイラーが提供されています。

本書の対象読者

読者はデータベースの管理とアプリケーション・プログラミングを理解していることに加えて、読者に以下の知識があ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、あるいはこれらのすべてのプログラミング言語
- アプリケーション・プログラム設計

本書の構成

本書には、アプリケーション開発で管理 API を使用するために必要な参照情報を記載しています。

本書の各章で説明する主なサブジェクト・エリアは、以下のとおりです。

管理 API およびデータ構造の概要

- 第 1 章『DB2® API』には、管理 API、インクルード・ファイル、およびサンプル・プログラムの一覧表があります。
- 第 2 章『変更された API およびデータ構造』では、サポートされるおよびサポートされない API とデータ構造のうち、これまでに変更されたものについて一覧表にまとめています。
- 第 3 章『API の説明の編成方法』では、API の説明の編成方法を説明し、DB2 API アプリケーションのインクルード・ファイルをリストしています。

API

- 第 4 章『管理 API』では、DB2 管理 API をアルファベット順にリストしています。

- 第 5 章『REXX での DB2 API の呼び出し』では、REXX アプリケーションから DB2 API を呼び出す方法について説明しています。
- 第 6 章『未確定トランザクション管理 API』では、ツール・ライター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機能を実行するための API のセットを説明しています。
- 第 7 章『並行アクセスを伴うスレッド化アプリケーション』では、スレッド化アプリケーションで使用できる DB2 API について説明しています。

プラグイン API

- 第 8 章『データベース管理のカスタマイズのための DB2 データベース・システム・プラグイン』では、読者およびサード・パーティー・ベンダーが特定のデータベース管理機能をカスタマイズするために使用できる、セキュリティー、バックアップ、リストア、ログ・アーカイブ、およびバックアップ・イメージの圧縮/解凍に関するプラグイン API について説明しています。

データ構造

- 第 9 章『API で使用されるデータ構造』では、API で使用されるデータ構造について説明しています。

付録

- 付録 A 『プリコンパイラーのカスタマイズ API』には、他のアプリケーション開発ツールがそれらの製品内に DB2 用のプリコンパイラー・サポートを直接インプリメントするための文書化された API のセットに関する情報へのリンクがあります。
- 付録 B 『DB2 ログ・レコード』では、さまざまな DB2 ログ・レコードの構造について説明しています。

強調表示規則

本書では、以下の強調表示規則を使用します。

太字	名前がシステムによって定義済みのコマンドやキーワードなどの項目を示します。
イタリック	以下のいずれかを示します。 • ユーザーが指定する必要がある名前または値 (変数) • 一般的な強調 • 初出の新規用語 • 別の情報ソースへの参照

モノスペース	以下のいずれかを示します。 • ファイルおよびディレクトリー • コマンド・プロンプトまたはウィンドウに入力するよう指示されている情報 • 特定のデータ値の例 • システムによって表示される内容に該当するテキストの例 • システム・メッセージの例 • プログラミング・コードのサンプル

第 1 章 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、それぞれのインクルード・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 (インクルード・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、インクルード・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
13 ページの表 2.

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

表 1. DB2 API、インクルード・ファイル、およびサンプル・プログラム

API のタイプ	DB2 API	インクルード・ファイル	サンプル・プログラム
データベース制御 API	67 ページの『db2DatabaseQuiesce - データベースの静止』	db2ApiDf	n/a
データベース制御 API	71 ページの『db2DatabaseUnquiesce - データベースの静止解除』	db2ApiDf	n/a
データベース制御 API	69 ページの『db2DatabaseRestart - データベースの再始動』	db2ApiDf	C: dbconn.sqc C++: dbconn.sqC
データベース制御 API	345 ページの『sqlecrea - データベースの作成』	sqlenv	C: dbcreate.c dbrecov.sqc dbsample.sqc C++: dbcreate.C dbrecov.sq COBOL: db_udcs.cbl dbconf.cbl ebcdicdb.cbl
データベース制御 API	343 ページの『sqlecran - データベース・パーティション・サーバー上へのデータベース作成』	sqlenv	n/a
データベース制御 API	359 ページの『sqledrpd - データベースのドロップ』	sqlenv	C: dbcreate.c C++: dbcreate.C COBOL: dbconf.cbl
データベース制御 API	358 ページの『sqledpan - データベース・パーティション・サーバーでのデータベースのドロップ』	sqlenv	n/a
データベース制御 API	378 ページの『sqlemgdb - 前のバージョンの DB2 データベースの現行バージョンへのマイグレーション』	sqlenv	C: dbmigrat.c C++: dbmigrat.C COBOL: migrate.cbl
データベース制御 API	419 ページの『db2XaListIndTrans - 未確定トランザクションのリスト』	db2ApiDf	n/a
データベース制御 API	326 ページの『sqle_activate_db - データベースの活動化』	sqlenv	n/a

表 1. DB2 API、インクルード・ファイル、およびサンプル・プログラム (続き)

API のタイプ	DB2 API	インクルード・ファイル	サンプル・プログラム
データベース制御 API	328 ページの『sqle_deactivate_db - データベースの非活動化』	sqlenv	n/a
データベース制御 API	325 ページの『sqlcspqy - DRDA 未確定トランザクションのリスト』	sqlxa	n/a
データベース制御 API	280 ページの『db2SetWriteForDB - データベースの入出力書き込みの中断または再開』	db2ApiDf	n/a
データベース制御 API	364 ページの『sqlefrce - システムからのユーザーおよび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
インスタンス制御 API	150 ページの『db2InstanceStart - インスタンスの開始』	db2ApiDf	C: instart.c C++: instart.C
インスタンス制御 API	155 ページの『db2InstanceStop - インスタンスの停止』	db2ApiDf	C: instart.c C++: instart.C
インスタンス制御 API	148 ページの『db2InstanceQuiesce - インスタンスの静止』	db2ApiDf	n/a
インスタンス制御 API	158 ページの『db2InstanceUnquiesce - インスタンスの静止解除』	db2ApiDf	n/a
インスタンス制御 API	334 ページの『sqleatin - インスタンスへのアタッチ』	sqlenv	C: inattach.c utilapi.c C++: inattach.C utilapi.C COBOL: dbinst.cbl
インスタンス制御 API	332 ページの『sqleatcp - インスタンスへのアタッチとパスワードの変更』	sqlenv	C: inattach.c C++: inattach.C COBOL: dbinst.cbl
インスタンス制御 API	363 ページの『sqledtin - インスタンスからのデタッチ』	sqlenv	C: inattach.c utilapi.c C++: inattach.C utilapi.C COBOL: dbinst.cbl
インスタンス制御 API	374 ページの『sqlegins - 現行インスタンスの取得』	sqlenv	C: ininfo.c C++: ininfo.C COBOL: dbinst.cbl
インスタンス制御 API	400 ページの『sqluadau - 現行ユーザーの権限の取得』 **注:** この API は、DB2 V9.5 では非推奨です。	sqlutil	n/a
インスタンス制御 API	299 ページの『db2UtilityControl - 実行されているユーティリティーの優先順位の設定』	db2ApiDf	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 API	55 ページの『db2CfgGet - データベース・マネージャー構成パラメーター、あるいはデータベース構成パラメーターの取得』	db2ApiDf	C: dbinfo.c dbrecov.sqc ininfo.c tscreate.sqc C++: dbinfo.C dbrecov.sqC ininfo.C tscreate.sqC
データベース・マネージャーおよびデータベース構成 API	58 ページの『db2CfgSet - データベース・マネージャー構成パラメーター、あるいはデータベース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

表 1. DB2 API、インクルード・ファイル、およびサンプル・プログラム (続き)

API のタイプ	DB2 API	インクルード・ファイル	サンプル・プログラム
データベース・マネージャーおよびデータベース構成 API	40 ページの『db2AutoConfig - 構成アドバイザーへのアクセス』	db2AuCfg	C: dbcfg.sqc C++: dbcfg.sqC
データベース・マネージャーおよびデータベース構成 API	44 ページの『db2AutoConfigFreeMemory - db2AutoConfig API によって割り振られたメモリーの解放』	db2AuCfg	C: dbcfg.sqc C++: dbcfg.sqC
データベース・モニター API	105 ページの『db2GetSnapshotSize - db2GetSnapshot API に必要な出力バッファー・サイズの見積もり』	db2ApiDf	n/a
データベース・モニター API	34 ページの『db2AddSnapshotRequest - スナップショット要求の追加』	db2ApiDf	n/a
データベース・モニター API	205 ページの『db2MonitorSwitches - モニター・スイッチ設定の取得あるいは更新』	db2ApiDf	C: utilsnap.c C++: utilsnap.C
データベース・モニター API	102 ページの『db2GetSnapshot - データベース・マネージャー操作状況のスナップショットの取得』	db2ApiDf	C: utilsnap.c C++: utilsnap.C
データベース・モニター API	238 ページの『db2ResetMonitor - データベース・システム・モニターのデータのリセット』	db2ApiDf	n/a
データベース・モニター API	62 ページの『db2ConvMonStream - バージョン 6 以前の形式へのモニター・ストリームの変換』	db2ApiDf	n/a
データベース・モニター API	140 ページの『db2Inspect - データベースの構造上の整合性の検査』	db2ApiDf	n/a
データベース・ヘルス・モニター API	31 ページの『db2AddContact - 通知メッセージを送信できる連絡先の追加』	db2ApiDf	n/a
データベース・ヘルス・モニター API	32 ページの『db2AddContactGroup - 通知メッセージを送信できる連絡先グループの追加』	db2ApiDf	n/a
データベース・ヘルス・モニター API	78 ページの『db2DropContact - 通知メッセージを送信できる連絡先リストからの連絡先の削除』	db2ApiDf	n/a
データベース・ヘルス・モニター API	79 ページの『db2DropContactGroup - 通知メッセージを送信できる連絡先リストからの連絡先グループの削除』	db2ApiDf	n/a
データベース・ヘルス・モニター API	88 ページの『db2GetAlertCfg - ヘルス・インディケーターのアラート構成設定の取得』	db2ApiDf	n/a
データベース・ヘルス・モニター API	92 ページの『db2GetAlertCfgFree - db2GetAlertCfg API によって割り振られたメモリーの解放』	db2ApiDf	n/a

表 1. DB2 API、インクルード・ファイル、およびサンプル・プログラム (続き)

API のタイプ	DB2 API	インクルード・ファイル	サンプル・プログラム
データベース・ヘルス・モニター API	93 ページの『db2GetContactGroup - 通知メッセージを送信できる単一の連絡先グループ中の連絡先のリストの取得』	db2ApiDf	n/a
データベース・ヘルス・モニター API	94 ページの『db2GetContactGroups - 通知メッセージを送信できる連絡先グループのリストの取得』	db2ApiDf	n/a
データベース・ヘルス・モニター API	96 ページの『db2GetContacts - 通知メッセージを送信できる連絡先リストの取得』	db2ApiDf	n/a
データベース・ヘルス・モニター API	97 ページの『db2GetHealthNotificationList - ヘルス・アラート通知を送信できる連絡先リストの取得』	db2ApiDf	n/a
データベース・ヘルス・モニター API	236 ページの『db2ResetAlertCfg - ヘルス・インディケーターのアラート構成のリセット』	db2ApiDf	n/a
データベース・ヘルス・モニター API	287 ページの『db2UpdateAlertCfg - ヘルス・インディケーターのアラート構成設定の更新』	db2ApiDf	n/a
データベース・ヘルス・モニター API	294 ページの『db2UpdateContact - 連絡先の属性の更新』	db2ApiDf	n/a
データベース・ヘルス・モニター API	295 ページの『db2UpdateContactGroup - 連絡先グループの属性の更新』	db2ApiDf	n/a
データベース・ヘルス・モニター API	297 ページの『db2UpdateHealthNotificationList - ヘルス・アラート通知を送信できる連絡先リストの更新』	db2ApiDf	n/a
データベース・ヘルス・モニター API	102 ページの『db2GetSnapshot - データベース・マネージャー操作状況のスナップショットの取得』	db2ApiDf	C: utilsnap.c C++: utilsnap.C
データベース・ヘルス・モニター API	105 ページの『db2GetSnapshotSize - db2GetSnapshot API に必要な出力バッファー・サイズの見積もり』	db2ApiDf	n/a
データベース・ヘルス・モニター API	99 ページの『db2GetRecommendations - アラート状態のヘルス・インディケーターを解決するための推奨の入手』	db2ApiDf	n/a
データベース・ヘルス・モニター API	101 ページの『db2GetRecommendationsFree - db2GetRecommendations API によって割り振られたメモリーの解放』	db2ApiDf	n/a
データ移動 API	80 ページの『db2Export - データベースからのデータのエクスポート』	sqlutil	C: tbmove.sqc C++: tbmove.sqC COBOL: expsamp.sqb impexp.sqb tload.sqb

表 1. DB2 API、インクルード・ファイル、およびサンプル・プログラム (続き)

API のタイプ	DB2 API	インクルード・ファイル	サンプル・プログラム
データ移動 API	125 ページの『db2Import - 表、階層、ニックネーム、ビューへのデータのインポート』	db2ApiDf	C: dtformat.sqc tbmove.sqc C++: tbmove.sqC COBOL: expsamp.sqb impexp.sqb
データ移動 API	174 ページの『db2Load - 表へのデータのロード』	db2ApiDf	C: dtformat.sqc tbload.sqc tbmove.sqc C++: tbmove.sqC
データ移動 API	197 ページの『db2LoadQuery - ロード操作の状況の取得』	db2ApiDf	C: tbmove.sqc C++: tbmove.sqC COBOL: loadqry.sqb
リカバリー API	44 ページの『db2Backup - データベースまたは表スペースのバックアップ』	db2ApiDf	C: dbrecov.sqc C++: dbrecov.sqC
リカバリー API	240 ページの『db2Restore - データベースまたは表スペースのリストア』	db2ApiDf	C: dbrecov.sqc C++: dbrecov.sqC
リカバリー API	222 ページの『db2Recover - データベースのリストアおよびロールフォワード』	db2ApiDf	n/a
リカバリー API	256 ページの『db2Rollforward - データベースのロールフォワード』	db2ApiDf	C: dbrecov.sqc C++: dbrecov.sqC
リカバリー API	118 ページの『db2HistoryOpenScan - 履歴ファイルのスキャンの開始』	db2ApiDf	C: dbrecov.sqc C++: dbrecov.sqC
リカバリー API	115 ページの『db2HistoryGetEntry - 履歴ファイルの次の項目の取得』	db2ApiDf	C: dbrecov.sqc C++: dbrecov.sqC
リカバリー API	114 ページの『db2HistoryCloseScan - 履歴ファイルのスキャンの終了』	db2ApiDf	C: dbrecov.sqc C++: dbrecov.sqC
リカバリー API	208 ページの『db2Prune - 履歴ファイル項目の削除、あるいはアクティブ・ログ・パスからのログ・ファイルの削除』	db2ApiDf	C: dbrecov.sqc C++: dbrecov.sqC
リカバリー API	122 ページの『db2HistoryUpdate - 履歴ファイル項目の更新』	db2ApiDf	C: dbrecov.sqc C++: dbrecov.sqC
リカバリー API	37 ページの『db2ArchiveLog - アクティブ・ログ・ファイルのアーカイブ』	db2ApiDf	n/a
高可用性災害時リカバリー (HADR) API	109 ページの『db2HADRStart - 高可用性災害時リカバリー (HADR) 操作の開始』	db2ApiDf	n/a
高可用性災害時リカバリー (HADR) API	111 ページの『db2HADRStop - 高可用性災害時リカバリー (HADR) 操作の停止』	db2ApiDf	n/a
高可用性災害時リカバリー (HADR) API	112 ページの『db2HADRTakeover - データベースへの高可用性災害時リカバリー (HADR) 1 次データベースとしてのテークオーバーの指示』	db2ApiDf	n/a
データベース・ディレクトリーおよび DCS ディレクトリー管理 API	337 ページの『sqlecadb - システム・データベース・ディレクトリーへのデータベースのカタログ』	sqlenv	C: ininfo.c C++: ininfo.C COBOL: dbcat.cbl

表 1. DB2 API、インクルード・ファイル、およびサンプル・プログラム (続き)

API のタイプ	DB2 API	インクルード・ファイル	サンプル・プログラム
データベース・ディレクトリーおよび DCS ディレクトリー管理 API	393 ページの『sqleuncd - システム・データベース・ディレクトリーからのデータベースのアンカタログ』	sqlenv	C: ininfo.c C++: ininfo.C COBOL: dbcat.cbl
データベース・ディレクトリーおよび DCS ディレクトリー管理 API	366 ページの『sqlegdad - データベース接続サービス (DCS) ディレクトリーへのデータベースのカタログ』	sqlenv	C: ininfo.c C++: ininfo.C COBOL: dcscat.cbl
データベース・ディレクトリーおよび DCS ディレクトリー管理 API	369 ページの『sqlegdel - データベース接続サービス (DCS) ディレクトリーからのデータベースのアンカタログ』	sqlenv	C: ininfo.c C++: ininfo.C COBOL: dcscat.cbl
データベース・ディレクトリーおよび DCS ディレクトリー管理 API	356 ページの『sqledcgd - 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・コメントの変更』	sqlenv	C: ininfo.c C++: ininfo.C COBOL: dbcmt.cbl
データベース・ディレクトリーおよび DCS ディレクトリー管理 API	77 ページの『db2DbDirOpenScan - 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・ディレクトリーのスキャンの開始』	db2ApiDf	C: ininfo.c C++: ininfo.C COBOL: dbcat.cbl dbcmt.cbl
データベース・ディレクトリーおよび DCS ディレクトリー管理 API	73 ページの『db2DbDirGetNextEntry - 次のシステム・データベース・ディレクトリー、あるいはローカル・データベース・ディレクトリー項目の取得』	db2ApiDf	C: ininfo.c C++: ininfo.C COBOL: dbcat.cbl dbcmt.cbl
データベース・ディレクトリーおよび DCS ディレクトリー管理 API	72 ページの『db2DbDirCloseScan - 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・ディレクトリーのスキャンの終了』	db2ApiDf	C: ininfo.c C++: ininfo.C COBOL: dbcat.cbl dbcmt.cbl
データベース・ディレクトリーおよび DCS ディレクトリー管理 API	373 ページの『sqlegdsc - データベース接続サービス (DCS) ディレクトリーのスキャンの開始』	sqlenv	C: ininfo.c C++: ininfo.C COBOL: dcscat.cbl
データベース・ディレクトリーおよび DCS ディレクトリー管理 API	372 ページの『sqlegdgt - データベース接続サービス (DCS) ディレクトリーの項目の取得』	sqlenv	C: ininfo.c C++: ininfo.C COBOL: dcscat.cbl
データベース・ディレクトリーおよび DCS ディレクトリー管理 API	368 ページの『sqlegdcl - データベース接続サービス (DCS) ディレクトリーのスキャンの終了』	sqlenv	C: ininfo.c C++: ininfo.C COBOL: dcscat.cbl
データベース・ディレクトリーおよび DCS ディレクトリー管理 API	371 ページの『sqlegdge - データベース接続サービス (DCS) ディレクトリーの特定項目の取得』	sqlenv	C: ininfo.c C++: ininfo.C COBOL: dcscat.cbl

表 1. DB2 API、インクルード・ファイル、およびサンプル・プログラム (続き)

API のタイプ	DB2 API	インクルード・ファイル	サンプル・プログラム
データベース・ディレクトリーおよび DCS ディレクトリー管理 API	292 ページの『db2UpdateAlternateServerForDB - 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の代替サーバーの更新』	db2ApiDf	n/a
クライアント/サーバー管理 API	383 ページの『sqleqryc - クライアント接続設定の照会』	sqlenv	C: cli_info.c C++: cli_info.C COBOL: client.cbl
クライアント/サーバー管理 API	385 ページの『sqleqryi - クライアント情報の照会』	sqlenv	C: cli_info.c C++: cli_info.C
クライアント/サーバー管理 API	389 ページの『sqlesetc - クライアント接続設定の指定』	sqlenv	C: cli_info.c dbcfg.sqc dbmcon.sqc C++: cli_info.C dbcfg.sqC dbmcon.sqC COBOL: client.cbl
クライアント/サーバー管理 API	391 ページの『sqleseti - クライアント情報の設定』	sqlenv	C: cli_info.c C++: cli_info.C
クライアント/サーバー管理 API	386 ページの『sqlesact - 会計情報ストリングの設定』	sqlenv	COBOL: setact.cbl
クライアント/サーバー管理 API	65 ページの『db2DatabasePing - ネットワーク応答時間のテストのためのデータベースの ping』	db2ApiDf	n/a
クライアント/サーバー管理 API	377 ページの『sqleisig - シグナル・ハンドラーのインストール』	sqlenv	COBOL: dbcmt.cbl
クライアント/サーバー管理 API	375 ページの『sqleintr - アプリケーション要求への割り込み』	sqlenv	n/a
Lightweight Directory Access Protocol (LDAP) ディレクトリー管理 API	164 ページの『db2LdapRegister - DB2 サーバーの LDAP サーバーへの登録』	db2ApiDf	n/a
Lightweight Directory Access Protocol (LDAP) ディレクトリー管理 API	169 ページの『db2LdapUpdate - LDAP サーバー上の DB2 サーバーの属性の更新』	db2ApiDf	n/a
Lightweight Directory Access Protocol (LDAP) ディレクトリー管理 API	163 ページの『db2LdapDeregister - LDAP サーバーからの DB2 サーバーおよびカタログされたデータベースの登録解除』	db2ApiDf	n/a
Lightweight Directory Access Protocol (LDAP) ディレクトリー管理 API	162 ページの『db2LdapCatalogNode - LDAP サーバーにあるノード名の別名の指定』	db2ApiDf	n/a

表 1. DB2 API、インクルード・ファイル、およびサンプル・プログラム (続き)

API のタイプ	DB2 API	インクルード・ファイル	サンプル・プログラム
Lightweight Directory Access Protocol (LDAP) ディレクトリー管理 API	168 ページの『db2LdapUncatalogNode - LDAP サーバーからのノード名に対応する別名の削除』	db2ApiDf	n/a
Lightweight Directory Access Protocol (LDAP) ディレクトリー管理 API	159 ページの『db2LdapCatalogDatabase - LDAP サーバーへのデータベースの登録』	db2ApiDf	n/a
Lightweight Directory Access Protocol (LDAP) ディレクトリー管理 API	167 ページの『db2LdapUncatalogDatabase - LDAP サーバーからのデータベース登録の解除』	db2ApiDf	n/a
Lightweight Directory Access Protocol (LDAP) ディレクトリー管理 API	172 ページの『db2LdapUpdateAlternateServerForDB - LDAP サーバーでのデータベースに対応する代替サーバーの更新』	db2ApiDf	n/a
アプリケーション・プログラミングおよび準備 API	303 ページの『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
アプリケーション・プログラミングおよび準備 API	398 ページの『sqlogstt - SQLSTATE メッセージの入手』	sql	C: utilapi.c C++: utilapi.C COBOL: checkerr.cbl
アプリケーション・プログラミングおよび準備 API	377 ページの『sqleisig - シグナル・ハンドラーのインストール』	sqlenv	COBOL: dbcmt.cbl
アプリケーション・プログラミングおよび準備 API	375 ページの『sqleintr - アプリケーション要求への割り込み』	sqlenv	n/a
アプリケーション・プログラミングおよび準備 API	305 ページの『sqlaprep - アプリケーション・プログラムのプリコンパイル』	sql	C: dbpkg.sqc C++: dbpkg.sqC
アプリケーション・プログラミングおよび準備 API	300 ページの『sqlabndx - アプリケーション・プログラムのバインドによるパッケージの作成』	sql	C: dbpkg.sqc dbsample.sqc C++: dbpkg.sqC

表 1. DB2 API、インクルード・ファイル、およびサンプル・プログラム (続き)

API のタイプ	DB2 API	インクルード・ファイル	サンプル・プログラム
アプリケーション・プログラミングおよび準備 API	307 ページの『sqlarbnd - パッケージの再バインド』	sql	C: dbpkg.sqc C++: dbpkg.sqC COBOL: rebind.sqb
COBOL、FORTRAN、REXX アプリケーション固有 API	396 ページの『sqlgaddr - 変数のアドレスの取得』	sqlutil	n/a
COBOL、FORTRAN、REXX アプリケーション固有 API	397 ページの『sqlgdref - アドレスの間接参照』	sqlutil	n/a
COBOL、FORTRAN、REXX アプリケーション固有 API	397 ページの『sqlgmcpy - あるメモリー領域から別のメモリー領域へデータのコピー』	sqlutil	n/a
表スペースおよび表管理 API	324 ページの『sqlbtcq - すべての表スペース・コンテナーの照会データの取得』	sqlutil	C: dbrecov.sqc tsinfo.sqc C++: dbrecov.sqC tsinfo.sqC COBOL: tabscont.sqb tspace.sqb
表スペースおよび表管理 API	317 ページの『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
表スペースおよび表管理 API	311 ページの『sqlbftcq - 表スペース・コンテナー中の行の照会データの取得』	sqlutil	C: tsinfo.sqc C++: tsinfo.sqC COBOL: tabscont.sqb tspace.sqb
表スペースおよび表管理 API	310 ページの『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
表スペースおよび表管理 API	322 ページの『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
表スペースおよび表管理 API	315 ページの『sqlbmtsq - すべての表スペースの照会データの取得』	sqlutil	C: dbrecov.sqc tsinfo.sqc C++: dbrecov.sqC tsinfo.sqC COBOL: tabspace.sqb tspace.sqb
表スペースおよび表管理 API	320 ページの『sqlbstpq - 単一の表スペースに関する情報の取得』	sqlutil	C: tsinfo.sqc C++: tsinfo.sqC COBOL: tabspace.sqb tspace.sqb
表スペースおよび表管理 API	318 ページの『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
表スペースおよび表管理 API	313 ページの『sqlbftpq - 表スペース中の行の照会データのフェッチ』	sqlutil	C: tsinfo.sqc C++: tsinfo.sqC COBOL: tabspace.sqb tspace.sqb
表スペースおよび表管理 API	311 ページの『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
表スペースおよび表管理 API	314 ページの『sqlbgtss - 表スペースの使用率に関する統計の取得』	sqlutil	C: tsinfo.sqc C++: tsinfo.sqC COBOL: tabspace.sqb tspace.sqb

表 1. DB2 API、インクルード・ファイル、およびサンプル・プログラム (続き)

API のタイプ	DB2 API	インクルード・ファイル	サンプル・プログラム
表スペースおよび表管理 API	410 ページの『sqluvqdp - 表の表スペースの静止』	sqlutil	C: tbmove.sqc C++: tbmove.sqC COBOL: tload.sqb
表スペースおよび表管理 API	268 ページの『db2Runstats - 表および索引の統計情報の更新』	db2ApiDf	C: tbreorg.sqc C++: tbreorg.sqC COBOL: dbstat.sqb
表スペースおよび表管理 API	228 ページの『db2Reorg - 索引または表の再編成』	db2ApiDf	C: tbreorg.sqc C++: tbreorg.sqC COBOL: dbstat.sqb
表スペースおよび表管理 API	363 ページの『sqlefmem - sqlbtcq および sqlbmtsq API によって割り振られたメモリーの解放』	sqlenv	C: dbrecov.sqc tsinfo.sqc C++: dbrecov.sqC tsinfo.sqC COBOL: tabscont.sqb tabspace.sqb tspace.sqb
ノード・ディレクトリー管理 API	353 ページの『sqlectnd - ノード・ディレクトリーへの項目のカタログ』	sqlenv	C: ininfo.c C++: ininfo.C COBOL: nodecat.cbl
ノード・ディレクトリー管理 API	395 ページの『sqleuncn - ノード・ディレクトリーからの項目のアンカタログ』	sqlenv	C: ininfo.c C++: ininfo.C COBOL: nodecat.cbl
ノード・ディレクトリー管理 API	382 ページの『sqlenops - ノード・ディレクトリー・スキャンの開始』	sqlenv	C: ininfo.c C++: ininfo.C COBOL: nodecat.cbl
ノード・ディレクトリー管理 API	380 ページの『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
ノード・ディレクトリー管理 API	379 ページの『sqlencls - ノード・ディレクトリー・スキャンの終了』	sqlenv	C: ininfo.c C++: ininfo.C COBOL: nodecat.cbl
ノード・ディレクトリー管理 API	292 ページの『db2UpdateAlternateServerForDB - 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の代替サーバーの更新』	db2ApiDf	n/a
サテライト同期 API	108 ページの『db2GetSyncSession - サテライト同期化セッション ID の取得』	db2ApiDf	n/a
サテライト同期 API	210 ページの『db2QuerySatelliteProgress - サテライト同期セッションの状況の取得』	db2ApiDf	n/a
サテライト同期 API	279 ページの『db2SetSyncSession - サテライト同期セッションの設定』	db2ApiDf	n/a
サテライト同期 API	285 ページの『db2SyncSatellite - サテライト同期化の開始』	db2ApiDf	n/a
サテライト同期 API	285 ページの『db2SyncSatelliteStop - サテライト同期化の一時停止』	db2ApiDf	n/a
サテライト同期 API	286 ページの『db2SyncSatelliteTest - サテライトが同期化可能かのテスト』	db2ApiDf	n/a
ログ・ファイル読み取り API	212 ページの『db2ReadLog - ログ・レコードの抽出』	db2ApiDf	C: dbrecov.sqc C++: dbrecov.sqC
ログ・ファイル読み取り API	216 ページの『db2ReadLogNoConn - データベース接続なしのデータベース・ログの読み取り』	db2ApiDf	n/a

表 1. DB2 API、インクルード・ファイル、およびサンプル・プログラム (続き)

API のタイプ	DB2 API	インクルード・ファイル	サンプル・プログラム
ログ・ファイル読み取り API	219 ページの『db2ReadLogNoConnInit - データベース接続なしのデータベース・ログ読み取りの初期設定』	db2ApiDf	n/a
ログ・ファイル読み取り API	221 ページの『db2ReadLogNoConnTerm - データベース接続なしのデータベース・ログの読み取り終了』	db2ApiDf	n/a
未確定トランザクション管理 API	419 ページの『db2XaListIndTrans - 未確定トランザクションのリスト』	db2ApiDf	n/a
未確定トランザクション管理 API	424 ページの『sqlxhfrg - トランザクション状況の forget』	sqlxa	n/a
未確定トランザクション管理 API	425 ページの『sqlxphcm - 未確定トランザクションのコミット』	sqlxa	n/a
未確定トランザクション管理 API	426 ページの『sqlxphrl - 未確定トランザクションのロールバック』	sqlxa	n/a
未確定トランザクション管理 API	325 ページの『sqlcspqy - DRDA 未確定トランザクションのリスト』	sqlxa	n/a
データベースへの並行アクセス取得のための API	427 ページの『sqleAttachToCtx - コンテキストへのアタッチ』	sql	C: dbthrds.sqc C++: dbthrds.sqC
データベースへの並行アクセス取得のための API	427 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』	sql	C: dbthrds.sqc C++: dbthrds.sqC
データベースへの並行アクセス取得のための API	428 ページの『sqleDetachFromCtx - コンテキストからのデタッチ』	sql	C: dbthrds.sqc C++: dbthrds.sqC
データベースへの並行アクセス取得のための API	429 ページの『sqleEndCtx - 特定のアプリケーション・コンテキストに関連付けられているメモリーのデタッチと解放』	sql	n/a
データベースへの並行アクセス取得のための API	430 ページの『sqleGetCurrentCtx - 現行コンテキストの入手』	sql	n/a
データベースへの並行アクセス取得のための API	431 ページの『sqleInterruptCtx - コンテキストへの割り込み』	sql	n/a
データベースへの並行アクセス取得のための API	432 ページの『sqleSetTypeCtx - アプリケーション・コンテキスト・タイプの設定』	sql	C: dbthrds.sqc C++: dbthrds.sqC
データベース・パーティション管理 API	330 ページの『sqleaddn - パーティション・データベース環境へのデータベース・パーティション・サーバーの追加』	sqlenv	n/a

表 1. DB2 API、インクルード・ファイル、およびサンプル・プログラム (続き)

API のタイプ	DB2 API	インクルード・ファイル	サンプル・プログラム
データベース・パーティション管理 API	361 ページの『sqledrpn - データベース・パーティション・サーバーがドロップ可能かどうかの検査』	sqlenv	n/a
データベース・パーティション管理 API	343 ページの『sqlecran - データベース・パーティション・サーバー上へのデータベース作成』	sqlenv	n/a
データベース・パーティション管理 API	358 ページの『sqledpan - データベース・パーティション・サーバーでのデータベースのドロップ』	sqlenv	n/a
データベース・パーティション管理 API	387 ページの『sqlesdeg - SQL ステートメントの最大実行時パーティション内並列処理レベル (つまり並列処理の程度) の設定』	sqlenv	C: ininfo.c C++: ininfo.C
データベース・パーティション管理 API	408 ページの『sqlugtpi - 表の分散情報の取得』	sqlutil	n/a
データベース・パーティション管理 API	405 ページの『sqlugrpn - 特定の行についてのデータベース・パーティション・サーバー番号の取得』	sqlutil	n/a
その他の API	36 ページの『db2AdminMsgWrite - 管理およびレプリケーション機能のためのログ・メッセージの書き込み』	db2ApiDf	n/a
その他の API	418 ページの『db2XaGetInfo - リソース・マネージャー情報の入手』	sqlxa	n/a

注: インクルード・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++ インクルード・ファイルのファイル拡張子は `.h` です。 COBOL インクルード・ファイルのファイル拡張子は `.cbl` です。インクルード・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	• sqlesetc API - クライアント接続設定の指定 • sqleseti API - クライアント情報の設定 • sqleqryc API - クライアント接続設定の照会 • sqleqryi API - クライアント情報の照会
dbcfg.sqc, dbcfg.sqC	• db2AutoConfig API - 構成アドバイザーへのアクセス • db2AutoConfigFreeMemory API - db2AutoConfig API によって割り振られたメモリーの解放 • sqlesetc API - クライアント接続設定の指定 • sqlaintp API - エラー・メッセージの入手
dbconn.sqc, dbconn.sqC	• db2DatabaseRestart API - データベースの再始動 • sqlefrce API - システムからのユーザーおよびアプリケーションの強制終了
dbcreate.c, dbcreate.C	• sqlecrea API - データベースの作成 • sqledrpd API - データベースのドロップ
dbinfo.c, dbinfo.C	• db2CfgGet API - データベース・マネージャー構成パラメーター、あるいはデータベース構成パラメーターの取得 • db2CfgSet API - データベース・マネージャー構成パラメーター、あるいはデータベース構成パラメーターの設定
dbmcon.sqc, dbmcon.sqC	• sqlesetc API - クライアント接続設定の指定
dbmigrat.c, dbmigrat.C	• sqlemgdb API - 前のバージョンの DB2 データベースの現行バージョンへのマイグレーション
dbpkg.sqc, dbpkg.sqC	• sqlaprep API - アプリケーション・プログラムのプリコンパイル • sqlabndx API - アプリケーション・プログラムのバインドによるパッケージの作成 • sqlarbnd API - パッケージの再バインド

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 API - 履歴ファイルのスキャンの終了 • db2HistoryGetEntry API - 履歴ファイルの次の項目の取得 • db2HistoryOpenScan API - 履歴ファイルのスキャンの開始 • db2HistoryUpdate API - 履歴ファイルの項目の更新 • db2Prune API - 履歴ファイル項目の削除、あるいはアクティブ・ログ・パスからのログ・ファイルの削除 • db2CfgGet API - データベース・マネージャー構成パラメーター、あるいはデータベース構成パラメーターの取得 • db2CfgSet API - データベース・マネージャー構成パラメーター、あるいはデータベース構成パラメーターの設定 • sqlbmtsq API - すべての表スペースの照会データの取得 • sqlbstsc API - 表スペース・コンテナーの設定 • sqlbtcq API - すべての表スペース・コンテナーの照会データの取得 • sqlecrea API - データベースの作成 • sqledrpd API - データベースのドロップ • sqlefmem API - sqlbtcq および sqlbmtsq API によって割り振られたメモリーの解放 • db2Backup API - データベースまたは表スペースのバックアップ • db2Restore API - データベースまたは表スペースのリストア • db2ReadLog API - ログの非同期読み取り • db2ReadLogNoConn API - データベース接続なしのログの読み取り • db2Rollforward API - データベースのロールフォワード
dbsample.sqc	• db2DatabaseRestart API - データベースの再始動 • sqlecrea API - データベースの作成 • sqlefrce API - システムからのユーザーおよびアプリケーションの強制終了 • sqlabndx API - アプリケーション・プログラムのバインドによるパッケージの作成
dbthrds.sqc, dbthrds.sqC	• sqleAttachToCtx API - コンテキストへのアタッチ • sqleBeginCtx API - 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 API - コンテキストからのデタッチ • sqleSetTypeCtx API - アプリケーション・コンテキスト・タイプの設定
dtformat.sqc	• db2Load API - 表へのデータのロード • db2Import API - 表、階層、ニックネーム、ビューへのデータのインポート
inattach.c, inattach.C	• sqleatcp API - インスタンスへのアタッチとパスワードの変更 • sqleatin API - インスタンスへのアタッチ • sqledtin API - インスタンスからのデタッチ

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 API - データベース・マネージャー構成パラメーター、あるいはデータベース構成パラメーターの取得 • db2CfgSet API - データベース・マネージャー構成パラメーター、あるいはデータベース構成パラメーターの設定 • sqlegins API - 現行インスタンスの取得 • sqlectnd API - ノード・ディレクトリーへの項目のカタログ • sqlenops API - ノード・ディレクトリー・スキャンの開始 • sqlengne API - ノード・ディレクトリー次項目の入手 • sqlencls API - ノード・ディレクトリー・スキャンの終了 • sqleuncn API - ノード・ディレクトリーからの項目のアンカタログ • sqlecadb API - システム・データベース・ディレクトリーへのデータベースのカタログ • db2DbDirOpenScan API - 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・ディレクトリーのスキャンの開始 • db2DbDirGetNextEntry API - 次のシステム・データベース・ディレクトリー、あるいはローカル・データベース・ディレクトリー項目の取得 • sqledcgd API - 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・コメントの変更 • db2DbDirCloseScan API - 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・ディレクトリーのスキャンの終了 • sqleuncd API - システム・データベース・ディレクトリーからのデータベースのアンカタログ • sqlegdad API - データベース接続サービス (DCS) ディレクトリーへのデータベースのカタログ • sqlegdsc API - データベース接続サービス (DCS) ディレクトリーのスキャンの開始 • sqlegdge API - データベース接続サービス (DCS) ディレクトリーの特定項目の取得 • sqlegdgt API - データベース接続サービス (DCS) ディレクトリーの項目の取得 • sqlegdcl API - データベース接続サービス (DCS) ディレクトリーのスキャンの終了 • sqlegdel API - データベース接続サービス (DCS) ディレクトリーからのデータベースのアンカタログ • sqlesdeg API - SQL ステートメントの最大実行時パーティション内並列処理レベル (つまり並列処理の程度) の設定

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
instart.c, instart.C	• sqlefrce API - システムからのユーザーおよびアプリケーションの強制終了 • db2InstanceStart API - インスタンスの開始 • db2InstanceStop API - インスタンスの停止
tbmove.sqc, tbmove.sqC	• db2Export API - データベースからのデータのエクスポート • db2Import API - 表、階層、ニックネーム、ビューへのデータのインポート • sqluvqdp API - 表の表スペースの静止 • db2Load API - 表へのデータのロード • db2LoadQuery API - ロード操作の状況の取得
tbreorg.sqc, tbreorg.sqC	• db2Reorg API - 索引または表の再編成 • db2Runstats API - 表および関連する索引の特性についての統計の更新
tscreate.sqc, tscreate.sqC	• db2CfgGet API - データベース・マネージャー構成パラメーター、あるいはデータベース構成パラメーターの取得
tsinfo.sqc, tsinfo.sqC	• sqlbstpq API - 単一の表スペースに関する情報の取得 • sqlbgtss API - 表スペースの使用率に関する統計の取得 • sqlbmtsq API - すべての表スペースの照会データの取得 • sqlefmem API - sqlbtcq および sqlbmtsq API によって割り振られたメモリーの解放 • sqlbotsq API - 表スペース照会のオープン • sqlbftpq API - 表スペース中の行の照会データのフェッチ • sqlbctsq API - 表スペース照会のクローズ • sqlbtcq API - すべての表スペース・コンテナーの照会データの取得 • sqlbotcq API - 表スペース・コンテナー照会のオープン • sqlbftcq API - 表スペース・コンテナー中の行の照会データの取得 • sqlbctcq API - 表スペース・コンテナー照会のクローズ
utilapi.c, utilapi.C	• sqlaintp API - エラー・メッセージの入手 • sqlogstt API - SQLSTATE メッセージの入手 • sqleatin API - インスタンスへのアタッチ • sqledtin API - インスタンスからのデタッチ
utilsnap.c, utilsnap.C	• db2GetSnapshot API - データベース・マネージャー操作状況のスナップショットの取得 • db2MonitorSwitches API - モニター・スイッチ設定の取得あるいは更新

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 API - エラー・メッセージの入手 • sqlogstt API - SQLSTATE メッセージの入手
client.cbl	• sqleqryc API - クライアント接続設定の照会 • sqlesetc API - クライアント接続設定の指定
db_udcs.cbl	• sqleatin API - インスタンスへのアタッチ • sqlecrea API - データベースの作成 • sqledrpd API - データベースのドロップ
dbcat.cbl	• sqlecadb API - システム・データベース・ディレクトリーへのデータベースのカタログ • db2DbDirCloseScan API - 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・ディレクトリーのスキャンの終了 • db2DbDirGetNextEntry API - 次のシステム・データベース・ディレクトリー、あるいはローカル・データベース・ディレクトリー項目の取得 • db2DbDirOpenScan API - 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・ディレクトリーのスキャンの開始 • sqleuncd API - システム・データベース・ディレクトリーからのデータベースのアンカタログ
dbcmt.cbl	• sqledcgd API - 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・コメントの変更 • db2DbDirCloseScan API - 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・ディレクトリーのスキャンの終了 • db2DbDirGetNextEntry API - 次のシステム・データベース・ディレクトリー、あるいはローカル・データベース・ディレクトリー項目の取得 • db2DbDirOpenScan API - 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・ディレクトリーのスキャンの開始 • sqleisig API - シグナル・ハンドラーのインストール
dbinst.cbl	• sqleatcp API - インスタンスへのアタッチとパスワードの変更 • sqleatin API - インスタンスへのアタッチ • sqledtin API - インスタンスからのデタッチ • sqlegins API - 現行インスタンスの取得
dbstat.sqb	• db2Reorg API - 索引または表の再編成 • db2Runstats API - 表および関連する索引の特性についての統計の更新

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
dcscat.cbl	• sqlegdad API - データベース接続サービス (DCS) ディレクトリーへのデータベースのカタログ • sqlegdcl API - データベース接続サービス (DCS) ディレクトリーのスキャンの終了 • sqlegdel API - データベース接続サービス (DCS) ディレクトリーからのデータベースのアンカタログ • sqlegdge API - データベース接続サービス (DCS) ディレクトリーの特定項目の取得 • sqlegdgt API - データベース接続サービス (DCS) ディレクトリーの項目の取得 • sqlegdsc API - データベース接続サービス (DCS) ディレクトリーのスキャンの開始
ebcdicdb.cbl	• sqleatin API - インスタンスへのアタッチ • sqlecrea API - データベースの作成 • sqledrpd API - データベースのドロップ
expsamp.sqb	• db2Export API - データベースからのデータのエクスポート • db2Import API - 表、階層、ニックネーム、ビューへのデータのインポート
impexp.sqb	• db2Export API - データベースからのデータのエクスポート • db2Import API - 表、階層、ニックネーム、ビューへのデータのインポート
loadqry.sqb	• db2LoadQuery API - ロード操作の状況の取得
migrate.cbl	• sqlemgdb API - 前のバージョンの DB2 データベースの現行バージョンへのマイグレーション
nodecat.cbl	• sqlectnd API - ノード・ディレクトリーへの項目のカタログ • sqlencls API - ノード・ディレクトリー・スキャンの終了 • sqlengne API - ノード・ディレクトリー次項目の入手 • sqlenops API - ノード・ディレクトリー・スキャンの開始 • sqleuncn API - ノード・ディレクトリーからの項目のアンカタログ
rebind.sqb	• sqlarbnd API - パッケージの再バインド
tabscont.sqb	• sqlbctcq API - 表スペース・コンテナー照会のクローズ • sqlbftcq API - 表スペース・コンテナー中の行の照会データの取得 • sqlbotcq API - 表スペース・コンテナー照会のオープン • sqlbtcq API - すべての表スペース・コンテナーの照会データの取得 • sqlefmem API - sqlbtcq および sqlbmtsq API によって割り振られたメモリーの解放

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
tabspace.sqb	• sqlbctsq API - 表スペース照会のクローズ • sqlbftpq API - 表スペース中の行の照会データのフェッチ • sqlbgtss API - 表スペースの使用率に関する統計の取得 • sqlbmtsq API - すべての表スペースの照会データの取得 • sqlbotsq API - 表スペース照会のオープン • sqlbstpq API - 単一の表スペースに関する情報の取得 • sqlefmem API - sqlbtcq および sqlbmtsq API によって割り振られたメモリーの解放
tload.sqb	• db2Export API - データベースからのデータのエクスポート • sqluvqdp API - 表の表スペースの静止
tspace.sqb	• sqlbctcq API - 表スペース・コンテナー照会のクローズ • sqlbctsq API - 表スペース照会のクローズ • sqlbftcq API - 表スペース・コンテナー中の行の照会データの取得 • sqlbftpq API - 表スペース中の行の照会データのフェッチ • sqlbgtss API - 表スペースの使用率に関する統計の取得 • sqlbmtsq API - すべての表スペースの照会データの取得 • sqlbotcq API - 表スペース・コンテナー照会のオープン • sqlbotsq API - 表スペース照会のオープン • sqlbstpq API - 単一の表スペースに関する情報の取得 • sqlbstsc API - 表スペース・コンテナーの設定 • sqlbtcq API - すべての表スペース・コンテナーの照会データの取得 • sqlefmem API - sqlbtcq および sqlbmtsq API によって割り振られたメモリーの解放
setact.cbl	• sqlesact API - 会計情報ストリングの設定

第 2 章 変更された API およびデータ構造

表 4. 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)	表スペース照会のフェッチ	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)	カタログ・データベース	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・マネージャーの停止	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・マネージャーの停止	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・スイッチの取得/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)	リセット・モニター	db2ResetMonitor (V7)
sqluadau (V8)	権限の取得	AUTH_LIST_AUTHORITIES_FOR_AUTHID 表関数 (V9.5)

表 4. 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)
sqlubkp (V5)	バックアップ・データベース	db2Backup (V8)
sqlubkup (V2)	バックアップ・データベース	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)
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)	データベースのリストア	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)	データベースのリストア	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)
SQL-AUTHORIZATIONS (V8)	権限構造	なし
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)

表 4. 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)
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)
db2DbDirInfo (V8.2)	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	db2DbDirInfoV9 (V9.1)
db2DbDirNextEntryStruct (V8.2)	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	db2DbDirNextEntryStructV9 (V9.1)
db2gDbDirNextEntryStruct (V8.2)	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	db2gDbDirNextEntryStrV9 (V9.1)

表 5. バックレベルがサポートされていない API およびデータ構造

名前	記述名	V9 でサポートされる API またはデータ構造
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
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
sqlxphqr	未確定トランザクションのリスト	db2XaListIndTrans
SQLXA-RECOVER	トランザクション API 構成	db2XaRecoverStruct

第 3 章 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論理データベース・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 インクルード・ファイル

API プロトタイプを含むインクルード・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: インクルード・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++ インクルード・ファイルのファイル拡張子は .h です。COBOL インクルード・ファイルのファイル拡張子は .cbl です。インクルード・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。以下に例を示します。

  ```
  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 呼び出しの構文です。

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インクルード・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、ほとんどのデータベース・マネージャー 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 つ以上の 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)。

DB2 API アプリケーションのインクルード・ファイル

C、C++、COBOL、および FORTRAN のアプリケーションで DB2 API を呼び出すのに使用されるインクルード・ファイルについて、以下で説明します。

C および C++ のインクルード・ファイル

DB2APIDF (db2ApiDf.h)
このファイルでは、名前が ″db2″ で始まるほとんどすべての DB2 API の構造、定数、およびプロトタイプが定義されます。

DB2AUCFG (db2AuCfg.h)
このファイルでは、DB2 API である db2AutoConfig と db2AutoConfigFreeMemory の構造、定数、およびプロトタイプが定義されます。

DB2SECPLUGIN (db2secPlugin.h)
このファイルでは、認証やグループ・メンバーシップ参照のためのカスタマイズ・セキュリティー・プラグインの開発に使用される API の構造、定数、およびプロトタイプが定義されます。

SQL (sql.h)
このファイルには、バインド・プログラム、プリコンパイラー、およびエラー・メッセージ検索 API 用の言語固有プロトタイプが含まれています。また、システム定数も定義されています。

SQLAPREP (sqlaprep.h)
このファイルには、独自のプリコンパイラーの作成に必要な定義が入っています。

SQLENV (sqlenv.h)
このファイルは、データベース環境 API に対する言語固有の呼び出し、およびそれらのインターフェースの構造、定数、戻りコードを定義します。

SQLMON (sqlmon.h)
このファイルは、データベース・システム・モニター API に対する言語固有の呼び出し、およびそれらのインターフェースの構造、定数、戻りコードを定義します。

SQLUTIL (sqlutil.h)
このファイルは、ユーティリティー API に対する言語固有の呼び出し、およびそれらのインターフェースに必要な構造、定数、コードを定義します。

SQLUVEND (sqluvend.h)
記憶管理ベンダーが使用する API の構造、定数およびプロトタイプを定義します。

SQLXA (sqlxa.h)
X/Open XA インターフェースを使用するアプリケーションが使用する関数プロトタイプと定数が含まれます。

COBOL のインクルード・ファイル

SQL (sql.cbl)
このファイルには、バインド・プログラム、プリコンパイラー、およびエラー・メッセージ検索 API 用の言語固有プロトタイプが含まれています。また、システム定数も定義されています。

SQLAPREP (sqlaprep.cbl)
このファイルには、独自のプリコンパイラーの作成に必要な定義が入っています。

SQLENV (sqlenv.cbl)
このファイルは、データベース環境 API に対する言語固有の呼び出し、およびそれらのインターフェースの構造、定数、戻りコードを定義します。

SQLMON (sqlmon.cbl)
このファイルは、データベース・システム・モニター API に対する言語固有の呼び出し、およびそれらのインターフェースの構造、定数、戻りコードを定義します。

SQLMONCT (sqlmonct.cbl)
このファイルには、データベース・システム・モニター API を呼び出すのに必要な定数定義とローカル・データ構造定義が含まれています。

SQLUTIL (sqlutil.cbl)
このファイルは、ユーティリティー API に対する言語固有の呼び出し、およびそれらのインターフェースに必要な構造、定数、コードを定義します。

FORTRAN のインクルード・ファイル

SQL (sql.f)
このファイルには、バインド・プログラム、プリコンパイラー、およびエラー・メッセージ検索 API 用の言語固有プロトタイプが含まれています。また、システム定数も定義されています。

SQLAPREP (sqlaprep.f)
このファイルには、独自のプリコンパイラーの作成に必要な定義が入っています。

SQLENV (sqlenv.f)
このファイルは、データベース環境 API に対する言語固有の呼び出し、およびそれらのインターフェースの構造、定数、戻りコードを定義します。

SQLMON (sqlmon.f)
このファイルは、データベース・システム・モニター API に対する言語固有の呼び出し、およびそれらのインターフェースの構造、定数、戻りコードを定義します。

SQLUTIL (sqlutil.f)
このファイルは、ユーティリティー API に対する言語固有の呼び出し、およびそれらのインターフェースに必要な構造、定数、コードを定義します。

第 4 章 管理 API

db2AddContact - 通知メッセージを送信できる連絡先の追加

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 インクルード・ファイル

db2ApiDf.h

API とデータ構造構文

```
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;
```

db2AddContact 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・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 構造を指すポインター。

db2AddContactData データ構造パラメーター

piUserid
入力。ユーザー名。

piPassword
入力。パラメーター piUserid によって指定されるユーザー ID のパスワードです。

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。

使用上の注意

この API は UNIX および Linux® ではサポートされません。しかし、SQL インターフェースによって同様の機能を使用することはできます。

---

db2AddContactGroup - 通知メッセージを送信できる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 インクルード・ファイル

```
db2ApiDf.h
```

API とデータ構造構文

```
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;
```

db2AddContactGroup 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・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 構造を指すポインター。

db2AddContactGroupData データ構造パラメーター

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。

db2ContactTypeData データ構造パラメーター

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。

使用上の注意

この API は UNIX および Linux ではサポートされません。しかし、SQL インターフェースによって同様の機能を使用することはできます。

db2AddSnapshotRequest - スナップショット要求の追加

この API は、db2GetSnapshotSize および db2GetSnapshot のスナップショット要求ストリームを準備します。

有効範囲

db2GetSnapshotSize および db2GetSnapshot API のスナップショット要求ストリームを準備します。出力 (db2AddSnapshotRequest API によって生成されるスナップショット要求) は、db2GetSnapshotSize および db2GetSnapshot API に渡されます。スナップショット要求には、スナップショット要求タイプと識別情報が含まれます。

許可

なし。

必要な接続

なし。

API インクルード・ファイル

```
db2ApiDf.h
```

API とデータ構造構文

```
SQL_API_RC SQL_API_FN
  db2AddSnapshotRequest (
   db2Uint32 versionNumber,
   void * pParmStruct,
   struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2AddSnapshotRqstData
{
   void *pioRequestData;
   db2Uint32 iRequestType;
   db2int32 iRequestFlags;
   db2Uint32 iQualType;
   void *piQualData;
} db2AddSnapshotRqstData;

SQL_API_RC SQL_API_FN
  db2gAddSnapshotRequest (
   db2Uint32 versionNumber,
   void * pParmStruct,
   struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2gAddSnapshotRqstData
{
   void *pioRequestData;
   db2Uint32 iRequestType;
   db2int32 iRequestFlags;
```

```
   db2Uint32 iQualType;
   void *piQualData;
   db2Uint32 iQualDataLen;
} db2gAddSnapshotRqstData;
```

db2AddSnapshotRequest 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・レベルを指定します。上記のような db2AddSnapshotData 構造を使用するには、db2Versio910 を指定します。この構造の別のバージョンを使用する場合には、include ディレクトリー内の db2ApiDf 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。指定するバージョン番号に対応する db2AddSnapshotRequestData 構造のバージョンを必ず使用してください。

pParmStruct

入力または出力 (あるいはその両方)。 db2AddSnapshotRequestData 構造を指すポインター。

pSqlca 出力。 sqlca 構造を指すポインター。

db2AddSnapshotRqstData データ構造パラメーター

pioRequestData

入出力。 db2AddSnapshotRequest API によって構成される要求データ。最初は、このパラメーターは NULL に設定されます。 pioRequestData に必要なメモリーは db2AddSnapshotRequest API によって割り振られます。使用が終わったら (例えば、db2GetSnapshot API 呼び出しの後)、pioRequestData を解放する必要があります。

iRequestType

入力。スナップショット要求タイプ (例えば、SQLMA_DB2)。

iRequestFlags

入力。ビットマップ・アクション・フラグ。値は SQLM_INSTREAM_ADD_REQUEST、SQLM_INSTREAM_ADD_QUAL、または SQLM_INSTREAM_ADD_REQQUAL です。 iRequestFlags が呼び出し側によって設定されていない場合、以下のようになります。

- iRequestType が設定されている場合、iRequestFlags ビット SQLM_INSTREAM_ADD_REQUEST が API によってオンにされます。
- piQualifierData ポインターがヌル以外の場合、SQLM_INSTREAM_ADD_QUAL が API によってオンにされます。

API 呼び出し時に、iRequestType、iQualifierType、iRequestFlags、および piQualifierData が 0 にリセットされます。

iQualType

入力。スナップショット要求にアタッチされる修飾子のタイプ (例えば、SQLM_INSTREAM_ELM_DBNAME)。

piQualData

入力。修飾子を記述するデータ。これは、ヌル終了ストリングを指すポインターです。

db2gAddSnapshotRqstData データ構造固有パラメーター

iQualDataLen
入力。piQualData パラメーター内の修飾子データの長さ。

---

db2AdminMsgWrite - 管理およびレプリケーション機能のためのログ・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 インクルード・ファイル

```
db2ApiDf.h
```

API とデータ構造構文

```
SQL_API_RC SQL_API_FN
  db2AdminMsgWrite (
   db2Uint32 versionNumber,
   void * pParmStruct,
   struct sqlca * pSqlca);

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

db2AdminMsgWrite 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 構造を指すポインター。

db2AdminMsgWriteStruct データ構造パラメーター

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。数値のプローブ・ポイントは、メッセージがソース・コード中のどの地点から出力されたかを特定するための、固有の内部 ID です。

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。

db2ArchiveLog - アクティブ・ログ・ファイルのアーカイブ

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。さらに、ユーザー出口が有効な場合、アーカイブ要求を発行します。

許可

以下のいずれか。

- 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既に存在している場合、API はエラーを戻します。

API インクルード・ファイル

db2ApiDf.h

API とデータ構造構文

```
SQL_API_RC SQL_API_FN
  db2ArchiveLog (
       db2Uint32 versionNumber,
       void * pDB2ArchiveLogStruct,
       struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2ArchiveLogStruct
{
   char *piDatabaseAlias;
   char *piUserName;
   char *piPassword;
   db2Uint16 iAllNodeFlag;
   db2Uint16 iNumNodes;
   SQL_PDB_NODE_TYPE *piNodeList;
   db2Uint32 iOptions;
} db2ArchiveLogStruct;

SQL_API_RC SQL_API_FN
  db2gArchiveLog (
       db2Uint32 versionNumber,
       void * pDB2ArchiveLogStruct,
       struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2gArchiveLogStruct
{
   db2Uint32 iAliasLen;
   db2Uint32 iUserNameLen;
   db2Uint32 iPasswordLen;
   char *piDatabaseAlias;
   char *piUserName;
   char *piPassword;
   db2Uint16 iAllNodeFlag;
   db2Uint16 iNumNodes;
   SQL_PDB_NODE_TYPE *piNodeList;
   db2Uint32 iOptions;
} db2gArchiveLogStruct;
```

db2ArchiveLog API パラメーター

versionNumber
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。

pSqlca 出力。 sqlca 構造を指すポインター。

db2ArchiveLogStruct データ構造パラメーター

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・データベース環境にのみ適用可能。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。

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。

db2gArchiveLogStruct データ構造固有パラメーター

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。

db2AutoConfig - 構成アドバイザーへのアクセス

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。

有効範囲

パーティション・データベース環境では、データベース推奨がすべてのデータベース・パーティションにデフォルトで適用されます。 db2AutoConfigInterface データ構造の iApply パラメーターに DB2_SG_APPLY_ON_ONE_NODE フラグが使用されると、変更がコーディネーター・パーティションだけに限定されます。バッファー・プールの変更は必ずシステム・カタログに適用される点に注意してください (DB2_SG_APPLY_ON_ONE_NODE は、バッファー・プール推奨には関係しない)。つまり、すべてのデータベース・パーティションが影響を受けるということです。

許可

sysadm

必要な接続

データベース

API インクルード・ファイル

db2AuCfg.h

API とデータ構造構文

```
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+1];
  char iDbAlias[SQL_ALIAS_SZ+1];
  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 db2CfgParam * pConfigs;
  void * pDataArea;
} db2ConfigValues;

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

typedef struct {
  db2Uint32 numElements;
  db2AutoConfigNameElement * pElements;
} db2AutoConfigNameArray;
typedef db2AutoConfigNameArray db2BpValues;

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

db2AutoConfig 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 構造を指すポインター。

db2AutoConfigInterface データ構造パラメーター

iProductID
入力。固有の製品 ID を指定します。 iProductID パラメーターの有効な値は以下のとおりです (include ディレクトリーの db2AuCfg.h で定義される)。

- DB2_SG_PID_DEFAULT
- DB2_SG_PID_WEBSPHERE_COMMERCE_SUITE
- DB2_SG_PID_SAP
- DB2_SG_PID_WEBSPHERE_ADVANCED_SERVER
- DB2_SG_PID_SIEBEL
- DB2_SG_PID_PS_EPM
- DB2_SG_PID_PS_ONLINE
- DB2_SG_PID_PS_BATCH

- DB2_SG_PID_PS
- DB2_SG_PID_LOTUS_DOMINO
- DB2_SG_PID_CONTENT_MANAGER

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。 iApply パラメーターの有効な値は以下のとおりです (include ディレクトリーの db2AuCfg.h で定義される)。

DB2_SG_NOT_APPLY
推奨をまったく適用しません。

DB2_SG_APPLY
すべての推奨を適用します。

DB2_SG_APPLY_DB
データベース (およびバッファー・プール) 推奨のみ適用します。

DB2_SG_APPLY_ON_ONE_NODE
現行データベース・パーティションにのみデータベース推奨を適用します (DB2_SG_APPLY および DB2_SG_APPLY_DB とともに使用した場合にのみ有効)。デフォルトでは、データベース推奨は、すべてのデータベース・パーティションに適用されます。

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。

db2AutoConfigElement データ構造パラメーター

token 入力または出力。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 入力または出力。トークンによって指定されたデータを保持します。

db2AutoConfigArray データ構造パラメーター

numElements
入力または出力。配列エレメントの数を示します。

pElements
入力または出力。エレメント配列を指すポインター。

db2ConfigValues データ構造パラメーター

numElements
入力または出力。配列エレメントの数を示します。

pConfigs
入力または出力。db2CfgParam 構造の配列を指すポインター。

pDataArea
入力または出力。構成の値が含まれるデータ域を指すポインター。

db2AutoConfigNameElement データ構造パラメーター

pName
出力。出力バッファー・プールの名前。

value 入力または出力。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。

db2AutoConfigNameArray データ構造パラメーター

numElements
入力または出力。配列エレメントの数を示します。

pElements
入力または出力。エレメント配列を指すポインター。

db2AutoConfigOutput データ構造パラメーター

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

maxagents と maxcagents 構成パラメーターは使用しない方がよいので、db2AutoConfig API の動作はこの API に渡される db2VersionNumber に依存します。バージョンが DB2 v9.5 以降の場合には、maxagents は戻されませんがそれより前のバージョンの場合には戻されます。今後のリリースでは、これらの構成パラメーターは完全に除去される予定です。

db2AutoConfigFreeMemory - db2AutoConfig API によって割り振られたメモリーの解放

db2AutoConfig API 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 インクルード・ファイル

```
db2AuCfg.h
```

API とデータ構造構文

```
SQL_API_RC SQL_API_FN
db2AutoConfigFreeMemory(
  db2Uint32 db2VersionNumber,
  void * pAutoConfigInterface,
  struct sqlca * pSqlca);
```

db2AutoConfigFreeMemory 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 構造を指すポインター。

db2Backup - データベースまたは表スペ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

パーティション・データベース環境では、デフォルトでこの 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。

パーティション・バックアップを実行するためのオプションが指定された場合、コマンドを呼び出すことができるのは、カタログ・ノードに対してだけです。すべてのデータベース・パーティション・サーバーをバックアップするためのオプションが指定されているなら、それは db2nodes.cfg ファイルの中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。それ以外の場合、API に指定されているデータベース・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。

- *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 インクルード・ファイル

db2ApiDf.h

API とデータ構造構文

```
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;
  db2int32 iAllNodeFlag;
  db2int32 iNumNodes;
  db2NodeType *piNodeList;
  db2int32 iNumMPPOutputStructs;
  struct db2BackupMPPOutputStruct *poMPPOutputStruct;
```

```
} db2BackupStruct;

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

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

typedef SQL_STRUCTURE db2BackupMPPOutputStruct
{
  db2NodeType nodeNumber;
  db2Uint64 backupSize;
  struct sqlca  sqlca;
} db2BackupMPPOutputStruct;

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;
  db2int32 iAllNodeFlag;
  db2int32 iNumNodes;
  db2NodeType *piNodeList;
  db2int32 iNumMPPOutputStructs;
  struct db2gBackupMPPOutputStruct *poMPPOutputStruct;
} 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 db2gBackupMPPOutputStruct
{
  db2NodeType nodeNumber;
  db2Uint64 backupSize;
  struct sqlca  sqlca;
} db2gBackupMPPOutputStruct;

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

db2Backup 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 構造を指すポインター。

db2BackupStruct データ構造パラメーター

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。

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。

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。

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 で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。 詳しくは、後出の 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。

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。

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。有効な値は以下のとおりです (インクルード・ディレクトリーの db2ApiDf ヘッダー・ファイルで定義される)。

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、例えば、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 (例えば、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。有効な値は以下のとおりです (インクルード・ディレクトリーの db2ApiDf ヘッダー・ファイルで定義される)。

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指定してください。

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、正常に実行された全バックアップ操作のうち最新のものが実行されて以来変更された、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、正常に実行された任意のタイプのバックアップ操作のうち最新のものが実行されて以来変更された、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。
EXCLUDE LOGS パラメーターが指定されない限り、データベース・マネージャーはバックアップ・イメージにデータベース・ログを含めます。デフォルト動作は、パーティション・データベースの非 SSV バックアップに適用されません。オフライン・バックアップの場合、このオプションは無効です。以下のバックアップ・シナリオの場合、デフォルトでログは含められます。

- 単一パーティション・データベースのオンライン・バックアップ
- 複数パーティション・データベースの、オンラインまたはオフラインのシングル・システム・ビュー (SSV) バックアップ
- オンラインまたはオフラインのスナップショット・バックアップ

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。ただし、スナップショット・バックアップの場合は例外であり、その場合は INCLUDE がデフォルトになります。このオプションは、パーティション・データベースの非 SSV バックアップのデフォルト動作です。
以下のバックアップ・シナリオの場合、デフォルトでログは含められません。

- 単一パーティション・データベースのオフライン・バックアップ
- シングル・システム・ビュー・バックアップを使用しない複数パーティション・データベースのオンラインまたはオフライン・バックアップ

DB2BACKUP_MPP
パーティション・データベースに適した方法でバックアップが実行されます。

iUtilImpactPriority
入力。バックアップ時に使用される優先度の値を指定します。

- この値がゼロ以外の場合、ユーティリティーはスロットルで実行されます。そうでない場合、ユーティリティーは非スロットルで実行されます。
- 複数のユーティリティーが並行して実行中の場合、このパラメーターは、スロットルされたタスク間の相対的な優先度を決定するために使用されます。例えば、2 つの並行バックアップが存在し、1 つは優先度 2、もう 1 つは優先度 4 だとします。両方ともスロットルされますが、優先度 4 のバックアップにより多くのリソースが割り振られます。優先度を 2 と 4 に設定する場合と、優先度を 5 と 10、または 30 と 60 に設定する場合とでは、違いはありません。優先度の値はまったく相対的です。

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。

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、DB2 によって、サーバー上に存在するファイルの名前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。

iAllNodeFlag
入力。パーティション・データベース環境のみ。バックアップ操作が、db2nodes.cfg で定義されているデータベース・パーティション・サーバーのすべてに適用されるのか、それとも一部に適用されるのかを示します。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。

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。バックアップを実行するデータベース・パーティション・サーバー番号の配列を指すポインター。

iNumMPPOutputStructs
入力。**piMPPOutputStruct** 配列内のエレメントの数を指定します。これは、このバックアップ操作に関係するデータベース・パーティション・サーバーの数以上でなければなりません。

piMPPOutputStruct
出力。特定のデータベース・パーティション・サーバーのための出力パラメーターを指定する db2BackupMPPOutputStruct 構造の配列へのポインター。

db2TablespaceStruct データ構造固有パラメーター

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

db2MediaListStruct データ構造パラメーター

locations
入力。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
入力。locations パラメーター内の項目数。

locationType
入力。メディア・タイプを示す文字。有効な値は以下のとおりです (include ディレクトリーの sqlutil ヘッダー・ファイルで定義される)。

SQLU_LOCAL_MEDIA: 'L'
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: 'X'
XBSA インターフェース。

SQLU_TSM_MEDIA: 'A'
Tivoli® Storage Manager。

SQLU_OTHER_MEDIA: 'O'
ベンダー・ライブラリー。

SQLU_SNAPSHOT_MEDIA: 'F'
スナップショット・バックアップを取ることを指定します。

SQLU_SNAPSHOT_MEDIA は、以下のいずれかと共には使用できません。

- DB2BACKUP_COMPRESS
- DB2BACKUP_TABLESPACE
- DB2BACKUP_INCREMENTAL
- **iNumBuffers**
- **iBufferSize**
- **iParallelism**
- **piComprOptions**
- **iUtilImpactPriority**
- スナップショット・リストアの場合、この構造の **numLocations** フィールドは 1 でなければなりません。

スナップショット・バックアップのデフォルトの動作は、すべてのコンテナー、ローカル・ボリューム・ディレクトリー、データベース・パス (DBPATH)、および 1 次ログとミラー・ログのパスを含む、データベースを構成するすべてのパスの FULL DATABASE OFFLINE バックアップです (明示的に EXCLUDE LOGS が指定されているのでない限り、すべてのスナップショット・バックアップでのデフォルトは INCLUDE LOGS です)。

IBM Data Server には、以下のストレージ・ハードウェアのための DB2 ACS API ドライバーが組み込まれています。

- IBM TotalStorage® SAN ボリューム・コントローラー
- IBM Enterprise Storage Server® Model 800
- IBM System Storage™ DS6000™
- IBM System Storage DS8000™
- IBM System Storage N Series
- NetApp V-series
- NetApp FAS

db2BackupMPPOutputStruct および db2gBackupMPPOutputStruct データ構造パラメーター

nodeNumber
オプションの適用対象となるデータベース・パーティション・サーバー。

backupSize
指定されたデータベース・パーティション上のバックアップのサイズ (KB)。

sqlca 指定されたデータベース・パーティションの sqlca。

db2gBackupStruct データ構造固有パラメーター

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)。

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.h に定義) と等しくなければなりません。

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) です。 SQLU_TIME_STAMP_LEN+1 (sqlutil.h に定義) と等しくなければなりません。

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。

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。

iComprLibraryLen
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。

db2Char データ構造パラメーター

pioData
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
入力。**pioData** バッファーのサイズ。

oLength
出力。 **pioData** バッファー内にあるデータの有効文字の数。

使用上の注意

スナップショット・バックアップを実行できるのは、**versionNumber** が db2Version950 以上の場合だけです。 **versionNumber** が db2Version950 より低い場合に、メディア・タイプ SQLU_SNAPSHOT_MEDIA を指定すると、DB2 データベースからエラーが戻されます。

この関数は、あらゆるラベル・ベース・アクセス制御 (LBAC) 規則の適用外です。保護データを含め、あらゆるデータをバックアップします。また、バックアップ中

のデータ自身も LBAC によっては保護されません。バックアップとそれをリストアする場所があれば、あらゆるユーザーがデータにアクセスできます。

定期的にデータベースをバックアップしていくと、非常に大きなデータベース・バックアップ・イメージ、多くのデータベース・ログ、およびロード・コピー・イメージが累積する場合があり、これらすべてが大量のディスク・スペースを占めることがあります。これらのリカバリー・オブジェクトの管理方法については、『リカバリー・オブジェクトの管理』を参照してください。

パーティション・データベース環境でのシングル・システム・ビュー (SSV) バックアップに関する使用上の注意

- SSV バックアップを実行するには、**iOptions** に DB2BACKUP_MPP および DB2BACKUP_DB か DB2BACKUP_TABLESPACE のいずれかを指定してください。
- SSV バックアップを実行できるのは、**versionNumber** が db2Version950 以上の場合だけです。 **versionNumber** が db2Version950 より低い場合に **iOptions** に DB2BACKUP_MPP を指定すると、DB2 データベースからエラーが戻されます。**versionNumber** が db2Version950 より低い場合に SSV バックアップに関連するその他のオプションを指定した場合、DB2 データベースはそれらのオプションを無視します。
- db2BackupStruct の中で直接指定されている **piMediaList** の値は、すべてのノードのデフォルト値として使用されます。
- db2BackupStruct に戻される **oBackupSize** の値は、すべてのノードのバックアップ・サイズの合計です。 db2BackupMPPOutputStruct に戻される **backupSize** の値は、指定されたデータベース・パーティションのバックアップのサイズです。
- **iAllNodeFlag**、**iNumNodes**、および **piNodeList** は、db2RollforwardStruct 内にある類似の名前の要素の場合と同じ働きをします。ただし、**iAllNodeFlag** については CAT_NODE_ONLY 値がないという例外があります。
- 呼び出し元アクションが DB2BACKUP_BACKUP として実行される SSV バックアップは、呼び出し元アクション DB2BACKUP_NOINTERRUPT が指定されたかのようにして実行されます。
- ***poMPPOutputStruct** は、バックアップに関係するデータベース・パーティション以上の数の要素を含む領域として、呼び出し元によって割り振られたメモリーを指すポインターです。

db2CfgGet - データベース・マネージャー構成パラメーター、あるいはデータベース構成パラメーターの取得

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 インクルード・ファイル

db2ApiDf.h

API とデータ構造構文

```
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;

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;
```

db2CfgGet 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・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 構造を指すポインター。

db2Cfg データ構造パラメーター

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 入力。実行するアクションのタイプを指定します。有効な値は以下のとおりです (インクルード・ディレクトリーの db2ApiDf ヘッダー・ファイルで定義される)。

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。

db2CfgReset
デフォルト値へのリセットを行います。

dbname
入力。データベース名。

db2CfgParam データ構造パラメーター

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要約』にリストされています。

ptrvalue
出力。構成パラメーターの値。

flags 出力。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。有効な値は以下のとおりです (インクルード・ディレクトリーの db2ApiDf ヘッダー・ファイルで定義される)。

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。

db2CfgParamComputed
検索されるパラメーターの値が computed かどうかを示します。特定の構成パラメーターが computed 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 db2CfgParamComputed キーワードに対して、ブール AND 演算を実行します。

上記のキーワードの両方に対するブール AND 演算が false である場合、検索されるパラメーター値が手動で設定されることを意味します。

db2gCfg データ構造固有パラメーター

dbname_len
入力。dbname パラメーターの長さ (バイト単位)。

db2gCfgParam データ構造固有パラメーター

ptrvalue_len
入力。ptrvalue パラメーターの長さ (バイト単位)。

使用上の注意

構成パラメーター maxagentsと maxcagents は使用すべきではありません。今後のリリースでは、これらの構成パラメーターは完全に除去される予定です。

db2CfgGet API は SQLF_KTN_MAXAGENTS および SQLF_KTN_MAXCAGENTS に関する要求を許容しますが、サーバーが DB2 v9.5 の場合には 0 が戻ります。

---

db2CfgSet - データベース・マネージャー構成パラメーター、あるいはデータベース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

デフォルトで、データベース構成ファイルの変更は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

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。

- 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、インスタンスへのアタッチメントは必要ありません。

API インクルード・ファイル

```
db2ApiDf.h
```

API とデータ構造構文

```
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;
   SQL_PDB_NODE_TYPE dbpartitionnum;
}  db2Cfg;

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

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;
```

db2CfgSet 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・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 構造を指すポインター。

db2Cfg データ構造パラメーター

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 入力。実行するアクションのタイプを指定します。有効な値は以下のとおりです (インクルード・ディレクトリーの db2ApiDf ヘッダー・ファイルで定義される)。

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。

db2CfgReset
デフォルト値へのリセットを行います。

db2CfgSingleDbpartition
特定のデータベース・パーティションでのデータベース構成を更新またはリセットするには、このフラグを設定して、dbpartitionnum の値を提供します。

dbname
入力。データベース名。

dbpartitionnum
入力。この API が構成値を設定するデータベース・パーティションを指定します。

db2CfgParam データ構造パラメーター

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要約』にリストされています。

ptrvalue
出力。構成パラメーターの値。

flags 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。有効な値は以下のとおりです (インクルード・ディレクトリーの db2ApiDf ヘッダー・ファイルで定義される)。

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。

db2CfgParamComputed
検索されるパラメーターの値が computed かどうかを示します。特定の構成パラメーターが computed 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 db2CfgParamComputed キーワードに対して、ブール AND 演算を実行します。

db2CfgParamManual
自動または計算されたパラメーターの設定を解除して、パラメーターを現行値に設定するのに使用されます。**ptrvalue** フィールドは無視され、NULL に設定できます。

db2gCfg データ構造固有パラメーター

dbname_len
入力。dbname パラメーターの長さ (バイト単位)。

db2gCfgParam データ構造固有パラメーター

ptrvalue_len
入力。ptrvalue パラメーターの長さ (バイト単位)。

使用上の注意

構成パラメーター maxagentsと maxcagents は使用すべきではありません。今後のリリースでは、これらの構成パラメーターは完全に除去される予定です。

db2CfgSet API は maxagents と maxcagents 構成パラメーターに対する更新を許容しますが、こうした更新は DB2 によって無視されます。

使用方法のサンプル

ケース 1: MAXAPPLS パラメーターを dbpartitionnum 30 で 50 に設定します。

ケース 2: MAXAPPLS パラメーターをすべての dbpartitionnum で 50 に設定します。

```
int updateDbConfig()
{
    struct sqlca sqlca = {0};
    db2Cfg cfgStruct = {0};
    db2CfgParam cfgParameters[2];
    char *dbAlias = "SAMPLE";

    /* initialize cfgParameters */
    cfgParameters[0].flags = 0;
    cfgParameters[0].token = SQLF_DBTN_TSM_OWNER;
    cfgParameters[0].ptrvalue = (char *)malloc(sizeof(char) * 65);
    cfgParameters[1].flags = 0;
    cfgParameters[1].token = SQLF_DBTN_MAXAPPLS;
    cfgParameters[1].ptrvalue = (char *)malloc(sizeof(sqluint16));

    /* set two DB Config. fields  */
    strcpy(cfgParameters[0].ptrvalue, "tsm_owner");
    *(sqluint16 *)(cfgParameters[1].ptrvalue) = 50;

    /* initialize cfgStruct to update db cfg on single partition*/
    cfgStruct.numItems = 2;
    cfgStruct.paramArray = cfgParameters;
    cfgStruct.flags = db2CfgDatabase | db2CfgImmediate;
    cfgStruct.flags |= db2CfgSingleDbpartition;
    cfgStruct.dbname = dbAlias;
    cfgStruct.dbpartitionnum = 30;

    /* CASE 1: update database configuration */
    db2CfgSet(db2Version950, (void *)&cfgStruct, &sqlca);

    /* set cfgStruct to update db cfg on all db partitions */
    cfgStruct.flags &= ~db2CfgSingleDbpartition;

    /* CASE 2: update database configuration */
    db2CfgSet(db2Version950, (void *)&cfgStruct, &sqlca);
    ..............
}
```

db2ConvMonStream - バージョン 6 以前の形式へのモニター・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しい自己記述型形式 (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・データで全探索を行う必要があります (例えば、 SQLM_ELM_DB2 エレメント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 インクルード・ファイル

db2ApiDf.h

API とデータ構造構文

```
SQL_API_RC SQL_API_FN
  db2ConvMonStream (
   db2Uint32 versionNumber,
   void * pParmStruct,
   struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2ConvMonStreamData
{
   void *poTarget;
   struct sqlm_header_info *piSource;
   db2Uint32 iTargetType;
   db2Uint32 iTargetSize;
   db2Uint32 iSourceType;
} db2ConvMonStreamData;

SQL_API_RC SQL_API_FN
  db2gConvMonStream (
   db2Uint32 versionNumber,
   void * pParmStruct,
   struct sqlca * pSqlca);
```

db2ConvMonStream 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。db2ConvMonStreamData 構造を指すポインター。

pSqlca 出力。 sqlca 構造を指すポインター。

db2ConvMonStreamData データ構造パラメーター

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 6. 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 7. 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

表 7. 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_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
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。

db2DatabasePing - ネットワーク応答時間のテストのためのデータベースの 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アクセスする場合に、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 インクルード・ファイル

```
db2ApiDf.h
```

API とデータ構造構文

```
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;
} db2DatabasePingStruct;

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 + 1];
   db2int32 RequestPacketSz;
   db2int32 ResponsePacketSz;
   db2Uint16 iNumIterations;
   db2Uint32 *poElapsedTime;
} db2gDatabasePingStruct;
```

db2DatabasePing API パラメーター

versionNumber
入力。アプリケーションが使用する DB2 データベースまたは 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 構造を指すポインター。

db2DatabasePingStruct データ構造パラメーター

iDbAlias
入力。データベース別名。将来の利用のために予約済み。値が設定されても無視されます。

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® Universal Database™ (UDB) for Linux, UNIX and 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 and 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。

db2gDatabasePingStruct データ構造固有パラメーター

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。

使用上の注意

この API は、DB2 UDB バージョン 7 クライアントから、DB2 Connect バージョン 8 経由で DB2 ホスト・データベース・サーバーへの接続に使用された場合、機能しません。

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するか、指定の時間 (単位: 分) だけ、現行の作業単位を完了するのを待ち (指定の時間内に作業単位が完了しなければ静止モードへの移行操作は失敗ということになります)、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、データベースに対しては、適切な権限を持ったユーザーによってシステム管理が行えます。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。

- sysadm
- dbadm

必要な接続

データベース

API インクルード・ファイル

```
db2ApiDf.h
```

API とデータ構造構文

```
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;

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;
```

db2DatabaseQuiesce 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・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。

db2DbQuiesceStruct データ構造パラメーター

piDatabaseName
入力。データベース名。

iImmediate
入力。有効な値は以下のとおりです。

TRUE=1
アプリケーションを即座に強制クローズする。

FALSE=0
据え置き強制クローズ。アプリケーションの現行の作業単位が完了してから終了するように、アプリケーションは iTimeout パラメーターに指定された時間 (単位: 分) だけ待機します。 iTimeout パラメーターに指定された時間内 (単位: 分) に据え置き強制クローズが完了できなければ、静止の操作は失敗に終わります。

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ー start_stop_time によって指定された値が使用されます。

db2gDbQuiesceStruct データ構造固有パラメーター

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

---

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 インクルード・ファイル

db2ApiDf.h

API とデータ構造構文

```
SQL_API_RC SQL_API_FN
  db2DatabaseRestart (
   db2Uint32 versionNumber,
   void * pParmStruct,
   struct sqlca * pSqlca);

typedef struct db2RestartDbStruct
{
   char *piDatabaseName;
   char *piUserId;
   char *piPassword;
   char *piTablespaceNames;
   db2int32 iOption;
} db2RestartDbStruct;

SQL_API_RC SQL_API_FN
  db2gDatabaseRestart (
   db2Uint32 versionNumber,
   void * pParmStruct,
   struct sqlca * pSqlca);

typedef struct db2gRestartDbStruct
{
   db2Uint32 iDatabaseNameLen;
   db2Uint32 iUserIdLen;
   db2Uint32 iPasswordLen;
   db2Uint32 iTablespaceNamesLen;
```

```
   char *piDatabaseName;
   char *piUserId;
   char *piPassword;
   char *piTablespaceNames;
} db2gRestartDbStruct;
```

db2DatabaseRestart 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 構造を指すポインター。

db2RestartDbStruct データ構造パラメーター

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。

db2gRestartDbStruct データ構造固有パラメーター

iDatabaseNameLen
入力。piDatabaseName パラメーターの長さ (バイト単位)。

iUserIdLen
入力。piUserId パラメーターの長さ (バイト単位)。

iPasswordLen
入力。piPassword パラメーターの長さ (バイト単位)。

iTablespaceNamesLen
入力。piTablespaceNames パラメーターの長さ (バイト単位)。

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 (例えば、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できますが、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。

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。

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。

- sysadm
- dbadm

必要な接続

データベース

API インクルード・ファイル

```
db2ApiDf.h
```

API とデータ構造構文

```
SQL_API_RC SQL_API_FN
  db2DatabaseUnquiesce (
              db2Uint32 versionNumber,
              void * pParmStruct,
              struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2DbUnquiesceStruct
{
              char *piDatabaseName;
} db2DbUnquiesceStruct;

SQL_API_RC SQL_API_FN
  db2gDatabaseUnquiesce (
              db2Uint32 versionNumber,
              void * pParmStruct,
              struct sqlca * pSqlca);

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

db2DatabaseUnquiesce 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・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。

db2DbUnquiesceStruct データ構造パラメーター

piDatabaseName
入力。データベース名。

db2gDbUnquiesceStruct データ構造固有パラメーター

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

---

db2DbDirCloseScan - 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・ディレクトリーのスキャンの終了

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 インクルード・ファイル

db2ApiDf.h

API とデータ構造構文

```
SQL_API_RC SQL_API_FN
  db2DbDirCloseScan (
       db2Uint32 versionNumber,
       void * pParmStruct,
       struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2DbDirCloseScanStruct
{
   db2Uint16 iHandle;
} db2DbDirCloseScanStruct;

SQL_API_RC SQL_API_FN
  db2gDbDirCloseScan (
       db2Uint32 versionNumber,
       void * pParmStruct,
       struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2gDbDirCloseScanStruct
{
   db2Uint16 iHandle;
} db2gDbDirCloseScanStruct;
```

db2DbDirCloseScan 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 構造を指すポインター。

db2DbDirCloseScanStruct データ構造パラメーター

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

---

db2DbDirGetNextEntry - 次のシステム・データベース・ディレクトリー、あるいはローカル・データベース・ディレクトリー項目の取得

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 インクルード・ファイル

```
db2ApiDf.h
```

API とデータ構造構文

```
SQL_API_RC SQL_API_FN
  db2DbDirGetNextEntry (
       db2Uint32 versionNumber,
       void * pParmStruct,
       struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2DbDirNextEntryStructV9
{
   db2Uint16 iHandle;
   struct db2DbDirInfoV9 *poDbDirEntry;
} db2DbDirNextEntryStructV9;

SQL_STRUCTURE db2DbDirInfoV9
{
   _SQLOLDCHAR alias[SQL_ALIAS_SZ];
   _SQLOLDCHAR dbname[SQL_DBNAME_SZ];
   _SQLOLDCHAR drive[SQL_DB_PATH_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];
};

SQL_API_RC SQL_API_FN
  db2gDbDirGetNextEntry (
       db2Uint32 versionNumber,
       void * pParmStruct,
       struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2gDbDirNextEntryStrV9
{
   db2Uint16 iHandle;
   struct db2DbDirInfoV9 *poDbDirEntry;
} db2gDbDirNextEntryStrV9;
```

db2DbDirGetNextEntry 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。db2DbDirGetNextEntryStructV9 構造を指すポインター。

pSqlca 出力。 sqlca 構造を指すポインター。

db2DbDirNextEntryStructV9 データ構造パラメーター

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

poDbDirEntry
出力。 db2DbDirInfoV9 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。

db2DbDirInfoV9 データ構造パラメーター

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_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。旧バージョン (DB2 V2 以前) のサーバー以外に接続するときは、この値を設定してください。

SQL_AUTHENTICATION_SVR_ENCRYPT
認証がターゲット・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
認証が、ターゲット・データベースを含むノードで行われ、接続がデータ暗号化を使用しなければならないことを指定します。

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。

スキャンする項目がない場合に db2DbDirGetNextEntry が呼び出されると、SQL1014N が SQLCA 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。

db2DbDirOpenScan - 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・ディレクトリーのスキャンの開始

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 API を使用してます。スキャンをクローズするには、db2DbDirCloseScan API 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 インクルード・ファイル

```
db2ApiDf.h
```

API とデータ構造構文

```
SQL_API_RC SQL_API_FN
  db2DbDirOpenScan (
       db2Uint32 versionNumber,
       void * pParmStruct,
       struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2DbDirOpenScanStruct
{
   char *piPath;
   db2Uint16 oHandle;
   db2Uint16 oNumEntries;
} db2DbDirOpenScanStruct;

SQL_API_RC SQL_API_FN
  db2gDbDirOpenScan (
       db2Uint32 versionNumber,
       void * pParmStruct,
       struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2gDbDirOpenScanStruct
{
   db2Uint32 iPath_len;
   char *piPath;
   db2Uint16 oHandle;
   db2Uint16 oNumEntries;
} db2gDbDirOpenScanStruct;
```

db2DbDirOpenScan 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 構造を指すポインター。

db2DbDirOpenScanStruct データ構造パラメーター

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 API 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 API 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。

db2gDbDirOpenScanStruct データ構造固有パラメーター

iPath_len
入力。piPath パラメーターの長さ (バイト単位)。

使用上の注意

この API が割り振ったストレージは、db2DbDirCloseScan API 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。

db2DropContact - 通知メッセージを送信できる連絡先リストからの連絡先の削除

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 インクルード・ファイル

```
db2ApiDf.h
```

API とデータ構造構文

```
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;
```

db2DropContact 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・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 構造を指すポインター。

db2DropContactData データ構造パラメーター

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。

使用上の注意

この API は UNIX および Linux ではサポートされません。しかし、SQL インターフェースによって同様の機能を使用することはできます。

db2DropContactGroup - 通知メッセージを送信できる連絡先リストからの連絡先グループの削除

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 インクルード・ファイル

```
db2ApiDf.h
```

API とデータ構造構文

```
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;
```

db2DropContactGroup 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・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 構造を指すポインター。

db2DropContactData データ構造パラメーター

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。

使用上の注意

この API は UNIX および Linux ではサポートされません。しかし、SQL インターフェースによって同様の機能を使用することはできます。

db2Export - データベースからのデータのエクスポート

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、または型付き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。

- 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 特権この関数に対しては、ラベル・ベースのアクセス制御 (LBAC) が実施されます。エクスポートするデータは、LBAC で保護されている場合、呼び出し元の LBAC 信用証明情報によって制限を受けることがあります。

必要な接続

データベース。暗黙接続が可能な場合には、デフォルト・データベースへの接続が確立されます。

API インクルード・ファイル

db2ApiDf.h

API とデータ構造構文

```
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;
   struct db2ExportIn *piExportInfoIn;
   struct sqlu_media_list *piXmlPathList;
   struct sqlu_media_list *piXmlFileList;
} db2ExportStruct;

typedef SQL_STRUCTURE db2ExportIn
{
   db2Uint16 *piXmlSaveSchema;
} db2ExportIn;

typedef SQL_STRUCTURE db2ExportOut
{
   db2Uint64 oRowsExported;
} db2ExportOut;

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;
   struct db2ExportIn *piExportInfoIn;
   struct sqlu_media_list *piXmlPathList;
   struct sqlu_media_list *piXmlFileList;
} db2gExportStruct;
```

db2Export 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・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 構造を指すポインター。

db2ExportStruct データ構造パラメーター

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 フィールドを SQLU_LOCAL_MEDIA に設定された sqlu_media_list 構造、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その sqlu_media_entry 構造を指すポインター。エクスポートした LOB データは、sqlu_media_entry 構造にリストされているすべてのパスに均一に分散されます。

piLobFileList
入力。media_type フィールドを SQLU_CLIENT_LOCATION に設定された sqlu_media_list 構造と、基本ファイル名の入ったその sqlu_location_entry 構造を指すポインター。

このリスト内の最初の名前を使用しているネーム・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。エクスポート操作での LOB ファイルの作成時には、現行パス (piLobPathList からの) にこのリストから現行ベース名を追加し、その後に 3 桁のシーケンス番号と .lob 拡張子を追加した形のファイル名が構成されます。例えば、現行の LOB パスが /u/foo/lob/path ディレクトリー、現行の LOB ファイル名が bar、そして LOBSINSEPFILES ファイル・タイプ修飾子が設定済みの場合、作成される LOB ファイルは、/u/foo/LOB/path/bar.001.lob、/u/foo/LOB/path/bar.002.lob、などとなります。 LOBSINSEPFILES ファイル・タイプ修

飾子を設定しない場合、すべての LOB 文書が連結され
て、/u/foo/lob/path/bar.001.lob という 1 つのファイルに書き込まれます。

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。このパラメーターの有効な値は以下のとおりです (インクルード・ディレクトリーの sqlutil ヘッダー・ファイルで定義される)。

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、piActionString に指定された 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 ステートメントの入った 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 ヘッダー・ファイルで定義) 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することができます。

piFileTypeMod

入力。2 バイトの長さフィールドと、1 つ以上の処理オプションを指定する文字の配列を収容する 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。以下の『エクスポート・ユーティリティーのファイル・タイプ修飾子』の関連リンク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。ファイルが既に存在する場合は、その情報が付加されます。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 (インクルード・ディレクトリーの sqlutil ヘッダー・ファイルで定義される)。

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 (例えば、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 (例えば、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。

piExportInfoIn

入力。db2ExportIn 構造を指すポインター。

piXmlPathList

入力。media_type フィールドを SQLU_LOCAL_MEDIA に設定された sqlu_media_list 構造、および XML 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その sqlu_media_entry 構造を指すポインター。エクスポートした XML データは、sqlu_media_entry 構造にリストされているすべてのパスに均一に分散されます。

piXmlFileList

入力。media_type フィールドを SQLU_CLIENT_LOCATION に設定された sqlu_media_list 構造と、基本ファイル名の入ったその sqlu_location_entry 構造を指すポインター。

このリスト内の最初の名前を使用しているネーム・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。エクスポート操作での XML ファイルの作成時には、現行パス (piXmlFileList からの) にこのリストから現行ベース名を追加し、その後に 3 桁のシーケンス番号と .xml 拡張子を追加した形のファイル名が構成されます。例えば、現行の XML パスが /u/foo/xml/path ディレクトリー、現行の XML ファイル名が bar、そして XMLINSEPFILES ファイル・タイプ修飾子が設定済みの場合、作成される XML ファイルは /u/foo/xml/path/bar.001.xml、/u/foo/xml/path/bar.002.xml、などとなります。 XMLINSEPFILES ファイル・タイプ修飾子を設定しない場合、すべての XML 文書が連結されて、/u/foo/xml/path/bar.001.xml という 1 つのファイルに書き込まれます。

db2ExportIn データ構造パラメーター

piXmlSaveSchema

入力。エクスポートされた各 XML 文書を検証するのに使用される XML スキーマの SQL ID を、エクスポートされたデータ・ファイルに保管する必要があることを指示します。指定可能な値は TRUE および FALSE です。

db2ExportOut データ構造パラメーター

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。

db2gExportStruct データ構造固有パラメーター

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。

使用上の注意

エクスポート操作を開始する前に、以下の 2 つの方法のいずれかで、すべての表操作を完了し、すべてのロックを解放する必要があります。

- WITH HOLD 節を使って定義したすべてのオープン・カーソルをクローズし、COMMIT ステートメントを実行して、データの変更をコミットします。
- ROLLBACK ステートメントを実行して、データ変更をロールバックします。

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。

エクスポート・ユーティリティーから警告が出された場合、そのメッセージは、メッセージ・ファイルに書き込まれますが、そのファイルを指定していない場合は、標準出力に書き込まれます。

外部列名配列 piDataDescriptor の列数 (sqldcol 構造の dcolnum フィールド) が、SELECT ステートメントによって生成された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・プログラムでフォーマット・オプションを使用しないでください。

DB2 Connect を使用して、DB2 for z/OS and OS/390®、DB2 for VM and VSE、および DB2 for System i® などの DRDA® サーバーから表をエクスポートすることができます。PC/IXF エクスポートのみサポートされます。

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。

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 piActionString パラメーターを指定して 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。ビューの索引は保管されません。piActionString の SELECT 節に結合が入っている場合も同様です。 piActionString パラメーターの WHERE 節、GROUP BY 節、または HAVING 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・ユーティリティーは表の NOT NULL WITH DEFAULT 属性を IXF ファイルに保管します。

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は、ターゲット表名と WHERE 節を指定することによってのみ表現することができます。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。

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。

db2GetAlertCfg - ヘルス・インディケーターのアラート構成設定の取得

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。インスタンス接続が存在しない場合は、デフォルトのインスタンス接続が作成されます。

API インクルード・ファイル

db2ApiDf.h

API とデータ構造構文

```
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;
   struct db2GetAlertCfgInd *pioIndicators;
} db2GetAlertCfgData;

typedef SQL_STRUCTURE db2GetAlertCfgInd
{
   db2Uint32 ioIndicatorID;
   sqlint64 oAlarm;
   sqlint64 oWarning;
   db2Uint32 oSensitivity;
   char *poFormula;
   db2Uint32 oActionEnabled;
   db2Uint32 oCheckThresholds;
   db2Uint32 oNumTaskActions;
   struct db2AlertTaskAction *poTaskActions;
   db2Uint32 oNumScriptActions;
   struct 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;
```

db2GetAlertCfg 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・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 構造を指すポインター。

db2GetAlertCfgData データ構造パラメーター

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。

db2GetAlertCfgInd データ構造パラメーター

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ゾーン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、しきい値を超過した場合、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、しきい値の超過、または状態変更が評価されます。しきい値の超過、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。

db2AlertTaskAction データ構造パラメーター

pTaskname
タスク名。

condition
アクションを実行する条件。

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。

db2AlertScriptAction データ構造パラメーター

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

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
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 API がエラーを戻さない場合はいつでも、db2GetAlertCfgFree API を呼び出して解放しなければなりません。 db2GetAlertCfgFree API の詳細については、インクルード・ディレクトリーの db2ApiDf.h を参照してください。

---

db2GetAlertCfgFree - db2GetAlertCfg API によって割り振られたメモリーの解放

db2GetAlertCfg 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 インクルード・ファイル

```
db2ApiDf.h
```

API とデータ構造構文

```
SQL_API_RC SQL_API_FN
  db2GetAlertCfgFree (
   db2Uint32 versionNumber,
   void * pParmStruct,
   struct sqlca * pSqlca);
```

db2GetAlertCfgFree 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 構造を指すポインター。

---

db2GetContactGroup - 通知メッセージを送信できる単一の連絡先グループ中の連絡先のリストの取得

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 インクルード・ファイル

```
db2ApiDf.h
```

API とデータ構造構文

```
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;
```

db2GetContactGroup 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・レベルを指定します。

pParmStruct
入力。db2ContactGroupData 構造を指すポインター。

pSqlca 出力。 sqlca 構造を指すポインター。

db2ContactGroupData データ構造パラメーター

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。

db2ContactTypeData データ構造パラメーター

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。

使用上の注意

この API は UNIX および Linux ではサポートされません。しかし、SQL インターフェースによって同様の機能を使用することはできます。

db2GetContactGroups - 通知メッセージを送信できる連絡先グループのリストの取得

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 インクルード・ファイル

```
db2ApiDf.h
```

API とデータ構造構文

```
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;
```

db2GetContactGroups 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・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 構造を指すポインター。

db2GetContactGroupsData データ構造パラメーター

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。

db2ContactGroupDesc データ構造パラメーター

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。

使用上の注意

この API は UNIX および Linux ではサポートされません。しかし、SQL インターフェースによって同様の機能を使用することはできます。

db2GetContacts - 通知メッセージを送信できる連絡先リストの取得

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 インクルード・ファイル

db2ApiDf.h

API とデータ構造構文

```
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;
```

db2GetContacts 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・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 構造を指すポインター。

db2GetContactsData データ構造パラメーター

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。

db2ContactData データ構造パラメーター

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。

使用上の注意

この API は UNIX および Linux ではサポートされません。しかし、SQL インターフェースによって同様の機能を使用することはできます。

db2GetHealthNotificationList - ヘルス・アラート通知を送信できる連絡先リストの取得

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。インスタンス接続が存在しない場合は、デフォルトのインスタンス接続が作成されます。

API インクルード・ファイル

db2ApiDf.h

API とデータ構造構文

```
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;
```

db2GetHealthNotificationList 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・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 構造を指すポインター。

db2GetHealthNotificationListData データ構造パラメーター

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。

db2ContactTypeData データ構造パラメーター

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。

db2GetRecommendations - アラート状態のヘルス・インディケーターを解決するための推奨の入手

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。インスタンス接続が存在しない場合は、デフォルトのインスタンス接続が作成されます。

API インクルード・ファイル

```
db2ApiDf.h
```

API とデータ構造構文

```
SQL_API_RC SQL_API_FN
  db2GetRecommendations (
   db2Uint32 versionNumber,
   void * pParmStruct,
   struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2GetRecommendationsData
{
   db2Uint32 iSchemaVersion;
   db2Uint32 iNodeNumber;
   db2Uint32 iIndicatorID;
   db2Uint32 iObjType;
   char *piObjName;
   char *piDbName;
   char *poRecommendation;
} db2GetRecommendationsData;
```

db2GetRecommendations 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・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 構造を指すポインター。

db2GetRecommendationsData データ構造パラメーター

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。このパラメーターは DB2HEALTH_RECSCHEMA_VERSION8_2 に設定します。

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データベー

ス・パーティション番号を示します。すべてのデータベース・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。複数の異なるデータベース・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、異なるデータベース・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。現在のデータベース・パーティションに関する推奨を検索するには、定数値 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。推奨を要求する対象である、アラート状態に入ったヘルス・インディケーター。値は、インクルード・ディレクトリーのヘッダー・ファイルである 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・パラメーター 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・パラメーター iObjType が DB2HEALTH_OBJTYPE_TS_CONTAINER、DB2HEALTH_OBJTYPE_TABLESPACE、あるいは 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。

db2GetRecommendationsFree - db2GetRecommendations API によって割り振られたメモリーの解放

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 インクルード・ファイル

```
db2ApiDf.h
```

API とデータ構造構文

```
SQL_API_RC SQL_API_FN
  db2GetRecommendationsFree (
   db2Uint32 versionNumber,
   void * pParmStruct,
   struct sqlca * pSqlca);
```

db2GetRecommendationsFree 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・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 構造を指すポインター。

db2GetSnapshot - データベース・マネージャー操作状況のスナップショットの取得

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。

- sysadm
- sysctrl
- sysmaint
- sysmon

必要な接続

インスタンス。インスタンス接続が存在しない場合は、デフォルトのインスタンス接続が作成されます。

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 インクルード・ファイル

```
db2ApiDf.h
```

API とデータ構造構文

```
SQL_API_RC SQL_API_FN
  db2GetSnapshot (
   db2Uint32 versionNumber,
   void * pParmStruct,
   struct sqlca * pSqlca);

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

SQL_API_RC SQL_API_FN
  db2gGetSnapshot (
   db2Uint32 versionNumber,
```

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

typedef SQL_STRUCTURE db2gGetSnapshotData
{
   void *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・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 以降を指定します。この構造の別のバージョンを使用する場合には、 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。指定したバージョン番号に対応する db2GetSnapshotData 構造のバージョンを必ず使用してください。

pParmStruct

入出力。db2GetSnapshotData 構造を指すポインター。

pSqlca 出力。 sqlca 構造を指すポインター。

db2GetSnapshotData データ構造パラメーター

piSqlmaData

入力。ユーザー割り当ての sqlma (モニター域) 構造、あるいは、db2AddSnapshotRequest API によって作成され、戻された要求データ構造、″poRequestData″ を指すポインター。この構造では、収集されるスナップショット・データのタイプ (複数の場合あり) が指定されます。 sqlma 構造を指すポインターを使用する場合、versionNumber パラメーターで db2GetSnapshot API に渡すバージョンは、db2Version900 よりも前 (db2Version810、db2Version822 など) にする必要があります。db2AddSnapshotRequest API によって poRequestData パラメーターで戻される要求データ構造を指すポインターを使用する場合、値 db2Version900 を db2GetSnapshot API の versionNumber パラメーターで渡す必要があります。

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、以前のリモート・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。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できるデータのみを戻します。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。

- SQLM_DBMON_VERSION1
- SQLM_DBMON_VERSION2
- SQLM_DBMON_VERSION5
- SQLM_DBMON_VERSION5_2
- SQLM_DBMON_VERSION6
- SQLM_DBMON_VERSION7
- SQLM_DBMON_VERSION8
- SQLM_DBMON_VERSION9
- SQLM_DBMON_VERSION9_5

注: SQLM_DBMON_VERSION5_2 以前の定数は使用すべきではなく、今後の DB2 リリースでは削除される可能性があります。

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 API 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、定数値 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 パラメーターが SQLM_DBMON_VERSION7 以降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、値 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修飾子。有効な値は以下のとおりです (include ディレクトリーの sqlmon ヘッダー・ファイルで定義される)。

- 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。

db2GetSnapshotSize - db2GetSnapshot API に必要な出力バッファー・サイズの見積もり

db2GetSnapshot API 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。

- sysadm
- sysctrl
- sysmaint
- sysmon

必要な接続

インスタンス。インスタンス接続が存在しない場合は、デフォルトのインスタンス接続が作成されます。

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接続が確立されます。

API インクルード・ファイル

db2ApiDf.h

API とデータ構造構文

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

typedef SQL_STRUCTURE db2GetSnapshotSizeData
{
   void *piSqlmaData;
   sqluint32 *poBufferSize;
   db2Uint32 iVersion;
   db2int32 iNodeNumber;
   db2Uint32 iSnapshotClass;
} db2GetSnapshotSizeData;

SQL_API_RC SQL_API_FN
  db2gGetSnapshotSize (
   db2Uint32 versionNumber,
   void * pParmStruct,
   struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2gGetSnapshotSizeData
{
   void *piSqlmaData;
   sqluint32 *poBufferSize;
   db2Uint32 iVersion;
   db2int32 iNodeNumber;
   db2Uint32 iSnapshotClass;
} db2gGetSnapshotSizeData;
```

db2GetSnapshotSize 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・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 以降を指定します。この構造の別のバージョンを使用する場合には、 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。指定したバージョン番号に対応する db2GetSnapshotSizeStruct 構造のバージョン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 構造を指すポインター。

db2GetSnapshotSizeData データ構造パラメーター

piSqlmaData
入力。ユーザー割り当ての sqlma (モニター域) 構造、あるいは、db2AddSnapshotRequest API によって作成され、戻された要求データ構造、″poRequestData″ を指すポインター。この構造では、収集されるスナップショット・データのタイプ (複数の場合あり) が指定されます。 sqlma 構造を指すポインターを使用する場合、versionNumber パラメーターで db2GetSnapshotSize API に渡すバージョンは、db2Version900 よりも前

(db2Version810、db2Version822 など) にする必要があります。
db2AddSnapshotRequest API によって poRequestData パラメーターで戻される要求データ構造を指すポインターを使用する場合、値 db2Version900 を db2GetSnapshotSize API の versionNumber パラメーターで渡す必要があります。

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。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できる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
- SQLM_DBMON_VERSION9
- SQLM_DBMON_VERSION9_5

注: SQLM_DBMON_VERSION5_2 以前の定数は使用すべきではなく、今後の DB2 リリースでは削除される可能性があります。

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 以降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、値 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修飾子。有効な値は以下のとおりです (include ディレクトリーの sqlmon ヘッダー・ファイルで定義される)。

- 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 API 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 (例えば、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 (例えば、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。

db2GetSyncSession - サテライト同期化セッション ID の取得

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 インクルード・ファイル

```
db2ApiDf.h
```

API とデータ構造構文

```
SQL_API_RC SQL_API_FN
  db2GetSyncSession (
   db2Uint32 versionNumber,
   void * pParmStruct,
   struct sqlca * pSqlca);

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

db2GetSyncSession 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 構造を指すポインター。

db2GetSyncSessionStruct データ構造パラメーター

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。

---

db2HADRStart - 高可用性災害時リカバリー (HADR) 操作の開始

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。

- 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 インクルード・ファイル

```
db2ApiDf.h
```

API とデータ構造構文

```
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;

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;
```

db2HADRStart 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・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 構造を指すポインター。

db2HADRStartStruct データ構造パラメーター

piDbAlias
入力。データベース別名を指すポインター。

piUserName
入力。コマンドを実行するときのユーザー名を指すポインター。

piPassword
入力。パスワードを含むストリングを指すポインター。

iDbRole
入力。どの HADR データベース役割を、指定のデータベース上で開始するかを指定します。有効な値は以下のとおりです。

DB2HADR_DB_ROLE_PRIMARY
1 次役割のデータベースに対する HADR 操作を開始します。

DB2HADR_DB_ROLE_STANDBY
スタンバイ役割のデータベースに対する HADR 操作を開始します。

iByForce
入力。iDbRole パラメーターが DB2HADR_DB_ROLE_STANDBY に設定される場合、この引数は無視されます。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。

db2gHADRStartStruct データ構造固有パラメーター

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。

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

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

db2HADRStop - 高可用性災害時リカバリー (HADR) 操作の停止

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。

- 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 インクルード・ファイル

db2ApiDf.h

API とデータ構造構文

```
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;

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;
```

db2HADRStop 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・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 構造を指すポインター。

db2HADRStopStruct データ構造パラメーター

piDbAlias
入力。データベース別名を指すポインター。

piUserName
入力。コマンドを実行するときのユーザー名を指すポインター。

piPassword
入力。パスワードを含むストリングを指すポインター。

db2gHADRStopStruct データ構造固有パラメーター

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。

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

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

db2HADRTakeover - データベースへの高可用性災害時リカバリー (HADR) 1 次データベースとしてのテークオーバーの指示

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。この API は、スタンバイ・データベースについてのみ呼び出せます。

許可

以下のいずれか。

- *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 インクルード・ファイル

```
db2ApiDf.h
```

API とデータ構造構文

```
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;

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;
```

db2HADRTakeover 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・レベルを指定します。

pParmStruct
入力。db2HADRTakeoverStruct 構造を指すポインター。

pSqlca 出力。 sqlca 構造を指すポインター。

db2HADRTakeoverStruct データ構造パラメーター

piDbAlias
入力。データベース別名を指すポインター。

piUserName
入力。コマンドを実行するときのユーザー名を指すポインター。

piPassword
入力。パスワードを含むストリングを指すポインター。

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。スタンバイ・データベースがリ

モート・キャッチアップ・ペンディング状態、あるいはピア状態にある時には、テークオーバーを強制実行する必要があります。

DB2HADR_FORCE_PEERWINDOW
このオプションが指定された場合、コマンドが正常に実行された後、ピア・ウィンドウ期間が終了する前に 1 次データベースがダウンする場合 (データベース構成パラメーター **HADR_PEER_WINDOW** をゼロ以外の値に設定) でも、コミット済みトランザクションの損失はありません。ピア・ウィンドウ期間が満了する前に 1 次データベースをダウンさせない場合、結果として *分割ブレイン* になります。 HADR ペアがピア状態または切断ピア状態でない場合 (ピア・ウィンドウが有効期限切れの場合) に実行されると、エラーが戻されます。

注: 1 次データベース・クロックとスタンバイ・データベース・クロックが相互に 5 秒以内の精度で同期していない場合、DB2HADR_FORCE_PEERWINDOW パラメーターが指定されたテークオーバー操作は誤った動作をする可能性があります。つまり、操作が失敗するはずの場合に成功したり、成功するはずの場合に失敗したりします。時刻同期サービス (例えば NTP) を使用して、クロックを同じソースに同期させる必要があります。

```
/* Values for iByForce                                    */
#define DB2HADR_NO_FORCE          0    /* Do not perform START or   */
                                       /* TAKEOVER HADR operation   */
                                       /* by force                  */
#define DB2HADR_FORCE             1    /* Do perform START or       */
                                       /* TAKEOVER HADR operation   */
                                       /* by force                  */
#define DB2HADR_FORCE_PEERWINDOW  2    /* Perform TAKEOVER HADR     */
                                       /* operation by force inside */
                                       /* the Peer Window only      */
```

db2gHADRTakeoverStruct データ構造固有パラメーター

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。

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

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

db2HistoryCloseScan - 履歴ファイルのスキャンの終了

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 API 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 API を呼び出す必要はありません。

API インクルード・ファイル

db2ApiDf.h

API とデータ構造構文

```
SQL_API_RC SQL_API_FN
  db2HistoryCloseScan (
       db2Uint32 versionNumber,
       void * piHandle,
       struct sqlca * pSqlca);

SQL_API_RC SQL_API_FN
  db2gHistoryCloseScan (
       db2Uint32 versionNumber,
       void * piHandle,
       struct sqlca * pSqlca);
```

db2HistoryCloseScan API パラメーター

versionNumber
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 API 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 構造を指すポインター。

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 API』を参照してください。

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。

db2HistoryGetEntry - 履歴ファイルの次の項目の取得

履歴ファイルの次項目を入手します。この API は、db2HistoryOpenScan API 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 インクルード・ファイル

```
db2ApiDf.h
```

API とデータ構造構文

```
SQL_API_RC SQL_API_FN
  db2HistoryGetEntry (
       db2Uint32 versionNumber,
       void * pParmStruct,
       struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2HistoryGetEntryStruct
{
   struct db2HistoryData *pioHistData;
   db2Uint16 iHandle;
   db2Uint16 iCallerAction;
} db2HistoryGetEntryStruct;

SQL_API_RC SQL_API_FN
  db2gHistoryGetEntry (
       db2Uint32 versionNumber,
       void * pParmStruct,
       struct sqlca * pSqlca);
```

db2HistoryGetEntry 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。db2HistoryGetEntryStruct 構造を指すポインター。

pSqlca 出力。 sqlca 構造を指すポインター。

db2HistoryGetEntryStruct データ構造パラメーター

pioHistData
入力。db2HistData 構造を指すポインター。

iHandle
入力。db2HistoryOpenScan API 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。有効な値は以下のとおりです (インクルード・ディレクトリーの db2ApiDf ヘッダー・ファイルで定義される)。

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。

使用上の注意

戻されるレコードは、db2HistoryOpenScan API への呼び出しで指定した値を使用して選択されたものです。

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

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
以降、3 番目、4 番目 ... と続きます

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)

---

db2HistoryOpenScan - 履歴ファイルのスキャンの開始

この API は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 API を呼び出してください。

API インクルード・ファイル

```
db2ApiDf.h
```

API とデータ構造構文

```
SQL_API_RC SQL_API_FN
  db2HistoryOpenScan (
       db2Uint32 versionNumber,
       void * pParmStruct,
       struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2HistoryOpenStruct
{
   char *piDatabaseAlias;
   char *piTimestamp;
   char *piObjectName;
   db2Uint32 oNumRows;
   db2Uint32 oMaxTbspaces;
   db2Uint16 iCallerAction;
   db2Uint16 oHandle;
} db2HistoryOpenStruct;

SQL_API_RC SQL_API_FN
  db2gHistoryOpenScan (
       db2Uint32 versionNumber,
       void * pParmStruct,
       struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2gHistoryOpenStruct
{
   char *piDatabaseAlias;
   char *piTimestamp;
   char *piObjectName;
   db2Uint32 iAliasLen;
   db2Uint32 iTimestampLen;
   db2Uint32 iObjectNameLen;
   db2Uint32 oNumRows;
   db2Uint32 oMaxTbspaces;
   db2Uint16 iCallerAction;
   db2Uint16 oHandle;
} db2gHistoryOpenStruct;
```

db2HistoryOpenScan 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
入力または出力。db2HistoryOpenStruct データ構造を指すポインター。

pSqlca 出力。 sqlca 構造を指すポインター。

db2HistoryOpenStruct データ構造パラメーター

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 呼び出しからの戻り時に、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。有効な値は以下のとおりです (インクルード・ディレクトリーの db2ApiDf ヘッダー・ファイルで定義される)。

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、この項目がフェッチされた直後に、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、この項目がフェッチされた直後に、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 API で使用されます。

db2gHistoryOpenStruct データ構造固有パラメーター

iAliasLen
入力。データベース別名ストリングの長さ (バイト単位) を指定します。

iTimestampLen
入力。タイム・スタンプ・ストリングの長さ (バイト単位) を指定します。

iObjectNameLen
入力。オブジェクト名ストリングの長さ (バイト単位) を指定します。

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. パラメーター値 oNumRows を戻す db2HistoryOpenScan API を呼び出す。
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. db2HistoryData 構造の iNumTablespaces フィールドを n に設定する。
4. ループの中で、以下を実行する。
   - db2HistoryGetEntry API を呼び出して履歴ファイルからフェッチを行う。
   - db2HistoryGetEntry API が SQL_RC_OK という SQLCODE 値を戻す場合、db2HistoryData 構造の oNumTablespaces フィールドを使用して、戻される表スペース項目の数を判別する。
   - db2HistoryGetEntry API が SQLUH_SQLUHINFO_VARS_WARNING という SQLCODE 値を戻す場合は、 DB2 が戻そうとする表スペースの中に、十分なスペースが割り当てられていないものがある、ということを意味する。db2HistoryData 構造を解放して再割り当てを行い、oDB2UsedTablespace 表スペース項目に十分なスペースが割り当てられるようにし、iDB2NumTablespace を oDB2UsedTablespace に設定する。
   - db2HistoryGetEntry API が SQLE_RC_NOMORE という SQLCODE 値を戻す場合は、すべての履歴ファイル項目が既に取得された。
   - 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. すべての情報のフェッチが終了したら、db2HistoryCloseScan API 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。

n 個の oTablespace 項目を持つ db2HistoryData 構造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 ヘッダー・ファイルで定義された) マクロ SQLUHINFOSIZE が用意されています。

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
リストされた履歴ファイルを持つデータベースの別名。

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

db2HistoryUpdate - 履歴ファイル項目の更新

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。

- *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 インクルード・ファイル

```
db2ApiDf.h
```

API とデータ構造構文

```
SQL_API_RC SQL_API_FN
  db2HistoryUpdate (
       db2Uint32 versionNumber,
       void * pParmStruct,
       struct sqlca * pSqlca);

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

typedef SQL_STRUCTURE db2HistoryEID
{
   SQL_PDB_NODE_TYPE ioNode;
```

```
   db2Uint32 ioHID;
} db2HistoryEID;

SQL_API_RC SQL_API_FN
  db2gHistoryUpdate (
       db2Uint32 versionNumber,
       void * pParmStruct,
       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;
```

db2HistoryUpdate 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。db2HistoryUpdateStruct 構造を指すポインター。

pSqlca 出力。 sqlca 構造を指すポインター。

db2HistoryUpdateStruct データ構造パラメーター

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。有効な装置タイプは次のとおりです。

D ディスク

K ディスケット

T テープ

F スナップショット・バックアップ

A Tivoli ストレージ・マネージャー

U ユーザー出口

P パイプ

N Null 装置

X XBSA

Q SQL ステートメント

O その他

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。有効な値は以下のとおりです。

A アクティブ。バックアップ・イメージはアクティブ・ログ・チェーン上にあります。ほとんどの項目はアクティブです。

I 非アクティブ。現行のログ・シーケンス (現行のログ・チェーンとも言う) に対応しなくなったバックアップ・イメージには、非アクティブのフラグが立てられます。

E 期限切れ。アクティブ・イメージの数が NUM_DB_BACKUPS を超えたために不要になったバックアップ・イメージは、期限切れのフラグが立てられます。

D 削除済み。リカバリーに使用可能でなくなったバックアップ・イメージは、削除済みとしてマークされることになります。

X do_not_delete。 PRUNE HISTORY コマンドの呼び出し、PRUNE HISTORY を指定した ADMIN_CMD プロシージャーの実行、db2Prune API の呼び出し、または自動リカバリー履歴ファイル・プルーニングの実行によって、do not delete マークの付いたリカバリー履歴項目は削除または整理されません。 do_not_delete 状況を使用することによって、重要なリカバリー・ファイル項目が整理されないよう、また、それらに関連するリカバリー・オブジェクトが削除されないよう保護できます。

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。

db2HistoryEID データ構造パラメーター

ioNode 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。

ノード番号を示します。

ioHID 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。

ローカル履歴ファイルの項目 ID を示します。

db2gHistoryUpdateStruct データ構造固有パラメーター

iNewLocationLen
入力。piNewLocation パラメーターの長さ (バイト単位) を指定します。

iNewDeviceLen
入力。piNewDeviceType パラメーターの長さ (バイト単位) を指定します。

iNewCommentLen
入力。piNewComment パラメーターの長さ (バイト単位) を指定します。

iNewStatusLen
入力。piNewStatus パラメーターの長さ (バイト単位) を指定します。

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から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、それらのイメージの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。

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)

db2Import - 表、階層、ニックネーム、ビューへのデータのインポート

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、ニックネーム、またはビューにデータを挿入します。ロード・ユーティリティーのほうが、この関数より高速です。ただし、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や、ニックネームへのロードをサポートしていません。

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、ビュー、またはニックネームに対する 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、ビュー、またはニックネームに対する 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 CREATEIN 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。暗黙接続が可能な場合には、デフォルト・データベースへの接続が確立されます。

API インクルード・ファイル

db2ApiDf.h

API とデータ構造構文

```
SQL_API_RC SQL_API_FN
  db2Import (
   db2Uint32 versionNumber,
   void * pParmStruct,
   struct sqlca * pSqlca);

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;
   struct sqllob *piLongActionString;
} db2ImportStruct;

typedef SQL_STRUCTURE db2ImportIn
{
   db2Uint64 iRowcount;
   db2Uint64 iRestartcount;
   db2Uint64 iSkipcount;
   db2int32 *piCommitcount;
   db2Uint32 iWarningcount;
   db2Uint16 iNoTimeout;
   db2Uint16 iAccessLevel;
   db2Uint16 *piXmlParse;
   struct db2DMUXmlValidate *piXmlValidate;
} db2ImportIn;

typedef SQL_STRUCTURE db2ImportOut
{
   db2Uint64 oRowsRead;
   db2Uint64 oRowsSkipped;
   db2Uint64 oRowsInserted;
   db2Uint64 oRowsUpdated;
   db2Uint64 oRowsRejected;
   db2Uint64 oRowsCommitted;
} db2ImportOut;

typedef SQL_STRUCTURE db2DMUXmlMapSchema
{
   struct db2Char                       iMapFromSchema;
   struct db2Char                       iMapToSchema;
} db2DMUXmlMapSchema;

typedef SQL_STRUCTURE db2DMUXmlValidateXds
{
```

```
   struct db2Char *piDefaultSchema;
   db2Uint32 iNumIgnoreSchemas;
   struct db2Char *piIgnoreSchemas;
   db2Uint32 iNumMapSchemas;
   struct db2DMUXmlMapSchema *piMapSchemas;
} db2DMUXmlValidateXds;

typedef SQL_STRUCTURE db2DMUXmlValidateSchema
{
   struct db2Char *piSchema;
} db2DMUXmlValidateSchema;

typedef SQL_STRUCTURE db2DMUXmlValidate
{
   db2Uint16 iUsing;
   struct db2DMUXmlValidateXds *piXdsArgs;
   struct db2DMUXmlValidateSchema *piSchemaArgs;
} db2DMUXmlValidate;

SQL_API_RC SQL_API_FN
  db2gImport (
   db2Uint32 versionNumber,
   void * pParmStruct,
   struct sqlca * pSqlca);

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;
   struct sqllob *piLongActionString;
} db2gImportStruct;

typedef SQL_STRUCTURE db2gImportIn
{
   db2Uint64 iRowcount;
   db2Uint64 iRestartcount;
   db2Uint64 iSkipcount;
   db2int32 *piCommitcount;
   db2Uint32 iWarningcount;
   db2Uint16 iNoTimeout;
   db2Uint16 iAccessLevel;
   db2Uint16 *piXmlParse;
   struct db2DMUXmlValidate *piXmlValidate;
} db2gImportIn;

typedef SQL_STRUCTURE db2gImportOut
{
   db2Uint64 oRowsRead;
   db2Uint64 oRowsSkipped;
   db2Uint64 oRowsInserted;
   db2Uint64 oRowsUpdated;
   db2Uint64 oRowsRejected;
   db2Uint64 oRowsCommitted;
} db2gImportOut;
```

db2Import 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
入出力。db2ImportStruct 構造を指すポインター。

pSqlca 出力。 sqlca 構造を指すポインター。

db2ImportStruct データ構造パラメーター

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 フィールドを SQLU_LOCAL_MEDIA に設定された sqlu_media_list、および LOB ファイルが置かれているクライアント上のパスをリストするその sqlu_media_entry 構造を指すポインター。ニックネームにインポートするときには、このパラメーターは無効です。

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収めた 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ペア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
非推奨。**piLongActionString** に換わりました。

piLongActionString
入力。4 バイト長のフィールドが含まれる sqllob 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 | 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]}
```

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。

注: 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。
ニックネームにインポートするときには、このパラメーターは無効です。

CREATE

注: CREATE パラメーターは推奨されておらず、今後のリリースで除去される可能性があります。さらに詳しくは、『IMPORT コマンドの推奨されなくなったオプション CREATE および REPLACE_CREATE』を参照してください。

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。 DB2 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。指定された表が既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。
ニックネームにインポートするときには、このパラメーターは無効です。

REPLACE_CREATE

注: REPLACE_CREATE パラメーターは推奨されておらず、今後のリリースで除去される可能性があります。さらに詳しくは、『IMPORT コマンドの推奨されなくなったオプション CREATE および REPLACE_CREATE』を参照してください。

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、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。

注: 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。
ニックネームにインポートするときには、このパラメーターは無効です。

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。以前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 つ以上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。

tname および tcolumn-list パラメーターは、 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。

ニックネームにインポートするときには、このパラメーターは無効です。

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)。同じ表または別のデータベース・マネージャー表へ再インポートできるよう、表からデータをエクスポートする場合に推奨される方式です。

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。 ニックネームにインポートするときには、WSF ファイル・タイプはサポートされません。

piFileTypeMod
入力。2 バイトの長さフィールドと、 1 つ以上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。以下の『インポート・ユーティリティーのファイル・タイプ修飾子』の関連リンク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。ファイルが既に存在する場合は、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 (例えば、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 (例えば、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には、データ・ファイル

内の識別された列に 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。

piXmlPathList
入力。media_type フィールドを SQLU_LOCAL_MEDIA に設定された sqlu_media_list、および XML ファイルが置かれているクライアント上のパスをリストするその sqlu_media_entry 構造を指すポインター。

db2lmportIn データ構造パラメーター

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。

iRestartcount
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。機能上は **iSkipcount** パラメーターと同等です。 **iRestartcount** と **iSkipcount** パラメーターは相互に排他的です。

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。機能上は **iRestartcount** と同等です。

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。このオプションのほうが、 **locktimeout** データベース構成パラメーターより優先されます。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。

最初の行の挿入時には、ターゲット表に意図的排他 (IX) ロックがかけられます。これで、表データへの同時の読み取りおよび書き出しアクセスが可能になります。オンライン・モードには、**REPLACE**、**CREATE**、または **REPLACE_CREATE** インポート・オプションとの互換性はありません。オンライン・モードとバッファー挿入との連携はサポートされません。インポート操作によって挿入後のデータが定期的にコミットされるので、表ロックへのロック・エスカレーションが削減され、アクティブなログ・スペースが使い果たされることはなくなります。このようなコミットは、**piCommitCount** パラメーターを使わなくても実行されます。各コミットごとに、インポートでは IX 表ロックが外されるので、コミットの完了後に再びロックの設定が試みられます。ニックネームにインポートするときにはこのパラメーターが必要で、有効な数値を使って **piCommitCount** パラメーターを指定する必要があります (AUTOMATIC は有効なオプションとは見なされません)。

piXmlParse

入力。XML 文書に対して行われる必要のある解析のタイプ。 include ディレクトリーに入っている db2ApiDf ヘッダー・ファイル内の有効値は、次のとおりです。

DB2DMU_XMLPARSE_PRESERVE_WS
空白が保持されます。

DB2DMU_XMLPARSE_STRIP_WS
空白が取り除かれます。

piXmlValidate

入力。db2DMUXmlValidate 構造を指すポインター。 XML 文書の XML スキーマ検証を行う必要があることを示します。

db2ImportOut データ構造パラメーター

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 (主キーの値が既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。

db2DMUXmlMapSchema データ構造パラメーター

iMapFromSchema
入力。マップ元の XML スキーマの SQL ID。

iMapToSchema
入力。マップ先の XML スキーマの SQL ID。

db2DMUXmlValidateXds データ構造パラメーター

piDefaultSchema
入力。XDS に SCH 属性が入っていない場合に検証で使用する必要のある XML スキーマの SQL ID。

iNumIgnoreSchemas
入力。XML スキーマが、XDS 内の SCH 属性によって参照される場合に、XML スキーマ検証中に無視される XML スキーマの数。

piIgnoreSchemas
入力。XML スキーマが、XDS 内の SCH 属性によって参照される場合に、XML スキーマ検証中に無視される XML スキーマのリスト。

iNumMapSchemas
入力。XML スキーマ検証中にマップされる XML スキーマの数。スキーマ・マップのペア中の最初のスキーマは、XDS 内の SCH 属性によって参照されるスキーマを表します。ペア中の 2 番目のスキーマは、スキーマ検証の実行で使用する必要のあるスキーマを表します。

piMapSchemas
入力。XML スキーマ・ペアのリスト。各ペアは、1 つのスキーマから別のものへのマッピングを表します。ペア中の最初のスキーマは、XDS 内の SCH 属性によって参照されるスキーマを表します。ペア中の 2 番目のスキーマは、スキーマ検証の実行で使用する必要のあるスキーマを表します。

db2DMUXmlValidateSchema データ構造パラメーター

piSchema
入力。使用する XML スキーマの SQL ID。

db2DMUXmlValidate データ構造パラメーター

iUsing 入力。XML スキーマ検証の実行で何を使用するかの指定。 include ディレクトリーに入っている db2ApiDf ヘッダー・ファイル内の有効値は、次のとおりです。

- **DB2DMU_XMLVAL_XDS**
  検証は、XDS に従って行う必要があります。これは、CLP ″**XMLVALIDATE USING XDS**″ 節に対応します。

- **DB2DMU_XMLVAL_SCHEMA**
  検証は、指定されたスキーマに従って行う必要があります。これは、CLP ″**XMLVALIDATE USING SCHEMA**″ 節に対応します。
- **DB2DMU_XMLVAL_SCHEMALOC_HINTS**
  検証は、XML 文書内に入っている schemaLocation ヒントに従って行う必要があります。これは、″**XMLVALIDATE USING SCHEMALOCATION HINTS**″ 節に対応します。

piXdsArgs
入力。CLP ″**XMLVALIDATE USING XDS**″ 節に対応する引数を表す db2DMUXmlValidateXds 構造を指すポインター。

このパラメーターが適用されるのは、同じ構造内の **iUsing** パラメーターを DB2DMU_XMLVAL_XDS に設定する場合のみです。

piSchemaArgs
入力。CLP ″**XMLVALIDATE USING SCHEMA**″ 節に対応する引数を表す db2DMUXmlValidateSchema 構造を指すポインター。

このパラメーターが適用されるのは、同じ構造内の **iUsing** パラメーターを DB2DMU_XMLVAL_SCHEMA に設定する場合のみです。

db2gImportStruct データ構造固有のパラメーター

iDataFileNameLen
入力。**piDataFileName** パラメーターの長さ (バイト単位) を指定します。

iFileTypeLen
入力。**piFileType** パラメーターの長さ (バイト単位) を指定します。

iMsgFileNameLen
入力。**piMsgFileName** パラメーターの長さ (バイト単位) を指定します。

使用上の注意

インポート操作を開始する前に、以下の 2 つの方法のいずれかで、すべての表操作を完了し、すべてのロックを解放する必要があります。

- WITH HOLD 節を使って定義したすべてのオープン・カーソルをクローズし、COMMIT ステートメントを実行して、データの変更をコミットします。
- ROLLBACK ステートメントを実行して、データ変更をロールバックします。

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・メッセージがメッセージ・ファイルに書き込まれ、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やり直すか、または正常にインポートされる行の数に設定した **iRestartcount** パラメーターを指定して **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 (例えば、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、参照整合性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。 (主キー定義は、データが前に SELECT * を使ってエクスポートされた場合、保存されます。)

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、常に保持されます。

piDataDescriptor でデフォルト以外の値を使用したり、 **piLongActionString** 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 を使用して、DB2 for OS/390、DB2 for VM and VSE、および DB2 for OS/400® などの DRDA サーバーからデータをインポートできます。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 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。

フェデレーテッドに関する考慮事項

db2Import API、および **INSERT**、**UPDATE**、または **INSERT_UPDATE** パラメーターを使用するときには、関係するニックネームに対する CONTROL 特権があることを確認してください。インポート操作で使用したいニックネームが既に存在することを確認する必要があります。

db2Inspect - データベースの構造上の整合性の検査

データベースの構造上の整合性を検査し、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。パーティション・データベース環境では、`db2nodes.cfg` に定義されている論理データベース・パーティションすべてのコレクションです。パーティション化された表では、データベースおよび表スペース・レベルの検査の有効範囲に、個々のデータ・パーティションとパーティション化されていない索引が含まれます。パーティション化された表の表レベルの検査では、単一のデータ・パーティションや索引ではなく、すべてのデータ・パーティションと、表中の索引が検査されます。

許可

以下のいずれか。

- *sysadm*
- *sysctrl*
- *sysmaint*
- *dbadm*
- 表に対する CONTROL 特権

必要な接続

データベース

API インクルード・ファイル

db2ApiDf.h

API とデータ構造構文

```
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 iFirstPage;
   db2Uint32 iNumberOfPages;
   db2Uint32 iFormatType;
   db2Uint32 iOptions;
   db2Uint32 iBeginCheckOption;
   db2int32 iLimitErrorReported;
   db2Uint16 iObjectErrorState;
   db2Uint16 iCatalogToTablespace;
   db2Uint16 iKeepResultfile;
   db2Uint16 iAllNodeFlag;
   db2Uint16 iNumNodes;
   db2Uint16 iLevelObjectData;
   db2Uint16 iLevelObjectIndex;
   db2Uint16 iLevelObjectLong;
   db2Uint16 iLevelObjectLOB;
   db2Uint16 iLevelObjectBlkMap;
   db2Uint16 iLevelExtentMap;
   db2Uint16 iLevelObjectXML;
   db2Uint32 iLevelCrossObject;
} db2InspectStruct;

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 iFirstPage;
   db2Uint32 iNumberOfPages;
   db2Uint32 iFormatType;
   db2Uint32 iOptions;
   db2Uint32 iBeginCheckOption;
   db2int32 iLimitErrorReported;
   db2Uint16 iObjectErrorState;
   db2Uint16 iCatalogToTablespace;
   db2Uint16 iKeepResultfile;
   db2Uint16 iAllNodeFlag;
   db2Uint16 iNumNodes;
   db2Uint16 iLevelObjectData;
   db2Uint16 iLevelObjectIndex;
   db2Uint16 iLevelObjectLong;
   db2Uint16 iLevelObjectLOB;
   db2Uint16 iLevelObjectBlkMap;
   db2Uint16 iLevelExtentMap;
   db2Uint16 iLevelObjectXML;
   db2Uint32 iLevelCrossObject;
} db2gInspectStruct;
```

db2Inspect 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・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。

db2InspectStruct データ構造パラメーター

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。

piNodeList
入力。操作を実行するデータベース・パーティション番号の配列を指すポインター。

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 (インクルード・ディレクトリーの db2ApiDf ヘッダー・ファイルで定義される)。

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。

DB2INSPECT_ACT_FORMAT_XML
XML オブジェクト・ページをフォーマットします。

DB2INSPECT_ACT_ROWCMPEST_TBL
表での行の圧縮の効果を見積もります。

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。

iLevelCrossObject フィールドの中で DB2INSPECT_LVL_XOBJ_INXDAT_RID が使用されている場合、指定された限度値、または上記の DEFAULT あるいは ALL 値は、オンライン索引/データ整合性検査中に報告するエラーの数を表します (エラーのあるページの数ではない)。

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。

iLevelCrossObject フィールドの中で DB2INSPECT_LVL_XOBJ_INXDAT_RID が使用されている場合、索引またはデータのオブジェクトがエラー状態なら、このフィールドで指定されている DB2INSPECT_ERROR_STATE_ALL は無視され、オンライン索引/データ整合性検査は実行されません。

iKeepResultfile
入力。結果ファイルの保持を指定します。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持されます。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持されます。

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。これはデフォルト値です。

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 (インクルード・ディレクトリーの db2ApiDf ヘッダー・ファイルで定義される)。

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。

iLevelObjectXML
入力。XML 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 (インクルード・ディレクトリーの db2ApiDf ヘッダー・ファイルで定義される)。

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。

iLevelCrossObject
クロス・オブジェクト整合性検査のために使用されるビット・ベースのフィールド。有効な値は以下のとおりです。

DB2INSPECT_LVL_XOBJ_NONE
オンライン索引/データ整合性検査は実行されません (0x00000000)。

DB2INSPECT_LVL_XOBJ_INXDAT_RID
RID 索引に対して INDEXDATA 検査が有効にされ (0x00000001)、読み取りと書き込みの両方を可能にする IS 表ロックを伴って実行されます。

db2gInspectStruct データ構造固有パラメーター

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。

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 ステートメントを実行して変更をコミットまたはロールバックし、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既に存在する場合は、操作は処理されません。

db2Inspect API を呼び出す場合、db2InspectStruct の中で `iLevelCrossObject` に適切な値を指定する必要があります。 DB2INSPECT_LVL_XOBJ_NONE が使用されている場合、オンライン索引/データ整合性検査は実行されません。オンライン索引/データ整合性検査を有効にするには、`iLevelCrossObject` フィールドに DB2INSPECT_LVL_XOBJ_INXDAT_RID を指定する必要があります。

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。例外は、索引/データ整合性検査の間です。データ・オブジェクトが他の表スペース中に存在する可能性があり、その場合、索引オブジェクトが検査対象表スペースに含まれている限り、検査を実行することにはメリットがあります。パーティション表

の場合、それぞれの索引は別個の表スペースに存在する可能性があります。索引/データ検査を実行するメリットがあるのは、指定された表スペース内に存在する索引だけです。

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、および sysadm、sysmaint、sysctrl のいずれかの権限を持つグループまたはユーザーのみが、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。

- sysadm
- sysctrl

必要な接続

なし

API インクルード・ファイル

db2ApiDf.h

API とデータ構造構文

```
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;

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;
```

db2InstanceQuiesce 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・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。

db2InsQuiesceStruct データ構造パラメーター

piInstanceName
入力。インスタンス名。

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。

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。有効な値は以下のとおりです。

TRUE=1
アプリケーションを即座に強制クローズする。

FALSE=0
据え置き強制クローズ。アプリケーションの現行の作業単位が完了してから終了するように、アプリケーションは iTimeout パラメーターに指定された時間 (単位: 分) だけ待機します。 iTimeout パラメーターに指定された時間内 (単位: 分) に据え置き強制クローズが完了できなければ、静止の操作は失敗に終わります。

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ー start_stop_time によって指定された値が使用されます。

db2gInsQuiesceStruct データ構造固有パラメーター

iInstanceNameLen
入力。piInstanceName の長さ (バイト単位) を指定します。

iUserIdLen
入力。piUserID の長さ (バイト単位) を指定します。

iGroupIdLen
入力。piGroupId の長さ (バイト単位) を指定します。

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 に定義されたすべての論理データベース・パーティション・サーバーを集めたものが有効範囲です。

許可

以下のいずれか。

- sysadm
- sysctrl
- sysmaint

必要な接続

なし

API インクルード・ファイル

```
db2ApiDf.h
```

API とデータ構造構文

```
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;

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;
```

db2InstanceStart 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・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。

db2InstanceStartStruct データ構造パラメーター

iIsRemote
入力。定数整数値である TRUE または FALSE に設定された標識。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。

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。

db2DasCommData データ構造パラメーター

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。

piNodeOrHostName
入力。データベース・パーティションまたはホスト名。

piUserId
入力。ユーザー名。

piUserPw
入力。ユーザー・パスワード。

db2StartOptionsStruct データ構造パラメーター

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 の有効な値は以下のとおりです (include ディレクトリーの sqlenv ヘッダー・ファイルで定義される)。

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_RESTART_PARALLEL
並列実行のための 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。

iTblspaceType
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
SYSTEM TEMPORARY 表スペースを作成しません。

SQLE_TABLESPACES_LIKE_NODE
SYSTEM 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
SYSTEM 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。

iTblspaceNode
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。

db2QuiesceStartStruct データ構造パラメーター

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。

piQUsrName
入力。静止ユーザー名。

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。

db2gInstanceStStruct データ構造固有パラメーター

iRemoteInstLen
入力。piRemoteInstName の長さ (バイト単位) を指定します。

db2gDasCommData データ構造固有パラメーター

iNodeOrHostNameLen
入力。piNodeOrHostName の長さ (バイト単位) を指定します。

iUserIdLen
入力。piUserId の長さ (バイト単位) を指定します。

iUserPwLen
入力。piUserPw の長さ (バイト単位) を指定します。

db2gQuiesceStartStruct データ構造固有パラメーター

iQUsrNameLen
入力。piQusrName の長さ (バイト単位) を指定します。

iQGrpNameLen
入力。piQGrpName の長さ (バイト単位) を指定します。

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 に定義されたすべての論理データベース・パーティション・サーバーを集めたものが有効範囲です。

許可

以下のいずれか。

- sysadm
- sysctrl
- sysmaint

必要な接続

なし

API インクルード・ファイル

```
db2ApiDf.h
```

API とデータ構造構文

```
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;

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;
```

db2InstanceStop 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・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。

db2InstanceStopStruct データ構造パラメーター

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。

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。

db2DasCommData データ構造パラメーター

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。

piNodeOrHostName
入力。データベース・パーティションまたはホスト名。

piUserId
入力。ユーザー名。

piUserPw
入力。ユーザー・パスワード。

db2StopOptionsStruct データ構造パラメーター

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。

db2gInstanceStopStruct データ構造固有パラメーター

iRemoteInstLen
入力。piRemoteInstName の長さ (バイト単位) を指定します。

db2gDasCommData データ構造固有パラメーター

iNodeOrHostNameLen
入力。piNodeOrHostName の長さ (バイト単位) を指定します。

iUserIdLen
入力。piUserId の長さ (バイト単位) を指定します。

iUserPwLen
入力。piUserPw の長さ (バイト単位) を指定します。

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。

- sysadm
- sysctrl

必要な接続

なし

API インクルード・ファイル

```
db2ApiDf.h
```

API とデータ構造構文

```
SQL_API_RC SQL_API_FN
  db2InstanceUnquiesce (
            db2Uint32 versionNumber,
```

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

typedef SQL_STRUCTURE db2InsUnquiesceStruct
{
            char *piInstanceName;
} db2InsUnquiesceStruct;

SQL_API_RC SQL_API_FN
  db2gInstanceUnquiesce (
            db2Uint32 versionNumber,
            void * pParmStruct,
            struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2gInsUnquiesceStruct
{
            db2Uint32 iInstanceNameLen;
            char *piInstanceName;
} db2gInsUnquiesceStruct;
```

db2InstanceUnquiesce 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・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。

db2InsUnquiesceStruct データ構造パラメーター

piInstanceName
入力。インスタンス名。

db2gInsUnquiesceStruct データ構造固有パラメーター

iInstanceNameLen
入力。piInstanceName の長さ (バイト単位) を指定します。

db2LdapCatalogDatabase - LDAP サーバーへのデータベースの登録

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 インクルード・ファイル

db2ApiDf.h

API とデータ構造構文

```
SQL_API_RC SQL_API_FN
  db2LdapCatalogDatabase (
   db2Uint32 versionNumber,
   void * pParmStruct,
   struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2LdapCatalogDatabaseStruct
{
   char *piAlias;
   char *piDatabaseName;
   char *piComment;
   char *piNodeName;
   char *piGWNodeName;
   char *piParameters;
   char *piARLibrary;
   unsigned short                       iAuthentication;
   char *piDCEPrincipalName;
   char *piBindDN;
   char *piPassword;
} db2LdapCatalogDatabaseStruct;
```

db2LdapCatalogDatabase 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 構造を指すポインター。

db2LdapCatalogDatabaseStruct データ構造パラメーター

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 が指定されない場合、現行ログオン・ユーザーの信用証明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・サーバーが LDAP をサポートしない場合。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。

- アプリケーションが、データベースに接続するために異なる名前を使用する必要がある場合。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 の際に、データベース名が既に 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、管理者には次の対応が可能です。 -- 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。 -- 異なる別名を使用して、LDAP に新規のデータベースを登録する。

db2LdapCatalogNode - LDAP サーバーにあるノード名の別名の指定

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 インクルード・ファイル

```
db2ApiDf.h
```

API とデータ構造構文

```
SQL_API_RC SQL_API_FN
  db2LdapCatalogNode (
   db2Uint32 versionNumber,
   void * pParmStruct,
   struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2LdapCatalogNodeStruct
{
   char *piAlias;
   char *piNodeName;
   char *piBindDN;
   char *piPassword;
} db2LdapCatalogNodeStruct;
```

db2LdapCatalogNode 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 構造を指すポインター。

db2LdapCatalogNodeStruct データ構造パラメーター

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 が指定されない場合、現行ログオン・ユーザーの信用証明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。

db2LdapDeregister - LDAP サーバーからの DB2 サーバーおよびカタログされたデータベースの登録解除

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 インクルード・ファイル

```
db2ApiDf.h
```

API とデータ構造構文

```
SQL_API_RC SQL_API_FN
  db2LdapDeregister (
   db2Uint32 versionNumber,
   void * pParmStruct,
   struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2LdapDeregisterStruct
{
   char *piNodeName;
   char *piBindDN;
   char *piPassword;
} db2LdapDeregisterStruct;
```

db2LdapDeregister 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 構造を指すポインター。

db2LdapDeregisterStruct データ構造パラメーター

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 が指定されない場合、現行ログオン・ユーザーの信用証明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。

db2LdapRegister - DB2 サーバーの LDAP サーバーへの登録

LDAP (Lightweight Directory Access Protocol) に DB2 サーバーを登録します。

注: NetBIOS は、現在サポートされていません。 SNA、およびその API の APPC、APPN、および CPI-C も、サポートされなくなりました。それらのプロトコルを使用している場合は、TCP/IP などのサポートされているプロトコルを使用してノードとデータベースを再カタログしてください。それらのプロトコルに言及している箇所がある場合、それらは無視してください。

許可

なし

必要な接続

なし

API インクルード・ファイル

db2ApiDf.h

API とデータ構造構文

```
SQL_API_RC SQL_API_FN
  db2LdapRegister (
   db2Uint32 versionNumber,
   void * pParmStruct,
   struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2LdapRegisterStruct
{
   char *piNodeName;
   char *piComputer;
   char *piInstance;
   unsigned short                      iNodeType;
   unsigned short                      iOsType;
   db2LdapProtocolInfo iProtocol;
   char *piComment;
   char *piBindDN;
   char *piPassword;
} db2LdapRegisterStruct;

typedef SQL_STRUCTURE db2LdapProtocolInfo
{
   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;
```

db2LdapRegister 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 構造を指すポインター。

db2LdapRegisterStruct データ構造パラメーター

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 オペレーティング・システムでは、これは Windows コンピューター名です。 UNIX ベースのシステムの場合は、TCP/IP ホスト名です。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

iOsType
入力。サーバー・マシンのオペレーティング・システムのタイプを指定します。オペレーティング・システムのタイプが指定されない場合、ローカル・サーバーに対してはローカルのオペレーティング・システムのタイプが使用され、リモート・サーバーに対してはオペレーティング・システムのタイプは使用されません。

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 が指定されない場合、現行ログオン・ユーザーの信用証明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。

db2LdapProtocolInfo データ構造パラメーター

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_TCPIP
TCP/IPv4 または TCP/IPv6 サポート向け

SQL_PROTOCOL_TCPIP4
TCP/IPv4 サポート向け

SQL_PROTOCOL_TCPIP6
TCP/IPv6 サポート向け

SQL_PROTOCOL_SOCKS
セキュリティー SOCKS を利用した TCP/IP 向け

SQL_PROTOCOL_SOCKS4
セキュリティー SOCKS を利用した TCP/IPv4 向け

SQL_PROTOCOL_NPIPE
Windows Named PIPE 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。 IP アドレスとしては、IPv4 のアドレスも IPv6 のアドレスも使用できます。IP アドレスは、選択されたプロトコル・タイプに適合するものである必要があります。例えば、SQL_PROTOCOL_TCPIP4 が選択された場合、指定される IP アドレスは IPv4 アドレスにしなくてはなりません。

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 (デフォルト)
- 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。

db2LdapUncatalogDatabase - LDAP サーバーからのデータベース登録の解除

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 インクルード・ファイル

db2ApiDf.h

API とデータ構造構文

```
SQL_API_RC SQL_API_FN
  db2LdapUncatalogDatabase (
   db2Uint32 versionNumber,
   void * pParmStruct,
   struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2LdapUncatalogDatabaseStruct
{
   char *piAlias;
   char *piBindDN;
   char *piPassword;
} db2LdapUncatalogDatabaseStruct;
```

db2LdapUncatalogDatabase 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 構造を指すポインター。

db2LdapUncatalogDatabaseStruct データ構造パラメーター

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 が指定されない場合、現行ログオン・ユーザーの信用証明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。

db2LdapUncatalogNode - LDAP サーバーからのノード名に対応する別名の削除

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 インクルード・ファイル

```
db2ApiDf.h
```

API とデータ構造構文

```
SQL_API_RC SQL_API_FN
  db2LdapUncatalogNode (
   db2Uint32 versionNumber,
   void * pParmStruct,
   struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2LdapUncatalogNodeStruct
{
   char *piAlias;
   char *piBindDN;
   char *piPassword;
} db2LdapUncatalogNodeStruct;
```

db2LdapUncatalogNode 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 構造を指すポインター。

db2LdapUncatalogNodeStruct データ構造パラメーター

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 が指定されない場合、現行ログオン・ユーザーの信用証明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。

db2LdapUpdate - LDAP サーバー上の DB2 サーバーの属性の更新

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。

注: NetBIOS は、現在サポートされていません。 SNA、およびその API の APPC、APPN、および CPI-C も、サポートされなくなりました。それらのプロトコルを使用している場合は、TCP/IP などのサポートされているプロトコルを使用してノードとデータベースを再カタログしてください。それらのプロトコルに言及している箇所がある場合、それらは無視してください。

許可

なし

必要な接続

なし

API インクルード・ファイル

```
db2ApiDf.h
```

API とデータ構造構文

```
SQL_API_RC SQL_API_FN
  db2LdapUpdate (
   db2Uint32 versionNumber,
   void * pParmStruct,
   struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2LdapUpdateStruct
{
   char *piNodeName;
   char *piComment;
   unsigned short                      iNodeType;
   db2LdapProtocolInfo iProtocol;
   char *piBindDN;
   char *piPassword;
} db2LdapUpdateStruct;

typedef SQL_STRUCTURE db2LdapProtocolInfo
{
   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;
```

db2LdapUpdate 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 構造を指すポインター。

db2LdapUpdateStruct データ構造パラメーター

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 が指定されない場合、現行ログオン・ユーザーの信用証明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。

db2LdapProtocolInfo データ構造パラメーター

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_TCPIP
TCP/IPv4 または TCP/IPv6 サポート向け

SQL_PROTOCOL_TCPIP4
TCP/IPv4 サポート向け

SQL_PROTOCOL_TCPIP6
TCP/IPv6 サポート向け

SQL_PROTOCOL_SOCKS
セキュリティー SOCKS を利用した TCP/IP 向け

SQL_PROTOCOL_SOCKS4
セキュリティー SOCKS を利用した TCP/IPv4 向け

SQL_PROTOCOL_NPIPE
Windows Named PIPE 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。 IP アドレスとしては、IPv4 のアドレスも IPv6 のアドレスも使用できます。IP アドレスは、選択されたプロトコル・タイプに適合するものである必要があります。例えば、SQL_PROTOCOL_TCPIP4 が選択された場合、指定される IP アドレスは IPv4 アドレスにしなくてはなりません。

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 (デフォルト)
- 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。

db2LdapUpdateAlternateServerForDB - LDAP サーバーでのデータベースに対応する代替サーバーの更新

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 インクルード・ファイル

```
db2ApiDf.h
```

API とデータ構造構文

```
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;
```

db2LdapUpdateAlternateServerForDB 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・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 構造を指すポインター。

db2LdapUpdateAltServerStruct データ構造パラメーター

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 IBM Data Server Runtime Client 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 が指定されない場合、現行ユーザーの信用証明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。

db2Load - 表へのデータのロード

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インポート・ユーティリティーよりも高速ですが、階層レベルでのデータのロードまたはニックネームへのロードをサポートしません。

許可

以下のいずれか。

- *sysadm*
- *dbadm*
- データベースに対する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。暗黙接続が可能な場合には、デフォルト・データベースへの接続が確立されます。 Linux、UNIX、または Windows クライアントから Linux、UNIX、または Windows データベース・サーバーへのユーティリティー・アクセスは、DB2 Connect ゲートウェイまたはループバック環境を経由してではなく、エンジンを使用した直接接続でなければなりません。

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 インクルード・ファイル

```
db2ApiDf.h
```

API とデータ構造構文

```
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;
   struct sqlu_media_list *piXmlPathList;
   struct sqllob *piLongActionString;
} db2LoadStruct;

typedef SQL_STRUCTURE db2LoadUserExit
{
   db2Char iSourceUserExitCmd;
   struct db2Char *piInputStream;
   struct db2Char *piInputFileName;
   struct db2Char *piOutputFileName;
   db2Uint16 *piEnableParallelism;
} db2LoadUserExit;

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;
   db2Uint16 *piXmlParse;
   db2DMUXmlValidate *piXmlValidate;
   db2Uint16 iSetIntegrityPending;
   struct db2LoadUserExit *piSourceUserExit;
} 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;

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;
   struct sqlu_media_list *piXmlPathList;
   struct sqllob *piLongActionString;
} 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;
   db2Uint16 iSetIntegrityPending;
   db2Uint16 *piXmlParse;
   db2DMUXmlValidate *piXmlValidate;
   struct db2LoadUserExit *piSourceUserExit;
} db2gLoadIn;

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;
   void *piReserved1;
   db2Uint16 iHostnameLen;
   db2Uint16 iFileTransferLen;
   db2Uint16 iPartFileLocLen;
```

```
   db2Uint16 iMapFileInputLen;
   db2Uint16 iMapFileOutputLen;
   db2Uint16 iDistfileLen;
} db2gPartLoadIn;


/* Definitions for iUsing value of db2DMUXmlValidate structure               */
#define DB2DMU_XMLVAL_XDS              1         /* Use XDS                  */
#define DB2DMU_XMLVAL_SCHEMA           2         /* Use a specified schema   */
#define DB2DMU_XMLVAL_SCHEMALOC_HINTS  3         /* Use schemaLocation hints */
#define DB2DMU_XMLVAL_ORIGSCHEMA       4         /* Use schema that document was
                                                    originally validated against
                                                    (load from cursor only) */
```

db2Load 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・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 構造を指すポインター。

db2LoadStruct データ構造パラメーター

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。有効な値は以下のとおりです (インクルード・ディレクトリーの sqlutil ヘッダー・ファイルで定義される)。

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固有の 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固有の 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。

SQLU_REMOTEFETCH
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_remotefetch_entry 構造によって情報が提供されます。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。

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。有効な値は以下のとおりです (インクルード・ディレクトリーの sqlutil ヘッダー・ファイルで定義される)。

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固有の 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固有の 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。

piFileType 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 フィールドを SQL_METH_L に設定する必要があります。ユーザーは、ロードする各列の開始位置と終了位置を指定します。

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
推奨されません。piLongActionString に換わりました。

piLongActionString
入力。4 バイト長のフィールドが含まれる *sqllob* 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 KEEPDICTIONARY|REPLACE RESETDICTIONARY|RESTART|TERMINATE
INTO tbname [(column_list)]
[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 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。

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。例外表は、ユニーク索引の規則、範囲制約、およびセキュリティー・ポリシーに違反する行のコピーを保管するために使用されます。

NORANGEEXC
範囲違反のためにリジェクトされた行は、例外表に挿入しないことを指定します。

NOUNIQUEEXC
ユニーク制約に違反しているためにリジェクトされた行は、例外表に挿入しないことを指定します。

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 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 または SQLU_REMOTEFETCH のいずれかであり、SQL 照会または表名を参照します。

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。関連リンクの「ロード・ユーティリティー用のファイル・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。このパラメーターの有効な値は以下のとおりです (インクルード・ディレクトリーの sqlutil ヘッダー・ファイルで定義される)。

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、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

出力。 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。有効な値は以下のとおりです (インクルード・ディレクトリーの sqlutil ヘッダー・ファイルで定義される)。

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 (例えば、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。

piXmlPathList
入力。media_type フィールドを SQLU_LOCAL_MEDIA に設定された sqlu_media_list、および xml ファイルが置かれているクライアント上のパスをリストするその sqlu_media_entry 構造を指すポインター。

db2LoadUserExit データ構造パラメーター

iSourceUserExitCmd
入力。データをユーティリティーに送るために使用される実行可能ファイルの完全修飾名。セキュリティー上の理由で、この実行可能ファイルはサーバー上の `sqllib/bin` ディレクトリー内に置く必要があります。
piSourceUserExit 構造が NULL ではない場合、このパラメーターは必須です。

piInputStream、piInputFileName、piOutputFileName、piEnableParallelism の各フィールドはオプションです。

piInputStream
入力。STDIN を経由してユーザー出口アプリケーションに直接渡される汎用バイト・ストリーム。このバイト・ストリームにどんなデータを含めるか、およびどんなフォーマットにするかについては、ユーザーが完全に制御できます。ロード・ユーティリティーは、このバイト・ストリームをサーバーに送り、プロセスの STDIN にフィードして、ユーザー出口アプリケーションに渡すだけです (コード・ページの変換またはバイト・ストリームの変更は行われません)。ユーザー出口アプリケーションは STDIN から引数を読み取り、適切な方法でデータを使用します。

このフィーチャーの重要な属性の 1 つは、機密情報 (ユーザー ID、パスワードなど) を隠す機能です。

piInputFileName

入力。完全修飾されたクライアント・サイド・ファイルの名前を含みます。そのファイルの内容は、プロセスの STDIN をフィードすることによりユーザー出口アプリケーションに渡されます。

piOutputFileName

入力。サーバー・サイド・ファイルの完全修飾名。ユーザー出口アプリケーションを実行しているプロセスの STDOUT および STDERR ストリームは、このファイルにストリーム入力されます。 piEnableParallelism が TRUE のとき、複数のファイル (ユーザー出口インスタンスごとに 1 つ) が作成されて、各ファイル名には 3 桁の数字のノード番号値が付加されます (`<filename>.000` など)。

piEnableParallelism

入力。ユーザー出口アプリケーションの起動を並列化するようにユーティリティーに指示するフラグ。

db2LoadIn データ構造パラメーター

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。将来の利用のために予約されています。

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。 SYSTEM 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。 savecount の値が十分な大きさでない場合、各整合点で実行される活動の同期化によってパフォーマンスに影響してしまいます。

デフォルト値は 0 ですが、それは、必要がなければ整合点は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当てられ, そのサイズは util_heap_sz データベース構成パラメーターで修正可能です。

値が指定されていない場合、実行時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・ヒープで使用できるフリー・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を確認するの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。ロード・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実行時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。ロード・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。値を指定しない場合、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、自動的に計算された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リ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。有効な値は以下のとおりです (インクルード・ディレクトリーの sqlutil ヘッダー・ファイルで定義される)。

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 (例えば、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

バージョン 9.1 では、このパラメーターは推奨されていません。代わりに iSetIntegrityPending パラメーターを使用します。

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。

iSetIntegrityPending

入力。表を SET INTEGRITY ペンディング状態にするように指定します。SQLU_SI_PENDING_CASCADE_IMMEDIATE が指定されている場合、 SET INTEGRITY 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。値として SQLU_SI_PENDING_CASCADE_DEFERRED が指定されている場合、SET INTEGRITY 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_SI_PENDING_CASCADE_DEFERRED がデフォルト値となります。

piSourceUserExit

入力。db2LoadUserExit 構造を指すポインター。

piXmlParse

入力。XML 文書に対して行われる必要のある解析のタイプ。 include ディレクトリーに入っている db2ApiDf ヘッダー・ファイル内の有効値は、次のとおりです。

DB2DMU_XMLPARSE_PRESERVE_WS
空白が保持されます。

DB2DMU_XMLPARSE_STRIP_WS
空白が取り除かれます。

piXmlValidate
入力。db2DMUXmlValidate 構造を指すポインター。 XML 文書の XML スキーマ検証を行う必要があることを示します。

```
/* XML Validate structure                                              */
typedef SQL_STRUCTURE db2DMUXmlValidate
{
   db2Uint16                        iUsing;      /* What to use to perform   */
                                                 /* validation               */
   struct db2DMUXmlValidateXds    *piXdsArgs;    /* Arguments for            */
                                                 /* XMLVALIDATE USING XDS    */
   struct db2DMUXmlValidateSchema *piSchemaArgs; /* Arguments for            */
                                                 /* XMLVALIDATE USING SCHEMA */
} db2DMUXmlValidate;
```

db2LoadOut データ構造パラメーター

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。

db2PartLoadIn データ構造パラメーター

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。このパラメーターは、推奨されていません。

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。このパラメーターは、推奨されていません。代わりに piSourceUserExit パラメーターを使用します。

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データベース・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、各出力データベース・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であり、LOAD_ONLY モードの場合は、各データベース・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。ロード出力データベース・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。

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。有効な値は以下のとおりです (インクルード・ディレクトリーの db2ApiDf ヘッダー・ファイルで定義される)。

- DB2LOAD_PARTITION_AND_LOAD
データは (多くの場合は並列で) 分散されて、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) 分散されて、それぞれのロード・データベース・パーティションの指定した位置にある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、各データベース・パーティション上の出力ファイルの名前の形式は filename.xxx となります。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 はデータベース・パーティションの番号です。SQL_CURSOR ファイル・タイプの場合、各データベース・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。各データベース・パーティション上のデータベース・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
データが既に分散されているとみなします。この場合、分散プロセスがスキップされ、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、各データベース・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 13 桁のデータベース・パーティション番号です。 SQL_CURSOR ファイル・タイプの場合、各データベース・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。各データベース・パーティション上のデータベース・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
データが既に分散されているとみなします。しかし、データ・ファイルにはデータベース・パーティション・ヘッダーがありません。分散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、各行が正しいデータベース・パーティション上にあるかがチェックされます。データベース・パーティション違反のある行は、dumpfile ファイル・タイプ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。データベース・パーティション違反が特定のロード・データベース・パーティションに存在する場合、1 つの警告が、そのデータベース・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。各データベース・パーティションの入力ファイル名の形式は filename.xxx となります。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 13 桁のデータベース・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分散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、個々のデータベース・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。有効な値は以下のとおりです (インクルード・ディレクトリーの db2ApiDf ヘッダー・ファイルで定義される)。

DB2LOAD_SETUP_ERRS_ONLY
このモードでは、セットアップ時にデータベース・パーティションで生じるエラー (例えば、データベース・パーティションへのアクセスに関する問題や、データベース・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、失敗したデータベース・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

りのデータベース・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データベース・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、各データベース・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、セットアップ時にデータベース・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、エラーのあるデータベース・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、残りのデータベース・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データベース・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。他のデータベース・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。すべてのデータベース・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、ロード操作が完了したデータベース・パーティション上では新たにロードされたデータが可視になり、エラーが発生したデータベース・パーティションでは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データベース・パーティション・レベルのエラーによって、影響を受けたデータベース・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・ロード中にデータベース・パーティション・エラーが生じた場合、すべてのデータベース・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。分散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。分散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。データベース・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、分散マップのヘッダーをデータベース・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、出力データベース・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。

db2LoadNodeList データ構造パラメーター

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。

db2LoadPortRange データ構造パラメーター

iPortMin
入力。ポート番号の下限。

iPortMax
入力。ポート番号の上限。

db2PartLoadOut データ構造パラメーター

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エンティティー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データベース・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データベース・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データベース・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。

db2LoadAgentInfo データ構造パラメーター

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

ロードがデータベース・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 SET INTEGRITY PENDING 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、データベース・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、またはそのようなデータベース・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、データベース・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、そのデータベース・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データベース・パーティションの数。

oAgentType

出力。エージェント・タイプ。有効な値は以下のとおりです (インクルード・ディレクトリーの db2ApiDf ヘッダー・ファイルで定義される)。

- DB2LOAD_LOAD_AGENT
- DB2LOAD_PARTITIONING_AGENT
- DB2LOAD_PRE_PARTITIONING_AGENT
- DB2LOAD_FILE_TRANSFER_AGENT
- DB2LOAD_LOAD_TO_FILE_AGENT

db2gLoadStruct データ構造固有パラメーター

iFileTypeLen
入力。iFileType パラメーターの長さ (バイト単位) を指定します。

iLocalMsgFileLen
入力。iLocalMsgFileName パラメーターの長さ (バイト単位) を指定します。

iTempFilesPathLen
入力。iTempFilesPath パラメーターの長さ (バイト単位) を指定します。

piXmlPathList
入力。media_type フィールドを SQLU_LOCAL_MEDIA に設定された sqlu_media_list、および xml ファイルが置かれているクライアント上のパスをリストするその sqlu_media_entry 構造を指すポインター。

db2gLoadIn データ構造固有パラメーター

iUseTablespaceLen
入力。piUseTablespace パラメーターの長さ (バイト単位)。

piXmlParse
入力。XML 文書に対して行われる必要のある解析のタイプ。 include ディレクトリーに入っている db2ApiDf ヘッダー・ファイル内の有効値は、次のとおりです。

DB2DMU_XMLPARSE_PRESERVE_WS
空白が保持されます。

DB2DMU_XMLPARSE_STRIP_WS
空白が取り除かれます。

piXmlValidate
入力。db2DMUXmlValidate 構造を指すポインター。 XML 文書の XML スキーマ検証を行う必要があることを示します。

```
/* XML Validate structure                                              */
typedef SQL_STRUCTURE db2DMUXmlValidate
{
   db2Uint16                       iUsing;       /* What to use to perform   */
                                                 /* validation               */
   struct db2DMUXmlValidateXds    *piXdsArgs;    /* Arguments for            */
                                                 /* XMLVALIDATE USING XDS    */
   struct db2DMUXmlValidateSchema *piSchemaArgs; /* Arguments for            */
                                                 /* XMLVALIDATE USING SCHEMA */
} db2DMUXmlValidate;
```

db2gPartLoadIn データ構造固有パラメーター

piReserved1
将来の利用のために予約されています。

iHostnameLen
入力。piHostname パラメーターの長さ (バイト単位)。

iFileTransferLen
入力。piFileTransferCmd パラメーターの長さ (バイト単位)。

iPartFileLocLen
入力。piPartFileLocation パラメーターの長さ (バイト単位)。

iMapFileInputLen
入力。piMapFileInput パラメーターの長さ (バイト単位)。

iMapFileOutputLen
入力。piMapFileOutput パラメーターの長さ (バイト単位)。

iDistfileLen
入力。piDistfile パラメーターの長さ (バイト単位)。

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、参照整合性を強制したり、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、SET INTEGRITY 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、SET INTEGRITY ペンディング状態になります。表の SET INTEGRITY 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。多次元クラスターされた (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。

db2LoadQuery - ロード操作の状況の取得

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 インクルード・ファイル

```
db2ApiDf.h
```

API とデータ構造構文

```
SQL_API_RC SQL_API_FN
  db2LoadQuery (
    db2Uint32 versionNumber,
    void * pParmStruct,
    struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2LoadQueryStruct
{
   db2Uint32 iStringType;
   char *piString;
```

```
   db2Uint32 iShowLoadMessages;
   struct db2LoadQueryOutputStruct *poOutputStruct;
   char *piLocalMessageFile;
} db2LoadQueryStruct;

typedef SQL_STRUCTURE db2LoadQueryOutputStruct
{
   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;

typedef SQL_STRUCTURE db2LoadQueryOutputStruct64
{
   db2Uint64 oRowsRead;
   db2Uint64 oRowsSkipped;
   db2Uint64 oRowsCommitted;
   db2Uint64 oRowsLoaded;
   db2Uint64 oRowsRejected;
   db2Uint64 oRowsDeleted;
   db2Uint32 oCurrentIndex;
   db2Uint32 oNumTotalIndexes;
   db2Uint32 oCurrentMPPNode;
   db2Uint32 oLoadRestarted;
   db2Uint32 oWhichPhase;
   db2Uint32 oWarningCount;
   db2Uint32 oTableState;
} db2LoadQueryOutputStruct64;

typedef SQL_STRUCTURE db2LoadQueryStruct64
{
   db2Uint32 iStringType;
   char *piString;
   db2Uint32 iShowLoadMessages;
   struct db2LoadQueryOutputStruct64 *poOutputStruct;
   char *piLocalMessageFile;
} db2LoadQueryStruct64;

SQL_API_RC SQL_API_FN
  db2gLoadQuery (
   db2Uint32 versionNumber,
   void * pParmStruct,
   struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2gLoadQueryStruct
{
   db2Uint32 iStringType;
   db2Uint32 iStringLen;
   char *piString;
   db2Uint32 iShowLoadMessages;
   struct db2LoadQueryOutputStruct *poOutputStruct;
   db2Uint32 iLocalMessageFileLen;
   char *piLocalMessageFile;
} db2gLoadQueryStruct;

typedef SQL_STRUCTURE db2gLoadQueryStru64
{
```

```
   db2Uint32 iStringType;
   db2Uint32 iStringLen;
   char *piString;
   db2Uint32 iShowLoadMessages;
   struct db2LoadQueryOutputStruct64 *poOutputStruct;
   db2Uint32 iLocalMessageFileLen;
   char *piLocalMessageFile;
} db2gLoadQueryStru64;
```

db2LoadQuery 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。バージョン 9 以降のバージョンの場合は、db2LoadQueryStruct64 構造を指すポインターです。それ以外のバージョンの場合は、db2LoadQueryStruct 構造を指すポインターです。

pSqlca 出力。 sqlca 構造を指すポインター。

db2LoadQueryStruct データ構造パラメーター

iStringType
入力。piString のタイプを指定します。有効な値は以下のとおりです (インクルード・ディレクトリーの db2ApiDf ヘッダー・ファイルで定義される)。

DB2LOADQUERY_TABLENAME
db2LoadQuery API が使用する表の名前を指定します。

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。有効な値は以下のとおりです (インクルード・ディレクトリーの db2ApiDf ヘッダー・ファイルで定義される)。

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。

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。

db2LoadQueryOutputStruct データ構造パラメーター

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。

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。

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。有効な値は以下のとおりです (インクルード・ディレクトリーの db2ApiDf ヘッダー・ファイルで定義される)。

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。

DB2LOADQUERY_INDEXCOPY_PHASE
索引コピー・フェーズ。

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。有効な値は以下のとおりです (インクルード・ディレクトリーの db2ApiDf ヘッダー・ファイルで定義される)。

DB2LOADQUERY_NORMAL
表の状態は表には影響を及ぼしません。

DB2LOADQUERY_SI_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_SI_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_SI_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_REORG_PENDING
REORG 推奨変更がこの表に対して行われています。この表をアクセス可能にするには、クラシック REORG の実行が必要です。

DB2LOADQUERY_READ_ACCESS
表データを読み取りアクセス照会に使用することができます。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
表は使用できません。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、表は使用できない状態になります。

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。

db2LoadQueryOutputStruct64 データ構造パラメーター

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。

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。

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。有効な値は以下のとおりです (インクルード・ディレクトリーの db2ApiDf ヘッダー・ファイルで定義される)。

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。

DB2LOADQUERY_INDEXCOPY_PHASE
索引コピー・フェーズ。

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。有効な値は以下のとおりです (インクルード・ディレクトリーの db2ApiDf ヘッダー・ファイルで定義される)。

DB2LOADQUERY_NORMAL
表の状態は表には影響を及ぼしません。

DB2LOADQUERY_SI_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_SI_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_SI_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_REORG_PENDING
REORG 推奨変更がこの表に対して行われています。この表をアクセス可能にするには、クラシック REORG の実行が必要です。

DB2LOADQUERY_READ_ACCESS
表データを読み取りアクセス照会に使用することができます。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
表は使用できません。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、表は使用できない状態になります。

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。

db2LoadQueryStruct64 データ構造パラメーター

iStringType
入力。piString のタイプを指定します。有効な値は以下のとおりです (インクルード・ディレクトリーの db2ApiDf ヘッダー・ファイルで定義される)。

DB2LOADQUERY_TABLENAME
db2LoadQuery API が使用する表の名前を指定します。

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。有効な値は以下のとおりです (インクルード・ディレクトリーの db2ApiDf ヘッダー・ファイルで定義される)。

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。

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。

db2gLoadQueryStruct データ構造固有パラメーター

iStringLen
入力。piString パラメーターの長さ (バイト単位) を指定します。

iLocalMessageFileLen
入力。piLocalMessageFile パラメーターの長さ (バイト単位) を指定します。

db2gLoadQueryStru64 データ構造固有パラメーター

iStringLen
入力。piString パラメーターの長さ (バイト単位) を指定します。

iLocalMessageFileLen
入力。piLocalMessageFile パラメーターの長さ (バイト単位) を指定します。

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、piLocalMsgFileName で指定されたファイルに状況を書き込みます。

db2MonitorSwitches - モニター・スイッチ設定の取得あるいは更新

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。

- sysadm
- sysctrl
- sysmaint
- sysmon

必要な接続

インスタンス。インスタンス接続が存在しない場合は、デフォルトのインスタンス接続が作成されます。

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 インクルード・ファイル

db2ApiDf.h

API とデータ構造構文

```
SQL_API_RC SQL_API_FN
  db2MonitorSwitches (
    db2Uint32 versionNumber,
    void * pParmStruct,
    struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2MonitorSwitchesData
{
   struct sqlm_recording_group *piGroupStates;
   void *poBuffer;
   db2Uint32 iBufferSize;
   db2Uint32 iReturnData;
   db2Uint32 iVersion;
   db2int32 iNodeNumber;
   db2Uint32 *poOutputFormat;
} db2MonitorSwitchesData;
```

```
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;
```

db2MonitorSwitches 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・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する場合には、 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。指定したバージョン番号に対応する db2MonitorSwitchesStruct 構造のバージョン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 構造を指すポインター。

db2MonitorSwitchesData データ構造パラメーター

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 が指示するデータ・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できる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
- SQLM_DBMON_VERSION9
- SQLM_DBMON_VERSION9_5

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。

注: SQLM_DBMON_VERSION5_2 以前の定数は使用すべきではなく、今後の DB2 リリースでは削除される可能性があります。

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、自己記述型フォーマット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 SQLMA_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。

db2Prune - 履歴ファイル項目の削除、あるいはアクティブ・ログ・パスからのログ・ファイルの削除

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。

- *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 インクルード・ファイル

db2ApiDf.h

API とデータ構造構文

```
SQL_API_RC SQL_API_FN
  db2Prune (
       db2Uint32 versionNumber,
       void * pParmStruct,
       struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2PruneStruct
{
  char *piString;
  db2HistoryEID iEID;
  db2Uint32 iAction;
  db2Uint32 iOptions;
} db2PruneStruct;

SQL_API_RC SQL_API_FN
  db2gPrune (
       db2Uint32 versionNumber,
       void * pParmStruct,
       struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2gPruneStruct
{
  db2Uint32 iStringLen;
  char *piString;
  db2HistoryEID iEID;
  db2Uint32 iAction;
  db2Uint32 iOptions;
} db2gPruneStruct;
```

db2Prune 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。db2PruneStruct 構造を指すポインター。

pSqlca 出力。 sqlca 構造を指すポインター。

db2PruneStruct データ構造パラメーター

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 のパラメーター値は無効です。

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。

iAction
入力。実行するアクションのタイプを指定します。有効な値は以下のとおりです (インクルード・ディレクトリーの db2ApiDf ヘッダー・ファイルで定義される)。

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。有効な値は以下のとおりです (インクルード・ディレクトリーの db2ApiDf ヘッダー・ファイルで定義される)。

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。

auto_del_rec_obj データベース構成パラメーターを ON に設定した場合、DB2PRUNE_OPTION_DELETE を指定した db2Prune を呼び出すと、関連するバックアップ・イメージおよびロード・コピー・イメージも削除されます。

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。

db2gPruneStruct データ構造固有パラメーター

iStringLen
入力。piString の長さ (バイト単位) を指定します。

使用上の注意

do_not_delete 状況の項目は、整理も削除もされません。リカバリー履歴ファイル項目の状況を do_not_delete に設定するには、UPDATE HISTORY コマンド、UPDATE_HISTORY を伴う ADMIN_CMD、または db2HistoryUpdate API を使用することができます。 do_not_delete 状況を使用することによって、重要なリカバリー履歴ファイル項目が整理または削除されないようにすることができます。

最新の全データベース・バックアップをメディアから削除する (さらに、履歴ファイルから項目が除去される) 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。

db2Prune を使用することによってスナップショット・バックアップ・データベース履歴ファイルの項目の整理を実行できますが、DB2PRUNE_OPTION_DELETE パラメーターを使用しても、関連する物理リカバリー・オブジェクトを削除することはできません。スナップショット・バックアップ・オブジェクトを削除する唯一の方法は、db2acsutil コマンドを使用することです。

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除去されます。指定しない場合、入力時に指定したタイム・スタンプ以下の場合でも、最新のリストア・セットが保持されます。

db2QuerySatelliteProgress - サテライト同期セッションの状況の取得

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 インクルード・ファイル

db2ApiDf.h

API とデータ構造構文

```
SQL_API_RC SQL_API_FN
  db2QuerySatelliteProgress (
   db2Uint32 versionNumber,
   void * pParmStruct,
   struct sqlca * pSqlca);

typedef struct db2QuerySatelliteProgressStruct
{
   db2int32 oStep;
   db2int32 oSubstep;
   db2int32 oNumSubsteps;
   db2int32 oScriptStep;
   db2int32 oNumScriptSteps;
   char *poDescription;
   char *poError;
   char *poProgressLog;
} db2QuerySatelliteProgressStruct;
```

db2QuerySatelliteProgress 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 構造を指すポインター。

db2QuerySatelliteProgressStruct データ構造パラメーター

oStep 出力。同期セッションの現行ステップ (include ディレクトリーの db2ApiDf ヘッダー・ファイルで定義される)。

oSubstep
出力。パラメーターによって示される同期ステップ (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。

db2ReadLog - ログ・レコードの抽出

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 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、データベース構成パラメーター logarchmeth1 および/または logarchmeth2 が OFF に設定されていない場合です。

許可

以下のいずれか。

- sysadm
- dbadm

必要な接続

データベース

API インクルード・ファイル

db2ApiDf.h

API とデータ構造構文

```
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;
} db2ReadLogStruct;

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;
```

db2ReadLog 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 構造を指すポインター。

pSqlca 出力。 sqlca 構造を指すポインター。

db2ReadLogStruct データ構造パラメーター

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 パラメーター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。この値が使用された場合に戻されるログ・レコードに関しては、『DB2 ログ・レコード』のトピックに詳述されています。他のすべてのログ・レコードについては IBM の内部使用専用であるため、詳述しません。

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。

db2ReadLogInfoStruct データ構造パラメーター

initialLSN
データベースがアクティブにされた後、データベースによって最初に使用された、あるいは使用される予定の LSN。

firstReadLSN
poLogBuffer パラメーターにある最初の LSN。

nextStartLSN
呼び出し元が読み取る次のログ・レコードの先頭。ログ・レコードの一部がフィルタリングされ、 poLogBuffer パラメーターで戻されない可能性があるので、この LSN を poLogBuffer パラメーターの最後のログ・レコードの代わりに、次の読み取りの先頭として使用すると、既にフィルタリングされたログ・レコードが再度スキャンされることを防止できます。

logRecsWritten
poLogBuffer パラメーターに書き込まれるログ・レコードの数。

logBytesWritten
poLogBuffer パラメーターに書き込まれるデータの総バイト数。

firstReusedLSN
データベース・リストア、あるいはロールフォワード操作のため、再利用される最初の LSN。

timeOfLSNReuse
firstReusedLSN に対応する LSN が再利用された時刻。 1970 年 1 月 1 日からの秒数で表されます。

currentTimeValue
データベースの提示する現在時刻。

db2TimeOfLog データ構造パラメーター

seconds
1970 年 1 月 1 日からの秒数。

accuracy
高精度カウンター。呼び出し元は、これによって、秒の単位まで同じになっているタイム・スタンプを比較し、イベントの起きた順序を特定できます。

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、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する必要があります。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 CHANGES 節を使って定義された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データベース・ログの照会 (値 DB2READLOG_QUERY の指定によって示される) 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 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。

この API は、DB2 ログからデータを読み取ります。この種のログには、ラベル・ベース・アクセス制御 (LBAC) は施行されません。したがって、この API を呼び出すアプリケーションは、呼び出し元に API を呼び出せるだけの適切な権限があり、ログ・レコード・フォーマットを解読する機能があれば、表データへのアクセスが可能です。

db2ReadLog API は、現行のデータベース接続を利用して機能します。同一のプロセスによって複数のデータベース接続が作成された場合は、並行アクセス API を使用して複数のコンテキストを管理します。

アプリケーションから db2ReadLog API を呼び出すと、事前にコミットかロールバックが行われずにアプリケーションがデータベースから切断された場合、エラーになることがあります。

- db2ReadLog API が CLI アプリケーションから呼び出された場合、CLI0116E エラーが生成される可能性があります。
- db2ReadLog API が C で書かれた組み込み SQL アプリケーションから呼び出された場合、SQL0428N エラーが生成される可能性があります。

予備手段 1: 非組み込み SQL アプリケーションを使用している場合は、db2ReadLog API を呼び出す前に自動コミット・モードをオンに設定します。

予備手段 2: db2ReadLog API を呼び出した後、データベースとの接続が切断される前に、COMMIT あるいは ROLLBACK ステートメントを発行します。

db2ReadLogNoConn - データベース接続なしのデータベース・ログの読み取り

ログ・レコードを DB2 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 API を呼び出して、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 を呼び出した後、db2ReadLogNoConnTerm API を呼び出して、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 インクルード・ファイル

```
db2ApiDf.h
```

API とデータ構造構文

```
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;
```

db2ReadLogNoConn 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 構造を指すポインター。

db2ReadLogNoConnStruct データ構造パラメーター

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。

db2ReadLogNoConnInfoStruct データ構造パラメーター

firstAvailableLSN
使用できるログ内で最初の使用できる 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。

currentTimeValue
将来の利用のために予約されています。

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 呼び出し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、使用できる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 ログ・レコードの詳細については、『DB2 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、使用できるログ・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。

この API は、DB2 ログからデータを読み取ります。この種のログには、ラベル・ベース・アクセス制御 (LBAC) は施行されません。したがって、この API を呼び出すアプリケーションは、呼び出し元に API を呼び出せるだけの適切な権限があり、ログ・レコード・フォーマットを解読する機能があれば、表データにアクセスできる可能性があります。

db2ReadLogNoConnInit - データベース接続なしのデータベース・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 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 インクルード・ファイル

```
db2ApiDf.h
```

API とデータ構造構文

```
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;
```

db2ReadLogNoConnInit 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 構造を指すポインター。

db2ReadLogNoConnInitStruct データ構造パラメーター

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。

db2ReadLogNoConnTerm - データベース接続なしのデータベース・ログの読み取り終了

本来は db2ReadLogNoConnInit API によって初期設定され、db2ReadLogNoConn API 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 インクルード・ファイル

```
db2ApiDf.h
```

API とデータ構造構文

```
SQL_API_RC SQL_API_FN
  db2ReadLogNoConnTerm (
       db2Uint32 versionNumber,
       void * pDB2ReadLogNoConnTermStruct,
       struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2ReadLogNoConnTermStruct
{
   char                                **poReadLogMemPtr;
} db2ReadLogNoConnTermStruct;
```

db2ReadLogNoConnTerm 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 構造を指すポインター。

db2ReadLogNoConnTermStruct データ構造パラメーター

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。

db2Recover - データベースのリストアおよびロールフォワード

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、インスタンス接続が必要です。

API インクルード・ファイル

```
db2ApiDf.h
```

API とデータ構造構文

```
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;
  char *piHistoryFile;
  db2Uint32 iNumChngHistoryFile;
  struct sqlu_histFile *piChngHistoryFile;
  char *piComprLibrary;
  void *piComprOptions;
  db2Uint32 iComprOptionsSize;
} db2RecoverStruct;

SQL_STRUCTURE sqlu_histFile
{
   SQL_PDB_NODE_TYPE nodeNum;
   unsigned short  filenameLen;
   char filename[SQL_FILENAME_SZ+1];
};

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;
  char *piHistoryFile;
  db2Uint32 iHistoryFileLen;
  db2Uint32 iNumChngHistoryFile;
  struct sqlu_histFile *piChngHistoryFile;
  char *piComprLibrary;
  db2Uint32 iComprLibraryLen;
  void *piComprOptions;
  db2Uint32 iComprOptionsSize;
} db2gRecoverStruct;
```

db2Recover 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・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 構造を指すポインター。

db2RecoverStruct データ構造パラメーター

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。

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。

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。

iRecoverCallerAction
入力。有効な値は以下のとおりです。

DB2RECOVER
リカバリー操作を開始します。リカバリー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、例えば、リカバリー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。

DB2RECOVER_RESTART
ユーザーが直前のリカバリーを無視して最初からやり直せるようにします。

DB2RECOVER_CONTINUE
警告メッセージを生成した装置の使用を続けます (例えば、新しいテープをマウントしたときなど)。

DB2RECOVER_LOADREC_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。

DB2RECOVER_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 (例えば、それ以上テープがない場合)。

DB2RECOVER_PARM_CHK_ONLY
リカバリー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。

DB2RECOVER_DEVICE_TERMINATE
リカバリー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、リカバリー・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カバリー・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。

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 ではなく現地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。

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・パスで検索され、次にオーバーフロー・ログ・パスで検索されます。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。

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。

piHistoryFile
履歴ファイル。

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。

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。

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、DB2 によって、サーバー上に存在するファイルの名前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します。 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。

sqlu_histFile データ構造パラメーター

nodeNum
入力。この項目をどのデータベース・パーティションに対して使用するかを指定します。

filenameLen
入力。ファイル名の長さ (バイト単位)。

filename
入力。このデータベース・パーティションの履歴ファイルへのパス。パスの最後はスラッシュでなければなりません。

db2gRecoverStruct データ構造固有パラメーター

iSourceDBAliasLen
piSourceDBAlias パラメーターの長さ (バイト単位) を指定します。

iUserNameLen
piUsername パラメーターの長さ (バイト単位) を指定します。

iPasswordLen
piPassword パラメーターの長さ (バイト単位) を指定します。

iStopTimeLen
piStopTime パラメーターの長さ (バイト単位) を指定します。

iOverflowLogPathLen
piOverflowLogPath パラメーターの長さ (バイト単位) を指定します。

iHistoryFileLen
piHistoryFile パラメーターの長さ (バイト単位) を指定します。

iComprLibraryLen
入力。piComprLibrary パラメーターで指定したライブラリー名の長さをバイト単位で指定します。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。

db2Reorg - 索引または表の再編成

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。

- SYSADM
- SYSCTRL
- SYSMAINT
- DBADM
- 表に対する CONTROL 特権

必要な接続

データベース

API インクルード・ファイル

```
db2ApiDf.h
```

API とデータ構造構文

```
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;
  char *pLongTempSpace;
} db2ReorgTable;

typedef SQL_STRUCTURE db2ReorgIndexesAll
{
  char *pTableName;
  char *pIndexName;
} db2ReorgIndexesAll;

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;
  db2Uint32 longTempSpaceLen;
  char *pLongTempSpace;
} db2gReorgTable;

typedef SQL_STRUCTURE db2gReorgIndexesAll
{
  db2Uint32 tableNameLen;
  char *pTableName;
  db2Uint32 indexNameLen;
  char *pIndexName;
} db2gReorgIndexesAll;
```

db2Reorg 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。

db2ReorgStruct データ構造パラメーター

reorgType
入力。再編成のタイプを指定します。有効な値は以下のとおりです (include ディレクトリーの db2ApiDf ヘッダー・ファイルで定義される)。

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。

DB2REORG_OBJ_INDEX
1 つの索引を再編成します。

reorgFlags
入力。再編成オプション。有効な値は以下のとおりです (include ディレクトリーの db2ApiDf ヘッダー・ファイルで定義される)。

DB2REORG_OPTION_NONE
デフォルトのアクション。

DB2REORG_LONGLOB
DB2REORG_OBJ_TABLE_OFFLINE が **reorgType** として指定されているときに使用され、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** として指定されているときに使用され、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** として指定されているときに使用され、オンライン再編成を開始します。

DB2REORG_PAUSE_ONLINE
DB2REORG_OBJ_TABLE_INPLACE が **reorgType** として指定されているときに使用され、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** として指定されているときに使用され、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** として指定されているときに使用され、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** として指定されているときに使用され、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** として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** として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
クリーンアップは必要ありません。DB2REORG_OBJ_INDEXESALL または DB2REORG_OBJ_INDEX が **reorgType** として指定されているときに使用されます。

DB2REORG_CLEANUP_ALL
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クリーンアップします。DB2REORG_OBJ_INDEXESALL または DB2REORG_OBJ_INDEX が **reorgType** として指定されているときに使用されます。

DB2REORG_CLEANUP_PAGES
DB2REORG_OBJ_INDEXESALL または DB2REORG_OBJ_INDEX が **reorgType** として指定されているときに使用され、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。疑似空ではない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
変換は必要ありません。DB2REORG_OBJ_INDEXESALL または DB2REORG_OBJ_INDEX が **reorgType** として指定されているときに使用されます。

DB2REORG_CONVERT
DB2REORG_OBJ_INDEXESALL が **reorgType** として指定されているときに使用され、タイプ 2 の索引に変換します。

DB2REORG_RESET_DICTIONARY
表の COMPRESS 属性が YES の場合、新しい行コンプレッション・ディクショナリーが作成されます。再編成の際に処理されるすべての行は、この新しいディクショナリーによる圧縮の対象になります。前のすべてのディクショナリーは、このディクショナリーに置き換わります。表の COMPRESS 属性が NO であり、表にコンプレッション・ディクショナリーが存在している場合、再編成処理によりそのディクショナリーは除去され、新たに編成された表のすべての行は非圧縮形式になります。このパラメーターは、DB2REORG_OBJ_TABLE_OFFLINE の **reorgType** に対してのみサポートされます。

DB2REORG_KEEP_DICTIONARY
表の COMPRESS 属性が YES で、ディクショナリーが存在する場合、そのディクショナリーは保持されます。表の COMPRESS 属性が YES で、ディクショナリーが存在しない場合、オプションはデフォルトの DB2REORG_RESET_DICTIONARY になるので、ディクショナリーが作成されます。再編成で処理されるすべての行は、圧縮の対象になります。表の COMPRESS 属性が NO の場合、ディクショナリー (存在する場合) は保持されて、新しく再編成された表のすべての行は非圧縮フォーマットになります。このパラメーターは、DB2REORG_OBJ_TABLE_OFFLINE の **reorgType** に対してのみサポートされます。

nodeListFlag
入力。再編成するノードを指定します。有効な値は以下のとおりです (include ディレクトリーの db2ApiDf ヘッダー・ファイルで定義される)。

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。

db2ReorgObject 共用体パラメーター

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。

db2ReorgTable data データ構造パラメーター

pTableName
入力。再編成する表の名前を指定します。

pOrderByIndex
入力。表の順序を決める索引を指定します。

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。 REORG コマンドは、列が表に追加されて (つまり ALTER TABLE ADD COLUMN から)、その列が追加される前に行が挿入される場合に、行を拡張することがあります。非パーティション表では、新しい表オブジェクトを作成するために十分な大きさの表スペースを、このパラメーターによって指定する必要があります。パーティション表は、単一のデータ・パーティションずつ再編成されます。この場合、表スペースには、表の最大のデータ・パーティションを入れることのできる十分のフリー・スペースが必要です。

このパラメーターが非パーティション表に対して指定されていない場合、表が存在する表スペースが使用されます。このパラメーターがパーティション表に対して指定されていない場合、各データ・パーティションが存在する表スペースがそのデータ・パーティションの一時ストレージに使用されます。各データ・パーティションの表スペースには、そのデータ・パーティションのコピーが入るだけの十分なフリー・スペースがなければなりません。

pLongTempSpace
入力。表の再編成の際にロング・オブジェクト (LONG VARCHAR および LOB 列) を作成するための TEMPORARY 表スペースを指定します。 **pSysTempSpace** パラメーターが指定されていない場合、このパラメーターは無視されます。このパラメーターが指定されていない場合で **pSysTempSpace** パラメーターが指定されているときは、DB2 は **pSysTempSpace** パラメーターで指定される表スペースに、ページ・サイズが異ならない限りロング・データ・オブジェクトを作成します。

ページ・サイズが異なるときは、**pSysTempSpace** が指定されていてもこのパラメーターは指定されていない場合、DB2 はロング・オブジェクトを作成して入れるために、同じページ・サイズの既存の表スペースを探すことを試行します。

db2ReorgIndexesAll データ構造パラメーター

pTableName
入力。索引再編成を行う表の名前を指定します。 DB2REORG_OBJ_INDEX

が **reorgType** として指定されている場合、**pTableName** パラメーターは必要ではなく、NULL とすることができます。ただし、**pTableName** パラメーターを指定する場合、それは、索引が定義されている表でなければなりません。

pIndexName
入力。再編成する索引の名前を指定します。このパラメーターは、**reorgType** パラメーターの値が DB2REORG_OBJ_INDEX に設定されている場合にのみ使用します。その他の場合には、**pIndexName** パラメーターを NULL に設定します。

db2gReorgTable データ構造固有パラメーター

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

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

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

longTempSpaceLen
入力。**pLongTempSpace** パラメーターに保管される名前の長さを指定します。

db2gReorgIndexesAll データ構造固有パラメーター

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

indexNameLen
入力。**pIndexName** パラメーターの長さ (バイト単位) 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。reorg 処理の際に、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。
- 表のデータがいくつかのノードに分散されており、影響するノードのいずれかで再編成が失敗した場合には、失敗したノードだけが再編成をロールバックします。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。
- 表の再編成では、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有効になっていなければなりません。
- COMPACT オプションと共に定義されていない LOB 列が表にある場合、 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。
- 以下の表は、reorg および表のタイプに基づいて選択されるデフォルトの表アクセスを示しています。

表 8. reorg および表のタイプに基づいて選択されるデフォルトの表アクセス

デフォルトの表アクセスに影響を与える可能性のある reorg のタイプおよび該当するフラグ		各表タイプに対して選択されるアクセス・モード	
reorgType	**reorgFlags (該当する場合)**	**非パーティション表**	**パーティション表**
DB2REORG_OBJ_TABLE_ OFFLINE		DB2REORG_ALLOW_ READ	DB2REORG_ALLOW_NONE
DB2REORG_OBJ_TABLE_ INPLACE		DB2REORG_ALLOW_WRITE	N/A
DB2REORG_OBJ_INDEXESALL		DB2REORG_ALLOW_ READ	DB2REORG_ALLOW_NONE
DB2REORG_OBJ_INDEXESALL	DB2REORG_CLEANUP_ALL, DB2REORG_CLEANUP_PAGES	DB2REORG_ALLOW_ READ	DB2REORG_ALLOW_ READ
DB2REORG_OBJ_INDEX		N/A	DB2REORG_ALLOW_ READ
DB2REORG_OBJ_INDEX	DB2REORG_CLEANUP_ALL, DB2REORG_CLEANUP_PAGES	N/A	DB2REORG_ALLOW_ READ

N/A: 現時点ではサポートされていないため、適用されません。

注: いくつかのアクセス・モードは、特定のタイプの表または索引でサポートされないことがあります。これらの場合、可能であれば、最も制限の小さいアクセス・モードが使用されます。 (最も制限の大きいアクセス・モードは DB2REORG_ALLOW_NONE であり、その次が DB2REORG_ALLOW_READ で、その次が最も制限の小さい DB2REORG_ALLOW_WRITE です)。既存の表または索引タイプに対するサポートが変更されるか、または新規の表または索引タイプが提供されると、デフォルトはより制限の大きいアクセス・モードからより制限の小さいアクセス・モードに変更されることがあります。 **reorgType** が DB2REORG_OBJ_TABLE_INPLACE ではない場合、デフォルトに選択された最も制限の小さいモードは DB2REORG_ALLOW_READ を超えることはありません。 DB2REORG_ALLOW_NONE、DB2REORG_ALLOW_READ、または DB2REORG_ALLOW_WRITE フラグのいずれも指定されていない場合、デフォルトのアクセス・モードが選択されます。

- 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。再編成された索引が使用できるようになるときには短いロックアウト期間が存在し、その間は表へのアクセスは許可されません。
- 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警告します。パーティション表に対する索引の再編成では、再作成の必要な索引がオフラインで再作成されてから、指定した索引が (まだ再作成されていないとの想定で) 再編成されます。この再編成では、ユーザーが指定するアクセス・モードが使用されます。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、索引がオフラインで再作成されていることについて警告します。

- インプレースではない表の再編成では、DB2REORG_RESET_DICTIONARY または DB2REORG_KEEP_DICTIONARY のどちらも指定されていない場合、デフォルトは DB2REORG_KEEP_DICTIONARY となります。
- アクセスがない索引の再編成が失敗した場合、一部または全部の索引は使用できなくなり、次回の表アクセスで再作成されます。
- 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

db2ResetAlertCfg - ヘルス・インディケーターの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。

- sysadm
- sysmaint
- sysctrl

必要な接続

インスタンス。インスタンス接続が存在しない場合は、デフォルトのインスタンス接続が作成されます。

API インクルード・ファイル

```
db2ApiDf.h
```

API とデータ構造構文

```
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;
```

db2ResetAlertCfg 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・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 構造を指すポインター。

db2ResetAlertCfgData データ構造パラメーター

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。有効な値は以下のとおりです (インクルード・ディレクトリーの db2ApiDf ヘッダー・ファイルで定義される)。

- 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。表スペース・コンテナーの名前は、<tablespace-numericalID>.<tablespace-container-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。

db2ResetMonitor - データベース・システム・モニターのデータのリセット

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、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。

- sysadm
- sysctrl
- sysmaint
- sysmon

必要な接続

インスタンス。インスタンス接続が存在しない場合は、デフォルトのインスタンス接続が作成されます。

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 インクルード・ファイル

db2ApiDf.h

API とデータ構造構文

```
SQL_API_RC SQL_API_FN
  db2ResetMonitor (
   db2Uint32 versionNumber,
   void * pParmStruct,
   struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2ResetMonitorData
{
   db2Uint32 iResetAll;
   char *piDbAlias;
   db2Uint32 iVersion;
```

```
   db2int32 iNodeNumber;
} db2ResetMonitorData;

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;
```

db2ResetMonitor 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・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 構造を指すポインター。

db2ResetMonitorData データ構造パラメーター

iResetAll
入力。リセット・フラグです。

piDbAlias
入力。データベース別名を指すポインター。

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。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できる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
- SQLM_DBMON_VERSION9
- SQLM_DBMON_VERSION9_5

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。

注: SQLM_DBMON_VERSION5_2 以前の定数は使用すべきではなく、今後の DB2 リリースでは削除される可能性があります。

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、値 SQLM_CURRENT_NODE を使用する必要があります。

db2gResetMonitorData データ構造固有パラメーター

iDbAliasLength
入力。piDbAlias パラメーターの長さ (バイト単位) を指定します。

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。

db2Restore - データベースまたは表スペースのリストア

db2Backup API を使用してバックアップされた損傷のある、または破壊された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。ただし、スナップショット・リストアの場合は例外であり、その場合はバックアップ・イメージ・データベース名は同じでなければなりません。

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

データベース (既存のデータベースにリストアする場合)。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。

インスタンス およびデータベース (新規データベースにリストアする場合)。データベースを作成するには、インスタンス接続が必要です。

スナップショット・リストアの場合、インスタンス とデータベース の接続が必要です。

(**DB2INSTANCE** 環境変数の値で定義された) 現行のインスタンスとは異なるインスタンスで新規データベースへのリストアを行うには、まず、新規データベースが存在することになるインスタンスにアタッチする必要があります。

API インクルード・ファイル

db2ApiDf.h

API とデータ構造構文

```
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;
  struct db2StoragePathsStruct *piStoragePaths;
  char *piRedirectScript;
} db2RestoreStruct;

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

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

typedef SQL_STRUCTURE db2StoragePathsStruct
{
  char                         **storagePaths;
  db2Uint32 numStoragePaths;
} db2StoragePathsStruct;

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;
  struct db2gStoragePathsStruct *piStoragePaths;
  char *piRedirectScript;
  db2Uint32 iRedirectScriptLen;
} 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 db2gStoragePathsStruct
{
  struct db2Char *storagePaths;
  db2Uint32 numStoragePaths;
} db2gStoragePathsStruct;

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

db2Restore API のパラメーター

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 構造を指すポインター。

db2RestoreStruct データ構造パラメーター

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。

piTargetDBAlias
入力。ターゲット・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** パラメーターの値が使用されます。

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。

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、このフィールドは任意指定です。

piTargetDBPath
入力。サーバー上のターゲット・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。

注: このパラメーターは廃止されていますが、現在のところまだ定義されています。

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。再作成の場合、これはデータベースの再作成にどの表スペースを使用するかを示す包含リストまたは除外リストにすることができます。 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 (再作成ではない場合のみ)。つまり、ログ保持またはユーザー出口が有効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。

- 再作成の場合、5 つの再作成タイプのうち DB2RESTORE_ALL_TBSP_IN_DB_EXC、DB2RESTORE_ALL_TBSP_IN_IMG_EXC、および DB2RESTORE_ALL_TBSP_IN_LIST の 3 つに関しては、このリストの指定が必要です。

piMediaList
入力。バックアップ・イメージのソース・メディア。

詳しくは、後出の 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。

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。

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。

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** パラメーターの長さ (バイト)。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・イメージの作成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。有効な値は以下のとおりです (インクルード・ディレクトリーの db2ApiDf ヘッダー・ファイルで定義される)。

- 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、例えば、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 (例えば、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、リストアを継続するには、ユーザーが **iCallerAction** パラメーターの値を DB2RESTORE_CONTINUE に設定してこの API 呼び出しをもう一度発行する必要があります。
- 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。有効な値は以下のとおりです (インクルード・ディレクトリーの db2ApiDf ヘッダー・ファイルで定義される)。

- 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をリストアします。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、他のリストア処理の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・イメージに含まれるログ・ファイルのセット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** パラメーターもまた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_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。リストアの成功後、データベースがロールフォワード・ペンディング状態であれば、データベースを使用できる状態にするためには db2Rollforward API を呼び出す必要があります。
- DB2RESTORE_GENERATE_SCRIPT - リダイレクト・リストアの実行に使用できるスクリプトを作成します。 **piRedirectScript** には有効なファイル名を含める必要があります。 **iCallerAction** は DB2RESTORE_RESTORE_STORDEF または DB2RESTORE_STORDEF_NOINTERRUPT のいずれかにする必要があります。

以下の値は、再作成操作でのみ使用します。

- DB2RESTORE_ALL_TBSP_IN_DB - イメージをリストアする時点でデータベースが認識しているすべての表スペースを使って、データベースをリストアします。データベースが既に存在する場合、この再作成によってデータベースが上書きされます。
- DB2RESTORE_ALL_TBSP_IN_DB_EXC - イメージをリストアする時点でデータベースが認識しているすべての表スペースを使って、データベースをリストアします。ただし、**piTablespaceList** パラメーターで指定されたリストに含まれるものは除外されます。データベースが既に存在する場合、この再作成によってデータベースが上書きされます。
- DB2RESTORE_ALL_TBSP_IN_IMG - リストアされるイメージに含まれる表スペースだけを使ってデータベースをリストアします。データベースが既に存在する場合、この再作成によってデータベースが上書きされます。
- DB2RESTORE_ALL_TBSP_IN_IMG_EXC - リストアされるイメージに含まれる表スペースだけを使ってデータベースをリストアします。ただし、

piTablespaceList パラメーターで指定されたリストに含まれるものは除外されます。データベースが既に存在する場合、この再作成によってデータベースが上書きされます。

- DB2RESTORE_ALL_TBSP_IN_LIST - **piTablespaceList** パラメーターで指定されたリストに含まれる表スペースだけを使ってデータベースをリストアします。データベースが既に存在する場合、この再作成によってデータベースが上書きされます。

注: リカバリー可能データベースのバックアップ・イメージである場合、上記の再作成アクションで WITHOUT ROLLING FORWARD (DB2RESTORE_NOROLLFWD) を指定することはできません。

piComprLibrary

入力。イメージが圧縮されている場合に、バックアップ・イメージを解凍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、空ストリングを指すポインターである場合、DB2 データベース・システムは、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い場合、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、リストア操作は失敗します。

piComprOptions

入力。この API パラメーターは、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 データベース・システム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「@」であれば、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。次いで DB2 データベース・システムは **piComprOptions** および **iComprOptionsSize** パラメーターの内容を、このファイルの内容およびサイズで置き換え、それらの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** の値が NULL ポインターである場合に限り、**iComprOptionsSize** パラメーターはゼロにします。

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必ず使用される、データベース・サーバー上のディレクトリーの絶対パスを指定します。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には、DB2RESTORE_LOGS 値を **iOptions** パラメーターに渡す必要があります。

スナップショット・リストアの場合、以下のいずれか 1 つを指定する必要があります。

- DB2RESTORE_LOGTARGET_INCLUDE ″INCLUDE″

  スナップショット・イメージからログ・ディレクトリー・ボリュームをリストアします。このオプションが指定されていて、バックアップ・イメージにログ・ディレクトリーが含まれている場合、それらはリストアされます。ディスク上に既存のログ・ディレクトリーとログ・ファイルは、バックアップ・イメージ中のログ・ディレクトリーと競合するのでなければ、変更なしでそのままになります。ディスク上に既存のログ・ディレクトリーがバックアップ・イメージ中のログ・ディレクトリーと競合する場合は、エラーが戻されます。

- DB2RESTORE_LOGTARGET_EXCLUDE ″EXCLUDE″

  ログ・ディレクトリー・ボリュームをリストアしません。このオプションが指定された場合、バックアップ・イメージからログ・ディレクトリーがリストアされません。ディスク上に既存のログ・ディレクトリーとログ・ファイルは、バックアップ・イメージ中のログ・ディレクトリーと競合するのでなければ、変更なしでそのままになります。データベースに属する 1 つのパスがリストアされ、そのために暗黙のうちに 1 つのログ・ディレクトリーがリストアされ、その結果、ログ・ディレクトリーが上書きされることになる場合、エラーが戻されます。

- DB2RESTORE_LOGTARGET_INCFORCE ″INCLUDE FORCE″

  スナップショット・イメージのリストア時に、既存のログ・ディレクトリーの上書きおよび置換を許可します。このオプションが指定されていて、バックアップ・イメージにログ・ディレクトリーが含まれている場合、それらはリストアされます。ディスク上に既存のログ・ディレクトリーとログ・ファイルは、バックアップ・イメージ中のログ・ディレクトリーと競合するのでなければ、変更なしでそのままになります。ディスク上に既存のログ・ディレクトリーがバックアップ・イメージ中のログ・ディレクトリーと競合する場合は、バックアップ・イメージ中のものによって上書きされます。

- DB2RESTORE_LOGTARGET_EXCFORCE ″EXCLUDE FORCE″

  スナップショット・イメージのリストア時に、既存のログ・ディレクトリーの上書きおよび置換を許可します。このオプションが指定された場合、バックアップ・イメージからログ・ディレクトリーがリストアされません。ディスク上に既存のログ・ディレクトリーとログ・ファイルは、バックアップ・イメージ中のログ・ディレクトリーと競合するのでなければ、変更なしでそのままになります。データベースに属する 1 つのパスがリストアされ、そのために暗黙のうちに 1 つのログ・ディレクトリーがリストアされ、その結果、ログ・ディレクトリーが上書きされることになる場合、リストアが続行され、競合するログ・ディレクトリーは上書きされます。

ここで DB2RESTORE_LOGTARGET_EXCLUDE はデフォルトです。

piStoragePaths

入力。自動ストレージに使用されるストレージ・パスのリストを表すフィールドを含む構造。データベースの自動ストレージが有効になっていない場合は、これを NULL に設定します。

piRedirectScript
入力。クライアント側に作成されるリダイレクト・リストア・スクリプトのファイル名。ファイル名には相対パスまたは絶対パスを指定できます。**iOptions** フィールドの DB2RESTORE_GENERATE_SCRIPT ビットを設定しておく必要があります。

db2TablespaceStruct データ構造固有パラメーター

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

db2MediaListStruct データ構造パラメーター

locations
入力。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
入力。locations パラメーター内の項目数。

locationType
入力。メディア・タイプを示す文字。有効な値は以下のとおりです (インクルード・ディレクトリーの sqlutil ヘッダー・ファイルで定義される)。

SQLU_LOCAL_MEDIA: 'L'
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: 'X'
XBSA インターフェース。

SQLU_TSM_MEDIA: 'A'
Tivoli Storage Manager。

SQLU_OTHER_MEDIA: 'O'
ベンダー・ライブラリー。

SQLU_SNAPSHOT_MEDIA: 'F'
データがスナップショット・バックアップからリストアされるように指定します。

SQLU_SNAPSHOT_MEDIA は、以下のいずれかと共には使用できません。

- 呼び出し元アクション: DB2RESTORE_RESTORE_STORDEF、DB2RESTORE_STORDEF_NOINTERRUPT、DB2RESTORE_TERMINATE_INCRE
- DB2RESTORE_REPLACE_HISTORY
- DB2RESTORE_TABLESPACE
- DB2RESTORE_COMPR_LIB

- DB2RESTORE_INCREMENTAL
- DB2RESTORE_HISTORY
- DB2RESTORE_LOGS
- **piStoragePaths** - 使用するには NULL または空でなければなりません。
- **piTargetDBPath**
- **piTargetDBAlias**
- **piNewLogPath**
- **iNumBuffers**
- **iBufferSize**
- **piRedirectScript**
- **iRedirectScriptLen**
- **iParallelism**
- **piComprLibrary**、**iComprLibraryLen**、**piComprOptions**、または **iComprOptionsSize**
- スナップショット・リストアの場合、この構造の **numLocations** フィールドは 1 でなければなりません。

また、表スペース・リストが関係するリストア操作では SNAPSHOT パラメーターを使用できません。

スナップショット・バックアップ・イメージからのデータ・リストア時のデフォルトの動作は、すべてのコンテナー、ローカル・ボリューム・ディレクトリー、データベース・パス (**DBPATH**)、最新のスナップショット・バックアップの 1 次ログとミラー・ログのパス (タイム・スタンプが指定されていない場合) を含む、データベースを構成するすべてのパスの FULL DATABASE OFFLINE リストアです (明示的に EXCLUDE LOGS が指定されているのでない限り、すべてのスナップショット・バックアップでのデフォルトは INCLUDE LOGS です)。タイム・スタンプが指定されている場合、そのスナップショット・バックアップ・イメージがリストアされます。

IBM Data Server には、以下のストレージ・ハードウェアのための DB2 ACS API ドライバーが組み込まれています。

- IBM TotalStorage SAN ボリューム・コントローラー
- IBM Enterprise Storage Server Model 800
- IBM System Storage DS6000
- IBM System Storage DS8000
- IBM System Storage N Series
- NetApp V-series
- NetApp FAS

db2StoragePathsStruct データ構造パラメーター

storagePaths

入力。自動ストレージ表スペースに使用される、サーバー上のストレージ・

パスの完全修飾名を含むストリングの配列。複数パーティション・データベースでは、すべてのデータベース・パーティションに対して同じストレージ・パスが使用されます。新しいストレージ・パスを使って複数パーティション・データベースをリストアする場合、他のデータベース・パーティションをリストアする前にまずカタログ・パーティションをリストアする必要があります。

numStoragePaths
入力。db2StoragePathsStruct 構造の **storagePaths** パラメーター内のストレージ・パスの数。

db2gRestoreStruct データ構造固有のパラメーター

iSourceDBAliasLen
入力。**piSourceDBAlias** パラメーターの長さ (バイト単位) を指定します。

iTargetDBAliasLen
入力。**piTargetDBAlias** パラメーターの長さ (バイト単位) を指定します。

iApplicationIdLen
入力。**poApplicationId** パラメーターの長さ (バイト単位) を指定します。SQLU_APPLID_LEN + 1 と等しくなければなりません。定数 SQLU_APPLID_LEN は、include ディレクトリー内にある sqlutil ヘッダー・ファイルで定義されています。

iTimestampLen
入力。**piTimestamp** パラメーターの長さ (バイト単位) を指定します。

iTargetDBPathLen
入力。**piTargetDBPath** パラメーターの長さ (バイト単位) を指定します。

iReportFileLen
入力。**piReportFile** パラメーターの長さ (バイト単位) を指定します。

iUsernameLen
入力。**piUsername** パラメーターの長さ (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。**piPassword** パラメーターの長さ (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。

iNewLogPathLen
入力。**piNewLogPath** パラメーターの長さ (バイト単位) を指定します。

iLogTargetLen
入力。**piLogTarget** パラメーターの長さ (バイト単位) を指定します。

iRedirectScriptLen
入力。**piRedirectScript** 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。スクリプト名が提供されない場合は、ゼロに設定してください。

db2Char データ構造パラメーター

pioData

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

入力。**pioData** バッファーのサイズ。

oLength

出力。 **pioData** バッファー内にあるデータの有効文字の数。

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) が既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、リストア実行のためにリストア・ユーティリティーが使用され、(呼び出し側アプリケーションを含む) いずれかのアプリケーションが同じワークステーション上のいずれかのデータベースに既に接続されている場合は、要求が失敗します。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、それが使用できないのでない限り、バックアップ・コピーによって置換されません。この場合、ファイルが置換されるときに警告メッセージが戻されます。
- db2Backup API を使ってデータベースまたは表スペースをバックアップ済みでなければなりません。
- 呼び出し側アクションの値が 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。呼び出し側アクションの値が 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、何らかのユーザー対話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。ユーザー対話の処理が終了した後、アプリケーションは、後続の呼び出しで処理を継続するか (DB2RESTORE_CONTINUE) それとも終了するか (DB2RESTORE_TERMINATE) を示す呼び出し側アクション値を設定して、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** に 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値を DB2RESTORE_DEVICE_TERMINATE に設定してください。例えば、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値 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値を DB2RESTORE_PARM_CHK に設定してください。
- リダイレクト・リストアを実行する場合には、呼び出し側アクション値を DB2RESTORE_RESTORE_STORDEF に設定します (sqlbstsc API と組み合わせて使用します)。
- 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・リカバリー用に構成されておらず、しかも (これらのいずれかのパラメーターが有効になっている) 使用できる現行の構成ファイルが存在する場合には、ユーザーはリストアの後、データベースの新しいバックアップを取るか、ログ保持パラメーターとユーザー出口パラメーターを無効にしてから、データベースに接続する必要があります。
- 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)、使用できなくなります。
- バックアップ内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の上にリストアされます。事実上、リストア対象のバックアップが取られた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・リカバリーが有効だった場合には、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に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できるようになります。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、DB2 サーバー上に存在する有効な完全修飾パス名を前提として、**LOGTARGET** オプション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、リストア操作が失敗してエラーが戻されます。

- RESTORE 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操作時に、抽出できないログ・ファイルがあった場合、リストア操作は失敗し、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** パス内の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、そのリストアは即時に終了し、エラーが戻されます。
- 自動増分リストア操作時には、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。手動増分リストア操作時には、**LOGTARGET** パスは、最後の RESTORE コマンド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、DB2 データベース・システムはそのことを検出し、データはリストア前に自動的に解凍されます。 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。db2Restore API でライブラリーが指定されていない場合、バックアップ・イメージに保管されているライブラリーが使用されます。バックアップ・イメージ内に保管されているライブラリーがない場合、データは解凍できず、リストア操作は失敗します。
- 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上で実行する必要があります。プラットフォームが異なる場合、2 つのシステムを扱うクロスプラットフォーム・リストア操作が通常は DB2 データベース・システムによってサポートされるような場合でも、リストア操作が失敗します。
- 自動ストレージに使用できるデータベースをリストアする場合、そのデータベースに関連したストレージ・パスは再定義することもできますし、以前のものをそのまま保持しておくこともできます。ストレージ・パス定義をそのまま保持するには、ストレージ・パスをリストア操作の一環として指定しないでください。指定する場合は、データベースに関連付ける新しいセットのストレージ・パスを指定してください。自動ストレージ表スペースは、リストア操作時に自動的に新規ストレージ・パスにリダイレクトされます。

スナップショット・リストア

従来の (スナップショットでない) リストアの場合と同じように、スナップショット・バックアップ・イメージのリストア時のデフォルトの動作は、ログ・ディレクトリーをリストアしない (DB2RESTORE_LOGTARGET_EXCLUDE) というものです。

いずれかのログ・ディレクトリーのグループ ID がリストアする他のパスのいずれかと共有されていることが DB2 マネージャーにより検出された場合、エラーが戻されます。その場合、ログ・ディレクトリーがリストアの一部でなければならないため、DB2RESTORE_LOGTARGET_INCLUDE または DB2RESTORE_LOGTARGET_INCFORCE を指定する必要があります。

DB2 マネージャーは、バックアップ・イメージからのパスのリストアが行われる前に既存のログ・ディレクトリー (1 次、ミラー、およびオーバーフロー) を保存するために、すべての方法を試みます。

ログ・ディレクトリーをリストアする場合、ディスク上に事前に存在するログ・ディレクトリーがバックアップ・イメージ中のログ・ディレクトリーと競合することが DB2 マネージャーによって検出されたなら、DB2 マネージャーによってエラーが報告されます。そのような場合、DB2RESTORE_LOGTARGET_INCFORCE を指定しているなら、そのエラーは抑止され、イメージ中のログ・ディレクトリーがリストアされ、それ以前から存在していたものはすべて削除されます。

特殊なケースとして DB2RESTORE_LOGTARGET_EXCLUDE オプションが指定されていて、ログ・ディレクトリーのパスがデータベース・ディレクトリー (例えば、/NODExxxx/SQLxxxxx/SQLOGDIR/) の下にあるという場合があります。この場合は、リストアによりログ・ディレクトリーはデータベース・パスとして上書きされ、その下位にあるすべての内容はリストアされます。このシナリオに該当することが DB2 マネージャーによって検出された場合、そのログ・ディレクトリー中にログ・ファイルが存在しているなら、エラーが報告されます。DB2RESTORE_LOGTARGET_EXCLUDE を指定した場合には、このエラーは抑止され、バックアップ・イメージ中のログ・ディレクトリーによって、ディスク上の競合ログ・ディレクトリーが上書きされます。

db2Rollforward - データベースのロールフォワード

データベースのログ・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・リカバリーを使用し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になってい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** またはデータベース構成パラメーター **logarchmeth2** が OFF 以外の値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・データベース環境の場合、この API はカタログ・パーティションから呼び出す必要があります。ロールフォワードされるパーティションは、TO 節に何を指定するかによって異なります。

- ポイント・イン・タイム・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。
- END OF LOGS ロールフォワードの呼び出しは、ON DATABASE PARTITION 節で指定されるデータベース・パーティション・サーバーに影響します。データベース・パーティション・サーバーが指定されていない場合、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・サーバーに影響します。
- バックアップの終了を指定するデータベースまたは表スペースのロールフォワード呼び出しは、db2nodes.cfg ファイルの中でリストが示されているすべてのデータベース・パーティション・サーバーに影響します。

特定のデータベース・パーティション・サーバー上のすべてのトランザクションが現在のデータベースに既に適用済みであり、したがってそれらのトランザクションのどれもロールフォワードの必要がない場合、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。

パーティション表を特定のポイント・イン・タイムまでロールフォワードする際には、その表を含む表スペースも、その同じポイント・イン・タイムまでロールフォワードする必要があります。しかし、表スペースをロールフォワードする際、その表スペース内のすべての表をロールフォワードする必要はありません。

許可

以下のいずれか。

- *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 インクルード・ファイル

db2ApiDf.h

API とデータ構造構文

```
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;
  db2Uint32 oRollforwardFlags;
} db2RfwdOutputStruct;

SQL_STRUCTURE sqlurf_newlogpath
{
   SQL_PDB_NODE_TYPE nodenum;
   unsigned short  pathlen;
   char logpath[SQL_LOGPATH_SZ+SQL_LOGFILE_NAME_SZ+1];
};

typedef SQL_STRUCTURE sqlu_tablespace_bkrst_list
{
   sqlint32 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;

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];
};

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;
```

```
typedef 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;
```

db2Rollforward 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 構造を指すポインター。

db2RollforwardStruct データ構造パラメーター

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。

db2RfwdInputStruct データ構造パラメーター

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。有効な値は以下のとおりです (インクルード・ディレクトリーの db2ApiDf ヘッダー・ファイルで定義される)。

DB2ROLLFORWARD_ROLLFWD
piStopTime パラメーターで指定されたポイント・イン・タイムへ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。ポイント・イン・タイムへの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進行中状態のままになります。

DB2ROLLFORWARD_STOP
使用できるログ・ファイルを使用してデータベースをロールフォワードしてからロールバックすることで、ロールフォワード・リカバリーを終了します。非コミットのトランザクションはバックアウトされ、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。この値の同義語は DB2ROLLFORWARD_RFWD_COMPLETE です。

DB2ROLLFORWARD_RFWD_STOP
piStopTime で指定されたポイント・イン・タイムへ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。この値の同義語は 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 (例えば、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 (例えば、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。アクティブ・ログ・ファイルに加えて、アーカイブ・ログ・ファイルを (ユーザーが) logpath に移動しておかないと、このユーティリティーからこれらのファイルを使用できるようにはなりません。このことは、データベースが logpath に十分なスペースを持ってい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 で検索され、次にオーバーフロー・ログ・パスで検索されます。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くことができます。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のデフォルト値は 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。有効な値は以下のとおりです (インクルード・ディレクトリーの db2ApiDf ヘッダー・ファイルで定義される)。

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、または特定の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。

パーティション表の場合、パーティション表のいずれかの部分が入った表スペースのポイント・イン・タイム (PIT) ロールフォワードを行うためには、その表が置かれている他のすべての表スペースも、同じポイント・イン・タイムにロールフォワードする必要があります。パーティション表の一部を収めた 1 つの表スペースのログの最後までロールフォワードすることも可能です。

アタッチ、デタッチ、またはドロップされたデータ・パーティションがあるパーティション表の場合、そのようなデータ・パーティションの表スペースもすべて PIT ロールフォワードに含める必要があります。パーティション化された表にアタッチ、デタッチ、またはドロップされたデータ・パーティションがあるかどうかを判別するには、SYSDATAPARTITIONS カタログ表の状況フィールドを照会します。

パーティション表は複数の表スペース内に置かれていることがあるので、一般的に、複数の表スペースのロールフォワードが必要になります。ドロップされた表のリカバリーを介してリカバリーされるデータは、piExportDir パラメーターに指定されたエクスポート・ディレクトリーに書き込まれます。すべての表スペースを 1 つのコマンドでロールフォワードできますが、関与する表スペースのサブセットのロールフォワード操作を繰り返してもかまいません。表のすべてのデータをリカバリーするのに必要な表スペースの完全セットを db2Rollforward API に指定しなかった場合、通知ログに警告が書き込まれます。管理通知ログに書かれているコマンドでリカバリーされなかったすべてのパーティションの完全詳細を示した警告が、ユーザーに戻されます。

表スペースのサブセットのロールフォワードが可能になれば、1 つのエクスポート・ディレクトリーに入り切らないほどのデータをリカバリーするようなケースの処理が簡単になります。

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。これはデフォルト値です。この値を使用する場合は piNodeList パラメーターを NULL に設定する必要があります。

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。この値を使用する場合は piNodeList パラメーターを NULL に設定する必要があります。

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。パーティション表の場合、drop-table-id は表全体を示します。したがって、表のすべてのデータ・パーティションを単一のロールフォワード・コマンドでリカバリーすることができます。

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の名前。

iRollforwardFlags
入力。ロールフォワード・フラグを指定します。有効な値は以下のとおりです (インクルード・ディレクトリーの db2ApiDf ヘッダー・ファイルで定義される)。

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、特定のポイント・イン・タイムにロールフォワードできるようにしま

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・ログの検索を無効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。ロールフォワードするログ・ファイルを制御することによって、待機マシンが実動マシンより必ず X 時間遅れるようにして、片方のシステムしかユーザーから影響を受けないようにすることができます。待機システムがアーカイブにアクセスせず、例えば 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。

DB2ROLLFORWARD_END_OF_BACKUP
データベースを*最小リカバリー時間* までロールフォワードする必要があることを指定します。

db2RfwdOutputStruct データ構造パラメーター

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。

oRollforwardFlags
出力。ロールフォワード出力フラグ。有効な値は以下のとおりです。

DB2ROLLFORWARD_OUT_LOCAL_TIME
最後にコミットされたトランザクションのタイム・スタンプは、UTC ではなく現地時間で表示されることをユーザーに示します。現地時間は、クライアントではなくサーバーの現地時間をベースにします。パーティション・データベース環境では、現地時間は、カタログ・パーティションの現地時間をベースにします。

sqlurf_newlogpath データ構造パラメーター

nodenum
入力。この構造が特定するデータベース・パーティション番号。

pathlen
入力。logpath フィールドの全長。

logpath
入力。特定ノードのロールフォワード操作で使用される完全修飾パス。

sqlu_tablespace_bkrst_list データ構造パラメーター

num_entry
入力。tablespace パラメーターが指すリストに載っている構造の数。

tablespace
入力。sqlu_tablespace_entry 構造のリストを指すポインター。

sqlu_tablespace_entry データ構造パラメーター

reserve_len
入力。tablespace_entry パラメーターの長さをバイト数で指定します。

tablespace_entry
入力。ロールフォワードする表スペースの名前。

filler メモリー内のデータ構造を正しく配置するために使用される充てん文字。

sqlurf_info データ構造パラメーター

nodenum
出力。この構造内に情報を収められたデータベース・パーティションの番号。

状態 出力。データベース・パーティションに対するロールフォワードに組み入れられたデータベースまたは表スペースの現在の状態。

nextarclog
出力。ロールフォワードが完了済みの場合、このフィールドは空です。ロールフォワードが未完了の場合、これは、ロールフォワードで処理される次のログ・ファイルの名前になります。

firstarcdel
出力。ロールフォワードで再生される最初のログ・ファイル。

lastarcdel
出力。ロールフォワードで再生される最後のログ・ファイル。

lastcommit
出力。最後にコミットされたトランザクションの時刻。

db2gRfwdInputStruct データ構造固有のパラメーター

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。

iDroppedTblIDLen
入力。ドロップ済みの表 ID (piDroppedTblID パラメーター) の長さをバイト数で指定します。ドロップ済みの表 ID が提供されていない場合は、ゼロに設定してください。

iExportDirLen
入力。ドロップ済みの表のエクスポート・ディレクトリー (piExportDir パラメーター) の長さをバイト数で指定します。ドロップ済みの表のエクスポート・ディレクトリーが提供されていない場合は、ゼロに設定してください。

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以後にデータベースで実行されたトランザクションを再構成します。

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) を使用して照会することができます。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 つ以上の表スペースが SQLB_ROLLFORWARD_PENDING または SQLB_ROLLFORWARD_IN_PROGRESS 状態にある場合、このフラグは 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 つ以上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、あるいは名前で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 を呼び出した時点では、SQLB_ROLLFORWARD_IN_PROGRESS 状態にある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入っ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、情報が使用できる限り、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。 lastcommit 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できなくなります。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 に変換されるため、この時刻はクライアントではなく、サーバーのタイム・ゾーンのローカル時刻になります。クライアントとサーバーのタイム・ゾーンが異なっている場合、サーバーのローカル時刻が使用されます。これは、クライアントのローカル時刻となるコントロール・センターのローカル時刻オプションとは異なります。タイム・スタンプ・ストリングが夏時間調整によるクロックの時刻

変更に近い場合は、停止時間が時刻変更の前か後かを確認し、正しく指定しておくことが重要です。

db2Runstats - 表および索引の統計情報の更新

表、関連する索引、統計ビューのすべてまたはいずれか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。

表に使用する場合、このユーティリティーは、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呼び出します。

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の部分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、表の部分を含んでいる必要はありません。

- 表の部分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。
- 表の部分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、表の部分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。統計ビューについての統計を収集する場合、すべてのデータベース・パーティションについて統計が収集されます。

統計ビューに使用された場合、このユーティリティーは、基礎表への変更が、ビューによって戻される行に大きな影響を与えた時に呼び出される必要があります。この種のビューは、″ALTER VIEW ... ENABLE QUERY OPTIMIZATION″ を使用した照会最適化に利用できるようにしておく必要があります。

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

表に使用される場合、以下のいずれか。

- sysadm
- sysctrl
- sysmaint
- 表に対する CONTROL 特権
- LOAD

統計ビューに使用される場合、以下のいずれか。

- sysadm
- sysctrl
- sysmaint
- dbadm
- ビューに対する CONTROL 特権

それに加え、ユーザーは、ビューの行にアクセスできる適切な権限または特権を持っている必要があります。特に、各表、ビュー、ビュー定義の中で参照されるニックネームに対して、ユーザーは以下のいずれかの権限または特権を持っている必要があります。

- sysadm または dbadm
- CONTROL 特権
- SELECT 特権

必要な接続

データベース

API インクルード・ファイル

db2ApiDf.h

API とデータ構造構文

```
SQL_API_RC SQL_API_FN
  db2Runstats (
                db2Uint32 versionNumber,
                void * data,
                struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2RunstatsData
{
  double iSamplingOption;
  unsigned char *piTablename;
  struct db2ColumnData        **piColumnList;
  struct db2ColumnDistData    **piColumnDistributionList;
  struct db2ColumnGrpData     **piColumnGroupList;
  unsigned char               **piIndexList;
  db2Uint32 iRunstatsFlags;
  db2int16 iNumColumns;
  db2int16 iNumColdist;
  db2int16 iNumColGroups;
  db2int16 iNumIndexes;
  db2int16 iParallelismOption;
  db2int16 iTableDefaultFreqValues;
  db2int16 iTableDefaultQuantiles;
  db2Uint32 iSamplingRepeatable;
  db2Uint32 iUtilImpactPriority;
} 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;

SQL_API_RC SQL_API_FN
  db2gRunstats (
                db2Uint32 versionNumber,
                void * data,
                struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2gRunstatsData
{
  double iSamplingOption;
  unsigned char *piTablename;
  struct db2gColumnData       **piColumnList;
  struct db2gColumnDistData   **piColumnDistributionList;
  struct 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;
} 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;
```

db2Runstats 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。

データ 入力。db2RunstatsData 構造を指すポインター。

pSqlca 出力。 sqlca 構造を指すポインター。

db2RunstatsData データ構造パラメーター

iSamplingOption
入力。表またはビューの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値 P として表します。この値は、100 以下の正の数でなければなりませんが、0 から 1 の間の数であるということはあり得ます。例えば、値 0.01 は 1 % の 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、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または統計ビュー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。例えば、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。

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 (インクルード・ディレクトリーの db2ApiDf ヘッダー・ファイルで定義される)。

DB2RUNSTATS_ALL_COLUMNS

表または統計ビュー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、表またはビュー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、統計ビューには使用できません。表に関しては、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。XML

タイプ列は、定義によるとキー列ではなく、iRunstatsFlags パラメーターの値が DB2RUNSTATS_KEY_COLUMNS に設定された場合の統計収集には含まれません。

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、分散統計は表または統計ビュー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。このオプションは、統計ビューには使用できません。

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。このオプションは、統計ビューには使用できません。

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。このオプションは、統計ビューには使用できません。

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。

SYSTEM サンプリングは、統計ビューには指定できません。ビュー・データのサンプリングには、BERNOULLI サンプリングのみ使用できます。

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または統計ビュー・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
表またはビューのカタログに既に登録済みの統計プロファイルを使って、表または統計ビュー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、実際には表またはビューの統計を収集しません。

DB2RUNSTATS_UNSET_PROFILE
SYSCAT.STATISTICS_PROFILE を NULL に設定することにより統計プロファイルを設定解除すると、システム・カタログから統計プロファイルが除去されます。統計プロファイルが存在しない場合、それを設定解除しようとするとエラーになります (SQLCODE -2315)。

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_UPDA_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、実際には表またはビューの統計を収集しません。

DB2RUNSTATS_EXCLUDING_XML
XML タイプ列についての統計を収集しません。統計は、XML タイプではない、すべての指定列に関して収集されます。このオプションは、XML 列を指定する他のすべてのメソッドに優先します。

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。将来の利用のために予約されています。有効な値は 0 です。

iTableDefaultFreqValues
入力。表またはビュー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。表またはビュー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。

iSamplingRepeatable
入力。表またはビューの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、表またはビューに対する何らかのアクティビティーによって表またはビューの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。

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。70 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。このオプションは、統計ビューには使用できません。

db2ColumnData データ構造パラメーター

piColumnName
入力。列名を表すストリングを指すポインター。

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。

db2ColumnDistData データ構造パラメーター

piColumnName
入力。列名を表すストリングを指すポインター。

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。例えばこれには、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、例えば 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。

db2ColumnGrpData データ構造パラメーター

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。

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。将来の利用のために予約されています。

iNumQuantiles
入力。将来の利用のために予約されています。

db2gRunstatsData データ構造固有パラメーター

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。

iTablenameLen
入力。表名またはビュー名の長さを示す値 (バイト単位)。

db2gColumnData データ構造固有パラメーター

iColumnNameLen
入力。列名の長さを示す値 (バイト単位)。

db2gColumnDistData データ構造固有パラメーター

iColumnNameLen
入力。列名の長さを示す値 (バイト単位)。

db2gColumnGrpData データ構造固有パラメーター

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)。

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 (例えば、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
- ビューによって戻される行を変更するために、ビューの基礎となる表の内容が大きく変更された場合

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持されます。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。 db2Runstats API が XML 列についてのみ統計を収集している場合は、以前に収集された基礎列統計と分散統計は保持されます。 XML 列についての統計が既に以前に収集されていた場合、その統計は、現行の db2Runstats API の呼び出しによって XML 列についての統計の収集が行われていなければドロップされますが、現行の db2Runstats API の呼び出しによって XML 列についての統計の収集が行われていれば、その統計に置き換えられます。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。

iRunstatsFlags パラメーターが値 DB2RUNSTATS_EXCLUDING_XML に設定されていれば、XML 列に関する統計収集は行われません。この値は、XML 列を指定する他のすべてのメソッドに優先します。

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。統計ビュー

を利用する可能性のある照会を含むパッケージは、そうしたビューについての統計が更新された後には、再バインドが必要になります。

統計ビューについての統計が収集される際には、SQL 照会が内部で実行されます。照会のために選択されたアクセス・プランを調べ、統計収集に関して何かパフォーマンス上の問題がないかを確認するには、EXPLAIN 機能を使用できます。EXPLAIN 表に照会アクセス・プランを保管するには、CURRENT EXPLAIN MODE 特殊レジスターを YES に設定します。

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。例えば、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 (FIRSTKEYCARD は、SYSCAT.INDEXES および SYSSTAT.INDEXES カタログ・ビューのカタログ統計フィールド) 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。例えば、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 (COLCARD は SYSCAT.COLUMNS および SYSSTAT.COLUMNS カタログ・ビューのカタログ統計フィールド) 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。

db2SelectDB2Copy - アプリケーションで使用する DB2 コピーの選択

特定の DB2 コピーまたは指定の場所を使用するためにアプリケーションに必要な環境を設定します。使用する DB2 コピー用に環境が既にセットアップされている場合、この API を呼び出す必要はありません。しかし、別の DB2 コピーを使用する必要がある場合は、この API を呼び出さなければなりません。プロセス内の DB2 dll ファイルをロードする前にこの API を呼び出します。この呼び出しはプロセスごとに 1 度のみ行うことができます。

許可

なし

必要な接続

なし

API インクルード・ファイル

```
db2ApiInstall.h
```

API とデータ構造構文

```
SQL_API_RC SQL_API_FN
  db2SelectDB2Copy (
   db2Uint32 versionNumber,
   void *pDB2SelectDB2CopyStruct);

typedef enum DB2CopyParmType
```

```
{
   DB2CopyInvalid=0,
   DB2CopyName,
   DB2CopyPath
} db2CopyParmType;

typedef struct DB2SelectDB2CopyStruct
{
   DB2CopyParmType  Type;
   char *psziDB2Copy;
} db2SelectDB2CopyStruct
```

db2SelectDB2Copy API パラメーター

versionNumber
入力。2 番目のパラメーター pDB2SelectInstallationStruct として渡される変数のバージョン番号およびリリース・レベルを指定します。

pDB2SelectDB2CopyStruct
入力。DB2SelectDB2CopyStruct 構造を指すポインター。

DB2SelectDB2CopyStruct データ構造パラメーター

Type 入力。これは、DB2CopyName または DB2CopyPath のいずれかです。

psziDB2Copy
入力。Type が DB2CopyName に指定されている場合、psziDB2Copy が DB2 コピーの名前です。Type が db2CopyPath に指定されている場合、psziDB2Copy が DB2 インストール・パスの名前です。これを NULL にすることはできません。

使用上の注意

API を使用するには、アプリケーションが静的に db2ApiInstall.lib にリンクするように強制する db2ApiInstall.h を含める必要があります。

さらに、この API は、DB2 ライブラリーのロード前に呼び出す必要があり、アプリケーションごとに 1 度のみ呼び出すことができます。 DB2 ライブラリーのロードは、/delayload オプションを DB2 ライブラリーのリンク時に使用することにより回避できます。または、LoadLibraryEx を使用し、LOAD_WITH_ALTERED_SEA を指定して、これらのライブラリーを動的にロードできます。

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 インクルード・ファイル

db2ApiDf.h

API とデータ構造構文

```
SQL_API_RC SQL_API_FN
  db2SetSyncSession (
   db2Uint32 versionNumber,
   void * pParmStruct,
   struct sqlca * pSqlca);

typedef struct db2SetSyncSessionStruct
{
   char *piSyncSessionID;
} db2SetSyncSessionStruct;
```

db2SetSyncSession 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 構造を指すポインター。

db2SetSyncSessionStruct データ構造パラメーター

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。

---

db2SetWriteForDB - データベースの入出力書き込みの中断または再開

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、同じ接続を維持し、書き込みを中断および再開してください。

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。

- sysadm
- sysctrl
- sysmaint

必要な接続

データベース

API インクルード・ファイル

db2ApiDf.h

API とデータ構造構文

```
SQL_API_RC SQL_API_FN
  db2SetWriteForDB (
   db2Uint32 versionNumber,
   void * pParmStruct,
   struct sqlca * pSqlca);

typedef struct db2SetWriteDbStruct
{
   db2int32 iOption;
   char *piTablespaceNames;
} db2SetWriteDbStruct;
```

db2SetWriteForDB 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・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 構造を指すポインター。

db2SetWriteDbStruct データ構造パラメーター

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。

db2SpmListIndTrans - SPM 未確定トランザクションのリスト

同期点マネージャーで未確定であるトランザクションのリストを表示します。

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

なし

必要な接続

同期点マネージャーとの接続。

API インクルード・ファイル

```
sqlxa.h
```

API とデータ構造構文

```
SQL_API_RC SQL_API_FN
db2SpmListIndTrans (
  db2Uint32 versionNumber,
  void * pParmStruct,
  struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2SpmListIndTransStruct
{
db2SpmRecoverStruct * piIndoubtData;
db2Uint32             iIndoubtDataLen;
db2Uint32             oNumIndoubtsReturned;
db2Uint32             oNumIndoubtsTotal;
db2Uint32             oReqBufferLen;
} db2XaListIndTransStruct;

typedef SQL_STRUCTURE db2SpmRecoverStruct
{
   SQLXA_XID    xid;
   char         luwid[SQLCSPQY_LUWID_SZ+1];
   char         corrtok[SQLCSPQY_APPLID_SZ+1];
   char         partner[SQLCSPQY_LUNAME_SZ+1];
   char         dbname[SQLCSPQY_DBNAME_SZ+1];
   char         dbalias[SQLCSPQY_DBNAME_SZ+1];
   char         role;
   char         uow_status;
   char         partner_status;
} db2SpmRecoverStruct;
```

db2SpmListIndTrans API パラメーター

versionNumber
入力。バージョンおよびリリース・レベルを指定します。

pParmStruct
入力。db2SpmListIndTransStruct 構造を指すポインター。

pSqlca 出力。 sqlca 構造を指すポインター。

db2SpmListIndTransStruct データ構造パラメーター

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 db2SpmRecoverStruct の形式です。アプリケーションは db2SpmRecoverStruct 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、必要なバッファー・サイズが計算され、oReqBufferLen で戻されます。 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 呼び出し時に使用できる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。

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。

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。

db2SpmRecoverStruct データ構造パラメーター

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。

luwid 出力。パートナー・システムで XA ID (XID) を識別するために同期点マネージャーが割り当てる作業論理単位 ID (LUWID) を指定します。

corrtok

出力。同期点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。

partner

出力。パートナー・システムの名前を指定します。

dbname

出力。パートナー・システムのデータベース

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。

role 出力。同期点マネージャーの役割。

SQLCSPQY_AR

同期点マネージャーは、アプリケーション・リクエスターです

SQLCSPQY_AS
同期点マネージャーは、アプリケーション・サーバーです

uow_status
出力。同期点マネージャーでの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。

SQLCSPQY_STATUS_COM
トランザクションは、同期点マネージャーでコミット状況にあります。トランザクションは、次の再同期インターバルでパートナー・システムによって再同期されるのを待機しています。

SQLCSPQY_STATUS_RBK
トランザクションは、同期点マネージャーでロールバック状況にあります。パートナー・システムが再同期を開始して未確定を解決するのを待機中です。

SQLCSPQY_STATUS_IDB
トランザクションは、同期点マネージャーで準備された状態にあります。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。

SQLCSPQY_STATUS_HCM
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。

SQLCSPQY_STATUS_HRB
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。

使用上の注意

一般的なアプリケーションは、現行の接続を同期点マネージャー*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、db2SpmListIndTransAPI 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 の最初の呼び出しのため、追加の未確定トランザクションがある場合は、このバッファーに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 oNumIndoubtsTotal 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。

* 同期点マネージャーに接続するには、DB2 Connect サーバーで使用されている同期点マネージャーの名前を判別します。これは、DB2 Connect サーバーでデータベース構成パラメーター spm_name を照会して判別できます。spm_name を接続 API でデータベースの別名として指定し、接続を発行します。

db2SyncSatellite - サテライト同期化の開始

サテライトを同期化します。サテライト同期化では、サテライトを、同じグループの他のサテライトと整合した状態にするための操作が行われます。

許可

なし

必要な接続

なし

API インクルード・ファイル

db2ApiDf.h

API とデータ構造構文

```
SQL_API_RC SQL_API_FN
  db2SyncSatellite (
   db2Uint32 versionNumber,
   void * pParmStruct,
   struct sqlca * pSqlca);
```

db2SyncSatellite 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 構造を指すポインター。

db2SyncSatelliteStop - サテライト同期化の一時停止

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 インクルード・ファイル

db2ApiDf.h

API とデータ構造構文

```
SQL_API_RC SQL_API_FN
  db2SyncSatelliteStop (
   db2Uint32 versionNumber,
   void * pParmStruct,
   struct sqlca * pSqlca);
```

db2SyncSatelliteStop 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 構造を指すポインター。

---

db2SyncSatelliteTest - サテライトが同期化可能かのテスト

サテライトが同期化できるかどうか、つまり、グループ内の他のサテライトと整合した状態にすることができるかをテストします。

許可

なし

必要な接続

なし

API インクルード・ファイル

```
db2ApiDf.h
```

API とデータ構造構文

```
SQL_API_RC SQL_API_FN
  db2SyncSatelliteTest (
   db2Uint32 versionNumber,
   void * pParmStruct,
   struct sqlca * pSqlca);
```

db2SyncSatelliteTest 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 構造を指すポインター。

db2UpdateAlertCfg - ヘルス・インディケーターのアラート構成設定の更新

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。

- sysadm
- sysctrl
- sysmaint

必要な接続

インスタンス。インスタンス接続が存在しない場合は、デフォルトのインスタンス接続が作成されます。

API インクルード・ファイル

db2ApiDf.h

API とデータ構造構文

```
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;
```

db2UpdateAlertCfg 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・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 構造を指すポインター。

db2UpdateAlertCfgData データ構造パラメーター

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。

db2AlertAttrib データ構造パラメーター

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

db2AlertActionUpdate データ構造パラメーター

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。

db2AlertActionDelete データ構造パラメーター

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

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。

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。

db2AlertActionNew データ構造パラメーター

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

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。

db2AlertScriptAction データ構造パラメーター

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

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

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。

db2AlertTaskAction データ構造パラメーター

pTaskname
タスク名。

condition
アクションを実行する条件。

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。

db2UpdateAlternateServerForDB - 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。

- sysadm
- sysctrl

必要な接続

なし

API インクルード・ファイル

db2ApiDf.h

API とデータ構造構文

```
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;

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;
```

db2UpdateAlternateServerForDB 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・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 構造を指すポインター。

db2UpdateAltServerStruct データ構造パラメーター

piDbAlias
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。

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。 IP アドレスとしては、IPv4 のアドレスも IPv6 のアドレスも使用できます。

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。

db2gUpdateAltServerStruct データ構造固有パラメーター

iDbAlias_len
入力。piDbAlias の長さ (バイト単位)。

iHostName_len
入力。piHostName の長さ (バイト単位)。

iPort_len
入力。piPort の長さ (バイト単位)。

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有効である場合、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。

db2UpdateContact - 連絡先の属性の更新

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。インスタンス接続が存在しない場合は、デフォルトのインスタンス接続が作成されます。

API インクルード・ファイル

```
db2ApiDf.h
```

API とデータ構造構文

```
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;
```

db2UpdateContact 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・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 構造を指すポインター。

db2UpdateContactData データ構造パラメーター

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。

db2ContactAttrib データ構造パラメーター

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。

使用上の注意

この API は UNIX および Linux ではサポートされません。しかし、SQL インターフェースによって同様の機能を使用することはできます。

db2UpdateContactGroup - 連絡先グループの属性の更新

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 インクルード・ファイル

```
db2ApiDf.h
```

API とデータ構造構文

```
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;
```

db2UpdateContactGroup 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・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 構造を指すポインター。

db2UpdateContactGroupData データ構造パラメーター

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。

db2ContactTypeData データ構造パラメーター

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。

使用上の注意

この API は UNIX および Linux ではサポートされません。しかし、SQL インターフェースによって同様の機能を使用することはできます。

---

db2UpdateHealthNotificationList - ヘルス・アラート通知を送信できる連絡先リストの更新

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。

- sysadm
- sysctrl
- sysmaint

必要な接続

インスタンス。インスタンス接続が存在しない場合は、デフォルトのインスタンス接続が作成されます。

API インクルード・ファイル

```
db2ApiDf.h
```

API とデータ構造構文

```
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;
```

db2UpdateHealthNotificationList 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・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 構造を指すポインター。

db2UpdateHealthNotificationListData データ構造パラメーター

iNumUpdates
入力。更新の数。

piUpdates
入力。db2HealthNotificationListUpdate 構造を指すポインター。

db2HealthNotificationListUpdate データ構造パラメーター

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。

db2ContactTypeData データ構造パラメーター

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。

db2UtilityControl - 実行されているユーティリティーの優先順位の設定

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

以下のいずれか。

- sysadm
- sysctrl
- sysmaint

必要な接続

インスタンス

API インクルード・ファイル

```
db2ApiDf.h
```

API とデータ構造構文

```
SQL_API_RC SQL_API_FN
  db2UtilityControl (
   db2Uint32 version,
   void * pUtilityControlStruct,
   struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2UtilityControlStruct
{
   db2Uint32 iID;
   db2Uint32 iAttribute;
   void *pioValue;
} db2UtilityControlStruct;

SQL_API_RC SQL_API_FN
  db2gUtilityControl (
   db2Uint32 version,
   void * pgUtilityControlStruct,
   struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2gUtilityControlStruct
{
   db2Uint32 iID;
   db2Uint32 iAttribute;
   void *pioValue;
} db2gUtilityControlStruct;
```

db2UtilityControl API パラメーター

version

入力。2 番目のパラメーター pUtilityControlStruct 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 構造を指すポインター。

db2UtilityControlStruct データ構造パラメーター

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。有効な値は以下のとおりです (インクルード・ディレクトリーの db2ApiDf ヘッダー・ファイルで定義される)。

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 - アプリケーション・プログラムのバインドによるパッケージの作成

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。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。

- sysadm または dbadm の権限
- 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ない場合は、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。
- パッケージが存在する場合は、スキーマに対する ALTERIN 特権
- パッケージに対する BIND 特権 (パッケージが存在する場合)

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 インクルード・ファイル

```
sql.h
```

API とデータ構造構文

```
SQL_API_RC SQL_API_FN
  sqlabndx (
        _SQLOLDCHAR * pBindFileName,
        _SQLOLDCHAR * pMsgFileName,
        struct sqlopt * pBindOptions,
        struct sqlca * pSqlca);

SQL_API_RC SQL_API_FN
  sqlgbndx (
        unsigned short MsgFileNameLen,
        unsigned short BindFileNameLen,
        struct sqlca * pSqlca,
        struct sqlopt * pBindOptions,
        _SQLOLDCHAR * pMsgFileName,
        _SQLOLDCHAR * pBindFileName);
```

sqlabndx API パラメーター

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・ファイルの名前の前にはアットマーク (@) を付けます。例えば、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 つ以上のバインド・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・ファイル名は複数行になることがあります。例えば、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。

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。ファイルが既に存在する場合、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。

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。

pSqlca 出力。 sqlca 構造を指すポインター。

sqlgbndx API 固有パラメーター

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。ファイルが既に存在する場合、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。

BindFileNameLen
入力。pBindFileName パラメーターの長さ (バイト単位)。

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)。例えば、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。例えば、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.h で定義されています。

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。

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 インクルード・ファイル

```
sql.h
```

API とデータ構造構文

```
SQL_API_RC SQL_API_FN
  sqlaintp (
        char * pBuffer,
        short BufferSize,
        short LineWidth,
        const char * pMsgFileName,
        struct sqlca * pSqlca);

SQL_API_RC SQL_API_FN
  sqlgintp (
        short BufferSize,
        short LineWidth,
        struct sqlca * pSqlca,
        _SQLOLDCHAR * pBuffer);
```

sqlaintp API パラメーター

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 ストリング終了文字を見込んで行われます。

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 構造を指すポインター。

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、使用できる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。

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。

sqlaprep - アプリケーション・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。

- sysadm または dbadm の権限
- 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ない場合は、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。
- パッケージが存在する場合は、スキーマに対する ALTERIN 特権
- パッケージに対する BIND 特権 (パッケージが存在する場合)

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 インクルード・ファイル

```
sql.h
```

API とデータ構造構文

```
SQL_API_RC SQL_API_FN
  sqlaprep (
        _SQLOLDCHAR * pProgramName,
        _SQLOLDCHAR * pMsgFileName,
        struct sqlopt * pPrepOptions,
        struct sqlca * pSqlca);

SQL_API_RC SQL_API_FN
  sqlgprep (
        unsigned short MsgFileNameLen,
        unsigned short ProgramNameLen,
        struct sqlca * pSqlca,
        struct sqlopt * pPrepOptions,
        _SQLOLDCHAR * pMsgFileName,
        _SQLOLDCHAR * pProgramName);
```

sqlaprep API パラメーター

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: UNIXC++ 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・システムにおいて、組み込み SQL を含む C++ アプリケーションの場合に望ましい拡張子は sqC です。しかし、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のための sqx 規則も通用します。

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。ファイルが既に存在する場合、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。

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。

pSqlca 出力。 sqlca 構造を指すポインター。

sqlgprep API 固有パラメーター

MsgFileNameLen

入力。pMsgFileName パラメーターの長さ (バイト単位)。

ProgramNameLen

入力。pProgramName パラメーターの長さ (バイト単位)。

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、接続が既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。

PRECOMPILE コマンドまたは sqlaprep API を使用する際、パッケージの名前は PACKAGE USING オプションを使用して指定できます。このオプションを使用すると、パッケージ名には最大 128 バイトまで指定できます。このオプションを使用しない場合には、パッケージの名前はプリコンパイラーによって生成されます。アプリケーション・プログラムのソース・ファイルの名前 (拡張子を取り去って大文字に変換したもの) には、最大 8 文字まで使用できていました。生成される名前も、旧バージョンの DB2 と互換性を保つために引き続き最大 8 バイトまで可能です。

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。

sqlarbnd - パッケージの再バインド

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。

- sysadm または dbadm の権限
- 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 列に記録した許可 ID は、パッケージの最新のバインド・プログラムの ID であり、再バインド用のバインド・プログラム許可 ID として使用されます。また、パッケージの表参照のためのデフォルト・スキーマ としても使用されます。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 インクルード・ファイル

```
sql.h
```

API とデータ構造構文

```
SQL_API_RC SQL_API_FN
  sqlarbnd (
        char * pPackageName,
        struct sqlca * pSqlca,
        struct sqlopt * pRebindOptions);

SQL_API_RC SQL_API_FN
  sqlgrbnd (
        unsigned short PackageNameLen,
        char * pPackageName,
        struct sqlca * pSqlca,
        struct sqlopt * pRebindOptions);
```

sqlarbnd API パラメーター

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_VERSION_OPT 再バインド・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 構造を指すポインター。

pRebindOptions
入力。SQLOPT 構造を指すポインター。これは API に再バインド・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。

sqlgrbnd API 固有パラメーター

PackageNameLen
入力。pPackageName パラメーターの長さ (バイト単位)。

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 .fs 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。例えば、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。パッケージが依存する機能インスタンスがドロップされると、パッケージは作動不能とマークされます (SYSCAT.PACKAGES システム・カタログの VALID 列は、X と設定されます。)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。例えば、データベース・マイグレーションの後、データベースに格納されているすべてのパッケージが、MIGRATE DATABASE コマンド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。これは、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 (例えば、 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。例えば、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・エラーを検出する必要がある場合。 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で REBIND が実行された場合、他のユーザーの作業論理単位が終了するまで、再バインドは起こりません。これは、再バインド中に SYSCAT.PACKAGES システム・カタログ表中のパッケージのレコードで、排他ロックが掛けられるためです。

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_VERSION_OPT 再バインド・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。再バインド・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。

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。

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。

- sysadm
- sysctrl
- sysmaint
- dbadm

必要な接続

データベース

API インクルード・ファイル

```
sqlutil.h
```

API とデータ構造構文

```
SQL_API_RC SQL_API_FN
  sqlbctcq (
   struct sqlca * pSqlca);

SQL_API_RC SQL_API_FN
  sqlgctcq (
   struct sqlca * pSqlca);
```

sqlbctcq API パラメーター

pSqlca 出力。 sqlca 構造を指すポインター。

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。

- sysadm
- sysctrl
- sysmaint
- dbadm
- load

必要な接続

データベース

API インクルード・ファイル

```
sqlutil.h
```

API とデータ構造構文

```
SQL_API_RC SQL_API_FN
  sqlbctsq (
   struct sqlca * pSqlca);

SQL_API_RC SQL_API_FN
  sqlgctsq (
   struct sqlca * pSqlca);
```

sqlbctsq API パラメーター

pSqlca 出力。 sqlca 構造を指すポインター。

sqlbftcq - 表スペース・コンテナー中の行の照会データの取得

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フェッチします。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。

- sysadm
- sysctrl
- sysmaint
- dbadm

必要な接続

データベース

API インクルード・ファイル

```
sqlutil.h
```

API とデータ構造構文

```
SQL_API_RC SQL_API_FN
  sqlbftcq (
   struct sqlca * pSqlca,
   sqluint32 MaxContainers,
   struct SQLB_TBSCONTQRY_DATA * pContainerData,
   sqluint32 * pNumContainers);

SQL_API_RC SQL_API_FN
  sqlgftcq (
   struct sqlca * pSqlca,
   sqluint32 MaxContainers,
   struct SQLB_TBSCONTQRY_DATA * pContainerData,
   sqluint32 * pNumContainers);
```

sqlbftcq API パラメーター

pSqlca 出力。 sqlca 構造を指すポインター。

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フェッチすることもできます。

sqlbftpq - 表スペース中の行の照会データのフェッチ

指定された行数の表スペースの照会データをフェッチします。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。

- sysadm
- sysctrl
- sysmaint
- dbadm
- load

必要な接続

データベース

API インクルード・ファイル

```
sqlutil.h
```

API とデータ構造構文

```
SQL_API_RC SQL_API_FN
  sqlbftpq (
   struct sqlca * pSqlca,
   sqluint32 MaxTablespaces,
   struct SQLB_TBSPQRY_DATA * pTablespaceData,
   sqluint32 * pNumTablespaces);

SQL_API_RC SQL_API_FN
  sqlgftpq (
   struct sqlca * pSqlca,
   sqluint32 MaxTablespaces,
   struct SQLB_TBSPQRY_DATA * pTablespaceData,
   sqluint32 * pNumTablespaces);
```

sqlbftpq API パラメーター

pSqlca 出力。 sqlca 構造を指すポインター。

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フェッチすることもできます。

sqlbgtss - 表スペースの使用率に関する統計の取得

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。

- sysadm
- sysctrl
- sysmaint
- dbadm
- load

必要な接続

データベース

API インクルード・ファイル

```
sqlutil.h
```

API とデータ構造構文

```
SQL_API_RC SQL_API_FN
  sqlbgtss (
   struct sqlca * pSqlca,
   sqluint32 TablespaceId,
   struct SQLB_TBS_STATS * pTablespaceStats);

SQL_API_RC SQL_API_FN
  sqlggtss (
   struct sqlca * pSqlca,
   sqluint32 TablespaceId,
   struct SQLB_TBS_STATS * pTablespaceStats);
```

sqlbgtss API パラメーター

pSqlca 出力。 sqlca 構造を指すポインター。

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。

sqlbmtsq - すべての表スペースの照会データの取得

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。

- sysadm
- sysctrl
- sysmaint
- dbadm
- load

必要な接続

データベース

API インクルード・ファイル

```
sqlutil.h
```

API とデータ構造構文

```
SQL_API_RC SQL_API_FN
  sqlbmtsq (
   struct sqlca * pSqlca,
   sqluint32 * pNumTablespaces,
   struct SQLB_TBSPQRY_DATA *** pppTablespaceData,
   sqluint32 reserved1,
   sqluint32 reserved2);

SQL_API_RC SQL_API_FN
  sqlgmtsq (
   struct sqlca * pSqlca,
   sqluint32 * pNumTablespaces,
   struct SQLB_TBSPQRY_DATA *** pppTablespaceData,
   sqluint32 reserved1,
   sqluint32 reserved2);
```

sqlbmtsq API パラメーター

pSqlca 出力。 sqlca 構造を指すポインター。

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、より下位のレベルのサービス、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使用できるなら、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フェッチしてください。

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。

- sysadm
- sysctrl
- sysmaint
- dbadm

必要な接続

データベース

API インクルード・ファイル

```
sqlutil.h
```

API とデータ構造構文

```
SQL_API_RC SQL_API_FN
  sqlbotcq (
   struct sqlca * pSqlca,
   sqluint32 TablespaceId,
   sqluint32 * pNumContainers);

SQL_API_RC SQL_API_FN
  sqlgotcq (
   struct sqlca * pSqlca,
   sqluint32 TablespaceId,
   sqluint32 * pNumContainers);
```

sqlbotcq API パラメーター

pSqlca 出力。 sqlca 構造を指すポインター。

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フェッチすることができます。

- sqlbtcq

コンテナー情報の完全なリストをフェッチします。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、フェッチ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。

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。

- sysadm
- sysctrl
- sysmaint
- dbadm
- load

必要な接続

データベース

API インクルード・ファイル

```
sqlutil.h
```

API とデータ構造構文

```
SQL_API_RC SQL_API_FN
  sqlbotsq (
   struct sqlca * pSqlca,
   sqluint32 TablespaceQueryOptions,
   sqluint32 * pNumTablespaces);

SQL_API_RC SQL_API_FN
  sqlgotsq (
   struct sqlca * pSqlca,
   sqluint32 TablespaceQueryOptions,
   sqluint32 * pNumTablespaces);
```

sqlbotsq API パラメーター

pSqlca 出力。 sqlca 構造を指すポインター。

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フェッチすることができます。

- sqlbstpq

  指定された表スペースに関する情報をフェッチします。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フェッチします。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、フェッチ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。

sqlbstpq - 単一の表スペースに関する情報の取得

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。

- sysadm
- sysctrl
- sysmaint
- dbadm
- load

必要な接続

データベース

API インクルード・ファイル

```
sqlutil.h
```

API とデータ構造構文

```
SQL_API_RC SQL_API_FN
  sqlbstpq (
   struct sqlca * pSqlca,
   sqluint32 TablespaceId,
   struct SQLB_TBSPQRY_DATA * pTablespaceData,
   sqluint32 reserved);

SQL_API_RC SQL_API_FN
  sqlgstpq (
   struct sqlca * pSqlca,
   sqluint32 TablespaceId,
   struct SQLB_TBSPQRY_DATA * pTablespaceData,
   sqluint32 reserved);
```

sqlbstpq API パラメーター

pSqlca 出力。 sqlca 構造を指すポインター。

TablespaceId
入力。照会する表スペースの ID です。

pTablespaceData
入出力。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。

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。

- sysadm
- sysctrl

必要な接続

データベース

API インクルード・ファイル

```
sqlutil.h
```

API とデータ構造構文

```
SQL_API_RC SQL_API_FN
  sqlbstsc (
   struct sqlca * pSqlca,
   sqluint32 SetContainerOptions,
   sqluint32 TablespaceId,
   sqluint32 NumContainers,
   struct SQLB_TBSCONTQRY_DATA * pContainerData);

SQL_API_RC SQL_API_FN
  sqlgstsc (
   struct sqlca * pSqlca,
   sqluint32 SetContainerOptions,
   sqluint32 TablespaceId,
   sqluint32 NumContainers,
   struct SQLB_TBSCONTQRY_DATA * pContainerData);
```

sqlbstsc API パラメーター

pSqlca 出力。 sqlca 構造を指すポインター。

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 つ以上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維持できます。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 DB2RESTORE_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 DB2RESTORE_CONTINUE に設定して、もう一度 db2Restore 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・スペースが使用できるようになるまで、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維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。

sqlbtcq - すべての表スペース・コンテナーの照会データの取得

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。

- sysadm
- sysctrl
- sysmaint
- dbadm

必要な接続

データベース

API インクルード・ファイル

```
sqlutil.h
```

API とデータ構造構文

```
SQL_API_RC SQL_API_FN
  sqlbtcq (
   struct sqlca * pSqlca,
   sqluint32 TablespaceId,
   sqluint32 * pNumContainers,
   struct SQLB_TBSCONTQRY_DATA ** ppContainerData);

SQL_API_RC SQL_API_FN
  sqlgtcq (
   struct sqlca * pSqlca,
   sqluint32 TablespaceId,
   sqluint32 * pNumContainers,
   struct SQLB_TBSCONTQRY_DATA ** ppContainerData);
```

sqlbtcq API パラメーター

pSqlca 出力。 sqlca 構造を指すポインター。

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、より下位のレベルのサービス、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使用できるなら、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フェッチしてください。

sqlcspqy - DRDA 未確定トランザクションのリスト

同期点マネージャー・パートナー接続の間にある未確定トランザクションのリストを表示します。この API は使用すべきではありません。 『db2SpmListIndTrans API - SPM 未確定トランザクションのリスト』を参照してください。

許可

なし

必要な接続

インスタンス

API インクルード・ファイル

```
sqlxa.h
```

API とデータ構造構文

```
extern int SQL_API_FN sqlcspqy(SQLCSPQY_INDOUBT     **indoubt_data,
                               sqlint32 *indoubt_count,
                               struct sqlca *sqlca);
```

sqlcspqy 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 構造を指すポインター。

使用上の注意

DRDA 未確定トランザクションが発生するのは、分散作業単位内のコーディネーターと参加者との間の通信が失われた場合です。

分散作業単位では、ユーザーやアプリケーションが、単一の作業単位内で複数の場所にあるデータを読んだり更新したりできます。そのような作業には、2 フェーズ・コミットが必要となります。

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。第 2 のフェーズでは、トランザクションをコミットまたはロールバックします。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 コマンドを発行する前に、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。 CONNECT ステートメントの dbalias として、データベース・マネージャー構成パラメーター spm_name を使います。

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、データベースが接続可能になり、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 つ以上のデータベース・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。

- 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まったくありません。

API インクルード・ファイル

```
sqlenv.h
```

API とデータ構造構文

```
SQL_API_RC SQL_API_FN
  sqle_activate_db (
        char * pDbAlias,
        char * pUserName,
        char * pPassword,
        void * pReserved,
        struct sqlca * pSqlca);

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);
```

sqle_activate_db API パラメーター

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 構造を指すポインター。

sqlg_activate_db API 固有パラメーター

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。

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シャットダウンするには 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 (例えば、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。

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。

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 つ以上のデータベース・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。

- 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まったくありません。

API インクルード・ファイル

```
sqlenv.h
```

API とデータ構造構文

```
SQL_API_RC SQL_API_FN
  sqle_deactivate_db (
        char * pDbAlias,
        char * pUserName,
        char * pPassword,
        void * pReserved,
        struct sqlca * pSqlca);

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);
```

sqle_deactivate_db API パラメーター

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 構造を指すポインター。

sqlg_deactivate_db API 固有パラメーター

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。

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 を使用しないとシャットダウンできません。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、最後の DB2 CONNECT RESET ステートメント (0 に等しいカウンター) ではデータベースがシャットダウンしないので、DEACTIVATE DATABASE を使用しなければなりません。

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。

---

sqleaddn - パーティション・データベース環境へのデータベース・パーティション・サーバーの追加

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 SYSTEM 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 SYSTEM 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。

- sysadm

- sysctrl

必要な接続

なし

API インクルード・ファイル

```
sqlenv.h
```

API とデータ構造構文

```
SQL_API_RC SQL_API_FN
  sqleaddn (
        void * pAddNodeOptions,
        struct sqlca * pSqlca);

SQL_API_RC SQL_API_FN
  sqlgaddn (
        unsigned short addnOptionsLen,
        struct sqlca * pSqlca,
        void * pAddNodeOptions);
```

sqleaddn API パラメーター

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。

pSqlca 出力。 sqlca 構造を指すポインター。

sqlgaddn API 固有パラメーター

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。

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・サーバー上のデータベース・パーティションに、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。

システム中のデータベースで、XML 列を持つユーザー表の作成が行われた場合や、XSR オブジェクトの登録が行われた場合、それが成功してもしなくても、この API は必ず失敗します。

データベースの自動ストレージが有効になっているかどうかを判別するために、sqleaddn API は、インスタンス中の各データベースについて、カタログ・パーティションと通信を行う必要があります。自動ストレージが有効になっていれば、その通信によってストレージ・パス定義が得られます。同様に、データベース・パーティションとともに SYSTEM TEMPORARY 表スペースが作成される場合、表スペース定義を検索するために、sqleaddn API 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ことが必要になる場合があります。 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自動ストレージおよび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。

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。

sqleatcp - インスタンスへのアタッチとパスワードの変更

インスタンス・レベルの関数 (例えば、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・パスワードの変更というオプション機能を可能にすることにより、sqleatin API の機能を拡張します。 DB2

データベース・システムでは、AIX、Linux、および Windows の各オペレーティング・システムでパスワードを変更することがサポートされています。

許可

なし

必要な接続

この API はインスタンス接続を確立します。

API インクルード・ファイル

```
sqlenv.h
```

API とデータ構造構文

```
SQL_API_RC SQL_API_FN
  sqleatcp (
        char * pNodeName,
        char * pUserName,
        char * pPassword,
        char * pNewPassword,
        struct sqlca * pSqlca);
```

sqleatcp API パラメーター

pNodeName
入力。アタッチする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。

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。

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。

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。

pSqlca 出力。 sqlca 構造を指すポインター。

使用上の注意

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。

アタッチ要求が成功すると、 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・サーバーの場合はデータベース・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、リモートで実行されることは決してありません。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。

アタッチを確立するノードは、sqlesetc API を呼び出すことによっても指定できます。

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。

sqleatin - インスタンスへのアタッチ

インスタンス・レベルの関数 (例えば、 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 はインスタンス接続を確立します。

API インクルード・ファイル

```
sqlenv.h
```

API とデータ構造構文

```
SQL_API_RC SQL_API_FN
  sqleatin (
        char * pNodeName,
        char * pUserName,
        char * pPassword,
        struct sqlca * pSqlca);

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);
```

sqleatin API パラメーター

pNodeName
入力。アタッチする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。

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。

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。

pSqlca 出力。 sqlca 構造を指すポインター。

sqlgatin API 固有パラメーター

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。

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。

アタッチ要求が成功すると、 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・サーバーの場合はデータベース・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、リモートで実行されることは決してありません。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。

アタッチを確立するノードは、sqlesetc API を呼び出すことによっても指定できます。

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することを希望する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。

sqlecadb - システム・データベース・ディレクトリーへの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・データベース・パーティション・サーバー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。

- SYSADM
- SYSCTRL

必要な接続

なし

API インクルード・ファイル

```
sqlenv.h
```

API とデータ構造構文

```
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);

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);
```

sqlecadb API パラメーター

pDbName
入力。データベース名を含むストリングを指定します。

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。

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.h 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。

pNodeName
入力。データベースが位置するデータベース・パーティション・サーバー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。

pPath

入力。Linux および UNIX 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・パスが指定されたものと見なされます。

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。

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.h から) は以下のとおりです。

SQL_AUTHENTICATION_SERVER
認証が、ターゲット・データベースを含むデータベース・パーティション・サーバー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データベース・パーティション・サーバー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
認証がターゲット・データベースを含むデータベース・パーティション・サーバーで行われることと、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
認証が、ターゲット・データベースを含むデータベース・パーティション・サーバーで行われ、接続がデータ暗号化を使用しなければならないことを指定します。

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。

SQL_AUTHENTICATION_SVRENC_AESO
認証が、ターゲット・データベースを含むデータベース・パーティション・サーバーで行われること、および認証ユーザー ID とパスワードが Advanced Encryption Standard (AES) 暗号化アルゴリズム

を使用して暗号化されることを指定します。この認証タイプは、DB2 バージョン 9.5 フィックスパック 3 以降で使用できます。

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のカタログ作成を実行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。

pPrincipal
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。 **authentication** が SQL_AUTHENTICATION_KERBEROS の場合に限り、この値を指定してください。

pSqlca 出力。 sqlca 構造を指すポインター。

sqlgcadb API 固有パラメーター

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。 **authentication** 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、0 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、0 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、0 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。

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。 **authentication** が SQL_AUTHENTICATION_KERBEROS の場合に限り、この値を指定してください。

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

type 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 **authentication** 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有効な場合、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。

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データベース・パーティション・サーバーで行われます。これはデフォルトです。

CLIENT
認証がアプリケーションを呼び出す確認データベース・パーティション・サーバーで行われます。

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
認証がターゲット・データベースを含むデータベース・パーティション・サーバーで行われることと、認証のユーザー ID およびパスワードが暗号化されることを指定します。

DATAENC
認証が、ターゲット・データベースを含むデータベース・パーティション・サーバーで行われ、接続がデータ暗号化を使用しなければならないことを指定します。

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。

SQL_AUTHENTICATION_SVRENC_AESO
認証が、ターゲット・データベースを含むデータベース・パーティション・サーバーで行われること、および認証ユーザー ID とパスワードが AES 暗号化アルゴリズムを使用して暗号化さ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 ネーム・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"'
```

sqlecran - データベース・パーティション・サーバー上へのデータベース作成

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。例えば、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。

- 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 インクルード・ファイル

```
sqlenv.h
```

API とデータ構造構文

```
SQL_API_RC SQL_API_FN
  sqlecran (
        char * pDbName,
        void * pReserved,
        struct sqlca * pSqlca);

SQL_API_RC SQL_API_FN
  sqlgcran (
        unsigned short reservedLen,
        unsigned short dbNameLen,
        struct sqlca * pSqlca,
        void * pReserved,
        char * pDbName);
```

sqlecran API パラメーター

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。

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。

pSqlca 出力。 sqlca 構造を指すポインター。

sqlgcran API 固有パラメーター

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。

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。

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。

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。

- 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) ノードで作成するには、まずそのノードにアタッチする必要があります。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 インクルード・ファイル

sqlenv.h

API とデータ構造構文

```
SQL_API_RC SQL_API_FN
  sqlecrea (
        char * pDbName,
        char * pLocalDbAlias,
        char * pPath,
        struct sqledbdesc * pDbDescriptor,
        SQLEDBTERRITORYINFO * pTerritoryInfo,
        char Reserved2,
        void * pDbDescriptorExt,
        struct sqlca * pSqlca);

SQL_API_RC SQL_API_FN
  sqlgcrea (
        unsigned short PathLen,
        unsigned short LocalDbAliasLen,
        unsigned short DbNameLen,
        struct sqlca * pSqlca,
        void * pReserved1,
        unsigned short Reserved2,
        SQLEDBTERRITORYINFO * pTerritoryInfo,
        struct sqledbdesc * pDbDescriptor,
        char * pPath,
        char * pLocalDbAlias,
        char * pDbName);
```

sqlecrea API parameters

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の付いたシステム・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。

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収容する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。

pPath 入力。Linux および UNIX システムでは、データベースを作成するパスを指定します。パスを指定しないと、データベース・マネージャー構成ファイル (dftdbpath パラメーター) に指定されているデフォルトのデータベース・パス上にデータベースが作成されます。 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。パスを指定しない場合、dftdbpath データベース・マネージャー構成パラメーターが NFS マウント・パスに設定されていないことを確認してください (例えば UNIX ベースのシステムの場合、パラメーターがインスタンス所有者の $HOME ディレクトリーを指定しないようにします)。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。

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、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ユーザーが使用できます。

提供される値は、照合シーケンス、データベース・コメント、または表スペース定義です。値を提供しないようにする場合は、提供する値を NULL にすることができます。このパラメーターで提供できる値については、SQLEDBDESC データ構造のトピックを参照してください。

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・セットを収容します。 NULL にすることもできます。データベースのデフォルト・コード・セットは UTF-8 (Unicode) です。あるデータベースのために特定のコード・セットおよびテリトリーが必要な場合、sqledbterritoryinfo 構造体によって必要なコード・セットおよびテリトリーを指定する必要があります。このフィールドが NULL の場合、データベースの照合値として NULL、 SQL_CS_SYSTEM、 SQL_CS_IDENTITY_16BIT、SQL_CS_UCA400_NO、 SQL_CS_UCA400_LTH、 SQL_CS_UCA400_LSK、または SQL_CS_UNICODE のいずれかが許可されます (sqlcode 1083)。

Reserved2
入力。将来の利用のために予約されています。

pDbDescriptorExt
入力。このパラメーターは、データベースの作成時に使用される拡張データベース記述ブロック (sqledbdescext) を参照します。拡張データベース記述ブロックは、データベースの自動ストレージを制御し、データベースのデフォルト・ページ・サイズを選択し、導入されている新しい表スペース属性の値を指定します。 NULL またはゼロに設定した場合、データベースのデフォルトのページ・サイズの 4 096 バイトが選択され、自動ストレージが有効になります。

pSqlca 出力。 sqlca 構造を指すポインター。

sqlgcrea API 固有パラメーター

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。

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/NODExxxx 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。
- 次のデータベース・ディレクトリーに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。ただし、パスを指定しなかった場合は、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、データベースが入っている各ファイル・システムに常駐しています。
  - 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。結果のディレクトリー項目には、データベース名とデータベース別名が入ることになります。

API がリモート・クライアントから呼び出された場合は、そのデータベース名やデータベース別名にあわせて、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、注釈およびコード・セット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・セット、テリトリー、および照合順序を保管します。照合順序が固有な重みで構成される場合、またはそれが識別順序である場合、データベース構成ファイルにフラグが設定されます。
- 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・ファイルを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 つ以上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。 RESTRICTIVE オプションを選択しない場合、PUBLIC に付与された CREATEIN 特権を使ってバインドを実行すると、NULLID と呼ばれ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。
- 以下の権限や特権を付与します。
  - データベース作成者に対する DBADM、 CONNECT、 CREATETAB、 BINDADD、 CREATE_NOT_FENCED、 IMPLICIT_SCHEMA、および LOAD の各種権限。
  - CONNECT、CREATETAB、BINDADD、および IMPLICIT_SCHEMA 権限を PUBLIC に。
  - USERSPACE1 表スペースの USE 特権を PUBLIC に。
  - 各システム・カタログに対する SELECT 特権を PUBLIC に。
  - 正常にバインドされたユーティリティーに対する BIND および EXECUTE 特権を PUBLIC に。
  - 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。

  注: RESTRICTIVE オプションを使用すると、RESTRICT_ACCESS データベース構成パラメーターが YES に設定されることになります。また、特権あるいは権限が自動的に PUBLIC に認可されることがなくなります。詳しくは、CREATE DATABASE コマンドの RESTRICTIVE オプションを参照してください。

dbadm 権限を使用すると、これらの特権を他のユーザーまたは PUBLIC に付与 (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/NODExxxx 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)。サブディレクトリー SQL00001 から SQLnnnnn は、このパスに常駐します。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 オペレーティング・システムでは、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。例えば、ISO8859-15 というコード・セット名の代わりに、ISO885915 と指定します。

sqlecrea API は、データベース記述子ブロック (SQLEDBDESC) と呼ばれるデータ構造を受け入れます。この構造内に独自の照合シーケンスを定義できます。

注: 1 バイト・データベースに対してのみ独自の照合シーケンスを定義できます。

データベースに照合シーケンスを指定するには、以下のようにします。

- 必要な SQLEDBDESC 構造を渡します。または
- NULL ポインターを渡します。オペレーティング・システムの照合シーケンス (現行のロケール・コードとコード・ページに基づいて) が使用されます。これは、SQL_CS_SYSTEM (0) と等しい SQLDBCSS を指定するのと同じです。

アプリケーションが既にデータベースに接続されている場合、CREATE DATABASE コマンドの実行は失敗します。

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最も顕著な値は、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、ホスト言語インクルード・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、この順序が原因でホスト 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、この順序が原因でホスト 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、この順序が原因でホスト 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、この順序が原因でホスト 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、この順序が原因でホスト CCSID 5026 (EBCDIC 日本語) に従ってソートが行われます。

データベースの作成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。それは、文字ストリングが比較される方法を決定します。これは索引の構造と照会の結果に影響します。 Unicode データベースでは、カタログ表およびビューが、データベース作成呼び出しに指定された照合とは関係なく常に IDENTITY 照合によって作成されます。非 Unicode データベースでは、カタログ表およびビューがデータベース照合によって作成されます。

新規のデータベースに異なる別名を定義するには、sqlecadb を使用します。

呼び出さないように特に指示しないかぎり、データベースの作成プロセス中に、デフォルトで構成アドバイザーが呼び出されます。

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。パスを指定しないと、データベース・マネージャー構成ファイル (dftdbpath 構成パラメーター) に指定されているデフォルトのデータベース・パス上にデータベースが作成されます。

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。パスを指定しない場合、dftdbpath データベース・マネージャー構成パラメーターが NFS マウント・パスに設定されていないことを確認してください (例えば UNIX ベースのシステムの場合、パラメーターがインスタンス所有者の $HOME ディレクトリーを指定しないようにします)。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
Unicode 以外のデータベースの場合、これはデフォルト・オプションであり、その照合シーケンスは、データベース・テリトリーをベースにします。Unicode データベースの場合、このオプションは IDENTITY オプションと同等です。

IDENTITY
ストリングがバイト単位で比較される一致照合順序。 Unicode データベースの場合は、これがデフォルトです。

USER udcs
照合シーケンスは、呼び出し側のアプリケーションによって、照合シーケンスを定義する 256 バイトのストリングを収めたホスト変数に指定されます。

numsegs
すべてのデフォルトの SMS 表スペース用のデータベース表ファイルを保管するために作成および使用されるディレクトリー数 (表スペース・コンテナー)。

dft_extentsize
データベースの表スペースのデフォルトのエクステント・サイズを指定します。

SMS_string
表スペースに属する予定の 1 つ以上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 つ以上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 4 KB 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。

sqlectnd - ノード・ディレクトリーへの項目のカタログ

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。

- sysadm
- sysctrl

必要な接続

なし

API インクルード・ファイル

```
sqlenv.h
```

API とデータ構造構文

```
SQL_API_RC SQL_API_FN
  sqlectnd (
        struct sqle_node_struct * pNodeInfo,
        void * pProtocolInfo,
        struct sqlca * pSqlca);

SQL_API_RC SQL_API_FN
  sqlgctnd (
        struct sqlca * pSqlca,
        struct sqle_node_struct * pNodeInfo,
        void * pProtocolInfo);
```

sqlectnd 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-LOCAL
- SQLE-NODE-NPIPE
- SQLE-NODE-TCPIP

pSqlca 出力。 sqlca 構造を指すポインター。

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 (例えば、sqllib) に格納されます。

ディレクトリーのキャッシュが有効な場合、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 コマンド)、再始動 (db2start コマンド) 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。

REXX API 構文、オプション 1

```
CATALOG LOCAL NODE nodename INSTANCE instance_name [WITH comment]
```

REXX API パラメーター、オプション 1

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、オプション 2

```
CATALOG NPIPE NODE nodename REMOTE computer_name INSTANCE instance_name
```

REXX API パラメーター、オプション 2

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、オプション 3

```
CATALOG TCPIP NODE nodename REMOTE hostname SERVER servicename
[WITH comment]
```

REXX API パラメーター、オプション 3

nodename
カタログするノードの別名。

hostname
ターゲット・データベースが常駐するノードのホスト名、または IPv4 あるいは IPv6 アドレス。

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。

REXX API 構文、オプション 4

```
CATALOG TCPIP4 NODE nodename REMOTE hostname SERVER servicename
[WITH comment]
```

REXX API パラメーター、オプション 4

nodename
カタログするノードの別名。

hostname
ターゲット・データベースが常駐するノードのホスト名、または IPv4 あるいは IPv6 アドレス。

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。

REXX API 構文、オプション 5

```
CATALOG TCPIP6 NODE nodename REMOTE hostname SERVER servicename
[WITH comment]
```

REXX API パラメーター、オプション 5

nodename
カタログするノードの別名。

hostname
ターゲット・データベースが常駐するノードのホスト名、または IPv4 あるいは IPv6 アドレス。

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。

sqledcgd - 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。

- sysadm
- sysctrl

必要な接続

なし

API インクルード・ファイル

```
sqlenv.h
```

API とデータ構造構文

```
SQL_API_RC SQL_API_FN
  sqledcgd (
        _SQLOLDCHAR * pDbAlias,
        _SQLOLDCHAR * pPath,
        _SQLOLDCHAR * pComment,
        struct sqlca * pSqlca);

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);
```

sqledcgd API パラメーター

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。

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。

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。

pSqlca 出力。 sqlca 構造を指すポインター。

sqlgdcgd API 固有パラメーター

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。

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。

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・ディレクトリー内の項目について記述します。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。

sqledpan - データベース・パーティション・サーバー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。

- sysadm
- sysctrl

必要な接続

なし。インスタンス接続は呼び出しの期間中に確立されます。

API インクルード・ファイル

```
sqlenv.h
```

API とデータ構造構文

```
SQL_API_RC SQL_API_FN
  sqledpan (
        char * pDbAlias,
        void * pReserved,
        struct sqlca * pSqlca);

SQL_API_RC SQL_API_FN
  sqlgdpan (
        unsigned short Reserved1,
        unsigned short DbAliasLen,
        struct sqlca * pSqlca,
        void * pReserved2,
        char * pDbAlias);
```

sqledpan API パラメーター

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。

pReserved
予約済み。 NULL にする必要があります。

pSqlca 出力。 sqlca 構造を指すポインター。

sqlgdpan API 固有パラメーター

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。

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。

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。

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。

sqledrpd - データベースのドロップ

データベースの内容とそのデータベースのすべてのログ・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。

- sysadm
- sysctrl

必要な接続

インスタンス。リモート・データベースをドロップする場合、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接続は、呼び出しの期間中に確立されます。

API インクルード・ファイル

```
sqlenv.h
```

API とデータ構造構文

```
SQL_API_RC SQL_API_FN
  sqledrpd (
        _SQLOLDCHAR * pDbAlias,
        _SQLOLDCHAR * pReserved2,
        struct sqlca * pSqlca);

SQL_API_RC SQL_API_FN
  sqlgdrpd (
        unsigned short Reserved1,
        unsigned short DbAliasLen,
        struct sqlca * pSqlca,
        _SQLOLDCHAR * pReserved2,
        _SQLOLDCHAR * pDbAlias);
```

sqledrpd API パラメーター

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。

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。

pSqlca 出力。 sqlca 構造を指すポインター。

sqlgdrpd API 固有パラメーター

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。

使用上の注意

sqledrpd API 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。

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。

sqledrpn - データベース・パーティション・サーバーがドロップ可能かどうかの検査

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。

- sysadm
- sysctrl

API インクルード・ファイル

```
sqlenv.h
```

API とデータ構造構文

```
SQL_API_RC SQL_API_FN
  sqledrpn (
        unsigned short Action,
        void * pReserved,
        struct sqlca * pSqlca);

SQL_API_RC SQL_API_FN
  sqlgdrpn (
        unsigned short Reserved1,
        struct sqlca * pSqlca,
        void * pReserved2,
        unsigned short Action);
```

sqledrpn API パラメーター

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

pReserved
予約済み。 NULL にする必要があります。

pSqlca 出力。 sqlca 構造を指すポインター。

sqlgdrpn API 固有パラメーター

Reserved1
pReserved2 の長さのために予約されています。

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。将来使用するために予約されています。

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。
2. 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。
3. 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。
4. 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。

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。

sqledtin - インスタンスからのデタッチ

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接続を除去します。

API インクルード・ファイル

```
sqlenv.h
```

API とデータ構造構文

```
SQL_API_RC SQL_API_FN
  sqledtin (
        struct sqlca * pSqlca);

SQL_API_RC SQL_API_FN
  sqlgdtin (
        struct sqlca * pSqlca);
```

sqledtin API パラメーター

pSqlca 出力。 sqlca 構造を指すポインター。

REXX API 構文

```
DETACH
```

sqlefmem - sqlbtcq および sqlbmtsq API によって割り振られたメモリーの解放

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 インクルード・ファイル

```
sqlenv.h
```

API とデータ構造構文

```
SQL_API_RC SQL_API_FN
  sqlefmem (
        struct sqlca * pSqlca,
        void * pBuffer);
```

```
SQL_API_RC SQL_API_FN
  sqlgfmem (
        struct sqlca * pSqlca,
        void * pBuffer);
```

sqlefmem API パラメーター

pSqlca 出力。 sqlca 構造を指すポインター。

pBuffer
入力。解放されるメモリーを指すポインター。

---

sqlefrce - システムからのユーザーおよび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： 割り込みが許されない操作 (例えばデータベース・リストア) を強制終了する場合は、その操作を再び実行して正常終了しなければデータベースは使用可能になりません。

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。

- 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 インクルード・ファイル

```
sqlenv.h
```

API とデータ構造構文

```
SQL_API_RC SQL_API_FN
  sqlefrce (
        sqlint32 NumAgentIds,
        sqluint32 * pAgentIds,
        unsigned short ForceMode,
        struct sqlca * pSqlca);
```

```
SQL_API_RC SQL_API_FN
  sqlgfrce (
        struct sqlca * pSqlca,
        unsigned short ForceMode,
        sqluint32 * pAgentIds,
        sqlint32 NumAgentIds);
```

sqlefrce API パラメーター

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。

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。

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。

pSqlca 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・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。

データベースの整合性を確保するため、強制終了できるのは、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 コマンドが出された後も、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。

ユーザーが強制終了されたとき、データベースの整合性を確保するために作業単位ロールバックが実行されます。

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。例えば、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。

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。

sqlegdad - データベース接続サービス (DCS) ディレクトリーへの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データベース名が 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。

- sysadm
- sysctrl

必要な接続

なし

API インクルード・ファイル

```
sqlenv.h
```

API とデータ構造構文

```
SQL_API_RC SQL_API_FN
  sqlegdad (
        struct sql_dir_entry * pDCSDirEntry,
        struct sqlca * pSqlca);

SQL_API_RC SQL_API_FN
  sqlggdad (
        struct sqlca * pSqlca,
        struct sql_dir_entry * pDCSDirEntry);
```

sqlegdad API パラメーター

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。

pSqlca 出力。 sqlca 構造を指すポインター。

使用上の注意

DB2 Connect プログラムは、次のような 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

データベース接続サービス・ディレクトリーが存在しない場合、データベース・マネージャーはそれを作成します。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有効な場合、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・ファイルのいず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。

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。

---

sqlegdcl - データベース接続サービス (DCS) ディレクトリーのスキャンの終了

sqlegdsc API 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 インクルード・ファイル

```
sqlenv.h
```

API とデータ構造構文

```
SQL_API_RC SQL_API_FN
  sqlegdcl (
        struct sqlca * pSqlca);
```

```
SQL_API_RC SQL_API_FN
  sqlggdcl (
        struct sqlca * pSqlca);
```

sqlegdcl API パラメーター

pSqlca 出力。 sqlca 構造を指すポインター。

REXX API 構文

```
CLOSE DCS DIRECTORY
```

sqlegdel - データベース接続サービス (DCS) ディレクトリーからの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。

- sysadm
- sysctrl

必要な接続

なし

API インクルード・ファイル

```
sqlenv.h
```

API とデータ構造構文

```
SQL_API_RC SQL_API_FN
  sqlegdel (
        struct sql_dir_entry * pDCSDirEntry,
        struct sqlca * pSqlca);

SQL_API_RC SQL_API_FN
  sqlggdel (
        struct sqlca * pSqlca,
        struct sql_dir_entry * pDCSDirEntry);
```

sqlegdel API パラメーター

pDCSDirEntry
入出力。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。

pSqlca 出力。 sqlca 構造を指すポインター。

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 DCS 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有効にされている場合、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。

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

sqlegdge - データベース接続サービス (DCS) ディレクトリーの特定項目の取得

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 インクルード・ファイル

```
sqlenv.h
```

API とデータ構造構文

```
SQL_API_RC SQL_API_FN
  sqlegdge (
        struct sql_dir_entry * pDCSDirEntry,
        struct sqlca * pSqlca);

SQL_API_RC SQL_API_FN
  sqlggdge (
        struct sqlca * pSqlca,
        struct sql_dir_entry * pDCSDirEntry);
```

sqlegdge API パラメーター

pDCSDirEntry
入出力。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を入力してください。

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。

pSqlca 出力。 sqlca 構造を指すポインター。

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

sqlegdgt - データベース接続サービス (DCS) ディレクトリーの項目の取得

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 インクルード・ファイル

```
sqlenv.h
```

API とデータ構造構文

```
SQL_API_RC SQL_API_FN
  sqlegdgt (
        short * pNumEntries,
        struct sql_dir_entry * pDCSDirEntries,
        struct sqlca * pSqlca);

SQL_API_RC SQL_API_FN
  sqlggdgt (
        struct sqlca * pSqlca,
        short * pNumEntries,
        struct sql_dir_entry * pDCSDirEntries);
```

sqlegdgt API パラメーター

pNumEntries
入出力。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。

pSqlca 出力。 sqlca 構造を指すポインター。

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 API 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。

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

sqlegdsc - データベース接続サービス (DCS) ディレクトリーのスキャンの開始

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。この API 呼び出しに関連するリソースの解放には、sqlegdgt API および sqlegdcl API を使用します。

許可

なし

必要な接続

なし

API インクルード・ファイル

```
sqlenv.h
```

API とデータ構造構文

```
SQL_API_RC SQL_API_FN
  sqlegdsc (
        short * pNumEntries,
        struct sqlca * pSqlca);

SQL_API_RC SQL_API_FN
  sqlggdsc (
        struct sqlca * pSqlca,
        short * pNumEntries);
```

sqlegdsc API パラメーター

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。

pSqlca 出力。 sqlca 構造を指すポインター。

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。コピーが既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。

REXX API 構文

```
OPEN DCS DIRECTORY
```

sqlegins - 現行インスタンスの取得

DB2INSTANCE 環境変数の値を戻します。

許可

なし

必要な接続

なし

API インクルード・ファイル

```
sqlenv.h
```

API とデータ構造構文

```
SQL_API_RC SQL_API_FN
  sqlegins (
        _SQLOLDCHAR * pInstance,
        struct sqlca * pSqlca);

SQL_API_RC SQL_API_FN
  sqlggins (
        struct sqlca * pSqlca,
        _SQLOLDCHAR * pInstance);
```

sqlegins API パラメーター

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。

pSqlca 出力。 sqlca 構造を指すポインター。

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 引数を指定して `sqleatin - アタッチ` を呼び出してください。

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。

sqleintr - アプリケーション要求への割り込み

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 インクルード・ファイル

```
sqlenv.h
```

API とデータ構造構文

```
SQL_API_RC SQL_API_INTR
        sqleintr ( void );

SQL_API_RC SQL_API_FN
  sqlgintr (
        void);
```

sqleintr API パラメーター

なし

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 9. 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。

表 9. *INTERRUPT* アクション (続き)

データベース活動	アクション
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。

REXX API 構文

```
INTERRUPT
```

例

```
call SQLDBS 'INTERRUPT'
```

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 インクルード・ファイル

```
sqlenv.h
```

API とデータ構造構文

```
SQL_API_RC SQL_API_FN
  sqleisig (
        struct sqlca * pSqlca);

SQL_API_RC SQL_API_FN
  sqlgisig (
        struct sqlca * pSqlca);
```

sqleisig API パラメーター

pSqlca 出力。 sqlca 構造を指すポインター。

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ムを必要とする場合、 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。

REXX API 構文

```
INSTALL SIGNAL HANDLER
```

sqlemgdb - 前のバージョンの DB2 データベースの現行バージョンへのマイグレーション

以前のバージョン (バージョン 8.x 以降) の DB2 データベースを現行バージョンに変換します。

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 インクルード・ファイル

```
sqlenv.h
```

API とデータ構造構文

```
SQL_API_RC SQL_API_FN
  sqlemgdb (
        _SQLOLDCHAR * pDbAlias,
        _SQLOLDCHAR * pUserName,
        _SQLOLDCHAR * pPassword,
        struct sqlca * pSqlca);

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);
```

sqlemgdb API パラメーター

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。

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。

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。

pSqlca 出力。 sqlca 構造を指すポインター。

sqlgmgdb API 固有パラメーター

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。

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マイグレーションするだけで、マイグレーション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。

マイグレーションの前にデータベースをカタログする必要があります。

REXX API 構文

```
MIGRATE DATABASE dbalias [USER username USING password]
```

REXX API パラメーター

dbalias
マイグレーション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。

password
ユーザー名の認証に使用されるパスワード。

sqlencls - ノード・ディレクトリー・スキャンの終了

sqlenops API 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 インクルード・ファイル

```
sqlenv.h
```

API とデータ構造構文

```
SQL_API_RC SQL_API_FN
  sqlencls (
        unsigned short Handle,
        struct sqlca * pSqlca);

SQL_API_RC SQL_API_FN
  sqlgncls (
        unsigned short Handle,
        struct sqlca * pSqlca);
```

sqlencls 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 構造を指すポインター。

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。

---

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 インクルード・ファイル

```
sqlenv.h
```

API とデータ構造構文

```
SQL_API_RC SQL_API_FN
  sqlengne (
        unsigned short Handle,
        struct sqleninfo ** ppNodeDirEntry,
        struct sqlca * pSqlca);

SQL_API_RC SQL_API_FN
  sqlgngne (
        unsigned short Handle,
        struct sqleninfo ** ppNodeDirEntry,
        struct sqlca * pSqlca);
```

sqlengne 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 構造を指すポインター。

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。

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 SQLENINFO 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.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)

---

sqlenops - ノード・ディレクトリー・スキャンの開始

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 API を呼び出します。スキャンをクローズするには、sqlencls API を呼び出します。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 インクルード・ファイル

```
sqlenv.h
```

API とデータ構造構文

```
SQL_API_RC SQL_API_FN
  sqlenops (
        unsigned short * pHandle,
        unsigned short * pNumEntries,
        struct sqlca * pSqlca);

SQL_API_RC SQL_API_FN
  sqlgnops (
        unsigned short * pHandle,
        unsigned short * pNumEntries,
        struct sqlca * pSqlca);
```

sqlenops API パラメーター

pHandle
出力。この API から戻された ID です。 sqlengne API と sqlencls API に、この ID を渡す必要があります。

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 構造を指すポインター。

使用上の注意

この API が割り振ったストレージは、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。

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。

sqleqryc - クライアント接続設定の照会

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。 sqle_conn_setting データ構造に、接続設定のタイプと値が設定されます。

許可

なし

必要な接続

なし

API インクルード・ファイル

```
sqlenv.h
```

API とデータ構造構文

```
SQL_API_RC SQL_API_FN
  sqleqryc (
        struct sqle_conn_setting * pConnectionSettings,
        unsigned short NumSettings,
        struct sqlca * pSqlca);

SQL_API_RC SQL_API_FN
  sqlgqryc (
        struct sqle_conn_setting * pConnectionSettings,
        unsigned short NumSettings,
        struct sqlca * pSqlca);
```

sqleqryc API パラメーター

pConnectionSettings
入出力。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 構造を指すポインター。

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 ステートメントが既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。

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。 SYNCPOINT オプションは無視され、後方互換性のためだけに使用できます。 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.6 据え置かれた PREPARE の現行接続設定。

sqleqryi - クライアント情報の照会

sqle_client_info データ構造のフィールドにデータを設定し、既に存在する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 インクルード・ファイル

```
sqlenv.h
```

API とデータ構造構文

```
SQL_API_RC SQL_API_FN
  sqleqryi (
        unsigned short DbAliasLen,
        char * pDbAlias,
        unsigned short NumItems,
        struct sqle_client_info* pClient_Info,
        struct sqlca * pSqlca);
```

sqleqryi 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、その戻り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 構造を指すポインター。

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。

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 インクルード・ファイル

```
sqlenv.h
```

API とデータ構造構文

```
SQL_API_RC SQL_API_FN
  sqlesact (
        char * pAccountingString,
        struct sqlca * pSqlca);

SQL_API_RC SQL_API_FN
```

```
sqlgsact (
      unsigned short AccountingStringLen,
      char * pAccountingString,
      struct sqlca * pSqlca);
```

sqlesact API パラメーター

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 構造を指すポインター。

sqlgsact API 固有パラメーター

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。

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 や下線記号 (_) および空白を使用するようにしてください。

sqlesdeg - SQL ステートメントの最大実行時パーティション内並列処理レベル (つまり並列処理の程度) 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 ステートメントの実行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。

- sysadm
- sysctrl

必要な接続

インスタンス。リモート・サーバーにおけるランタイムの最大並列処理度を変更するには、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 インクルード・ファイル

```
sqlenv.h
```

API とデータ構造構文

```
SQL_API_RC SQL_API_FN
  sqlesdeg (
        sqlint32 NumAgentIds,
        sqluint32 * pAgentIds,
        sqlint32 Degree,
        struct sqlca * pSqlca);

SQL_API_RC SQL_API_FN
  sqlgsdeg (
        struct sqlca * pSqlca,
        sqlint32 Degree,
        sqluint32 * pAgentIds,
        sqlint32 NumAgentIds);
```

sqlesdeg API パラメーター

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。

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。

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。

pSqlca 出力。 sqlca 構造を指すポインター。

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、例えば、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。

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。

---

sqlesetc - クライアント接続設定の指定

アプリケーション用の接続設定を指定します。 sqle_conn_setting データ構造を使用して、接続設定のタイプと値を指定します。

許可

なし

必要な接続

なし

API インクルード・ファイル

```
sqlenv.h
```

API とデータ構造構文

```
SQL_API_RC SQL_API_FN
  sqlesetc (
        struct sqle_conn_setting * pConnectionSettings,
        unsigned short NumSettings,
        struct sqlca * pSqlca);

SQL_API_RC SQL_API_FN
  sqlgsetc (
        struct sqle_conn_setting * pConnectionSettings,
        unsigned short NumSettings,
        struct sqlca * pSqlca);
```

sqlesetc 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 構造を指すポインター。

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 (例えば、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。

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 以下の事柄に基づいて、SQLRULES の設定方法を指定します。

- タイプ 2 の CONNECT を DB2 規則に従って処理するか、それとも ISO/ANS SQL92 の標準 (STD) の規則に従って処理するか。
- 結果セット内の LOB 列の形式をアプリケーションが指定する方法。

DB2

- SQL CONNECT ステートメントで、現在の接続と、確立されている (*休止* 状態の) 別の接続との間で切り換えることができるようにします。
- このデフォルト設定を使用する場合、アプリケーションは LOB 値と LOB ロケーターのいずれを取り出すかを最初のフェッチ要求時にのみ指定できます。その後のフェッチ要求では、同じ形式を LOB 列に使用する必要があります。

STD

- SQL CONNECT ステートメントでは、*新しい* 接続を確立することしかできないようにします。 休止接続へ切り替えるには、SQL SET CONNECTION ステートメントを使います。
- アプリケーションは、それぞれのフェッチ要求で LOB 値を取り出すのか LOB ロケーターを取り出すのかを変

更することができます。つまり、BLOCKING バインド・オプションの設定値にかかわりなく、1 つ以上の LOB 列を持つカーソルをブロック化することはできません。

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。 SYNCPOINT オプションは無視され、後方互換性のためだけに使用できます。

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。

sqleseti - クライアント情報の設定

接続が既に存在する場合、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 (sqle_client_info データ構造のフィールドを設定すること) 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 インクルード・ファイル

```
sqlenv.h
```

API とデータ構造構文

```
SQL_API_RC SQL_API_FN
  sqleseti (
        unsigned short DbAliasLen,
        char * pDbAlias,
        unsigned short NumItems,
        struct sqle_client_info* pClient_Info,
        struct sqlca * pSqlca);
```

sqleseti 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 構造を指すポインター。

使用上の注意

別名が提供された場合、別名への接続が既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

で保持されます。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持されます。

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。例えば、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 バージョン 5 以降、DB2 UDB バージョン 7 以降、および DB2 i5/OS® V6R1 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。

CLI プログラムで、接続前に sqleseti API を呼び出すと機能しません。 CLI プログラムで、接続確立後に sqleseti API を呼び出すと、予測不能な動作が生じる可能性があります。代わりに、対応する CLI 関数、SQLSetConnectAttr() あるいは SQLSetEnvAttr() を使用するようお勧めします。

sqleuncd - システム・データベース・ディレクトリーからの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。

- sysadm
- sysctrl

必要な接続

なし

API インクルード・ファイル

sqlenv.h

API とデータ構造構文

```
SQL_API_RC SQL_API_FN
  sqleuncd (
        _SQLOLDCHAR * pDbAlias,
        struct sqlca * pSqlca);

SQL_API_RC SQL_API_FN
  sqlguncd (
        unsigned short DbAliasLen,
        struct sqlca * pSqlca,
        _SQLOLDCHAR * pDbAlias);
```

sqleuncd API パラメーター

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。

pSqlca 出力。 sqlca 構造を指すポインター。

sqlguncd API 固有パラメーター

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。

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、以前の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有効にされている場合、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。

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。

---

sqleuncn - ノード・ディレクトリーからの項目のアンカタログ

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。

- sysadm
- sysctrl

必要な接続

なし

API インクルード・ファイル

```
sqlenv.h
```

API とデータ構造構文

```
SQL_API_RC SQL_API_FN
  sqleuncn (
        _SQLOLDCHAR * pNodeName,
        struct sqlca * pSqlca);

SQL_API_RC SQL_API_FN
  sqlguncn (
        unsigned short NodeNameLen,
        struct sqlca * pSqlca,
        _SQLOLDCHAR * pNodeName);
```

sqleuncn API パラメーター

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。

pSqlca 出力。 sqlca 構造を指すポインター。

sqlguncn API 固有パラメーター

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。

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有効にされている場合、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。

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。

sqlgaddr - 変数のアドレスの取得

ある変数のアドレスを別の変数の中に入れます。この API は、FORTRAN や COBOL など、ポインター操作を提供しないホスト・プログラミング言語で使用されます。

許可

なし

必要な接続

なし

API インクルード・ファイル

```
sqlutil.h
```

API とデータ構造構文

```
SQL_API_RC SQL_API_FN
  sqlgaddr (
   char * pVariable,
   char ** ppOutputAddress);
```

sqlgaddr 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。

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。この API は、FORTRAN や COBOL など、ポインター操作を提供しないホスト・プログラミング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 インクルード・ファイル

```
sqlutil.h
```

API とデータ構造構文

```
SQL_API_RC SQL_API_FN
  sqlgdref (
   unsigned int NumBytes,
   char * pTargetBuffer,
   char ** ppSourceBuffer);
```

sqlgdref 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。

sqlgmcpy - あるメモリー領域から別のメモリー領域へデータのコピー

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。この API は、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・プログラミング言語で使用されます。

許可

なし

必要な接続

なし

API インクルード・ファイル

```
sqlutil.h
```

API とデータ構造構文

```
SQL_API_RC SQL_API_FN
  sqlgmcpy (
   void * pTargetBuffer,
   const void * pSource,
   sqluint32 NumBytes);
```

sqlgmcpy 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。

---

sqlogstt - SQLSTATE メッセージの入手

SQLSTATE 値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 インクルード・ファイル

```
sql.h
```

API とデータ構造構文

```
SQL_API_RC SQL_API_FN
  sqlogstt (
        char * pBuffer,
        short BufferSize,
        short LineWidth,
        char * pSqlstate);

SQL_API_RC SQL_API_FN
  sqlggstt (
        short BufferSize,
        short LineWidth,
        char * pSqlstate,
        char * pBuffer);
```

sqlogstt API パラメーター

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 ストリング終了文字を見込んで行われます。

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。

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、使用できる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。

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。

sqluadau - 現行ユーザーの権限の取得

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、現行ユーザーのインスタンス・レベルおよびデータベース・レベルの権限をそれぞれ報告します。報告されるインスタンス・レベルの権限は、sysadm_group、sysmaint_group、sysctrl_group といったデータベース・マネージャー構成パラメーターに設定できるもので、データベース・レベルの権限は、GRANT (データベース権限) ステートメントによって付与できるものです。

注: この API は使用されなくなりますが、AUTH_LIST_AUTHORITIES_FOR_AUTHID 表関数を使用することで同じ機能を得ることができます。

許可

なし

必要な接続

データベース

API インクルード・ファイル

```
sqlutil.h
```

API とデータ構造構文

```
SQL_API_RC SQL_API_FN
  sqluadau (
   struct sql_authorizations * pAuthorizations,
   struct sqlca * pSqlca);

SQL_API_RC SQL_API_FN
  sqlgadau (
   struct sql_authorizations * pAuthorizations,
   struct sqlca * pSqlca);
```

sqluadau API パラメーター

pAuthorizations
入力または出力。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 構造を指すポインター。

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。

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

sqludrdt - データベース・パーティション・グループ間でのデータの再配分

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配分します。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。再配分アルゴリズムは、現行のデータ分散に基づいて移動するデータベース・パーティションを選択します。この API は、REDISTRIBUTE DATABASE PARTITION GROUP コマンドの NOT ROLLFORWARD RECOVERABLE オプションをサポートしません。

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のカタログ・パーティションになっているかを判別するには、LIST DATABASE DIRECTORY コマンドを使用します。

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。

- sysadm
- sysctrl
- dbadm

API インクルード・ファイル

```
sqlutil.h
```

API とデータ構造構文

```
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);

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);
```

sqludrdt API パラメーター

pNodeGroupName
再配分されるデータベース・パーティション・グループの名前。

pTargetPMapFileName
ターゲット分散マップを含むファイルの名前。ファイル名の一部としてディレクトリー・パスを指定していない場合、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 の場合に使用されます。このファイルは文字フォーマットでなければならず、4 096 項目 (複数区画データベース・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・データベース・パーティション・グループの場合) のいずれかを含みます。ファイル内の項目はノード番号を示します。項目はフリー・フォーマットで構いません。

pDataDistFileName
入力分散情報を含むファイルの名前。ファイル名の一部としてディレクトリー・パスを指定していない場合、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 の場合に使用されます。このファイルは文字フォーマットでなければならず、4 096 の正整数項目を含みます。ファイル内の各項目により、対応するデータベース・パーティションの重みを示す必要があります。 4 096 の値の合計は、4 294 967 295 以下である必要があります。

pAddList
データ再配分中にデータベース・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。リスト内の項目の形式は、SQL_PDB_NODE_TYPE 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
データ再配分中にデータベース・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。リスト内の項目の形式は、SQL_PDB_NODE_TYPE 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配分のタイプを示す単一文字。可能な値は以下のとおりです。

U 平衡配分を実現するためにデータベース・パーティション・グループの再配分を指定します。 pDataDistFileName が NULL の場合、現行のデータ分散は均等であると見なされます (つまり、各データベース・パーティションは同量のデータを表します)。
pDataDistFileName パラメーターが非 NULL の場合、このファイル内の値は、現行のデータ分散を表していると見なされます。
DataRedistOption が U の場合、pTargetPMapFileName パラメーターは NULL でなければなりません。追加リストで指定されているデータベース・パーティションは追加され、ドロップ・リストで指定されているデータベース・パーティションはデータベース・パーティション・グループからドロップされます。

T pTargetPMapFileName パラメーターを使用したデータベース・パーティション・グループの再配分を指定します。このオプションでは、パラメーター pDataDistFileName、pAddList、および pDropList は NULL でなければならず、AddCount および DropCount の両方のパラメーターはゼロでなければなりません。

C 失敗した再配分操作の続行を指定します。このオプションでは、パラメーター pTargetPMapFileName、pDataDistFileName、pAddList、および pDropList は NULL でなければならず、AddCount および DropCount の両方のパラメーターはゼロでなければなりません。

R 失敗した再配分操作のロールバックを指定します。このオプションでは、パラメーター pTargetPMapFileName、pDataDistFileName、pAddList、および pDropList は NULL でなければならず、AddCount および DropCount の両方のパラメーターはゼロでなければなりません。

pSqlca 出力。 sqlca 構造を指すポインター。

sqlgdrdt API 固有パラメーター

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
ターゲット分散マップ・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。

使用上の注意

再配分操作が完了すると、メッセージ・ファイルが以下に書き込まれます。

- UNIX ベース・システムの場合、$HOME/sqllib/redist ディレクトリー。サブディレクトリーとファイル名については、次の形式が使用されます。
  database-name.nodegroup-name.timestamp。
- Windows オペレーティング・システムの場合、$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ついては、次の形式が使用されます。
  database-name¥first-eight-characters-of-the-nodegroup-name¥date¥time。

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。

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・グループに追加するには、ALTER DATABASE PARTITION GROUP ステートメントを使用します。このステートメントを使用すると、データベース・パーティション・グループに関連する表スペース用のコンテナーを定義できます。

再配分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・グループの再配分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。明示的な再バインドにより、無効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。再配分メッセージ・ファイルには、再配分を受けたすべての表のリストが入ります。

データベース・パーティション・グループの再配分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もお勧めします。

複製されたサマリー表や、DATA CAPTURE CHANGES 節を使用して定義された表を含むデータベース・パーティション・グループは、再配分させることはできません。

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、再配分を行うことはできません。

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。

sqlugrpn - 特定の行についてのデータベース・パーティション・サーバー番号の取得

分散キー値に基づいてデータベース・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、対応する分散キー値の文字表示があります。出力は、分散ストラテジーによって生成されたデータベース・パーティション番号と、分散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。分散マップ情報が提供されていない場合には、データベース・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 ファイル内のデータベース・パーティション・サーバーから呼び出す必要があります。この API は、クライアントとサーバーの間でコード・ページやバイトの並び順などに違いがあると、誤ったデータベース・パーティション情報が戻される危険があるため、クライアントから呼び出すべきではありません。

許可

なし

API インクルード・ファイル

```
sqlutil.h
```

API とデータ構造構文

```
SQL_API_RC SQL_API_FN
  sqlugrpn (
   unsigned short num_ptrs,
   unsigned char ** ptr_array,
   unsigned short * ptr_lens,
   unsigned short territory_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);

SQL_API_RC SQL_API_FN
  sqlggrpn (
   unsigned short num_ptrs,
   unsigned char ** ptr_array,
   unsigned short * ptr_lens,
   unsigned short territory_code,
   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);
```

sqlugrpn 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 パラメーター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 に指定された分散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パーティション化することにつなが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・キーの各部に対応する値の文字表示の長さを表す符号なし整数の配列。

territory_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
データベース・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
分散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
すべての分散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。例えば、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 進数列分散キー値はパック 10 進数形式になります。

SQL_BINARYNUMERICS_FORMAT
すべての数値分散キー値はビッグ・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、分散キーとして定義できるものと同じです。

注: CHAR、VARCHAR、GRAPHIC、および VARGRAPHIC データ・タイプは、この API を呼び出す前にデータベース・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、分散マップを提供しなければなりません。つまり、 part_info パラメーターの pmaplen フィールド (part_info->pmaplen) を 2 または 8192 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。分散キーを定義しなければなりません。つまり、part_info パラメーターの sqld フィールド (part_info->sqld) をゼロより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。

sqlugtpi - 表の分散情報の取得

これによってアプリケーションは、表の分散情報を入手できます。分散情報には、分散マップと、分散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、表の任意の行のデータベース・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、分散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 インクルード・ファイル

```
sqlutil.h
```

API とデータ構造構文

```
SQL_API_RC SQL_API_FN
  sqlugtpi (
   unsigned char * tablename,
   struct sqlupi * part_info,
   struct sqlca * sqlca);

SQL_API_RC SQL_API_FN
  sqlggtpi (
   unsigned short tn_length,
   unsigned char * tablename,
   struct sqlupi * part_info,
   struct sqlca * sqlca);
```

sqlugtpi API パラメーター

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 構造を指すポインター。

sqlggtpi API 固有パラメーター

tn_length
表名の長さを示す 2 バイトの符号なし整数。

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以下の 3 つです。

- 静止: SHARE
- 静止: UPDATE
- 静止: EXCLUSIVE

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。

- sysadm
- sysctrl
- sysmaint
- dbadm
- load

必要な接続

データベース

API インクルード・ファイル

```
sqlutil.h
```

API とデータ構造構文

```
SQL_API_RC SQL_API_FN
  sqluvqdp (
   char * pTableName,
   sqlint32 QuiesceMode,
   void * pReserved,
   struct sqlca * pSqlca);

SQL_API_RC SQL_API_FN
  sqlgvqdp (
   unsigned short TableNameLen,
   char * pTableName,
   sqlint32 QuiesceMode,
   void * pReserved,
   struct sqlca * pSqlca);
```

sqluvqdp API パラメーター

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 構造を指すポインター。

sqlgvqdp API 固有パラメーター

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)。

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意図的共有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、表スペースは意図的排他 (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。

静止プログラム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 (例えば、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、ユーザーが既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。

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。

第 5 章 REXX での DB2 API の呼び出し

DB2 API を呼び出すには、次の構文により SQLDBS ルーチンを使用します。

```
CALL SQLDBS 'command string'
```

SQLDBS ルーチンを使用しても、使用する DB2 API を呼び出せない場合、 REXX アプリケーション内から DB2 コマンド行プロセッサー (CLP) を呼び出して、その API を呼び出すことができます。ただし、DB2 CLP は、標準出力装置または特定のファイルのどちらかに出力するように命令します。その理由で、REXX アプリケーションは、呼び出された DB2 API からその出力に直接アクセスできません。また、呼び出された API が正常に処理されたかどうかを容易には判断することもできません。 SQLDB2 API は DB2 CLP へのインターフェースを提供しています。そのインターフェースは、各呼び出し後にコンパウンドの REXX 変数である SQLCA を設定して、呼び出された API ごとに処理が正常だったか失敗だったかを、直接 REXX アプリケーションにフィードバックできます。

SQLDB2 ルーチンを使用して、次の構文により DB2 API を呼び出すことができます。

```
CALL SQLDB2 'command string'
```

`'command string'` は、コマンド行プロセッサー (CLP) で処理可能なストリングです。

SQLDB2 を使用して DB2 API を呼び出すことは、次の場合の他は CLP を直接呼び出すのと同じです。

- 実行可能 CLP の呼び出しが SQLDB2 の呼び出しで置換された場合 (他のすべての CLP オプションとパラメーターは、同じように指定されている)。
- SQLDB2 の呼び出し後に REXX のコンパウンド変数 SQLCA が設定されたが、実行可能 CLP の呼び出し後には設定されなかった場合。
- SQLDB2 を呼び出した時、CLP のディスプレイ出力のデフォルトはオフに設定されていますが、実行可能 CLP を呼び出す場合、ディスプレイ出力はオンに設定されています。 CLP のディスプレイ出力をオンに切り替えるには、 SQLDB2 に `+o` または `-o-` オプションを渡します。

SQLDB2 の呼び出し後、REXX 変数だけが SQLCA に設定されるので、 DB2 API を呼び出すには、このルーチンだけを使用してください。 DB2 API は SQLDBS インターフェースですが、SQLCA 以外のデータを戻しませんし、現在のところ組み込まれてはいません。 したがって SQLDB2 では、以下の DB2 API だけがサポートされています。

- データベースの活動化 (Activate Database)
- ノードの追加 (Add Node)
- DB2 バージョン 1 用バインド (Bind for DB2 Version 1)(1) (2)
- DB2 バージョン 2 または 5 用バインド (Bind for DB2 Version 2 or 5)(1)
- ノードでのデータベース作成 (Create Database at Node)
- ノードでのデータベースのドロップ (Drop Database at Node)
- ノードのドロップの検査 (Drop Node Verify)

- データベースの非活動化 (Deactivate Database)
- 登録取り消し (Deregister)
- ロード (Load)(3)
- ロードの照会
- プログラムのプリコンパイル (Precompile Program)(1)
- パッケージの再バインド (Rebind Package)(1)
- データベース・パーティション・グループでの再配分
- 登録 (Register)
- データベース・マネージャーの始動
- データベース・マネージャーの停止

SQLDB2 がサポートする DB2 API に関する注意:

1. これらのコマンドは、SQLDB2 インターフェースを介して CONNECT ステートメントを必要とします。 SQLDB2 インターフェースを使用した接続では、SQLEXEC インターフェースに接続できません。また、SQLEXEC インターフェースを使用した接続では、SQLDB2 インターフェースに接続できません。
2. SQLDB2 インターフェースを介して Windows ベースのプラットフォームでサポートされます。
3. Load API 用の任意指定の出力パラメーター `poLoadInfoOut` は、 REXX のアプリケーションに戻りません。

注: SQLDB2 ルーチンは、上にリストされた DB2 API のためだけに使用されるように意図されていますが、 SQLDBS ルーチンを通じてサポートされていない、他の DB2 API のためにも使用することができます。あるいは、REXX アプリケーション内から CLP を介して DB2 API にアクセスすることも可能です。

分離レベルの変更

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。この API は、REXX アプリケーションからのみ呼び出すことができます。

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。これはデフォルトです。

UR 非コミット読み取り。

RS 読み取り固定。

NC　　コミットなし。

第 6 章 未確定トランザクション管理 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。

トランザクション・マネージャー (TM) が *再同期* アクションを実行するのをリソース所有者 (データベース管理者など) が待てないときに未確定トランザクションにヒューリスティック機能を実行させるための、API のセットがツール・ライター用に提供されています。この状態は、たとえば通信回線が切断されて未確定トランザクションが必要なリソースと拘束しようとしている場合に発生します。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、そのトランザクションにより使用されている表や索引のロック、ログのスペース、およびそのストレージなどが含まれます。各未確定トランザクションごとに、データベース・マネージャー で処理できる並行トランザクションの最大数も (1 つずつ) 減っていきます。

ヒューリスティックな API には、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機能、およびログ・レコードを削除してログ・ページを解放することにより、ヒューリスティックにコミットされたかロールバックされたトランザクションを取り消す機能があります。

重要: ヒューリスティックな API の使用には注意が必要で、最後の手段としてのみ使用すべきです。TM が再同期イベントを開始しなければなりません。 TM に再同期アクションを開始するオペレーター・コマンドがある場合には、これを使用します。ユーザーが TM による再同期の開始を待てない場合には、ヒューリスティックなアクションが必要です。

これらのアクションの実行には決まった方法はありませんが、以下の指針が役立ちます。

- db2XaListIndTrans 関数を使って、未確定トランザクションを表示します。これらは status = '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 ID であり、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) のグローバル・トランザクション ID と同一です。
- アプリケーションと稼働環境の知識を使用して、参加する他の RM を識別します。
- トランザクション・マネージャーが CICS で、唯一の RM が CICS® リソースの場合、ヒューリスティックなロールバックを実行します。
- トランザクション・マネージャーが CICS でない場合、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断するのに利用します。
- 最低 1 つの RM がコミットまたはロールバックしている場合、ヒューリスティックなコミットまたはロールバックを実行します。
- それらがすべてが準備済みの場合、ヒューリスティックなロールバックを実行します。
- 少なくとも 1 つの RM が使用できない場合、ヒューリスティックなロールバックを実行します。

トランザクション・マネージャーが使用でき、かつ未確定トランザクションの原因が、第 2 フェーズまたはそれ以前の再同期で RM が使用不能になっていることにある場合、 DBA は他の RM に対しどのようなアクションがとられたかを TM のログから判別し、それと同じアクションを実行します。 *gtrid* は TM と RM の間のマッチング・キーです。

ヒューリスティックな手法で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原因でログ満杯状態が発生した場合を除いて sqlxhfrg を実行しないでください。forget 関数を実行すると、この未確定トランザクションが占有していたログ・スペースが解放されます。トランザクション・マネージャーが最終的にこの未確定トランザクションに再同期アクションを実行する場合、この RM にレコードが見つからないため、TM が誤った判断をして他の RM をコミットまたはロールバックすることがあります。総じて、レコードの欠落は、RM がロールバックしていることを暗黙に示します。

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

インスタンス - SPM 名接続

必要な接続

データベース

API インクルード・ファイル

```
sqlxa.h
```

API とデータ構造構文

```
SQL_API_RC SQL_API_FN
  db2XaGetInfo(db2Uint32 versionNumber,
               void * pParmStruct,
               struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2XaGetInfoStruct
{
   db2int32 iRmid;
   struct sqlca         oLastSqlca;
} db2XaGetInfoStruct;
```

db2XaGetInfo 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。

db2XaGetInfoStruct データ構造パラメーター

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。

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

なし

必要な接続

データベース

API インクルード・ファイル

```
sqlxa.h
```

API とデータ構造構文

```
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[SQLXA_USERID_SZ];
   char log_full;
   char connected;
   char indoubt_status;
   char originator;
   char reserved[8];
```

```
   sqluint32 rmn;
   rm_entry rm_list[SQLXA_MAX_FedRM];
   } db2XaRecoverStruct;

typedef SQL_STRUCTURE rm_entry
{
  char name[SQLQG_MAX_SERVER_NAME_LEN];
  SQLXA_XID xid;
} rm_entry;
```

db2XaListIndTrans 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 構造を指すポインター。

db2XaListIndTransStruct データ構造パラメーター

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。

iIndoubtDataLen
入力。piIndoubtData パラメーターで指示されたバッファーのサイズ (バイト単位)。

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 呼び出し時に使用できる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。

db2XaRecoverStruct データ構造パラメーター

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。

connected
アプリケーションが接続されているかどうかを示します。

CONNECTED に有効な値 (sqlxa で定義) は、以下のとおりです。

SQLXA_TRUE
真。トランザクションは、通常の同期点処理を受け、 2 フェーズ・コミットの 2 番目のフェーズを待っています。

SQLXA_FALSE
false。トランザクションは、以前の障害により未確定のままで、現在はトランザクション・マネージャーからの再同期化を待っています。

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。

発信元のパラメーターの値が SQLXA_ORIG_FXA に設定されている場合、indoubt_status パラメーターの有効な値は以下のとおりです (include ディレクトリーの sqlxa.h で定義される)。

SQLXA_TS_MFCACK
トランザクションで、1 つ以上のフェデレーテッド・データ・ソースからのコミット確認通知が欠落していることを示します。

SQLXA_TS_MFRACK
トランザクションで、1 つ以上のフェデレーテッド・データ・ソースからのロールバック確認通知が欠落していることを示します。

originator
未確定トランザクションの発信元を示します。

ORIGINATOR に指定できる値は以下のとおりです (include ディレクトリーの sqlxa.h で定義される)。

SQLXA_ORIG_PE
MPP 環境で DB2 によって発信されたトランザクション。

SQLXA_ORIG_XA
XA によって発信されたトランザクション。

SQLXA_ORIG_FXA
フェデレーテッド 2 フェーズ・コミット処理の 2 番目のフェーズで発信されたトランザクション。これは、このトランザクションが 2 フェーズ・コミット・プロトコルの 2 番目のフェーズに入ったが、1 つ以上のフェデレーテッド・データ・ソースがこの 2 番目のフェーズを完了できないか、またはフェデレーテッド・サーバーと通信できないことを示します。

reserved
最初のバイトは、未確定トランザクションのタイプを示すのに使用されます。 0 は RM、1 は TM を示します。

rmn 出力。トランザクションのコミットまたはロールバックに失敗したフェデレーテッド・データ・ソースの数

rm_list
出力。障害のあるフェデレーテッド・データ・ソースのリストで、各項目にはサーバー名および XID が含まれます。

rm_entry データ構造パラメーター

name 出力。フェデレーテッド・データ・ソースの名前。

xid 出力。フェデレーテッド・トランザクションを固有に識別する、フェデレーテッド・データベースによってフェデレーテッド・データ・ソースに割り当てられた XA ID を指定します。

使用上の注意

SQLXA_MAX_FEDRM は 16 に定義されています。大半のフェデレーテッド・トランザクションには 10 より少ないデータ・ソースが関係しています。16 を超えるフェデレーテッド・データ・ソースがトランザクションのコミットまたはロールバックに失敗した場合、db2XaListIndTrans API により、そのうち 16 のみがこの未確定トランザクションに対して戻されます。非フェデレーテッドの未確定トランザクションの場合、rmn パラメーターは 0 に設定され、未確定トランザクションには関係するフェデレーテッド・データ・ソースがないことを示します。

フェデレーテッド未確定トランザクションに 16 を超える失敗したフェデレーテッド・データ・ソースが関係する場合、ヒューリスティック処理が呼び出されると、すべてのデータ・ソース (db2XaListIndTrans API によって戻されるかどうかにかかわりなく) が未確定トランザクションをコミットまたはロールバックします。未確定トランザクションのコミットまたはロールバックを正常に行ったフェデレーテッド・データ・ソースは、フェデレーテッド未確定トランザクションに対して障害のあるフェデレーテッド・データ・ソースのリストから除去されます。
db2XaListIndTrans API への次の呼び出しでは、未確定トランザクションのコミットまたはロールバックに失敗したフェデレーテッド・データ・ソースのみが、フェデレーテッド未確定トランザクションのリストに残ります。

フェデレーテッド未確定トランザクションのデータ・ソースのリストを入手するには、DB2 バージョン 9.1 ヘッダー・ファイルを使用してアプリケーションをコンパ

イルし、バージョン番号 db2Version900 またはそれ以上 (将来のリリース) を db2XaListIndTrans API へパスします。これより低いバージョン番号をパスした場合、API はそれでも未確定トランザクションのリストを戻しますが、フェデレーテッド・データ・ソース情報は除外されます。どちらの場合でも、アプリケーションによって使用されるヘッダー・ファイルのバージョンは、API へパスされるバージョン番号と同期でなければなりません。そうでない場合、結果は予測不能です。

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 oNumIndoubtsTotal 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。

sqlxhfrg - トランザクション状況の forget

リソース・マネージャーに、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) トランザクションに保持されていたリソースを解放させます。未確定 XA トランザクションが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後に、この API を呼び出すことがあります。

許可

なし

必要な接続

データベース

API インクルード・ファイル

```
sqlxa.h
```

API とデータ構造構文

```
extern int SQL_API_FN sqlxhfrg(
   SQLXA_XID *pTransId,
   struct sqlca *pSqlca
   );
```

sqlxhfrg API パラメーター

pTransId
入力。ヒューリスティックに「forget」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 構造を指すポインター。

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。

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

なし

必要な接続

データベース

API インクルード・ファイル

```
sqlxa.h
```

API とデータ構造構文

```
extern int SQL_API_FN sqlxphcm(
   int exe_type,
   SQLXA_XID *pTransId,
   struct sqlca *pSqlca
   );
```

sqlxphcm 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 構造を指すポインター。

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 API が呼び出されるまで、トランザクションの状態を記憶します。

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

なし

必要な接続

データベース

API インクルード・ファイル

```
sqlxa.h
```

API とデータ構造構文

```
extern int SQL_API_FN sqlxphrl(
   int exe_type,
   SQLXA_XID *pTransId,
   struct sqlca *pSqlca
   );
```

sqlxphrl 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 構造を指すポインター。

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 API が呼び出されるまで、トランザクションの状態を記憶します。

第 7 章 並行アクセスを伴うスレッド化アプリケーション

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 インクルード・ファイル

```
sql.h
```

API とデータ構造構文

```
SQL_API_RC SQL_API_FN
  sqleAttachToCtx (
        void * pCtx,
        void * reserved,
        struct sqlca * pSqlca);
```

sqleAttachToCtx 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。

pSqlca 出力。 sqlca 構造を指すポインター。

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 API』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 インクルード・ファイル

```
sql.h
```

API とデータ構造構文

```
SQL_API_RC SQL_API_FN
  sqleBeginCtx (
        void ** ppCtx,
        sqlint32 lOptions,
        void * reserved,
        struct sqlca * pSqlca);
```

sqleBeginCtx 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。スレッドが既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。

pSqlca 出力。 sqlca 構造を指すポインター。

sqleDetachFromCtx - コンテキストからのデタッチ

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デタッチします。コンテキストがデタッチされるのは、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 インクルード・ファイル

sql.h

API とデータ構造構文

```
SQL_API_RC SQL_API_FN
  sqleDetachFromCtx (
        void * pCtx,
        void * reserved,
        struct sqlca * pSqlca);
```

sqleDetachFromCtx 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。

pSqlca 出力。 sqlca 構造を指すポインター。

---

sqleEndCtx - 特定のアプリケーション・コンテキストに関連付けられているメモリーのデタッチと解放

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 インクルード・ファイル

sql.h

API とデータ構造構文

```
SQL_API_RC SQL_API_FN
  sqleEndCtx (
        void ** ppCtx,
        sqlint32 lOptions,
        void * reserved,
        struct sqlca * pSqlca);
```

sqleEndCtx 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、以前にデタッチ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: デタッチは、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。

pSqlca 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、インスタンス接続を確立する API (例えば、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デタッチすることが必要です。

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 インクルード・ファイル

sql.h

API とデータ構造構文

```
SQL_API_RC SQL_API_FN
  sqleGetCurrentCtx (
        void ** ppCtx,
        void * reserved,
        struct sqlca * pSqlca);
```

sqleGetCurrentCtx 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。

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。

pSqlca 出力。 sqlca 構造を指すポインター。

---

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 インクルード・ファイル

sql.h

API とデータ構造構文

```
SQL_API_RC SQL_API_FN
  sqleInterruptCtx (
        void * pCtx,
        void * reserved,
        struct sqlca * pSqlca);
```

sqleInterruptCtx 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。

pSqlca 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

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 インクルード・ファイル

```
sql.h
```

API とデータ構造構文

```
SQL_API_RC SQL_API_FN
  sqleSetTypeCtx (
        sqlint32 lOptions);
```

sqleSetTypeCtx 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 API
- sqleAttachToCtx API
- sqleDetachFromCtx API
- sqleEndCtx API

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 は無効になります。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 API 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。

第 8 章 データベース管理をカスタマイズするための DB2 データベース・システム・プラグイン

DB2 データベース製品にはプラグイン・インターフェースが組み込まれており、ユーザーとサード・パーティー・ベンダーは、これを使用して一部のデータベース管理機能をカスタマイズできます。

現在、DB2 データベース・システムには以下の 3 つのタイプのプラグインがあります。

- セキュリティー・プラグイン。DB2 データベース・システムの認証と、グループ・メンバーシップの検索動作をカスタマイズするためのものです。
- バックアップおよびリストア・プラグイン。DB2 データベース・システム提供のバックアップおよびリストア機能ではサポートされていない装置で、データのバックアップとリストアを行うためのものです。
- 圧縮プラグイン。バックアップ・イメージを圧縮および解凍するためのものです。

上記 3 つのプラグインを使用して提供される機能は、DB2 データベース・システム製品に組み込まれていますが、DB2 データベース・システムの動作をカスタマイズまたは増強する場合は、独自のプラグインを作成するか、ベンダーから購入できます。

各プラグインは、API およびデータ構造の動的にロード可能なライブラリーです。API およびデータ構造のプロトタイプは DB2 データベース・システムで提供されており、インプリメンテーションはベンダーから提供されます。DB2 データベース・システムは、一部の API およびデータ構造のインプリメンテーションを提供しています。DB2 データベース・システムによりインプリメントされるプラグイン API およびデータ構造のリストについては、それぞれのプラグインのトピックを参照してください。このインプリメンテーションは、UNIX システムでは共用ライブラリーの形になっており、Windows プラットフォームでは DLL の形になっています。DB2 データベース・システムが特定のプラグインを検索する実際の場所については、それぞれのプラグインのトピックを参照してください。

プラグイン API は、2 つの点で DB2 API (db2Export、db2Backup など) とは異なります。まず、プラグイン API のインプリメンテーションは、多くの場合、ベンダーから提供されます。一方、DB2 API のインプリメンテーションは、DB2 により提供されます。次に、プラグイン API が DB2 から呼び出されるのに対し、DB2 API はユーザーがクライアント・アプリケーションから呼び出します。このため、プラグイン API のトピックにパラメーターが入力としてリストされている場合は、DB2 がこのパラメーターの値を入力し、パラメーターが出力としてリストされている場合は、ベンダーによる API のインプリメンテーションがパラメーターの値を入力する役目を担うということになります。

プラグインの有効化

グループ検索プラグインの展開

DB2 セキュリティー・システムのグループ検索動作をカスタマイズするには、独自のグループ検索プラグインを開発するか、または第三者からこれを購入できます。

データベース管理システムに適したグループ検索プラグインを入手した後、それを展開することができます。

- グループ検索プラグインをデータベース・サーバー上に展開するには、以下のステップを実行します。
  1. グループ検索プラグイン・ライブラリーをサーバーのグループ・プラグイン・ディレクトリーにコピーします。
  2. データベース・マネージャー構成パラメーター *group_plugin* をプラグインの名前で更新します。
- グループ検索プラグインをデータベース・クライアント上に展開するには、以下のステップを実行します。
  1. グループ検索プラグイン・ライブラリーをクライアントのグループ・プラグイン・ディレクトリーにコピーします。
  2. データベース・クライアント上で、データベース・マネージャー構成パラメーター *group_plugin* をプラグインの名前で更新します。

ユーザー ID/パスワード・プラグインのデプロイ

DB2 セキュリティー・システムのユーザー ID/パスワード認証動作をカスタマイズするには、独自のユーザー ID/パスワード認証プラグインを開発するか、または第三者からこれを購入できます。

すべてのユーザー ID/パスワード・ベース認証プラグインは、意図しているそれらプラグインの使用法に応じて、クライアントのプラグイン・ディレクトリーか、サーバーのプラグイン・ディレクトリーのいずれかに置く必要があります。プラグインがクライアントのプラグイン・ディレクトリーに置かれる場合、それはローカル許可検査のためと、クライアントがサーバーと接続しようとするときのクライアントの妥当性検査のために使用されます。プラグインがサーバーのプラグイン・ディレクトリーに置かれる場合、それはサーバーへの着信接続の処理のためと、 USER または GROUP キーワードが指定されずに GRANT ステートメントが発行された場合の許可 ID の存在とその有効性の検査のために使用されます。ほとんどの場合、ユーザー ID/パスワード認証で必要となるのは、サーバー・サイドのプラグインのみです。一般にそれほど役に立ちませんが、クライアントのユーザー ID/パスワード・プラグインのみを使用することも可能です。非常に稀なことですが、クライアントとサーバーの両方に、一致するユーザー ID/パスワード・プラグインが必要になることもあります。

注: *既存の プラグインの新しい* バージョンをデプロイする場合は、DB2 サーバー、またはプラグインを使用するすべてのアプリケーションをまず停止する必要があります。新しいバージョンを (同じ名前で) 上書きコピーするときに、プラグインを使用するプロセスがまだ実行中になっていると、未定義の動作 (トラップなど) が

発生します。この制約事項は、プラグインを初めてデプロイする場合や、プラグインが使用中になっていない場合には適用されません。

データベース管理システムに適したユーザー ID/パスワード認証プラグインを入手した後、それらをデプロイすることができます。

- ユーザー ID/パスワード認証プラグインをデータベース・サーバー上にデプロイするには、データベース・サーバー上で以下のステップを実行します。
  1. ユーザー ID/パスワード認証プラグイン・ライブラリーをサーバーのプラグイン・ディレクトリーにコピーします。
  2. データベース・マネージャー構成パラメーター *srvcon_pw_plugin* をサーバー・プラグインの名前で更新します。このプラグインは、サーバーが接続 (CONNECT) 要求およびアタッチ (ATTACH) 要求を処理する際に使用します。
  3. 次のいずれかを行ってください。
     - データベース・マネージャー構成パラメーター *srvcon_auth* に、CLIENT、SERVER、SERVER_ENCRYPT、DATA_ENCRYPT、または DATA_ENCRYPT_CMP の認証タイプを設定します。あるいは、
     - データベース・マネージャー構成パラメーター *srvcon_auth* に NOT_SPECIFIED を設定し、 *authentication* に CLIENT、 SERVER、 SERVER_ENCRYPT、 DATA_ENCRYPT、または DATA_ENCRYPT_CMP の認証タイプを設定します。
- ユーザー ID/パスワード認証プラグインをデータベース・クライアント上にデプロイするには、各クライアント上で以下のステップを実行します。
  1. ユーザー ID/パスワード認証プラグイン・ライブラリーをクライアントのプラグイン・ディレクトリーにコピーします。
  2. データベース・マネージャー構成パラメーター *clnt_pw_plugin* をクライアント・プラグインの名前で更新します。このプラグインは、認証がどこで行われているかに関係なく、つまり、データベース構成パラメーター *authentication* が CLIENT に設定されているとき以外にもロードされ、呼び出されます。
- ユーザー ID/パスワード認証プラグインを使用したクライアント、サーバー、またはゲートウェイでのローカル許可に関しては、各クライアント、サーバー、またはゲートウェイ上で以下のステップを実行します。
  1. ユーザー ID/パスワード認証プラグイン・ライブラリーをクライアント、サーバー、またはゲートウェイ上のクライアント・プラグイン・ディレクトリーにコピーします。
  2. データベース・マネージャー構成パラメーター *clnt_pw_plugin* をプラグインの名前で更新します。
  3. *authentication* データベース・マネージャー構成パラメーターに、 CLIENT、SERVER、 SERVER_ENCRYPT、 DATA_ENCRYPT、または DATA_ENCRYPT_CMP を設定します。

GSS-API プラグインのデプロイ

DB2 セキュリティー・システムの認証動作をカスタマイズするには、 GSS-API を使用して独自の認証プラグインを開発するか、または第三者からこれを購入できます。

Kerberos 以外のプラグイン・タイプの場合は、クライアントとサーバー上に、同じプラグイン・タイプだけでなく、一致するプラグイン名がなければなりません。クライアントとサーバー上のプラグインは、同一のベンダーのものである必要はありませんが、これらは互換性のある GSS-API トークンを生成して使用する必要があります。 Kerberos プラグインは標準化されているので、クライアントとサーバー上にはどのような組み合わせの Kerberos プラグインをデプロイしても構いません。しかし、それほど標準化されていない *x.509* 証明書などの種々の GSS-API メカニズムのインプリメンテーション間には、DB2 データベース・システムとの部分的な互換性しかない可能性があります。すべての GSS-API 認証プラグインは、意図しているそれらプラグインの使用法に応じて、クライアントのプラグイン・ディレクトリーか、サーバーのプラグイン・ディレクトリーのいずれかに置く必要があります。プラグインがクライアントのプラグイン・ディレクトリーに置かれる場合、それはローカル許可検査のためと、クライアントがサーバーと接続しようとするときに使用されます。プラグインがサーバーのプラグイン・ディレクトリーに置かれる場合、それはサーバーへの着信接続の処理のためと、 USER または GROUP キーワードが指定されずに GRANT ステートメントが発行された場合の許可 ID の存在とその有効性の検査のために使用されます。

注: *既存の プラグインの新しい* バージョンをデプロイする場合は、DB2 サーバー、またはプラグインを使用するすべてのアプリケーションをまず停止する必要があります。新しいバージョンを (同じ名前で) 上書きコピーするときに、プラグインを使用するプロセスがまだ実行中になっていると、未定義の動作 (トラップなど) が発生します。この制約事項は、プラグインを初めてデプロイする場合や、プラグインが使用中になっていない場合には適用されません。

データベース管理システムに適した GSS-API 認証プラグインを入手した後、それらをデプロイすることができます。

- GSS-API 認証プラグインをデータベース・サーバー上にデプロイするには、サーバー上で以下のステップを実行します。
  1. GSS-API 認証プラグイン・ライブラリーをサーバー・プラグイン・ディレクトリーにコピーします。このディレクトリーには、多数の GSS-API プラグインをコピーすることができます。
  2. データベース・マネージャー構成パラメーター *srvcon_gssplugin_list* を、GSS-API プラグイン・ディレクトリーにインストールされたプラグインの名前の順番のコンマ区切りのリストで更新します。
  3. 次のいずれかを行ってください。
     - データベース・マネージャー構成パラメーター *srvcon_auth* を GSSPLUGIN または GSS_SERVER_ENCRYPT に設定することは、サーバーが GSSAPI PLUGIN 認証方式を使用できるようにする一つの方法です。あるいは、
     - データベース・マネージャー構成パラメーター *srvcon_auth* を NOT_SPECIFIED に設定し、*authentication* を GSSPLUGIN または GSS_SERVER_ENCRYPT に設定することは、サーバーが GSSAPI PLUGIN 認証方式を使用できるようにする一つの方法です。
- GSS-API 認証プラグインをデータベース・クライアント上にデプロイするには、各クライアント上で以下のステップを実行します。
  1. GSS-API 認証プラグイン・ライブラリーをクライアントのプラグイン・ディレクトリーにコピーします。このディレクトリーには、多数の GSS-API プラ

グインをコピーすることができます。クライアントは、クライアント上で使用できる、サーバーのプラグインのリストに含まれている最初の GSS-API を選出することによって、接続 (CONNECT) またはアタッチ (ATTACH) 操作中の認証用の GSS-API プラグインを選択します。

2. オプション: クライアントがアクセスするデータベースをカタログし、クライアントが GSS-API 認証プラグインのみを認証メカニズムとして受け入れることを示します。以下に例を示します。

   ```
   CATALOG DB testdb AT NODE testnode AUTHENTICATION GSSPLUGIN
   ```

- GSS-API 認証プラグインを使用したクライアント、サーバー、またはゲートウェイでのローカル許可に関しては、以下のステップを実行します。
  1. GSS-API 認証プラグイン・ライブラリーをクライアント、サーバー、またはゲートウェイ上のクライアント・プラグイン・ディレクトリーにコピーします。
  2. データベース・マネージャー構成パラメーター *local_gssplugin* をプラグインの名前で更新します。
  3. *authentication* データベース・マネージャー構成パラメーターに、GSSPLUGIN、または GSS_SERVER_ENCRYPT を設定します。

Kerberos プラグインのデプロイ

DB2 セキュリティー・システムの Kerberos 認証動作をカスタマイズするには、独自の Kerberos 認証プラグインを開発するか、または第三者からこれを購入できます。 Kerberos セキュリティー・プラグインは IPv6 をサポートしないことに注意してください。

注: *既存の* プラグインの*新しい* バージョンをデプロイする場合は、DB2 サーバー、またはプラグインを使用するすべてのアプリケーションをまず停止する必要があります。新しいバージョンを (同じ名前で) 上書きコピーするときに、プラグインを使用するプロセスがまだ実行中になっていると、未定義の動作 (トラップなど) が発生します。この制約事項は、プラグインを初めてデプロイする場合や、プラグインが使用中になっていない場合には適用されません。

データベース管理システムに適した Kerberos 認証プラグインを入手した後、それらをデプロイすることができます。

- Kerberos 認証プラグインをデータベース・サーバー上にデプロイするには、サーバー上で以下のステップを実行します。
  1. Kerberos 認証プラグイン・ライブラリーをサーバー・プラグイン・ディレクトリーにコピーします。
  2. 順番のコンマ区切りのリストとして指定されるデータベース・マネージャー構成パラメーター **srvcon_gssplugin_list** を更新して、 Kerberos サーバー・プラグイン名を含めます。このリストの中の 1 つのプラグインのみを Kerberos プラグインにすることができます。このリストがブランクで、 **authentication** に KERBEROS または KRB_SVR_ENCRYPT が設定されている場合は、デフォルトの DB2 Kerberos プラグイン IBMkrb5 が使用されます。
  3. 必要に応じ、**srvcon_auth** データベース・マネージャー構成パラメーターを設定して、現行の認証タイプをオーバーライドします。 **srvcon_auth** データベース・マネージャー構成パラメーターを設定しない場合、DB2 データベー

ス・マネージャーは、**authentication** 構成パラメーターの値を使用します。**authentication** 構成パラメーターが、以下の認証タイプのいずれかに現在設定されている場合、Kerberos プラグインをデプロイして使用できます。

- KERBEROS
- KRB_SERVER_ENCRYPT
- GSSPLUGIN
- GSS_SERVER_ENCRYPT

現行の認証タイプをオーバーライドする必要がある場合は、**srvcon_auth** 構成パラメーターを以下の認証タイプのいずれかに設定します。

- KERBEROS
- KRB_SERVER_ENCRYPT
- GSSPLUGIN
- GSS_SERVER_ENCRYPT

- Kerberos 認証プラグインをデータベース・クライアント上にデプロイするには、各クライアント上で以下のステップを実行します。
  1. Kerberos 認証プラグイン・ライブラリーをクライアントのプラグイン・ディレクトリーにコピーします。
  2. データベース・マネージャー構成パラメーター **clnt_krb_plugin** を Kerberos プラグインの名前で更新します。 **clnt_krb_plugin** がブランクの場合、 DB2 はクライアントは Kerberos 認証を使用できないとみなします。サーバーがプラグインをサポートできない場合にのみ、この設定は適切です。サーバーとクライアントの両方がセキュリティー・プラグインをサポートする場合、デフォルトのサーバー・プラグイン *IBMkrb5* が **clnt_krb_plugin** のクライアントの値よりも優先して使用されます。 Kerberos 認証プラグインを使用したクライアント、サーバー、またはゲートウェイでのローカル許可に関しては、以下のステップを実行します。
     a. Kerberos 認証プラグイン・ライブラリーをクライアント、サーバー、またはゲートウェイ上のクライアント・プラグイン・ディレクトリーにコピーします。
     b. データベース・マネージャー構成パラメーター **clnt_krb_plugin** をプラグインの名前で更新します。
     c. **authentication** データベース・マネージャー構成パラメーターに、KERBEROS または KRB_SERVER_ENCRYPT を設定します。
  3. オプション: クライアントがアクセスするデータベースをカタログし、クライアントが Kerberos 認証プラグインのみを使用することを示します。以下に例を示します。

```
CATALOG DB testdb AT NODE testnode AUTHENTICATION KERBEROS
        TARGET PRINCIPAL service/host@REALM
```

注: Kerberos をサポートするプラットフォームの場合、 `IBMkrb5` ライブラリーはクライアント・プラグイン・ディレクトリーに置かれます。 Kerberos プラグインは GSS-API プラグインを使用してインプリメントされているので、 DB2 はこのライブラリーを有効な GSS-API プラグインとして認識します。

セキュリティー・プラグインの作成

DB2 によるセキュリティー・プラグインのロード方法

各プラグイン・ライブラリーには、以下のような、プラグイン・タイプに応じた特定の名前を持つ初期化関数が含まれていなければなりません。

- サーバー・サイド認証プラグイン: db2secServerAuthPluginInit()
- クライアント・サイド認証プラグイン: db2secClientAuthPluginInit()
- グループ・プラグイン: db2secGroupPluginInit()

この関数は、プラグイン初期化関数と呼ばれます。プラグイン初期化関数は、指定されたプラグインを初期化し、プラグインの関数を呼び出すために必要な情報を DB2 に提供します。プラグイン初期化関数は、以下のパラメーターを受け入れます。

- プラグインを呼び出す DB2 インスタンスがサポートできる関数ポインター構造の最高バージョン番号
- インプリメンテーションを必要とするすべての API を指すポインターを収めた構造を指すポインター
- db2diag.log ファイルにログ・メッセージを追加する関数を指すポインター
- エラー・メッセージ・ストリングを指すポインター
- エラー・メッセージの長さ

以下は、グループ検索プラグインの初期化関数の関数シグニチャーです。

```
SQL_API_RC SQL_API_FN db2secGroupPluginInit(
  db2int32 version,
  void *group_fns,
  db2secLogMessage *logMessage_fn,
  char **errormsg,
  db2int32  *errormsglen);
```

注: プラグイン・ライブラリーを C++ でコンパイルする場合は、 extern "C" を使用してすべての関数を宣言する必要があります。 DB2 は、基礎オペレーティング・システムの動的ローダーを利用して、C++ のユーザー作成プラグイン・ライブラリーの内部で使用されている C++ コンストラクターおよびデストラクターを処理します。

初期化関数は、規定の関数名を使用しなければならない、プラグイン・ライブラリー内の唯一の関数です。その他のプラグイン関数は、初期化関数から戻された関数ポインターを通して参照されます。サーバー・プラグインは、DB2 サーバーの始動時にロードされます。クライアント・プラグインは、クライアント上で必要とされるときにロードされます。 DB2 は、プラグイン・ライブラリーをロードするとすぐに、この初期化関数の位置を解決して呼び出します。この関数固有のタスクは、以下のとおりです。

- 関数ポインターを、適切な関数構造を指すポインターにキャストする
- ライブラリー内の他の関数を指すポインターに入力する
- 戻される関数ポインター構造のバージョン番号に入力する

DB2 はプラグイン初期化関数を複数回呼び出すことがあります。このことが起こるのは、アプリケーションが動的に DB2 クライアント・ライブラリーをロードして

からこれをアンロードして再ロードし、再ロードの前と後の両方にプラグインから認証関数を実行した場合です。このような場合は、プラグイン・ライブラリーがアンロードされず、したがって再ロードもされないことがあります。ただし、この動作はオペレーティング・システムによって異なります。

別の例として、データベース・サーバー自身がクライアントとして振る舞うことがある、ストアード・プロシージャーやフェデレーテッド・システム呼び出しの実行時にも、DB2 がプラグイン初期化関数を複数回呼び出すことがあります。データベース・サーバー上のクライアント・プラグインとサーバー・プラグインが同じファイル内にある場合、 DB2 はプラグイン初期化関数を 2 回呼び出す可能性があります。

db2secGroupPluginInit が複数回呼び出されたことを検出した場合、プラグインは、プラグイン・ライブラリーを終了して再初期化するよう指示されたものとして、このイベントを処理する必要があります。したがって、プラグイン初期化関数は、db2secPluginTerm を呼び出すと実行されるクリーンアップ・タスクをすべて実行してから、再び関数ポインターのセットを戻す必要があります。

UNIX または Linux ベースのオペレーティング・システムが稼働している DB2 サーバーでは、DB2 は異なるプロセスでプラグイン・ライブラリーを複数回ロードして再初期化することがあります。

セキュリティー・プラグイン・ライブラリーの開発に関する制約事項

以下は、プラグイン・ライブラリーの作成に関連した制約事項です。

C-linkage
プラグイン・ライブラリーは、C-linkage とリンクされていなければなりません。プロトタイプ、プラグインのインプリメントに必要なデータ構造、およびエラー・コード定義を規定するヘッダー・ファイルは、C/C++ の場合にのみ準備されます。プラグイン・ライブラリーが C++ としてコンパイルされている場合は、 DB2 がロード時に解決する関数を extern ″C″ を用いて宣言する必要があります。

.NET 共通言語ランタイムはサポートされていません。
プラグイン・ライブラリーのソース・コードのコンパイルおよびリンクにおいて、 .NET 共通言語ランタイム (CLR) はサポートされません。

シグナル・ハンドラー
プラグイン・ライブラリーは、シグナル・ハンドラーをインストールしたり、シグナル・マスクを変更したりしてはなりません。なぜなら、これをすると、DB2 のシグナル・ハンドラーが妨げられるからです。 DB2 のシグナル・ハンドラーが妨げられると、プラグイン・コード自体にあるトラップを含めたエラーを報告してリカバリーする DB2 の機能が著しく妨げられます。さらに、プラグイン・ライブラリーは、C++ 例外を出してはなりません。なぜなら、これも DB2 のエラー処理を妨げるからです。

スレッド・セーフ
プラグイン・ライブラリーは、スレッド・セーフおよび再入可能でなければなりません。プラグイン初期化関数は、再入可能でなくてもよい唯一の API

です。プラグイン初期化関数は異なるプロセスから複数回呼び出される可能性がありますが、その場合は、プラグインがすべての使用済みリソースをクリーンアップして、プラグイン自体を再初期化します。

終了ハンドラー、および標準 C ライブラリーとオペレーティング・システム呼び出しのオーバーライド

プラグイン・ライブラリーは、標準 C ライブラリーやオペレーティング・システム呼び出しをオーバーライドしてはなりません。さらに、プラグイン・ライブラリーは、終了ハンドラーや `pthread_atfork` ハンドラーをインストールしてはなりません。終了ハンドラーはプログラムが終了する前にアンロードされる可能性があるので、終了ハンドラーを使用することはお勧めしません。

ライブラリーの従属関係

Linux または UNIX では、プラグイン・ライブラリーをロードするプロセスは、`setuid` か `setgid` になります。このことは、プロセスが `$LD_LIBRARY_PATH`、 `$SHLIB_PATH`、または `$LIBPATH` 環境変数を利用して従属ライブラリーを検索できないことを意味します。したがって、従属ライブラリーが次のような他の方法でアクセス可能にされていない限り、プラグイン・ライブラリーが追加のライブラリーに従属していてはなりません。

- `/lib` または `/usr/lib` の中に入れる。
- それらが常駐するディレクトリーを OS ワイド (Linux 上の `ld.so.conf` ファイル内など) で指定する。
- プラグイン・ライブラリー自体の `RPATH` で指定する。

この制限は、Windows オペレーティング・システムには当てはまりません。

シンボルの重複

可能であれば、プラグイン・ライブラリーは、シンボルの重複の可能性を減らすオプションとして使用できるオプション (アンバインドされた外部シンボル参照を削減するオプションなど) を用いてコンパイルおよびリンクする必要があります。例えば、HP、Solaris、および Linux 上で ″-Bsymbolic″ リンカー・オプションを使用するなら、シンボルの重複に関係した問題を防ぐことができます。ただし、AIX で作成されたプラグインの場合は、`"-brtl"` リンカー・オプションは明示的にも暗黙的にも使用しないでください。

32 ビット・アプリケーションと 64 ビット・アプリケーション

32 ビット・アプリケーションは、32 ビット・プラグインを使用する必要があります。 64 ビット・アプリケーションは、64 ビット・プラグインを使用する必要があります。詳細については、32 ビットと 64 ビットの考慮事項に関するトピックを参照してください。

テキスト・ストリング

入力テキスト・ストリングがヌル終了になっているという保証はなく、出力ストリングがヌル終了である必要はありません。その代わり、すべての入力ストリングに対して整数の長さが指定され、戻される長さとして整数を指すポインターが指定されます。

許可 ID パラメーターの引き渡し

DB2 がプラグインに渡す許可 ID (authid) パラメーター (入力 authid パラメーター) には、埋め込みブランクが除かれた大文字の authid が含まれま

す。プラグインが DB2 に戻す authid パラメーター (出力 authid パラメーター) には特別な処理は必要ありませんが、DB2 は authid を大文字に変換し、内部 DB2 規格に準じてブランクを埋め込みます。

パラメーターのサイズ制限

プラグイン API は、パラメーターの長さ制限として以下を使用します。

```
#define DB2SEC_MAX_AUTHID_LENGTH 255
#define DB2SEC_MAX_USERID_LENGTH 255
#define DB2SEC_MAX_USERNAMESPACE_LENGTH 255
#define DB2SEC_MAX_PASSWORD_LENGTH 255
#define DB2SEC_MAX_DBNAME_LENGTH 128
```

特定のプラグイン・インプリメンテーションでは、許可 ID、ユーザー ID、およびパスワードの最大長は、小さくする必要があるか、あるいは強制的に小さくされる可能性があります。特に、DB2 データベース・システムに付属しているオペレーティング・システム認証プラグインは、オペレーティング・システムの限界が上記の限界より低い場合、オペレーティング・システムが施行する最大ユーザー長、最大グループ長、および最大ネーム・スペース長の限界の制約を受けます。

AIX でのセキュリティー・プラグイン・ライブラリーの拡張子

AIX システムでは、セキュリティー・プラグイン・ライブラリーは、*.a* または *.so* というファイル名拡張子を持つことができます。プラグイン・ライブラリーをロードするのに使用するメカニズムは、次のように、どの拡張子が使用されているかによって異なります。

- ファイル名拡張子が *.a* のプラグイン・ライブラリーは、共用オブジェクト・メンバーを含むアーカイブであると想定されます。そのようなメンバーには、*shr.o* (32 ビット) または *shr64.o* (64 ビット) という名前を付けなければなりません。 32 ビットおよび 64 ビットの両方のメンバーを 1 つのアーカイブに収容することができ、それによって、両方のタイプのプラットフォームにデプロイすることができます。

  例えば、32 ビット・アーカイブ・スタイルのプラグイン・ライブラリーを作成するには、次のようにします。

  ```
  xlc_r -qmkshrobj -o shr.o MyPlugin.c -bE:MyPlugin.exp
  ar rv MyPlugin.a shr.o
  ```

- ファイル名拡張子が *.so* のプラグイン・ライブラリーは、動的にロード可能な共用オブジェクトであると想定されます。そのようなオブジェクトは、その作成時に使用したコンパイラーおよびリンカーのオプションに応じて、32 ビットまたは 64 ビットのどちらかになります。例えば、32 ビットのプラグイン・ライブラリーを作成するには、次のようにします。

  ```
  xlc_r -qmkshrobj -o MyPlugin.so MyPlugin.c -bE:MyPlugin.exp
  ```

AIX 以外のすべてのプラットフォームでは、セキュリティー・プラグイン・ライブラリーは、常に動的にロード可能な共有オブジェクトであるとみなされます。

セキュリティー・プラグインに関する制約事項

以下は、セキュリティー・プラグインの使用に関する制限事項です。

DB2 データベース・ファミリーのサポートに関する制約事項

GSS-API プラグインを使用して、Linux、UNIX、およびWindows 上の DB2 クライアントと、DB2 for z/OS などの別の DB2 ファミリー・サーバーとの間の接続を認証することはできません。また、クライアントとして機能する他の DB2 データベース・ファミリー製品から Linux、UNIX、または Windows 上の DB2 サーバーへの接続も認証できません。

ただし、Linux、UNIX、または Windows 上の DB2 クライアントを使用して他の DB2 データベース・ファミリー・サーバーに接続する場合には、クライアント・サイドのユーザー ID/パスワード・プラグイン (IBM 提供のオペレーティング・システム認証プラグインなど) を使用したり、独自のユーザー ID/パスワード・プラグインを作成したりすることができます。また、組み込みの Kerberos プラグインの使用や、自分独自のプラグインのインプリメントを行ってもかまいません。

Linux、 UNIX、または Windows 上の DB2 クライアントでは、GSSPLUGIN 認証タイプを使用してデータベースをカタログしてはなりません。

AUTHID ID に関する制限DB2 データベース・システムのバージョン 9.5 以降では、128 バイトの許可 ID を持つことができますが、その許可 ID がオペレーティング・システムのユーザー ID またはグループ名として解釈される場合、オペレーティング・システムの命名上の制約が適用されます (例えば、ユーザー ID の制限は 8 または 30 文字、グループ名の制限は 30 文字です)。このため、128 バイトの許可 ID を付与できますが、この許可 ID を持つユーザーとしては接続することができません。独自のセキュリティー・プラグインを作成した場合は、許可 ID の拡張されたサイズを最大限に活用することができます。例えば、セキュリティー・プラグインに 30 バイトのユーザー ID を与えて、接続可能な認証中に、セキュリティー・プラグインが 128 バイトの許可 ID を返すことができます。

WebSphere® フェデレーション・サーバーのサポートに関する制約事項

DB2 II は、GSS_API プラグインからの委任証明書を使用して、データ・ソースへのアウトバウンド接続を確立することをサポートしていません。データ・ソースへの接続には、引き続き CREATE USER MAPPING コマンドを使用する必要があります。

データベース管理サーバーのサポートに関する制約事項

DB2 Administration Server (DAS) はセキュリティー・プラグインをサポートしていません。DAS はオペレーティング・システムの認証メカニズムのみをサポートします。

DB2 クライアントでのセキュリティー・プラグインに関する問題および制約事項 (Windows)

Windows オペレーティング・システム上の DB2 クライアント内でデプロイする予定のセキュリティー・プラグインの開発時には、プラグイン終了関数の中でどの補助ライブラリーもアンロードしないでください。この制約事項は、グループ、ユーザー ID とパスワード、Kerberos、および GSS-API プラグインを含む、すべてのタイプのクライアント・セキュリティー・プラグインに対して適用されます。このよ

うな、db2secPluginTerm、db2secClientAuthPluginTerm、および db2secServerAuthPluginTerm といった終了 API は、どの Windows プラットフォームでも呼び出されないので、該当するリソース・クリーンアップを行う必要があります。

この制約事項は、Windows での DLL のアンロードに関連したクリーンアップ問題に関係しています。

AIX 上での .a または .so の拡張子の付いたプラグイン・ライブラリーのロード

AIX では、セキュリティー・プラグイン・ライブラリーには、.a または .so のファイル名拡張子をつけることができます。プラグイン・ライブラリーをロードするのに使用するメカニズムは、次のように、どの拡張子が使用されているかによって異なります。

- .a のファイル名拡張子の付いたプラグイン・ライブラリー

  .a のファイル名拡張子の付いたプラグイン・ライブラリーは、共有オブジェクト・メンバーを収容するアーカイブであるとみなされます。そのようなメンバーには、shr.o (32 ビット) または shr64.o (64 ビット) という名前を付けなければなりません。 32 ビットおよび 64 ビットの両方のメンバーを 1 つのアーカイブに収容することができ、それによって、両方のタイプのプラットフォームにデプロイすることができます。

  例えば、32 ビット・アーカイブ・スタイルのプラグイン・ライブラリーを作成するには、次のようにします。

  ```
  xlc_r -qmkshrobj -o shr.o MyPlugin.c -bE:MyPlugin.exp
  ar rv MyPlugin.a shr.o
  ```

- .so のファイル名拡張子の付いたプラグイン・ライブラリー

  .so のファイル名拡張子の付いたプラグイン・ライブラリーは、動的にロード可能な共有オブジェクトであるとみなされます。そのようなオブジェクトは、その作成時に使用したコンパイラーおよびリンカーのオプションに応じて、32 ビットまたは 64 ビットのどちらかになります。例えば、32 ビットのプラグイン・ライブラリーを作成するには、次のようにします。

  ```
  xlc_r -qmkshrobj -o MyPlugin.so MyPlugin.c -bE:MyPlugin.exp
  ```

AIX 以外のすべてのプラットフォームでは、セキュリティー・プラグイン・ライブラリーは、常に動的にロード可能な共有オブジェクトであるとみなされます。

GSS-API セキュリティー・プラグインでは、メッセージの暗号化と署名はサポートされない

メッセージの暗号化および署名は、GSS-API セキュリティー・プラグインでは使用できません。

セキュリティー・プラグインの戻りコード

すべてのセキュリティー・プラグイン API は、API の実行の成功や失敗を示すために整数の値を戻す必要があります。戻りコード値 0 は、API が正常に実行したことを示します。 -3、-4、および -5 以外のすべての負の戻りコードは、API がエラーを検出したことを示します。

-3、-4、または -5 が付く戻りコードを除き、セキュリティー・プラグイン API から戻されるすべての負の戻りコードは、SQLCODE -1365、SQLCODE -1366、または SQLCODE -30082 にマップされます。 -3、-4、および -5 の値は、許可 ID が有効なユーザーまたはグループを表しているかどうかを示すために使用されます。

すべてのセキュリティー・プラグイン API の戻りコードは、DB2 の組み込みディレクトリー SQLLIB/include にある db2secPlugin.h で定義されます。

すべてのセキュリティー・プラグインの戻りコードに関する詳細については、以下の表で説明しています。

表 10. セキュリティー・プラグインの戻りコード

戻りコード	定義値	意味	関連 API
0	DB2SEC_PLUGIN_OK	プラグイン API が正常に実行されました。	すべて
-1	DB2SEC_PLUGIN_UNKNOWNERROR	プラグイン API で想定外のエラーが発生しました。	すべて
-2	DB2SEC_PLUGIN_BADUSER	入力として渡されたユーザー ID が定義されていません。	db2secGenerateInitialCred db2secValidatePassword db2secRemapUserid db2secGetGroupsForUser
-3	DB2SEC_PLUGIN _INVALIDUSERORGROUP	このユーザーまたはグループがありません。	db2secDoesAuthIDExist db2secDoesGroupExist
-4	DB2SEC_PLUGIN _USERSTATUSNOTKNOWN	ユーザー状況が不明です。これは DB2 ではエラーとして扱われません。これは、GRANT ステートメントが、 authid がユーザーまたはオペレーティング・システム・グループのどちらを表しているか判別するために使用します。	db2secDoesAuthIDExist
-5	DB2SEC_PLUGIN _GROUPSTATUSNOTKNOWN	グループ状況が不明です。これは DB2 ではエラーとして扱われません。これは、GRANT ステートメントが、 authid がユーザーまたはオペレーティング・システム・グループのどちらを表しているか判別するために使用します。	db2secDoesGroupExist
-6	DB2SEC_PLUGIN_UID_EXPIRED	ユーザー ID が期限切れです。	db2secValidatePassword db2GetGroupsForUser db2secGenerateInitialCred

表 10. セキュリティー・プラグインの戻りコード (続き)

戻りコード	定義値	意味	関連 API
-7	DB2SEC_PLUGIN_PWD_EXPIRED	パスワードが期限切れです。	db2secValidatePassword db2GetGroupsForUser db2secGenerateInitialCred
-8	DB2SEC_PLUGIN_USER_REVOKED	ユーザーが失効しています。	db2secValidatePassword db2GetGroupsForUser
-9	DB2SEC_PLUGIN _USER_SUSPENDED	ユーザーが一時失効しています。	db2secValidatePassword db2GetGroupsForUser
-10	DB2SEC_PLUGIN_BADPWD	パスワードが無効です。	db2secValidatePassword db2secRemapUserid db2secGenerateInitialCred
-11	DB2SEC_PLUGIN _BAD_NEWPASSWORD	新規パスワードが無効です。	db2secValidatePassword db2secRemapUserid
-12	DB2SEC_PLUGIN _CHANGEPASSWORD _NOTSUPPORTED	パスワード変更はサポートされていません。	db2secValidatePassword db2secRemapUserid db2secGenerateInitialCred
-13	DB2SEC_PLUGIN_NOMEM	メモリー不足のため、プラグインがメモリーを割り振れませんでした。	すべて
-14	DB2SEC_PLUGIN_DISKERROR	プラグインがディスク・エラーを検出しました。	すべて
-15	DB2SEC_PLUGIN_NOPERM	ファイルの許可が不適切なため、プラグインがファイルにアクセスできませんでした。	すべて
-16	DB2SEC_PLUGIN_NETWORKERROR	プラグインがネットワーク・エラーを検出しました。	すべて
-17	DB2SEC_PLUGIN _CANTLOADLIBRARY	プラグインが必要なライブラリーをロードできません。	db2secGroupPluginInit db2secClientAuthPluginInit db2secServerAuthPluginInit
-18	DB2SEC_PLUGIN_CANT _OPEN_FILE	欠落ファイルや不適切なファイル許可以外の理由のために、プラグインがファイルをオープンして読み取ることができません。	すべて
-19	DB2SEC_PLUGIN_FILENOTFOUND	ファイル・システムにファイルがないために、プラグインがファイルをオープンして読み取ることができません。	すべて

表 10. セキュリティー・プラグインの戻りコード (続き)

戻りコード	定義値	意味	関連 API
-20	DB2SEC_PLUGIN_CONNECTION_DISALLOWED	接続できるデータベース、または特定のデータベースに接続できない TCP/IP アドレスについての制約事項のために、プラグインが接続を拒否しています。	すべてのサーバー・サイドのプラグイン API。
-21	DB2SEC_PLUGIN_NO_CRED	GSS API プラグインのみ: 初期クライアント証明書がありません。	db2secGetDefaultLoginContext db2secServerAuthPluginInit
-22	DB2SEC_PLUGIN_CRED_EXPIRED	GSS API プラグインのみ: クライアント証明書が期限切れです。	db2secGetDefaultLoginContext db2secServerAuthPluginInit
-23	DB2SEC_PLUGIN_BAD_PRINCIPAL_NAME	GSS API プラグインのみ: プリンシパル名が無効です。	db2secProcessServerPrincipalName
-24	DB2SEC_PLUGIN_NO_CON_DETAILS	この戻りコードは、db2secGetConDetails コールバック (例えば、DB2 からプラグインへの) によって戻され、 DB2 がクライアントの TCP/IP アドレスを判別できないことを示します。	db2secGetConDetails
-25	DB2SEC_PLUGIN_BAD_INPUT_PARAMETERS	プラグイン API を呼び出すとき、いくつかのパラメーターが無効か、または欠落しています。	すべて
-26	DB2SEC_PLUGIN_INCOMPATIBLE_VER	プラグインによって報告された API のバージョンに、DB2 との互換性がありません。	db2secGroupPluginInit db2secClientAuthPluginInit db2secServerAuthPluginInit
-27	DB2SEC_PLUGIN_PROCESS_LIMIT	プラグインが新規プロセスを作成するために、十分なリソースを使用できません。	すべて
-28	DB2SEC_PLUGIN_NO_LICENSES	プラグインがユーザー・ライセンスの問題を検出しました。基礎メカニズムのライセンスが限界に達している可能性があります。	すべて

セキュリティー・プラグインのエラー・メッセージ処理

セキュリティー・プラグイン API でエラーが発生すると、 API は errormsg フィールドに ASCII テキスト・ストリングを戻して、戻りコードよりも具体的な問題の説明を提示することがあります。

例えば、errormsg ストリングに、 "File /home/db2inst1/mypasswd.txt does not exist." などのメッセージが含められます。 DB2 はこのストリングをまるごと DB2 管理通知ログに書き込み、さらに、短縮版をいくつかの SQL メッセージにトークンとして組み込みます。 SQL メッセージ内のトークンは限られた長さにしかできないため、これらのメッセージは短くし、これらのメッセージの重要な変数の

部分がストリングの先頭に来るようにしてください。デバッグに役立てるため、エラー・メッセージにはセキュリティー・プラグインの名前を追加することを考慮してください。

パスワード期限切れエラーなどの緊急でないエラーに関しては、errormsg ストリングは、 DIAGLEVEL データベース・マネージャー構成パラメーターに 4 が設定されている場合にのみダンプされます。

これらのエラー・メッセージ用のメモリーは、セキュリティー・プラグインによって割り振られる必要があります。したがって、プラグインは、このメモリーを解放するための API である db2secFreeErrormsg を備えていなければなりません。

errormsg フィールドは、 API がゼロ以外の値を戻した場合にのみ DB2 によってチェックされます。そのため、プラグインは、エラーがない場合は、この戻りエラー・メッセージ用のメモリーを割り振るべきではありません。

初期化時には、メッセージ・ロギング関数ポインター logMessage_fn が、グループ、クライアント、およびサーバーのプラグインに渡されます。プラグインはこの関数を使用してデバッグ情報を db2diag.log に記録できます。以下に例を示します。

```
// Log an message indicate init successful
(*(logMessage_fn))(DB2SEC_LOG_CRITICAL,
                   "db2secGroupPluginInit successful",
                    strlen("db2secGroupPluginInit successful"));
```

db2secLogMessage 関数の各パラメーターについて詳しくは、各プラグイン・タイプの初期化 API を参照してください。

セキュリティー・プラグイン API の呼び出し順序

DB2 データベース・マネージャーがセキュリティー・プラグイン API を呼び出す主なシナリオを以下に示します。

- クライアントでのデータベース接続 (暗黙的および明示的)
  - CLIENT
  - サーバー・ベース (SERVER、SERVER_ENCRYPT、DATA_ENCRYPT)
  - GSSAPI および Kerberos
- クライアント、サーバー、またはゲートウェイでのローカル許可
- サーバーでのデータベース接続
- サーバーでの GRANT ステートメント
- サーバーで許可 ID が所属するグループのリストを取得する

注: DB2 データベース・サーバーは、ローカル許可が必要な db2start、db2stop、および db2trc などのデータベース・アクションを、クライアント・アプリケーションと同様に取り扱います。

DB2 データベース・マネージャーがセキュリティー・プラグイン API を呼び出す順序は、これらの各操作ごとに異なります。これらの各シナリオにおいて、DB2 データベース・マネージャーが呼び出す API の順序を以下に示します。

CLIENT - 暗黙的

ユーザー構成認証タイプが CLIENT の場合、DB2 クライアント・アプリケーションは以下のセキュリティー・プラグイン API を呼び出します。

- `db2secGetDefaultLoginContext();`
- `db2secValidatePassword();`
- `db2secFreetoken();`

暗黙的な認証の場合、すなわち、特定のユーザー ID やパスワードを指定せずに接続する場合は、ユーザー ID/パスワード・プラグインを使用していると、`db2secValidatePassword` API が呼び出されます。必要に応じ、プラグイン作成者はこの API によって暗黙的な認証を禁止することができます。

CLIENT - 明示的

明示的な認証の場合、すなわち、ユーザー ID とパスワードの両方が指定されているデータベースに接続する場合は、*authentication* データベース・マネージャー構成パラメーターが CLIENT に設定されていると、DB2 クライアント・アプリケーションは、インプリメンテーションが必要とする場合には、以下のセキュリティー・プラグイン API を複数回呼び出します。

- `db2secRemapUserid();`
- `db2secValidatePassword();`
- `db2secFreeToken();`

サーバー・ベース (SERVER、SERVER_ENCRYPT、DATA_ENCRYPT) - 暗黙的

暗黙的な認証の場合、クライアントとサーバーがユーザーID/パスワードの認証を折衝している場合 (例えば、サーバー側の *srvcon_auth* パラメーターが SERVER、SERVER_ENCRYPT、DATA_ENCRYPT、または DATA_ENCRYPT_CMP に設定されている場合)、クライアント・アプリケーションは以下のセキュリティー・プラグイン API を呼び出します。

- `db2secGetDefaultLoginContext();`
- `db2secFreeToken();`

サーバー・ベース (SERVER、SERVER_ENCRYPT、DATA_ENCRYPT) - 明示的

明示的な認証の場合、クライアントとサーバーがユーザーID/パスワードの認証を折衝している場合 (例えば、サーバー側の srvcon_auth パラメーターが SERVER、SERVER_ENCRYPT、DATA_ENCRYPT、または DATA_ENCRYPT_CMP に設定されている場合)、クライアント・アプリケーションは以下のセキュリティー・プラグイン API を呼び出します。

- `db2secRemapUserid();`

GSSAPI および Kerberos - 暗黙的

暗黙的な認証の場合、クライアントとサーバーが GSS-API または Kerberos 認証を折衝している場合 (例えば、サーバー側の *srvcon_auth* パラメーターが KERBEROS、KRB_SERVER_ENCRYPT、GSSPLUGIN、または GSS_SERVER_ENCRYPT に設定されている場合)、クライアント・アプリケーションは以下のセキュリティー・プラグイン API を呼び出します。(gss_init_sec_context() を呼び出すときは、GSS_C_NO_CREDENTIAL が入力証明書として使用されます。)

- `db2secGetDefaultLoginContext();`
- `db2secProcessServerPrincipalName();`

- gss_init_sec_context();
- gss_release_buffer();
- gss_release_name();
- gss_delete_sec_context();
- db2secFreeToken();

マルチフロー GSS-API サポートを使用すると、インプリメンテーションが必要とする場合には、 gss_init_sec_context() を複数回呼び出すことができます。

GSSAPI および Kerberos - 明示的

折衝された認証タイプが GSS-API または Kerberos の場合は、クライアント・アプリケーションが GSS-API プラグイン用に、以下のセキュリティー・プラグイン API をこの順序で呼び出します。特に記述されていない場合、これらの API は暗黙的な認証と明示的な認証の両方に使用されます。

- db2secProcessServerPrincipalName();
- db2secGenerateInitialCred(); (明示的な認証の場合のみ)
- gss_init_sec_context();
- gss_release_buffer ();
- gss_release_name();
- gss_release_cred();
- db2secFreeInitInfo();
- gss_delete_sec_context();
- db2secFreeToken();

サーバーから相互認証トークンが戻され、インプリメンテーションが必要とする場合には、API gss_init_sec_context() が複数回呼び出されることがあります。

クライアント、サーバー、またはゲートウェイでのローカル許可

ローカル許可の場合は、使用される DB2 コマンドが、以下のセキュリティー・プラグイン API を呼び出します。

- db2secGetDefaultLoginContext();
- db2secGetGroupsForUser();
- db2secFreeToken();
- db2secFreeGroupList();

これらの API が、ユーザー ID/パスワードと GSS-API の両方の認証メカニズム用に呼び出されます。

サーバーでのデータベース接続

データベース・サーバー上でのデータベース接続の場合は、DB2 エージェント・プロセスまたはスレッドが、ユーザー ID/パスワード認証メカニズム用に以下のセキュリティー・プラグイン API を呼び出します。

- db2secValidatePassword(); (*authentication* データベース構成パラメーターが CLIENT でない場合のみ)
- db2secGetAuthIDs();
- db2secGetGroupsForUser();

- db2secFreeToken();
- db2secFreeGroupList();

データベースへの接続の場合は、DB2 エージェント・プロセスまたはスレッドが、GSS-API 認証メカニズム用に以下のセキュリティー・プラグイン API を呼び出します。

- gss_accept_sec_context();
- gss_release_buffer();
- db2secGetAuthIDs();
- db2secGetGroupsForUser();
- gss_delete_sec_context();
- db2secFreeGroupListMemory();

サーバーでの GRANT ステートメント

USER または GROUP キーワードを指定しない GRANT ステートメント (例えば、 "GRANT CONNECT ON DATABASE TO user1") の場合、 DB2 エージェント・プロセスは user1 がユーザー、グループ、またはその両方のいずれであるかを判別できなければなりません。そのため、DB2 エージェント・プロセスまたはスレッドは以下のセキュリティー・プラグイン API を呼び出します。

- db2secDoesGroupExist();
- db2secDoesAuthIDExist();

サーバーで authid が所属するグループのリストを取得する

データベース・サーバーで、許可 ID が所属するグループのリストを取得する必要がある場合、DB2 エージェント・プロセスまたはスレッドは以下のセキュリティー・プラグイン API を、許可 ID のみを入力として呼び出します。

- db2secGetGroupsForUser();

他のセキュリティー・プラグインからのトークンはありません。

セキュリティー・プラグイン

DB2 データベース・システムでの認証は、セキュリティー・プラグイン を使用して行われます。セキュリティー・プラグインは、動的にロード可能なライブラリーであり、認証セキュリティー・サービスを提供します。

DB2 データベース・システムは、以下のタイプのプラグインを提供します。

- グループ検索プラグイン: 特定のユーザーのグループ・メンバーシップ情報を検索します。
- クライアント認証プラグイン: DB2 クライアント上で認証を管理します。
- サーバー認証プラグイン: DB2 サーバー上で認証を管理します。

DB2 は、次の 2 つのプラグイン認証メカニズムをサポートしています。

ユーザー ID/パスワード認証

これには、ユーザー ID とパスワードを使用する認証が関係します。以下の認証タイプは、ユーザー ID/パスワード認証プラグインを使用してインプリメントされます。

- CLIENT
- SERVER
- SERVER_ENCRYPT
- DATA_ENCRYPT
- DATA_ENCRYPT_CMP

上記のような認証タイプによって、ユーザー認証がどこでどのように行われるかが決まります。使用される認証タイプは、*authentication* データベース・マネージャー構成パラメーターで指定した認証タイプによって異なります。 SRVCON_AUTH パラメーターを指定した場合、接続またはアタッチの操作の処理時には、このパラメーターのほうが AUTHENTICATION よりも優先されます。

GSS-API 認証

GSS-API の正式名称は、*Generic Security Service Application Program Interface Version 2* (IETF RFC2743) および *Generic Security Service API Version 2: C-Bindings* (IETF RFC2744) です。 Kerberos 認証も、GSS-API を使用してインプリメントされます。以下の認証タイプは、GSS-API 認証プラグインを使用してインプリメントされます。

- KERBEROS
- GSSPLUGIN
- KRB_SERVER_ENCRYPT
- GSS_SERVER_ENCRYPT

KRB_SERVER_ENCRYPT および GSS_SERVER_ENCRYPT は、GSS-API 認証とユーザー ID/パスワード認証の両方をサポートしますが、GSS-API 認証のほうが望ましい認証タイプです。

注: 認証タイプによって、ユーザーがどこでどのように認証されるかが決まります。特定の認証タイプを使用するには、認証データベース・マネージャーの構成パラメーターを更新します。

各プラグインは独立して使用するか、1 つ以上の他のプラグインと併せて使用することができます。例えば、サーバー認証プラグインだけを使用して、クライアントおよびグループの認証に対しては DB2 のデフォルトをとることができます。または、グループまたはクライアントの認証プラグインのみを使用することもできます。クライアントとサーバーの両方のプラグインが必要となるのは、GSS-API 認証プラグインの場合のみです。

デフォルトの動作として、オペレーティング・システム・レベルの認証メカニズムをインプリメントするユーザー ID/パスワード・プラグインが使用されます。これより前のリリースでは、デフォルト動作として、プラグインのインプリメンテーションなしで、オペレーティング・システム・レベルの認証が直接使用されます。クライアント・サイドの Kerberos サポートは、Solaris、AIX、Windows、および

Linux オペレーティング・システムで使用できます。 Windows プラットフォームでは、デフォルトで Kerberos サポートが使用可能になっています。

DB2 データベース・システムには、グループ検索用、ユーザー ID/パスワード認証用、および Kerberos 認証用のプラグインのセットが組み込まれています。セキュリティー・プラグイン・アーキテクチャーを用いる場合は、独自のプラグインを作成するか、またはサード・パーティーからプラグインを購入することによって、DB2 のクライアントおよびサーバーの認証動作をカスタマイズすることができます。

DB2 クライアント上のセキュリティー・プラグインのデプロイメント

DB2 クライアントは 1 つのグループ・プラグイン、1 つのユーザー ID/パスワード認証プラグインをサポートでき、特定の GSS-API プラグインについて DB2 サーバーと折衝します。この折衝では、クライアントが DB2 サーバー側のインプリメントされている GSS-API プラグインのリストをスキャンして、クライアントにインプリメントされている認証プラグインと一致する認証プラグイン名を探します。サーバー側のプラグインのリストは、サーバー上にインプリメントされているすべてのプラグインの名前を収めている *srvcon_gssplugin_list* データベース・マネージャー構成パラメーター値に指定されます。以下の図は、DB2 クライアント上のセキュリティー・プラグイン・インフラストラクチャーを描写しています。

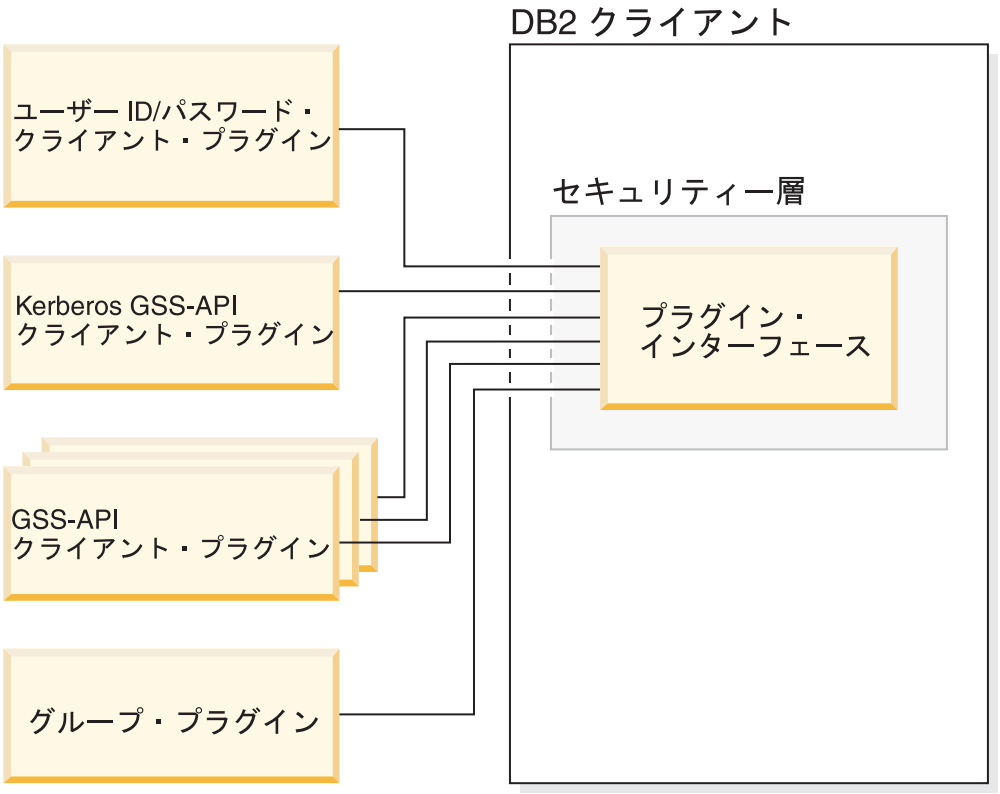


図 1. DB2 クライアント上のセキュリティー・プラグインのデプロイメント

DB2 サーバー上のセキュリティー・プラグインのデプロイメント

DB2 サーバーは 1 つのグループ・プラグイン、1 つのユーザー ID/パスワード認証プラグイン、および複数の GSS-API プラグインをサポートできます。複数の

GSS-API プラグインは、*srvcon_gssplugin_list* データベース・マネージャー構成パラメーター値内にリストとして指定されます。このリストの中の 1 つの GSS-API プラグインのみ、Kerberos プラグインにすることができます。

サーバー・サイドのセキュリティー・プラグインに加えて、データベース・サーバー上にもクライアント許可プラグインをデプロイする必要があるかもしれません。db2start および db2trc などのインスタンス・レベルの操作の実行時には、DB2 データベース・マネージャーはクライアント認証プラグインを使用して、その操作に対する許可検査を実行します。したがって、*authentication* データベース・マネージャー構成パラメーターで指定したサーバー・プラグインに対応するクライアント認証プラグインをインストールしておく必要があります。 *authentication* と *srvcon_auth* には重大な違いがあります。特に、これらを別々の異なる値に設定することで、データベース接続の認証用に一方のメカニズムを使用し、ローカル許可用にもう一方のメカニズムを使用することができます。最も一般的な使用法としては、*srvcon_auth* を GSSPLUGIN として設定し、*authentication* を SERVER として設定します。データベース・サーバー上でクライアント認証プラグインを使用しない場合、db2start などのインスタンス・レベルの操作は失敗します。例えば、認証タイプが SERVER で、ユーザー指定のクライアント・プラグインが使用されていない場合、DB2 データベース・システムは、IBM 提供のデフォルト・クライアント・オペレーティング・システム・プラグインを使用します。以下の図は、DB2 サーバー上のセキュリティー・プラグイン・インフラストラクチャーを描写しています。

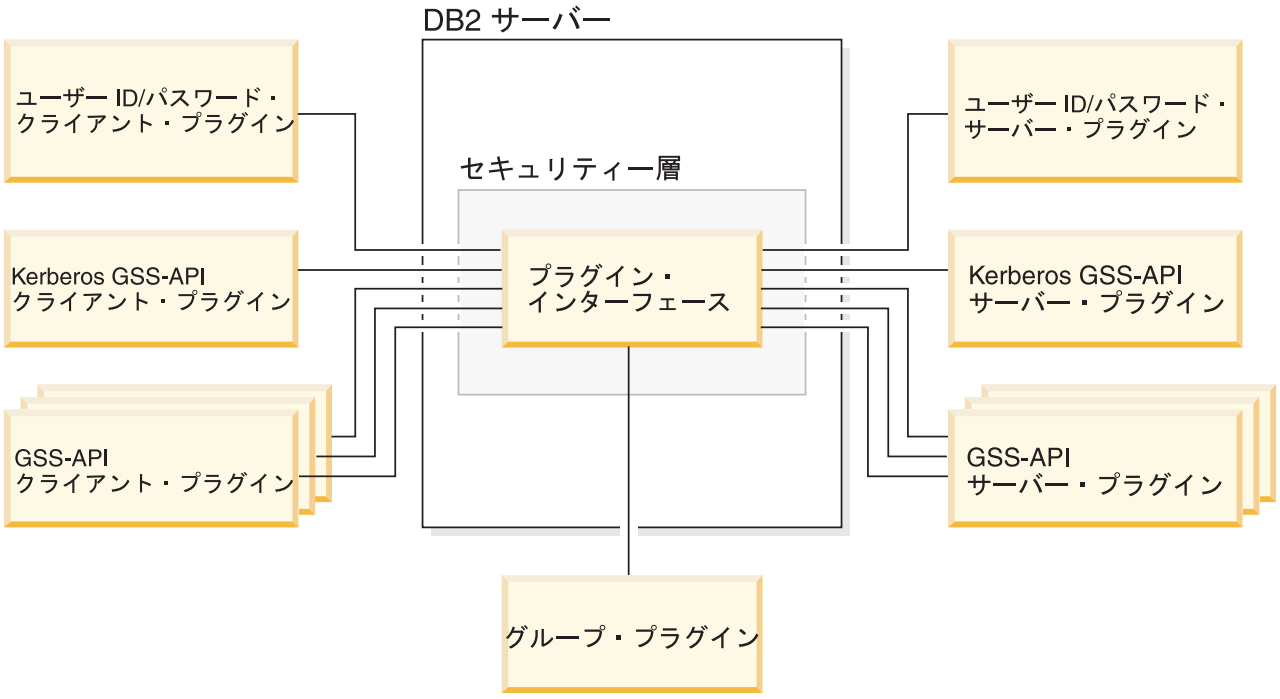


図 2. DB2 サーバー上のセキュリティー・プラグインのデプロイメント

注: セキュリティー・プラグインのデプロイメントのコーディング、確認、およびテストが十分に行われないと、インストールされている DB2 データベース・システムの保全性が損なわれることがあります。 DB2 データベース・システムでは、

多数の一般的なタイプの障害に対する予防措置がとられていますが、ユーザー作成のセキュリティー・プラグインのデプロイ時には、完全な保全性が確保されるとは限りません。

セキュリティー・プラグインの有効化

システム管理者は、プラグインに関係した特定のデータベース・マネージャー構成パラメーターを更新することにより、各認証メカニズムに使用するプラグインの名前を指定できます。これらのパラメーターがヌルの場合、DB2 提供のグループ検索、ユーザー ID/パスワード管理、または Kerberos (サーバー上で、認証に Kerberos が設定されている場合) 用のプラグインがデフォルトになります。 DB2 ではデフォルト GSS-API プラグインは用意されていません。したがって、システム管理者は、*authentication* パラメーターに認証タイプ GSSPLUGIN を指定する場合は、*srvcon_gssplugin_list* に GSS-API 認証プラグインも指定しなければなりません。

DB2 によるセキュリティー・プラグインのロード方法

データベース・マネージャーの始動時に、データベース・マネージャー構成パラメーターで識別されたサポートされるすべてのプラグインがロードされます。

接続またはアタッチの操作中にサーバーとの折衝済みのセキュリティー・メカニズムに適したプラグインが、DB2 クライアントによってロードされます。クライアント・アプリケーションが原因で、複数のセキュリティー・プラグインが並行してロードされて使用される可能性もあります。このような事態が発生するのは、例えば、さまざまなインスタンスからのそれぞれ異なるデータベースへの同時接続をもったスレッド化されたプログラムにおいてです。

接続またはアタッチの操作以外のアクションでは、許可も必要になります (データベース・マネージャー構成の更新、データベース・マネージャーの開始と停止、DB2 トレースのオン/オフなど)。そのようなアクションの場合、DB2 クライアント・プログラムは、別のデータベース・マネージャー構成パラメーターに指定されているプラグインをロードします。 *authentication* が GSSPLUGIN に設定されている場合、DB2 データベース・マネージャーは、*local_gssplugin* で指定されたプラグインを使用します。 *authentication* が KERBEROS に設定されている場合、DB2 データベース・マネージャーは、*clnt_krb_plugin* で指定されたプラグインを使用します。その他の場合は、DB2 データベース・マネージャーは *clnt_pw_plugin* で指定されたプラグインを使用します。

セキュリティー・プラグイン API は、IPv4 プラットフォームまたは IPv6 プラットフォームから呼び出すことができます。 IPv4 アドレスは、a.b.c.d という可読形式の 32 ビット・アドレスです。a から d はそれぞれ、0 から 255 までの 10 進数を表します。 IPv6 アドレスは、a:b:c:d:e:f:g:h の形式の 128 ビット・アドレスです。a から h はそれぞれ、4 桁の 16 進数字を表します。

セキュリティー・プラグインの作成

セキュリティー・プラグインを作成する場合は、DB2 データベース・マネージャーが使用する標準認証機能をインプリメントする必要があります。自分独自のカスタマイズしたセキュリティー・プラグインを使用する場合、CLP または動的 SQL ステートメントを介して発行する接続ステートメント上で、最大 255 文字までのユー

ザー ID を使用することができます。使用できるプラグイン・タイプに関してインプリメントする必要がある機能は、以下のとおりです。

グループ検索
ユーザーが所属するグループのリストを取得します。

ユーザー ID/パスワード認証
- デフォルトのセキュリティー・コンテキストを識別します (クライアントのみ)。
- パスワードの検証と、任意選択でパスワードの変更を行います。
- 指定されたストリングが有効なユーザーを表すかどうかを判別します (サーバーのみ)。
- クライアント上で提示されたユーザー ID またはパスワードを、サーバーへの送信前に修正します (クライアントのみ)。
- 特定のユーザーに関連した DB2 許可 ID を戻します。

GSS-API 認証
- 必要な GSS-API 関数をインプリメントします。
- デフォルトのセキュリティー・コンテキストを識別します (クライアントのみ)。
- ユーザー ID とパスワードに基づいて初期信用証明情報を生成し、任意選択でパスワードを変更します (クライアントのみ)。
- セキュリティー・チケットの作成と受け入れを行います。
- 特定の GSS-API セキュリティー・コンテキストに関連した DB2 許可 ID を戻します。

セキュリティー・プラグイン・ライブラリーの位置

セキュリティー・プラグインを (自分で作成するか、またはサード・パーティーから購入して) 取得したら、データベース・サーバー上の特定の場所にコピーします。

DB2 クライアントは、クライアント・サイドのユーザー認証プラグインを次のディレクトリーで探します。
- UNIX 32 ビット: `$DB2PATH/security32/plugin/client`
- UNIX 64 ビット: `$DB2PATH/security64/plugin/client`
- WINDOWS 32 ビットおよび 64 ビット: `$DB2PATH¥security¥plugin¥`*instance name*`¥client`

注: Windows ベースのプラットフォームの場合、サブディレクトリーの *instance name* および *client* は自動的には作成されません。これらは、インスタンスの所有者が手動で作成しなければなりません。

DB2 データベース・マネージャーは、サーバー・サイドのユーザー認証プラグインを次のディレクトリーで探します。
- UNIX 32 ビット: `$DB2PATH/security32/plugin/server`
- UNIX 64 ビット: `$DB2PATH/security64/plugin/server`

- WINDOWS 32 ビットおよび 64 ビット: $DB2PATH¥security¥plugin¥*instance name*¥server

注: Windows ベースのプラットフォームの場合、サブディレクトリーの *instance name* および *server* は自動的には作成されません。これらは、インスタンスの所有者が手動で作成しなければなりません。

DB2 データベース・マネージャーは、グループ・プラグインを次のディレクトリーで探します。

- UNIX 32 ビット: $DB2PATH/security32/plugin/group
- UNIX 64 ビット: $DB2PATH/security64/plugin/group
- WINDOWS 32 ビットおよび 64 ビット: $DB2PATH¥security¥plugin¥*instance name*¥group

注: Windows ベースのプラットフォームの場合、サブディレクトリーの *instance name* および *group* は自動的には作成されません。これらは、インスタンスの所有者が手動で作成しなければなりません。

セキュリティー・プラグインの命名規則

セキュリティー・プラグインのライブラリーには、プラットフォーム固有のファイル名拡張子が付いていなければなりません。 C または C++ で書かれたセキュリティー・プラグイン・ライブラリーには、プラットフォーム固有のファイル名拡張子が付いていなければなりません。

- Windows: .dll
- AIX: .a または .so。両方の拡張子が存在する場合、.a 拡張子が使用されます。
- Linux、 HP IPF、および Solaris: .so
- PA-RISC 上の HPUX: .sl または .so。両方の拡張子が存在する場合、.sl 拡張子が使用されます。

注: また、ユーザーは、DB2 Universal JDBC ドライバーをもったセキュリティー・プラグインを開発することもできます。
例えば、MyPlugin というセキュリティー・プラグイン・ライブラリーがあるとします。サポートされている各オペレーティング・システムに対する適切なライブラリー・ファイル名は、次のとおりです。

- Windows 32 ビット: MyPlugin.dll
- Windows 64 ビット: MyPlugin64.dll
- AIX 32 または 64 ビット: MyPlugin.a またはMyPlugin.so
- SUN 32 または 64-bit、Linux 32 または 64 ビット、IPF 上の HP 32 または 64 ビット: MyPlugin.so
- PA-RISC 上の HP-UX 32 または 64 ビット: MyPlugin.sl または MyPlugin.so

注: 接尾部 ″64″ は、64 ビットWindows のセキュリティー・プラグインのライブラリー名にのみ必要です。

セキュリティー・プラグインの名前を使ってデータベース・マネージャー構成を更新する際は、接尾部 ″64″ のないライブラリーの絶対パス名を使用し、ファイル拡

張子と名前の修飾パス部分を省略してください。オペレーティング・システムが何であっても、MyPlugin というセキュリティー・プラグイン・ライブラリーは、次のように登録されます。

```
UPDATE DBM CFG USING CLNT_PW_PLUGIN MyPlugin
```

セキュリティー・プラグイン名は、大文字小文字の区別があり、ライブラリー名と完全に一致している必要があります。 DB2 データベース・システムは、関連するデータベース・マネージャー構成パラメーターの値を使用してライブラリー・パスを組み立て、そのライブラリー・パスを使用してセキュリティー・プラグイン・ライブラリーをロードします。

セキュリティー・プラグイン名の競合を防ぐために、使用する認証方式、およびそのプラグインを作成した会社の識別シンボルを使って、プラグインに名前を付けてください。例えば、Foo, Inc. という会社が FOOsomemethod という認証方式をインプリメントするプラグインを作成した場合、そのプラグインには FOOsomemethod.dll といった名前を付けることができます。

プラグイン名の最大長 (ファイル拡張子および接尾部の 64 を含まない) は、 32 バイトまでに制限されています。データベース・サーバーによってサポートされるプラグインの最大数はありませんが、データベース・マネージャー構成内のコンマで区切られたプラグインのリストの最大長は 255 バイトです。次のように、この 2 つの制限を識別する 2 つの定義が、インクルード・ファイル sqlenv.h にあります。

```
#define SQL_PLUGIN_NAME_SZ     32     /* plug-in name */
#define SQL_SRVCON_GSSPLUGIN_LIST_SZ 255 /* GSS API plug-in list */
```

セキュリティー・プラグイン・ライブラリー・ファイルには、以下のファイル許可がなければなりません。

- インスタンス所有者によって所有される。
- システム上のすべてのユーザーが読み取れる。
- システム上のすべてのユーザーが実行できる。

セキュリティー・プラグインの 2 部構成ユーザー ID のサポート

Windows 上の DB2 データベース・マネージャーは、2 部構成ユーザー ID の使用と、2 部構成ユーザー ID から 2 部構成許可 ID へのマッピングをサポートしています。

例えば、ドメインとユーザー ID からなる Windows オペレーティング・システムの 2 部構成ユーザー ID (例えば MEDWAY¥pieter) について考えます。この例では、MEDWAY はドメインであり、pieter はユーザー名です。 DB2 データベース・システムでは、この 2 部構成ユーザー ID を、1 部構成許可 ID と 2 部構成許可 ID のどちらにマップするかを指定することができます。

2 部構成ユーザー ID から 2 部構成許可 ID へのマッピングもサポートされていますが、デフォルト動作ではありません。デフォルトでは、 1 部構成ユーザー ID と 2 部構成ユーザー ID はどちらも 1 部構成許可 ID にマップされます。 2 部構成ユーザー ID から 2 部構成許可 ID へのマッピングもサポートされていますが、デフォルト動作ではありません。

2 部構成ユーザー ID から 1 部構成ユーザー ID へのデフォルト・マッピングを使用すれば、ユーザーは次のようにしてデータベースに接続することができます。

```
db2 connect to db user MEDWAY¥pieter using pw
```

この状況下でデフォルト動作を使用すると、ユーザー ID MEDWAY¥pieter は許可 ID PIETER に解決されます。 2 部構成ユーザー ID から 2 部構成許可 ID へのマッピングのサポートが有効になっている場合、許可 ID は MEDWAY¥PIETER となります。

2 部構成ユーザー ID から 2 部構成許可 ID へのマップをDB2 で有効にするために、以下の 2 セットの認証プラグインが DB2 に用意されています。

- 一方のセットは、1 部構成ユーザー ID を 1 部構成許可 ID にマップし、2 部構成ユーザー ID を 1 部構成許可 ID にマップするだけです。
- もう一方のセットは、1 部構成ユーザー ID または 2 部構成ユーザー ID の両方を、2 部構成許可 ID にマップします。

作業環境におけるユーザー名を、さまざまな場所で定義された複数のアカウント (ローカル・アカウント、ドメイン・アカウント、およびトラステッド・ドメイン・アカウントなど) にマップできる場合は、2 部構成許可 ID のマッピングを有効にするプラグインを指定することができます。

PIETER などの 1 部構成許可 ID と、ドメインとユーザー ID を結合した MEDWAY¥PIETERのような 2 部構成許可 ID は、機能的に個別の許可 ID であることに注意してください。このような許可 ID の一方に関連付けられている特権セットは、他方の許可 ID に関連付けられている特権セットとは完全に異なります。 1 部構成許可 ID と 2 部構成許可 ID を扱う際には注意が必要です。

次の表は、DB2 データベース・システムに用意されているプラグインの種類と、特定の認証インプリメンテーション用のプラグイン名を示しています。

表 11. DB2 セキュリティー・プラグイン

認証タイプ	1 部構成のユーザー ID のプラグインの名前	2 部構成のユーザー ID のプラグインの名前
ユーザー ID/パスワード (クライアント)	IBMOSauthclient	IBMOSauthclientTwoPart
ユーザー ID/パスワード (サーバー)	IBMOSauthserver	IBMOSauthserverTwoPart
Kerberos	IBMkrb5	IBMkrb5TwoPart

注: Windows 64 ビット・プラットフォームでは、ここにリストされているプラグイン名に ″64″ という文字が付加されます。

ユーザー ID/パスワード・プラグインまたは Kerberos プラグインを必要とする認証タイプを指定している場合は、上記の表の「1 部構成ユーザー ID のプラグインの名前」列にリストされているプラグインがデフォルトで使用されます。

2 部構成ユーザー ID を 2 部構成許可 ID にマップするには、2 部構成プラグイン (これはデフォルトのプラグインではありません) の使用を指定する必要がありま

す。セキュリティー・プラグインは、セキュリティー関連のデータベース・マネージャー構成パラメーターを設定することによって、インスタンス・レベルで指定します。それは次のようにします。

2 部構成ユーザー ID を 2 部構成許可 ID にマップするサーバー認証の場合は、以下のように設定する必要があります。

- `srvcon_pw_plugin` を `IBMOSauthserverTwoPart` に設定
- `clnt_pw_plugin` を `IBMOSauthclientTwoPart` に設定

2 部構成ユーザー ID を 2 部構成許可 ID にマップするクライアント認証の場合は、以下のように設定する必要があります。

- `srvcon_pw_plugin` を `IBMOSauthserverTwoPart` に設定
- `clnt_pw_plugin` を `IBMOSauthclientTwoPart` に設定

2 部構成ユーザー ID を 2 部構成許可 ID にマップする Kerberos 認証の場合は、以下のように設定する必要があります。

- `srvcon_gssplugin_list` を `IBMOSkrb5TwoPart` に設定
- `clnt_krb_plugin` を `IBMkrb5TwoPart` に設定

セキュリティー・プラグイン・ライブラリーは、Microsoft® Windows Security Account Manager 互換形式で指定された 2 パーツのユーザー ID を受け入れます。これは、例えば *domain¥user ID* の形式です。接続時の DB2 の認証プロセスおよび許可プロセスでは、ドメインとユーザー ID の両方の情報が使用されます。

新規データベースを作成する際は、既存データベース内の 1 部構成許可 ID と競合しないよう、2 部構成プラグインをインプリメントすることを検討するようお勧めします。 2 部構成許可 ID を使用する新規のデータベースは、1 部構成許可 ID を使用するデータベースとは別のインスタンス内で作成する必要があります。

セキュリティー・プラグイン API のバージョン管理

DB2 データベース・システムは、セキュリティー・プラグイン API のバージョン番号をサポートします。そのようなバージョン番号は、DB2 UDB、バージョン 8.2 では、1 で始まる整数になります。

DB2 がセキュリティー・プラグイン API に渡すバージョン番号は、DB2 がサポートできる最高のバージョン番号であり、構造のバージョン番号に対応します。プラグインは、それより高い API バージョンをサポートできる場合、DB2 が要求したバージョン用の関数ポインターを戻さなければなりません。プラグインがそれより低いバージョンの API しかサポートしない場合、プラグインはその低いバージョン用の関数ポインターを入れる必要があります。いずれのケースでも、セキュリティー・プラグイン API は、サポートしている API のバージョン番号を、関数構造のバージョン・フィールドに入れて戻す必要があります。

DB2 の場合、セキュリティー・プラグインのバージョン番号は、必要な場合にのみ変化します (例えば、API のパラメーターが変更された場合など)。バージョン番号が DB2 のリリース番号とともに自動的に変わるわけではありません。

セキュリティー・プラグインの 32 ビットと 64 ビットに関する考慮事項

通常、32 ビット DB2 インスタンスは、32 ビット・セキュリティー・プラグインを使用し、64 ビット DB2 インスタンスは 64 ビット・セキュリティー・プラグインを使用します。しかし、64 ビット・インスタンス上では、DB2 は、32 ビット・プラグイン・ライブラリーを必要とする 32 ビット・アプリケーションをサポートします。

32 ビットと 64 ビットの両方のアプリケーションが実行できるデータベース・インスタンスは、ハイブリッド・インスタンスといいます。ハイブリッド・インスタンスがあり、32 ビット・アプリケーションを実行しようとしている場合は、必要な 32 ビット・セキュリティー・プラグインが、 32 ビット・プラグイン・ディレクトリー内に用意されていることを確認してください。 Linux および UNIX オペレーティング・システム (Linux on IPF を除く) 上の 64 ビット DB2 インスタンスでは、security32と security64 のディレクトリーが現れます。 X64 または IPF 上での Windows における 64 ビット DB2 インスタンスの場合、32 ビットと 64 ビットの両方のセキュリティー・プラグインが同一のディレクトリー内にありますが、64 ビット・プラグイン名には、接尾部の 64 が付いています。

32 ビット・インスタンスを 64 ビット・インスタンスに移行する予定の場合は、64 ビット用に再コンパイルされたセキュリティー・プラグイン用のバージョンを取得する必要があります。

64 ビット・プラグイン・ライブラリーを提供しないベンダーからセキュリティー・プラグインを購入した場合、32 ビット・アプリケーションを実行する 64 ビット・スタブをインプリメントできます。この場合、セキュリティー・プラグインは、ライブラリーではなく外部プログラムになります。

セキュリティー・プラグインの問題判別

セキュリティー・プラグインの問題は、SQL エラー経由と管理通知ログ経由の 2 とおりの方法で報告されます。

以下は、セキュリティー・プラグインに関係した SQLCODE 値です。

- SQLCODE -1365 は、db2start または db2stop の間にプラグイン・エラーが発生すると戻されます。
- SQLCODE -1366 は、ローカル許可の問題がある場合に戻されます。
- SQLCODE -30082 は、すべての接続に関係したプラグイン・エラーで戻されます。

管理通知ログは、セキュリティー・プラグインのデバッグおよび管理のための良い情報源です。 UNIX 上で管理通知ログを参照するには、sqllib/db2dump/*instance name*.nfy を調べます。 Windows オペレーティング・システム上で管理通知ログを参照するには、「イベント ビューア」ツールを使用します。「イベント ビューア」ツールは、Windows オペレーティング・システムの「スタート」ボタンから「設定」->「コントロール パネル」->「管理ツール」->「イベント ビューア」の順にナビゲートすると見つかります。以下は、セキュリティー・プラグインに関係した管理通知ログ値です。

- 13000 は、GSS-API セキュリティー・プラグイン API の呼び出しがエラーによって失敗し、オプションのエラー・メッセージが戻されたことを示します。

  ```
  SQLT_ADMIN_GSS_API_ERROR (13000)
  Plug-in "plug-in name" received error code "error code" from
  GSS API  "gss api name" with the error message "error message"
  ```

- 13001 は、DB2 セキュリティー・プラグイン API の呼び出しがエラーで失敗し、オプションのエラー・メッセージが戻されたことを示します。

  ```
  SQLT_ADMIN_PLUGIN_API_ERROR(13001)
  Plug-in "plug-in name" received error code "error code" from DB2
  security plug-in API "gss api name" with the error message
  "error message"
  ```

- 13002 は、DB2 がプラグインをアンロードできなかったことを示します。

  ```
  SQLT_ADMIN_PLUGIN_UNLOAD_ERROR (13002)
  Unable to unload plug-in "plug-in name". No further action required.
  ```

- 13003 は、無効なプリンシパル名を示します。

  ```
  SQLT_ADMIN_INVALID_PRIN_NAME (13003)
  The principal name "principal name" used for "plug-in name"
  is invalid. Fix the principal name.
  ```

- 13004 は、プラグイン名が無効なことを示します。パス区切り記号 (UNIX の場合は ″/″、Windows の場合は ″¥″) をプラグイン名の一部として使用することはできません。

  ```
  SQLT_ADMIN_INVALID_PLGN_NAME (13004)
  The plug-in name "plug-in name" is invalid. Fix the plug-in name.
  ```

- 13005 は、セキュリティー・プラグインがロードできなかったことを示します。プラグインを正しいディレクトリーに入れ、該当するデータベース・マネージャー構成パラメーターを更新してください。

  ```
  SQLT_ADMIN_PLUGIN_LOAD_ERROR (13005)
  Unable to load plug-in "plug-in name". Verify the plug-in existence and
  directory where it is located is correct.
  ```

- 13006 は、セキュリティー・プラグインによって予期しないエラーが検出されたことを示します。すべての db2support 情報を収集し、可能であれば db2trc を取り込んでから、IBM サポートに問い合わせてください。

  ```
  SQLT_ADMIN_PLUGIN_UNEXP_ERROR (13006)
  Plug-in encountered unexpected error. Contact IBM Support for further assistance.
  ```

注: Windows 64 ビットのデータベース・サーバー上でセキュリティー・プラグインを使用しているときに、セキュリティー・プラグインのロード・エラーが表示された場合は、32 ビットおよび 64 ビットの場合の考慮事項とセキュリティー・プラグインの命名規則に関するトピックを参照してください。 64 ビット・プラグイン・ライブラリーには、ライブラリー名に 64 という接尾部が付いていなければなりませんが、セキュリティー・プラグインのデータベース・マネージャー構成パラメーターへの入力にはこの接尾部は含めません。

セキュリティー・プラグインの API

ユーザーが DB2 データベース・システムの認証およびグループ・メンバーシップの検索の動作をカスタマイズできるように、既存のプラグイン・モジュールの変更や、新規セキュリティー・プラグイン・モジュールの作成の際に使用できる API が DB2 データベース・システムに用意されています。

セキュリティー・プラグイン・モジュールを作成するときは、DB2 データベース・マネージャーが呼び出す標準の認証またはグループ・メンバーシップの検索関数をインプリメントする必要があります。使用できる 3 つのタイプのプラグイン・モジュールに関してインプリメントする必要がある機能は、以下のとおりです。

グループ検索
特定のユーザーのグループ・メンバーシップ情報を検索し、指定されたストリングが有効なグループ名を表しているかどうかを判別します。

ユーザー ID/パスワード認証
この認証は、デフォルトのセキュリティー・コンテキストを識別し (クライアントのみ)、パスワードを検証して必要があれば変更し、指定されたストリングが有効なユーザーを表しているかどうか判別し (サーバーのみ)、クライアントで規定されているユーザー ID またはパスワードをサーバーへの送信の前に変更し (クライアントのみ)、指定されたユーザーに関連付けられた DB2 許可 ID を戻します。

GSS-API 認証
この認証は、必要な GSS-API 関数をインプリメントし、デフォルトのセキュリティー・コンテキストを識別し (クライアント・サイドのみ)、ユーザー ID およびパスワードを基に初期証明書を生成し、必要があればパスワードを変更し (クライアント・サイドのみ)、セキュリティー・チケットを作成して受け入れ、指定された GSS-API セキュリティー・コンテキストに関連付けられた DB2 許可 ID を戻します。

以下は、プラグイン API の説明に使用される用語の定義です。

プラグイン
DB2 が、ユーザー作成の認証またはグループ・メンバーシップの検索関数にアクセスするためにロードする動的にロード可能なライブラリー。

暗黙的な認証
ユーザー ID またはパスワードが指定されないデータベースへの接続。

明示的な認証
ユーザー ID とパスワードの両方が指定されるデータベースへの接続。

Authid データベース内での権限および特権が付与された個人またはグループを表す内部 ID。内部では、DB2 authid は大文字に変換されます。これは、8 文字以上です (8 文字になるようブランクが埋め込まれます)。現在のところ、DB2 は、7 ビット ASCII で表記できる authid、ユーザー ID、パスワード、グループ名、ネーム・スペース、およびドメイン名を必要とします。

ローカル許可
許可をインプリメントしているサーバーまたはクライアントでのローカルな許可です。これは、データベース・マネージャーの開始と停止、DB2 トレースのオン/オフ、データベース・マネージャー構成の更新などのアクション (データベース接続以外のアクション) を実行する権限がユーザーにあるかどうかを検査します。

ネーム・スペース
ユーザーの集合またはグループ。この中で個々のユーザー ID はユニークでなければなりません。一般的な例としては、Windows ドメインと Kerberos レルムがあります。例えば、Windows ドメイン ″usa.company.com″ では、

すべてのユーザー名がユニークでなければなりません。例えば、″user1@usa.company.com″ などとなります。他のドメインにある同一のユーザー ID (例えば、″user1@canada.company.com″) は、別のユーザーを表します。完全修飾ユーザー ID には、ユーザー ID とネーム・スペースのペア (例えば ″user@domain.name″ または ″domain¥user″) が含まれます。

入力 DB2 が、セキュリティー・プラグイン API パラメーターに値を入力することを示します。

出力 セキュリティー・プラグイン API が API パラメーターの値を入力することを示します。

グループ検索プラグイン用の API

グループ検索プラグイン・モジュール用には、以下の API をインプリメントする必要があります。

- db2secGroupPluginInit

  注: db2secGroupPluginInit API は、以下のプロトタイプを持つ API を指すポインター *logMessage_fn を入力としてとります。

  ```
  SQL_API_RC (SQL_API_FN db2secLogMessage)
  (
  db2int32 level,
  void    *data,
  db2int32 length
  );
  ```

  db2secLogMessage API により、プラグインはデバッグまたは通知の目的で、メッセージを db2diag.log に記録することができます。この API は DB2 データベース・システムによって提供されるため、インプリメントする必要はありません。

- db2secPluginTerm
- db2secGetGroupsForUser
- db2secDoesGroupExist
- db2secFreeGroupListMemory
- db2secFreeErrormsg
- 外部で解決できなければならない唯一の API は、db2secGroupPluginInit です。この API は、void * パラメーターをとり、それは以下のタイプにキャストする必要があります。

  ```
  typedef struct db2secGroupFunctions_1
  {
  db2int32 version;
  db2int32 plugintype;
  SQL_API_RC (SQL_API_FN * db2secGetGroupsForUser)
  (
  const char *authid,
  db2int32    authidlen,
  const char *userid,
  db2int32    useridlen,
  const char *usernamespace,
  db2int32    usernamespacelen,
  db2int32    usernamespacetype,
  const char *dbname,
  db2int32    dbnamelen,
  const void *token,
  db2int32    tokentype,
  ```

```
db2int32    location,
const char *authpluginname,
db2int32    authpluginnamelen,
void      **grouplist,
db2int32   *numgroups,
char      **errormsg,
db2int32   *errormsglen
);

SQL_API_RC (SQL_API_FN * db2secDoesGroupExist)
(
const char *groupname,
db2int32    groupnamelen,
char      **errormsg,
db2int32   *errormsglen
);

SQL_API_RC (SQL_API_FN * db2secFreeGroupListMemory)
(
void     *ptr,
char    **errormsg,
db2int32 *errormsglen
);

SQL_API_RC (SQL_API_FN * db2secFreeErrormsg)
(
char *msgtobefree
);

SQL_API_RC (SQL_API_FN * db2secPluginTerm)
(
char    **errormsg,
db2int32 *errormsglen
);

} db2secGroupFunctions_1;
```

db2secGroupPluginInit API は、外部で使用できる残りの関数のアドレスを割り当てます。

注: _1 はこれが API のバージョン 1 に対応する構造であることを示します。後続のインターフェース・バージョンの拡張子は _2、_3 というようになります。

db2secDoesGroupExist API - グループの存在のチェック

authid がグループを表すかどうかを判断します。

グループ名が存在する場合、API は、正常に完了したことを示すために値 DB2SEC_PLUGIN_OK を戻すことができなければなりません。グループ名が有効でない場合は、値 DB2SEC_PLUGIN_INVALIDUSERORGROUP も戻されなければなりません。入力が有効なグループかどうか判別できない場合は、API が値 DB2SEC_PLUGIN_GROUPSTATUSNOTKNOWN を戻すこともできます。無効なグループ (DB2SEC_PLUGIN_INVALIDUSERORGROUP) や不明なグループ (DB2SEC_PLUGIN_GROUPSTATUSNOTKNOWN) の値が戻される場合、 DB2 は USER キーワードおよび GROUP キーワードのない GRANT ステートメントを発行するときに、 authid がグループかユーザーかを判別できない可能性があり、その結果 SQLCODE -569、SQLSTATE 56092 のエラーがユーザーに戻されます。

API とデータ構造構文

```
SQL_API_RC ( SQL_API_FN *db2secDoesGroupExist)
                      ( const char *groupname,
                        db2int32 groupnamelen,
                        char      **errormsg,
                        db2int32 *errormsglen );
```

db2secDoesGroupExist API パラメーター

groupname
入力。末尾ブランクなしの大文字の authid。

groupnamelen
入力。groupname パラメーター値のバイト単位の長さ。

errormsg
出力。 db2secDoesGroupExist API が正常に実行されない場合にこのパラメーターで戻されることのある、プラグインによって割り振られた ASCII エラー・メッセージ・ストリングのアドレスを指すポインター。

errormsglen
出力。 errormsg パラメーターのエラー・メッセージ・ストリングのバイト単位の長さを示す整数を指すポインター。

db2secFreeErrormsg API - エラー・メッセージのメモリーの解放

直前の API 呼び出しのエラー・メッセージを保持するために使用されているメモリーを解放します。これは、エラー・メッセージを一緒に戻さない唯一の API です。この API がエラーを戻す場合、DB2 はそれをログに記録して続行します。

API とデータ構造構文

```
SQL_API_RC ( SQL_API_FN *db2secFreeErrormsg)
                      ( char *errormsg );
```

db2secFreeErrormsg API パラメーター

msgtofree
入力。以前の API 呼び出しで割り振られたエラー・メッセージを指すポインター。

db2secFreeGroupListMemory API - グループ・リストのメモリーの解放

直前の db2secGetGroupsForUser API の呼び出しのグループのリストを保持するのに使用されているメモリーを解放します。

API とデータ構造構文

```
SQL_API_RC ( SQL_API_FN *db2secFreeGroupListMemory)
                      ( void *ptr,
                        char      **errormsg,
                        db2int32 *errormsglen );
```

db2secFreeGroupListMemory API パラメーター

ptr 入力。解放されるメモリーを指すポインター。

errormsg
出力。プラグインによって割り振られた ASCII エラー・メッセージ・ストリングのアドレスを指すポインター。このエラー・メッセージ・ストリングは、db2secFreeGroupListMemory API が正常に実行されない場合にこのパラメーターに戻されることがあります。

errormsglen
出力。 errormsg パラメーターのエラー・メッセージ・ストリングのバイト単位の長さを示す整数を指すポインター。

db2secGetGroupsForUser API - ユーザーのグループのリストの取得

ユーザーが所属するグループのリストを戻します。

API とデータ構造構文

```
SQL_API_RC ( SQL_API_FN *db2secGetGroupsForUser)
                       ( const char *authid,
                         db2int32 authidlen,
                         const char *userid,
                         db2int32 useridlen,
                         const char *usernamespace,
                         db2int32 usernamespacelen,
                         db2int32 usernamespacetype,
                         const char *dbname,
                         db2int32 dbnamelen,
                         void *token,
                         db2int32 tokentype,
                         db2int32 location,
                         const char *authpluginname,
                         db2int32 authpluginnamelen,
                         void      **grouplist,
                         db2int32 *numgroups,
                         char      **errormsg,
                         db2int32 *errormsglen );
```

db2secGetGroupsForUser API パラメーター

authid 入力。このパラメーター値は SQL authid です。これは、その値が DB2 により大文字ストリングに変換され、末尾ブランクは付かないという意味です。 DB2 は常に、authid パラメーターに対して非ヌル値を提供します。 API は、他の入力パラメーターに関係なく、 authid が所属するグループのリストを戻せなければなりません。これが判別できない場合は、短縮されたリストまたは空のリストを戻しても差し支えありません。

ユーザーが存在しない場合、この API は戻りコード DB2SEC_PLUGIN_BADUSER を戻す必要があります。 authid には関連するグループがなくても差し支えないため、 DB2 は存在しないユーザーのケースをエラーとして扱いません。これには、db2secGetAuthids API がオペレーティング・システムに存在しない authid を戻す可能性があります。この authid にはグループが関連付けられていませんが、それでもこれには直接特権を割り当てることができます。

API がその authid を使用するだけでは完全なグループのリストを戻せない場合、グループ・サポートに関連した特定の SQL 関数になんらかの制限が生じる可能性があります。考えられる問題シナリオのリストについて詳しくは、このトピックの「使用上の注意」セクションを参照してください。

authidlen
入力。authid パラメーター値のバイト単位の長さ。 DB2 データベース・マネージャーは常に、authidlen パラメーターに対してゼロ以外の値を提供します。

userid 入力。これは authid に対応するユーザー ID です。非接続のシナリオで、サーバー上でこの API が呼び出されたときは、DB2 はこのパラメーターに値を入れません。

useridlen
入力。userid パラメーター値のバイト単位の長さ。

usernamespace
入力。取得されたユーザー ID が属するネーム・スペース。ユーザー ID が使用できない場合、DB2 データベース・マネージャーはこのパラメーターに値を入れません。

usernamespacelen
入力。usernamespace パラメーター値のバイト単位の長さ。

usernamespacetype
入力。ネーム・スペースのタイプ。 usernamespacetype パラメーターの有効な値 (db2secPlugin.h で定義されている) は以下のとおりです。

- DB2SEC_NAMESPACE_SAM_COMPATIBLE は domain¥myname などのユーザー名スタイルに対応します。
- DB2SEC_NAMESPACE_USER_PRINCIPAL は myname@domain.ibm.com などのユーザー名スタイルに対応します。

現在のところ、DB2 データベース・システムは値 DB2SEC_NAMESPACE_SAM_COMPATIBLE しかサポートしていません。ユーザー ID がない場合、usernamespacetype パラメーターの値は DB2SEC_USER_NAMESPACE_UNDEFINED (db2secPlugin.h で定義された) に設定されます。

dbname
入力。接続先のデータベースの名前。このパラメーターは、非接続シナリオでは NULL にすることができます。

dbnamelen
入力。dbname パラメーター値のバイト単位の長さ。非接続シナリオでは、dbname パラメーターが NULL の場合、このパラメーターは 0 に設定されます。

token 入力。認証プラグインによって提供されるデータを指すポインター。これは DB2 では使用されません。これを使用することにより、プラグイン作成者はユーザーおよびグループ情報を調整することができるようになります。このパラメーターは、必ずしもすべての事例で使用できない可能性があり (例えば、非接続シナリオで)、その場合のパラメーターの値は NULL になります。使用されている認証プラグインが GSS-API ベースの場合、このトークンには GSS-API コンテキスト・ハンドル (gss_ctx_id_t) が設定されます。

tokentype
入力。認証プラグインによって提供されるデータのタイプを示します。使用されている認証プラグインが GSS-API ベースの場合、このトークンには

GSS-API コンテキスト・ハンドル (gss_ctx_id_t) が設定されます。使用されている認証プラグインがユーザー ID/パスワード・ベースの場合、これは汎用タイプになります。 tokentype パラメーターの有効な値 (db2secPlugin.h で定義されている) は以下のとおりです。

- DB2SEC_GENERIC: トークンがユーザー ID/パスワード・ベースのプラグインからのものであることを示します。
- DB2SEC_GSSAPI_CTX_HANDLE: トークンが GSS-API (Kerberos を含む) ベースのプラグインからのものであることを示します。

location

入力。DB2 がクライアント・サイドとサーバー・サイドのどちらでこの API を呼び出すかを示します。 location パラメーターの有効な値 (db2secPlugin.h で定義されている) は以下のとおりです。

- DB2SEC_SERVER_SIDE: API はデータベース・サーバーで呼び出されます。
- DB2SEC_CLIENT_SIDE: API はクライアントで呼び出されます。

authpluginname

入力。トークンのデータを提供した認証プラグインの名前。 db2secGetGroupsForUser API は、正しいグループ・メンバーシップを判別するためにこの情報を使用することがあります。 authid が認証されない場合 (例えば、authid が現行接続ユーザーと一致しない場合) には、このパラメーターには DB2 によって値が入力されないことがあります。

authpluginnamelen

入力。authpluginname パラメーター値のバイト単位の長さ。

grouplist

出力。ユーザーが所属するグループのリスト。グループのリストは、連結された varchar (varchar とは、最初のバイトが後続のバイトの数を示す文字配列です) が含まれている、プラグインによって割り振られたメモリーのセクションを指すポインターとして戻されなければなりません。長さは unsigned char (1 バイト) であり、このためグループ名の最大長は 255 文字までに制限されます。例えば、「¥006GROUP1¥007MYGROUP¥008MYGROUP3」などです。各グループ名は、有効な DB2 authid でなければなりません。この配列のメモリーは、プラグインによって割り振られる必要があります。したがって、プラグインは、 DB2 がメモリーを解放するために呼び出す db2secFreeGroupListMemory API などの API を備えている必要があります。

numgroups

出力。 grouplist パラメーターに含まれるグループの数。

errormsg

出力。 db2secGetGroupsForUser API が正常に実行されない場合にこのパラメーターで戻されることのある、プラグインによって割り振られた ASCII エラー・メッセージ・ストリングのアドレスを指すポインター。

errormsglen

出力。 errormsg パラメーターのエラー・メッセージ・ストリングのバイト単位の長さを示す整数を指すポインター。

使用上の注意

以下は、この API によって DB2 に不完全なグループのリストが戻された場合に、問題が生じる可能性のあるシナリオのリストです。

- DYNAMICRULES BIND (あるいは、パッケージがスタンドアロン・アプリケーションとして実行している場合は、 DEFINEDBIND または INVOKEDBIND) が指定された組み込み SQL アプリケーション。 DB2 は SYSADM メンバーシップをチェックします。そして、アプリケーションが、 SYSADM のメンバーであることによって付与される暗黙的な DBADM 権限に依存している場合、このアプリケーションは失敗します。
- CREATE SCHEMA ステートメントで代替許可が提供される。 CREATE SCHEMA ステートメント内にネストされた CREATE ステートメントがある場合、 AUTHORIZATION NAME パラメーターに対してグループ検索が実行されます。
- DYNAMICRULES DEFINERUN/DEFINEBIND が指定された組み込み SQL アプリケーションがあり、そのパッケージがルーチン・コンテキストで実行している。 DB2 はルーチン定義者の SYSADM メンバーシップをチェックします。そして、アプリケーションが、 SYSADM のメンバーであることによって付与される暗黙的な DBADM 権限に依存している場合、このアプリケーションは失敗します。
- MPP 環境での jar ファイルの処理。 MPP 環境では、jar 処理要求が、セッション authid とともにコーディネーター・ノードから送信されます。カタログ・ノードは要求を受信すると、セッション authid (jar 処理要求を実行するユーザー) の特権に基づいて jar ファイルを処理します。
  - jar ファイルのインストール。セッション authid は、SYSADM、DBADM、または CREATEIN のいずれかの (jar スキーマに対する暗黙的または明示的な) 権限を有している必要があります。セッション authid の含まれるグループに対しては上記の権限が付与されているが、セッション authid に明示的には付与されていない場合や、データベース構成パラメーターによって定義されたグループのメンバーシップによって SYSADM メンバーシップが判別されたために、 SYSADM のみが保持されている場合は、操作は失敗します。
  - jar ファイルの除去。セッション authid は、SYSADM、DBADM、または DROPIN のいずれかの (jar スキーマに対する暗黙的または明示的な) 権限を有しているか、 jar ファイルの定義者である必要があります。セッション authid の含まれるグループに対しては上記の権限が付与されているが、セッション authid に明示的には付与されておらず、セッション authid が Jar ファイルの定義者でもない場合や、データベース構成パラメーターによって定義されたグループのメンバーシップによって SYSADM メンバーシップが判別されたために、 SYSADM のみが保持されている場合は、操作は失敗します。
  - jar ファイルの置き換え。これは、jar ファイルを除去した後に、jar ファイルをインストールするのと同じことです。上記の両方が当てはまります。
- ビューの再生成。これは ALTER TABLE、ALTER COLUMN、SET DATA TYPE VARCHAR/VARGRAPHIC ステートメントによって、またはマイグレーションの際に起動されます。 DB2 データベース・マネージャーはビュー定義者の SYSADM メンバーシップをチェックします。アプリケーションが、 SYSADM グループのメンバーであることによって付与される暗黙的な DBADM 権限に依存している場合、このアプリケーションは失敗します。

- SET SESSION_USER ステートメントが発行される場合。その後の DB2 操作は、このステートメントで指定された authid のコンテキストの下で実行されます。必要な特権が SESSION_USER のグループのいずれかによって所有されているものの、 SESSION_USER authid に明示的に付与されてはいない場合、それらの操作は失敗します。

db2secGroupPluginInit API - グループ・プラグインの初期化

プラグインのロードの直後に DB2 データベース・マネージャーが呼び出す、グループ検索プラグイン用の初期化 API。

API とデータ構造構文

```
SQL_API_RC SQL_API_FN db2secGroupPluginInit
                              ( db2int32 version,
                                void *group_fns,
                                db2secLogMessage *logMessage_fn,
                                char     **errormsg,
                                db2int32 *errormsglen );
```

db2secGroupPluginInit API パラメーター

version

入力。そのプラグインをロードするインスタンスによってサポートされる API の最上位バージョン。値 DB2SEC_API_VERSION (db2secPlugin.h 内) には、DB2 データベース・マネージャーが現在サポートしている API の最新のバージョン番号が含まれます。

group_fns

出力。 db2secGroupFunctions_<version_number> (group_functions_<version_number> としても知られる) 構造を指すポインター。db2secGroupFunctions_<version_number> 構造には、グループ検索プラグイン用にインプリメントされた API を指すポインターが含まれます。将来、これらの API には異なるバージョンが存在する可能性があるので (例えば、db2secGroupFunctions_<version_number>)、group_fns パラメーターは、プラグインがインプリメントしているバージョンに対応する db2secGroupFunctions_<version_number> 構造を指すポインターとしてキャストされます。 group_functions_<version_number> 構造の最初のパラメーターは、プラグインがインプリメントしている API のバージョンを DB2 に知らせます。注: DB2 のバージョンが、プラグインがインプリメントしている API のバージョンと同じかそれより大きい場合に限り、キャストが行われます。バージョン番号は、プラグインがインプリメントしている API のバージョンを表しており、pluginType は DB2SEC_PLUGIN_TYPE_GROUP に設定されていなければなりません。

logMessage_fn

入力。DB2 データベース・システムによってインプリメントされる db2secLogMessage API を指すポインター。 db2secGroupPluginInit API は、db2secLogMessage API を呼び出して、デバッグまたは通知の目的でメッセージを db2diag.log に記録することができます。 db2secLogMessage API の最初のパラメーター (level) は、db2diag.log ファイルに記録される診断エラーのタイプを指定し、最後の 2 つのパラメーターはそれぞれメッセージ・

ストリングとその長さです。 (db2secPlugin.h で定義された) dbesecLogMessage API の最初のパラメーターの有効な値は以下のとおりです。

- DB2SEC_LOG_NONE: (0) ロギングなし
- DB2SEC_LOG_CRITICAL: (1) 重大エラーを検出した
- DB2SEC_LOG_ERROR: (2) エラーを検出した
- DB2SEC_LOG_WARNING: (3) 警告
- DB2SEC_LOG_INFO: (4) 通知

メッセージ・テキストが diag.log に表示されるのは、db2secLogMessage API の 'level' パラメーターの値が 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。そのため、例えば DB2SEC_LOG_INFO 値を使用する場合、メッセージ・テキストは diaglevel データベース・マネージャー構成パラメーターに 4 が設定されている場合にのみ db2diag.log に表示されます。

errormsg
出力。プラグインによって割り振られた ASCII エラー・メッセージ・ストリングのアドレスを指すポインター。db2secGroupPluginInit API が正常に実行されない場合にこのパラメーターに戻されることがあります。

errormsglen
出力。 errormsg パラメーターのエラー・メッセージ・ストリングのバイト単位の長さを示す整数を指すポインター。

db2secPluginTerm - グループ・プラグイン・リソースのクリーンアップ

グループ検索プラグインによって使用されるリソースを解放します。

この API は、DB2 データベース・マネージャーがグループ検索プラグインをアンロードする直前に呼び出されます。これは、プラグイン・ライブラリーが保持しているリソースの適切なクリーンアップを実行する、という方法でインプリメントされる必要があります。例えば、プラグインによって割り振られたメモリーを解放し、まだオープンしているファイルをクローズし、ネットワーク接続をクローズします。これらのリソースを解放するためにその記録を保持することは、プラグインが行います。この API は Windows プラットフォームでは呼び出されません。

API とデータ構造構文

```
SQL_API_RC ( SQL_API_FN *db2secPluginTerm)
                      ( char      **errormsg,
                        db2int32 *errormsglen );
```

db2secPluginTerm API parameters

errormsg
出力。プラグインによって割り振られた ASCII エラー・メッセージ・ストリングのアドレスを指すポインター。db2secPluginTerm API が正常に実行されない場合にこのパラメーターに戻されることがあります。

errormsglen

出力。 errormsg パラメーターのエラー・メッセージ・ストリングのバイト単位の長さを示す整数を指すポインター。

ユーザー ID/パスワード認証プラグインの API

ユーザー ID/パスワード・プラグイン・モジュール用には、以下のクライアント・サイド API をインプリメントする必要があります。

- db2secClientAuthPluginInit

  注: db2secClientAuthPluginInit API は、以下のプロトタイプを持つ API を指すポインター *logMessage_fn を入力としてとります。

  ```
  SQL_API_RC (SQL_API_FN db2secLogMessage)
  (
  db2int32 level,
  void    *data,
  db2int32 length
  );
  ```

  db2secLogMessage API により、プラグインはデバッグまたは通知の目的で、メッセージを db2diag.log に記録することができます。この API は DB2 データベース・システムによって提供されるため、インプリメントする必要はありません。
- db2secClientAuthPluginTerm
- db2secGenerateInitialCred (gssapi 専用)
- db2secRemapUserid (オプション)
- db2secGetDefaultLoginContext
- db2secValidatePassword
- db2secProcessServerPrincipalName (これは GSS-API 専用)
- db2secFreeToken (DLL で保持されているメモリーを解放するための関数)
- db2secFreeErrormsg
- db2secFreeInitInfo
- 外部で解決できなければならない唯一の API は、db2secClientAuthPluginInit です。この API は void * パラメーターをとり、それは以下のいずれかにキャストする必要があります。

  ```
  typedef struct db2secUseridPasswordClientAuthFunctions_1
  {
  db2int32 version;
  db2int32 plugintype;

  SQL_API_RC (SQL_API_FN * db2secGetDefaultLoginContext)
  (
  char        authid[DB2SEC_MAX_AUTHID_LENGTH],
  db2int32   *authidlen,
  char        userid[DB2SEC_MAX_USERID_LENGTH],
  db2int32   *useridlen,
  db2int32    useridtype,
  char        usernamespace[DB2SEC_MAX_USERNAMESPACE_LENGTH],
  db2int32   *usernamespacelen,
  db2int32   *usernamespacetype,
  const char *dbname,
  db2int32    dbnamelen,
  void      **token,
  char      **errormsg,
  ```

```
db2int32   *errormsglen
);
/* Optional */
SQL_API_RC (SQL_API_FN * db2secRemapUserid)
(
char        userid[DB2SEC_MAX_USERID_LENGTH],
db2int32   *useridlen,
char        usernamespace[DB2SEC_MAX_USERNAMESPACE_LENGTH],
db2int32   *usernamespacelen,
db2int32   *usernamespacetype,
char        password[DB2SEC_MAX_PASSWORD_LENGTH],
db2int32   *passwordlen,
char        newpassword[DB2SEC_MAX_PASSWORD_LENGTH],
db2int32   *newpasswordlen,
const char *dbname,
db2int32    dbnamelen,
char      **errormsg,
db2int32   *errormsglen
);

SQL_API_RC (SQL_API_FN * db2secValidatePassword)
(
const char *userid,
db2int32    useridlen,
const char *usernamespace,
db2int32    usernamespacelen,
db2int32    usernamespacetype,
const char *password,
db2int32    passwordlen,
const char *newpassword,
db2int32    newpasswordlen,
const char *dbname,
db2int32    dbnamelen,
db2Uint32   connection_details,
void      **token,
char      **errormsg,
db2int32   *errormsglen
);

SQL_API_RC (SQL_API_FN * db2secFreeToken)
(
void     **token,
char     **errormsg,
db2int32  *errormsglen
);

SQL_API_RC (SQL_API_FN * db2secFreeErrormsg)
(
char *errormsg
);

SQL_API_RC (SQL_API_FN * db2secClientAuthPluginTerm)
(
char     **errormsg,
db2int32  *errormsglen
);
}
```

または

```
typedef struct db2secGssapiClientAuthFunctions_1
{
db2int32 version;
db2int32 plugintype;

SQL_API_RC (SQL_API_FN * db2secGetDefaultLoginContext)
(
```

```
char        authid[DB2SEC_MAX_AUTHID_LENGTH],
db2int32   *authidlen,
char        userid[DB2SEC_MAX_USERID_LENGTH],
db2int32   *useridlen,
db2int32    useridtype,
char        usernamespace[DB2SEC_MAX_USERNAMESPACE_LENGTH],
db2int32   *usernamespacelen,
db2int32   *usernamespacetype,
const char *dbname,
db2int32    dbnamelen,
void      **token,
char      **errormsg,
db2int32   *errormsglen
);

SQL_API_RC (SQL_API_FN * db2secProcessServerPrincipalName)
(
const void *data,
gss_name_t *gssName,
char      **errormsg,
db2int32   *errormsglen
);

SQL_API_RC (SQL_API_FN * db2secGenerateInitialCred)
(
const char    *userid,
db2int32       useridlen,
const char    *usernamespace,
db2int32       usernamespacelen,
db2int32       usernamespacetype,
const char    *password,
db2int32       passwordlen,
const char    *newpassword,
db2int32       newpasswordlen,
const char    *dbname,
db2int32       dbnamelen,
gss_cred_id_t *pGSSCredHandle,
void         **initInfo,
char         **errormsg,
db2int32      *errormsglen
);

SQL_API_RC (SQL_API_FN * db2secFreeToken)
(
void      *token,
char     **errormsg,
db2int32  *errormsglen
);

SQL_API_RC (SQL_API_FN * db2secFreeErrormsg)
(
char *errormsg
);


SQL_API_RC (SQL_API_FN * db2secFreeInitInfo)
(
void      *initInfo,
char     **errormsg,
db2int32  *errormsglen
);

SQL_API_RC (SQL_API_FN * db2secClientAuthPluginTerm)
(
char     **errormsg,
db2int32  *errormsglen
);
```

```
/* GSS-API specific functions -- refer to db2secPlugin.h
   for parameter list*/

  OM_uint32 (SQL_API_FN * gss_init_sec_context )(<parameter list>);
  OM_uint32 (SQL_API_FN * gss_delete_sec_context )(<parameter list>);
  OM_uint32 (SQL_API_FN * gss_display_status )(<parameter list>);
  OM_uint32 (SQL_API_FN * gss_release_buffer )(<parameter list>);
  OM_uint32 (SQL_API_FN * gss_release_cred )(<parameter list>);
  OM_uint32 (SQL_API_FN * gss_release_name )(<parameter list>);
  }
```

ユーザー ID/パスワード・プラグインを作成する場合は、db2secUseridPasswordClientAuthFunctions_1 構造を使用する必要があります。GSS-API (Kerberos を含む) プラグインを作成する場合は、db2secGssapiClientAuthFunctions_1 構造を使用する必要があります。

ユーザー ID/パスワード・プラグイン・ライブラリー用には、以下のサーバー・サイド API をインプリメントする必要があります。

- db2secServerAuthPluginInit

  db2secServerAuthPluginInit API は、以下のプロトタイプを持つ、db2secLogMessage API を指すポインター *logMessage_fn、および db2secGetConDetails API を指すポインター *getConDetails_fn を入力としてとります。

```
SQL_API_RC (SQL_API_FN db2secLogMessage)
(
db2int32 level,
void    *data,
db2int32 length
);

SQL_API_RC (SQL_API_FN db2secGetConDetails)
(
db2int32    conDetailsVersion,
const void *pConDetails
);
```

  db2secLogMessage API により、プラグインはデバッグまたは通知の目的で、メッセージを db2diag.log に記録することができます。 db2secGetConDetails API により、プラグインは、データベース接続を持とうとしているクライアントに関する詳細を取得することができます。 db2secLogMessage API と db2secGetConDetails API はどちらも DB2 データベース・システムによって提供されるので、インプリメントする必要はありません。同様に、db2secGetConDetails API は、その 2 番目のパラメーター pConDetails として、以下の構造の 1 つを指すポインターをとります。

  db2sec_con_details_1:

```
typedef struct db2sec_con_details_1
{
  db2int32  clientProtocol;
  db2Uint32 clientIPAddress;
  db2Uint32 connect_info_bitmap;
  db2int32  dbnameLen;
  char      dbname[DB2SEC_MAX_DBNAME_LENGTH + 1];
} db2sec_con_details_1;
```

db2sec_con_details_2:

```
typedef struct db2sec_con_details_2
{
  db2int32  clientProtocol;     /* See SQL_PROTOCOL_ in sqlenv.h */
  db2Uint32 clientIPAddress;    /* Set if protocol is TCPIP4     */
  db2Uint32 connect_info_bitmap;
  db2int32  dbnameLen;
  char dbname[DB2SEC_MAX_DBNAME_LENGTH + 1];
  db2Uint32 clientIP6Address[4];/* Set if protocol is TCPIP6     */
} db2sec_con_details_2;
```

db2sec_con_details_3:

```
typedef struct db2sec_con_details_3
{
  db2int32 clientProtocol; /* See SQL_PROTOCOL_ in sqlenv.h */
  db2Uint32 clientIPAddress; /* Set if protocol is TCPIP4 */
  db2Uint32 connect_info_bitmap;
  db2int32 dbnameLen;
  char dbname[DB2SEC_MAX_DBNAME_LENGTH + 1];
  db2Uint32 clientIP6Address[4];/* Set if protocol is TCPIP6 */
  db2Uint32 clientPlatform; /* SQLM_PLATFORM_* from sqlmon.h */
  db2Uint32 _reserved[16];
} db2sec_con_details_3;
```

conDetailsVersion の考えられる値は、API のバージョンを表す DB2SEC_CON_DETAILS_VERSION_1、DB2SEC_CON_DETAILS_VERSION_2 および DB2SEC_CON_DETAILS_VERSION_3 です。

注: db2sec_con_details_1、db2sec_con_details_2 または db2sec_con_details_3 を使用しているときには、以下の事柄を考慮してください。

– db2sec_con_details_1 構造と DB2SEC_CON_DETAILS_VERSION_1 値を使用している既存のプラグインは、db2GetConDetails API を呼び出すと、バージョン 8.2 で行っていたように作業を続けます。この API が IPv4 プラットフォームで呼び出される場合、クライアント IP アドレスが構造の clientIPAddress フィールドで戻されます。この API が IPv6 プラットフォームで呼び出される場合、値 0 が clientIPAddress フィールドで戻されます。クライアント IP アドレスを IPv6 プラットフォームで取り出すには、db2sec_con_details_2 構造と DB2SEC_CON_DETAILS_VERSION_2 値を使用するように、または db2sec_con_details_3 構造と DB2SEC_CON_DETAILS_VERSION_3 値を使用するようにセキュリティー・プラグイン・コードを変更する必要があります。

– 新規プラグインは db2sec_con_details_3 構造と DB2SEC_CON_DETAILS_VERSION_3 値を使用する必要があります。db2secGetConDetails API が IPv4 プラットフォームで呼び出される場合、クライアント IP アドレスが db2sec_con_details_3 構造の clientIPAddress フィールドで戻され、この API が IPv6 プラットフォームで呼び出される場合、クライアント IP アドレスが db2sec_con_details_3 構造の clientIP6Address フィールドで戻されます。接続詳細構造の *clientProtocol* フィールドは、SQL_PROTOCOL_TCPIP (IPv4 で v1 の構造を持つ)、SQL_PROTOCOL_TCPIP4 (IPv4 で v2 または v3 の構造を持つ)、または SQL_PROTOCOL_TCPIP6 (IPv6で v2 の構造を持つ) のいずれかに設定されます。

– 構造 db2sec_con_details_3 は、SQLM_PLATFORM_AIX のような sqlmon.h で定義されるプラットフォーム・タイプ定数を使用するクライアント・プラット

フォーム・タイプ (通信層でレポートされる) を特定する追加フィールド (*clientPlatform*) を含んでいる点を除けば、構造 db2sec_con_details_2 と同じです。

- db2secServerAuthPluginTerm
- db2secValidatePassword
- db2secGetAuthIDs
- db2secDoesAuthIDExist
- db2secFreeToken
- db2secFreeErrormsg
- 外部で解決できなければならない唯一の API は、db2secServerAuthPluginInit です。この API は void * パラメーターをとり、それは以下のいずれかにキャストする必要があります。

```
typedef struct db2secUseridPasswordServerAuthFunctions_1
{
db2int32 version;
db2int32 plugintype;

  /* parameter lists left blank for readability
    see above for parameters */
SQL_API_RC (SQL_API_FN * db2secValidatePassword)(<parameter list>);
SQL_API_RC (SQL_API_FN * db2secGetAuthIDs)(<parameter list);
SQL_API_RC (SQL_API_FN * db2secDoesAuthIDExist)(<parameter list>);
SQL_API_RC (SQL_API_FN * db2secFreeToken)(<parameter list>);
SQL_API_RC (SQL_API_FN * db2secFreeErrormsg)(<parameter list>);
SQL_API_RC (SQL_API_FN * db2secServerAuthPluginTerm)();
} userid_password_server_auth_functions;
```

または

```
typedef struct db2secGssapiServerAuthFunctions_1
{
db2int32 version;
db2int32 plugintype;
gss_buffer_desc serverPrincipalName;
gss_cred_id_t ServerCredHandle;
SQL_API_RC (SQL_API_FN * db2secGetAuthIDs)(<parameter list);
SQL_API_RC (SQL_API_FN * db2secDoesAuthIDExist)(<parameter list>);
SQL_API_RC (SQL_API_FN * db2secFreeErrormsg)(<parameter list>);
SQL_API_RC (SQL_API_FN * db2secServerAuthPluginTerm)();

/* GSS-API specific functions
refer to db2secPlugin.h for parameter list*/
OM_uint32 (SQL_API_FN * gss_accept_sec_context )(<parameter list>);
OM_uint32 (SQL_API_FN * gss_display_name )(<parameter list>);
OM_uint32 (SQL_API_FN * gss_delete_sec_context )(<parameter list>);
OM_uint32 (SQL_API_FN * gss_display_status )(<parameter list>);
OM_uint32 (SQL_API_FN * gss_release_buffer )(<parameter list>);
OM_uint32 (SQL_API_FN * gss_release_cred )(<parameter list>);
OM_uint32 (SQL_API_FN * gss_release_name )(<parameter list>);

} gssapi_server_auth_functions;
```

ユーザー ID/パスワード・プラグインを作成する場合は、db2secUseridPasswordServerAuthFunctions_1 構造を使用する必要があります。GSS-API (Kerberos を含む) プラグインを作成する場合は、db2secGssapiServerAuthFunctions_1 構造を使用する必要があります。

db2secClientAuthPluginInit API - クライアント認証プラグインの初期化

プラグインのロードの直後に DB2 データベース・マネージャーが呼び出す、クライアント認証プラグイン用の初期化 API。

API とデータ構造構文

```
SQL_API_RC SQL_API_FN db2secClientAuthPluginInit
                              ( db2int32 version,
                                void *client_fns,
                                db2secLogMessage *logMessage_fn,
                                char    **errormsg,
                                db2int32 *errormsglen );
```

db2secClientAuthPluginInit API パラメーター

version

入力。DB2 データベース・マネージャーが現在サポートしている API の最大のバージョン番号。 DB2SEC_API_VERSION 値 (db2secPlugin.h 内) には、DB2 が現在サポートしている API の最新のバージョン番号が含まれます。

client_fns

出力。 GSS-API 認証が使用される場合は、db2secGssapiClientAuthFunctions_<version_number> 構造 (gssapi_client_auth_functions_<version_number> としても知られる) のために DB2 データベース・マネージャーによって提供されたメモリーを指すポインター。ユーザー ID/パスワード認証が使用される場合は、db2secUseridPasswordClientAuthFunctions_<version_number> 構造 (userid_password_client_auth_functions_<version_number> としても知られる) のために DB2 データベース・マネージャーによって提供されたメモリーを指すポインター。 db2secGssapiClientAuthFunctions_<version_number> 構造は、GSS-API 認証プラグイン用にインプリメントされた API を指すポインターを含んでおり、db2secUseridPasswordClientAuthFunctions_<version_number> 構造は、ユーザー ID/パスワード認証プラグイン用にインプリメントされた API を指すポインターを含んでいます。 DB2 の将来のバージョンでは、異なるバージョンの API が存在している可能性があるので、client_fns パラメーターは、プラグインがインプリメントしているバージョンに対応する gssapi_client_auth_functions_<version_number> 構造を指すポインターとしてキャストします。

gssapi_client_auth_functions_<version_number> 構造または userid_password_client_auth_functions_<version_number> 構造の最初のパラメーターは、プラグインがインプリメントしている API のバージョンを DB2 データベース・マネージャーに知らせます。

注: DB2 のバージョンが、プラグインがインプリメントしている API のバージョンと同じかそれより大きい場合に限り、キャストが行われます。

gssapi_server_auth_functions_<version_number> または userid_password_server_auth_functions_<version_number> 構造内では、plugintype パスワードを DB2SEC_PLUGIN_TYPE_USERID_PASSWORD、

DB2SEC_PLUGIN_TYPE_GSSAPI、または DB2SEC_PLUGIN_TYPE_KERBEROS のいずれかに設定する必要があります。将来のバージョンの API では、他の値も定義される可能性があります。

logMessage_fn

入力。DB2 データベース・マネージャーによってインプリメントされる db2secLogMessage API を指すポインター。 db2secClientAuthPluginInit API は、db2secLogMessage API を呼び出して、デバッグまたは通知の目的でメッセージを db2diag.log に記録することができます。 db2secLogMessage API の最初のパラメーター (level) は、db2diag.log ファイルに記録される診断エラーのタイプを指定し、最後の 2 つのパラメーターはそれぞれメッセージ・ストリングとその長さです。 (db2secPlugin.h で定義された) dbesecLogMessage API の最初のパラメーターの有効な値は以下のとおりです。

- DB2SEC_LOG_NONE (0) ロギングなし
- DB2SEC_LOG_CRITICAL (1) 重大エラーを検出した
- DB2SEC_LOG_ERROR (2) エラーを検出した
- DB2SEC_LOG_WARNING (3) 警告
- DB2SEC_LOG_INFO (4) 通知

メッセージ・テキストが db2diag.log に表示されるのは、db2secLogMessage API の 'level' パラメーターの値が 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。例えば DB2SEC_LOG_INFO 値を使用する場合、メッセージ・テキストは diaglevel データベース・マネージャー構成パラメーターに 4 が設定されている場合にのみ db2diag.log に表示されます。

errormsg

出力。プラグインによって割り振られた ASCII エラー・メッセージ・ストリングのアドレスを指すポインター。db2secClientAuthPluginInit API が正常に実行されない場合にこのパラメーターに戻されることがあります。

errormsglen

出力。 errormsg パラメーターのエラー・メッセージ・ストリングのバイト単位の長さを示す整数を指すポインター。

db2secClientAuthPluginTerm API - クライアント認証プラグイン・リソースのクリーンアップ

クライアント認証プラグインによって使用されるリソースを解放します。

この API は、DB2 データベース・マネージャーがクライアント認証プラグインをアンロードする直前に呼び出します。これは、プラグイン・ライブラリーが保持しているリソースの適切なクリーンアップを実行する、という方法でインプリメントされる必要があります。例えば、プラグインによって割り振られたメモリーを解放し、まだオープンしているファイルをクローズし、ネットワーク接続をクローズします。これらのリソースを解放するためにその記録を保持することは、プラグインが行います。この API は Windows プラットフォームでは呼び出されません。

API とデータ構造構文

```
SQL_API_RC ( SQL_API_FN *db2secClientAuthPluginTerm)
                      ( char     **errormsg,
                        db2int32 *errormsglen);
```

db2secClientAuthPluginTerm API パラメーター

errormsg
出力。 db2secClientAuthPluginTerm API が正常に実行されない場合にこのパラメーターで戻されることのある、プラグインによって割り振られた ASCII エラー・メッセージ・ストリングのアドレスを指すポインター。

errormsglen
出力。 errormsg パラメーターのエラー・メッセージ・ストリングのバイト単位の長さを示す整数を指すポインター。

db2secDoesAuthIDExist - 認証 ID の存在の検査

authid が個々のユーザーを表しているかどうか (例えば、この API がこの authid を外部ユーザーにマップできるかどうか) を判別します。

この API は、これが正常 (authid が有効) な場合は値 DB2SEC_PLUGIN_OK を、無効な場合は DB2SEC_PLUGIN_INVALID_USERORGROUP を、authid の存在を判別できない場合は DB2SEC_PLUGIN_USERSTATUSNOTKNOWN を戻す必要があります。

API とデータ構造構文

```
SQL_API_RC ( SQL_API_FN *db2secDoesAuthIDExist)
                      ( const char *authid,
                        db2int32 authidlen,
                        char       **errormsg,
                        db2int32 *errormsglen );
```

db2secDoesAuthIDExist API パラメーター

authid 入力。検証する authid。これは、末尾ブランクなしの大文字になります。

authidlen
入力。authid パラメーター値のバイト単位の長さ。

errormsg
出力。 db2secDoesAuthIDExist API が正常に実行されない場合にこのパラメーターで戻されることのある、プラグインによって割り振られた ASCII エラー・メッセージ・ストリングのアドレスを指すポインター。

errormsglen
出力。 errormsg パラメーターのエラー・メッセージ・ストリングの長さを示す整数を指すポインター。

db2secFreeInitInfo API - db2secGenerateInitialCred が保持しているリソースのクリーンアップ

db2secGenerateInitialCred API によって割り振られたすべてのリソースを解放します。これには、例えば、基礎メカニズム・コンテキストのハンドルや、 GSS-API 証明書キャッシュ用に作成された証明書キャッシュが含まれます。

API とデータ構造構文

```
SQL_API_RC ( SQL_API_FN *db2secFreeInitInfo)
                     ( void *initinfo,
                       char     **errormsg,
                       db2int32 *errormsglen);
```

db2secFreeInitInfo API パラメーター

initinfo

入力。DB2 データベース・マネージャーに認識されていないデータを指すポインター。プラグインはこのメモリーを使用して、証明書ハンドルの生成プロセスで割り振られたリソースのリストを保守できます。これらのリソースは、この API を呼び出すことによって解放されます。

errormsg

出力。プラグインによって割り振られた ASCII エラー・メッセージ・ストリングのアドレスを指すポインター。db2secFreeInitInfo API が正常に実行されない場合にこのパラメーターに戻されることがあります。

errormsglen

出力。 errormsg パラメーターのエラー・メッセージ・ストリングのバイト単位の長さを示す整数を指すポインター。

db2secFreeToken API - トークンが保持しているメモリーの解放

トークンによって保持されたメモリーを解放します。この API は、DB2 データベース・マネージャーが token パラメーターによって保持されているメモリーを必要としなくなったときに呼び出します。

API とデータ構造構文

```
SQL_API_RC ( SQL_API_FN *db2secFreeToken)
                     ( void *token,
                       char       **errormsg,
                       db2int32 *errormsglen );
```

db2secFreeToken API パラメーター

token 入力。解放されるメモリーを指すポインター。

errormsg

出力。プラグインによって割り振られた ASCII エラー・メッセージ・ストリングのアドレスを指すポインター。db2secFreeToken API が正常に実行されない場合にこのパラメーターに戻されることがあります。

errormsglen

出力。 errormsg パラメーターのエラー・メッセージ・ストリングのバイト単位の長さを示す整数を指すポインター。

db2secGenerateInitialCred API - 初期証明書の生成

渡されるユーザー ID およびパスワードを基に初期 GSS-API 証明書を取得します。 Kerberos の場合、これは発券許可証 (TGT) になります。pGSSCredHandle パラメーターに戻される証明書ハンドルは、 gss_init_sec_context API で使用されるハンドルであり、INITIATE 証明書か BOTH 証明書のいずれかでなければなりません。 db2secGenerateInitialCred API は、ユーザー ID、そしておそらくパスワードが

指定されている場合にのみ呼び出されます。それ以外の場合、DB2 データベース・マネージャーは gss_init_sec_context API を呼び出すときに値 GSS_C_NO_CREDENTIAL を指定して、現行ログイン・コンテキストから取得されるデフォルト証明書が使用されることを示します。

API とデータ構造構文

```
SQL_API_RC ( SQL_API_FN *db2secGenerateInitialCred)
                       ( const char *userid,
                         db2int32 useridlen,
                         const char *usernamespace,
                         db2int32 usernamespacelen,
                         db2int32 usernamespacetype,
                         const char *password,
                         db2int32 passwordlen,
                         const char *newpassword,
                         db2int32 newpasswordlen,
                         const char *dbname,
                         db2int32 dbnamelen,
                         gss_cred_id_t *pGSSCredHandle,
                         void          **InitInfo,
                         char          **errormsg,
                         db2int32 *errormsglen );
```

db2secGenerateInitialCred API パラメーター

userid 入力。データベース・サーバー上でパスワードが検証されるユーザー ID。

useridlen
入力。userid パラメーター値のバイト単位の長さ。

usernamespace
入力。取得されたユーザー ID が属するネーム・スペース。

usernamespacelen
入力。usernamespace パラメーター値のバイト単位の長さ。

usernamespacetype
入力。ネーム・スペースのタイプ。

password
入力。検証されるパスワード。

passwordlen
入力。password パラメーター値のバイト単位の長さ。

newpassword
入力。パスワードが変更される場合の新規パスワード。変更が要求されない場合、newpassword パラメーターは NULL に設定されます。これが非ヌルの場合、API は、旧パスワードを新規パスワードに設定する前に検証する必要があります。 API は、パスワードの変更要求を受け入れなくても構いませんが、受け入れない場合は、旧パスワードを検証せずに即時に戻り値 DB2SEC_PLUGIN_CHANGEPASSWORD_NOTSUPPORTED を戻す必要があります。

newpasswordlen
入力。newpassword パラメーター値のバイト単位の長さ。

dbname
入力。接続先のデータベースの名前。この API はこのパラメーターを無視

しても差し支えありません。あるいは、特定のデータベースへのアクセスを、有効なパスワードを特別に持つユーザーのみに限定する方針をとっている場合は、この関数は値 DB2SEC_PLUGIN_CONNECTION_DISALLOWED を戻すことができます。

dbnamelen
入力。dbname パラメーター値のバイト単位の長さ。

pGSSCredHandle
出力。 GSS-API 証明書ハンドルを指すポインター。

InitInfo
出力。 DB2 に認識されていないデータを指すポインター。プラグインはこのメモリーを使用して、証明書ハンドルの生成プロセスで割り振られたリソースのリストを保守できます。 DB2 データベース・マネージャーは認証プロセスの最後に db2secFreeInitInfo API を呼び出し、その時点でこれらのリソースは解放されます。 db2secGenerateInitialCred API は、このようなリストを保守する必要がない場合は、NULL を戻す必要があります。

errormsg
出力。 db2secGenerateInitialCred API が正常に実行されない場合にこのパラメーターで戻されることのある、プラグインによって割り振られた ASCII エラー・メッセージ・ストリングのアドレスを指すポインター。

注: この API では、戻り値が無効なユーザー ID またはパスワードを示している場合には、エラー・メッセージは作成されるべきではありません。エラー・メッセージは、API の中に、この API が正しく完了することを妨げる内部エラーがある場合にのみ戻される必要があります。

errormsglen
出力。 errormsg パラメーターのエラー・メッセージ・ストリングのバイト単位の長さを示す整数を指すポインター。

db2secGetAuthIDs API - 認証 ID の取得

認証ユーザーの SQL authid を戻します。この API は、ユーザー ID/パスワードと GSS-API の両方の認証方式において、データベース接続時に呼び出されます。

API とデータ構造構文

```
SQL_API_RC ( SQL_API_FN *db2secGetAuthIDs)
                      ( const char *userid,
                        db2int32 useridlen,
                        const char *usernamespace,
                        db2int32 usernamespacelen,
                        db2int32 usernamespacetype,
                        const char *dbname,
                        db2int32 dbnamelen,
                        void      **token,
                        char SystemAuthID[DB2SEC_MAX_AUTHID_LENGTH],
                        db2int32 *SystemAuthIDlen,
                        char InitialSessionAuthID[DB2SEC_MAX_AUTHID_LENGTH],
                        db2int32 *InitialSessionAuthIDlen,
                        char username[DB2SEC_MAX_USERID_LENGTH],
                        db2int32 *usernamelen,
                        db2int32 *initsessionidtype,
                        char      **errormsg,
                        db2int32 *errormsglen );
```

db2secGetAuthIDs API パラメーター

userid 入力。認証ユーザー。 GSS-API 認証では、通常は使用されません。ただし、認証なしのユーザー切り替え操作を認めるトラステッド・コンテキストが定義されている場合は例外です。そのような場合は、ユーザーの切り替え要求で指定されているユーザー名がこのパラメーターで渡されます。

useridlen
入力。userid パラメーター値のバイト単位の長さ。

usernamespace
入力。取得されたユーザー ID が属するネーム・スペース。

usernamespacelen
入力。usernamespace パラメーター値のバイト単位の長さ。

usernamespacetype
入力。ネーム・スペース・タイプ値。現時点でサポートされる唯一のネーム・スペース・タイプ値は、DB2SEC_NAMESPACE_SAM_COMPATIBLE です (domain¥myname などのユーザー名スタイルに相当します)。

dbname
入力。接続先のデータベースの名前。 API はこれを無視しても差し支えありません。あるいはプラグインは、同一のユーザーが異なるデータベースに接続する際に別々の authid を戻すこともできます。このパラメーターは、NULL にすることができます。

dbnamelen
入力。dbname パラメーター値のバイト単位の長さ。 dbname パラメーターが NULL の場合、このパラメーターは 0 に設定されます。

token 入力または出力。プラグインが db2secGetGroupsForUser API に渡すデータ。 GSS-API の場合、これはコンテキスト・ハンドル (gss_ctx_id_t) です。通常、トークンは入力のみのパラメーターであり、この値は db2secValidatePassword から取り込まれます。認証がクライアントで行われ、そのために db2secValidatePassword API が呼び出されない場合には、これは出力パラメーターにもなります。認証なしのユーザー切り替え操作を認めるトラステッド・コンテキストが定義されている環境では、db2secGetAuthIDs API でこのトークン・パラメーターの NULL 値を受け付け、上記の userid 入力パラメーターと useridlen 入力パラメーターに基づいて、システム許可 ID を派生させることができるように設定しなければなりません。

SystemAuthID
出力。認証ユーザーの ID に対応するシステム許可。サイズは 255 バイトですが、DB2 データベース・マネージャーは現在最大 30 バイトまで使用します。

SystemAuthIDlen
出力。 SystemAuthID パラメーター値のバイト単位の長さ。

InitialSessionAuthID
出力。この接続セッションに使用される authid。これは通常 SystemAuthID パラメーターと同じですが、SET SESSION AUTHORIZATION ステートメ

ントを発行する場合などのある特定の場合には異なることがあります。サイズは 255 バイトですが、DB2 データベース・マネージャーは現在最大 30 バイトまで使用します。

InitialSessionAuthIDlen
出力。 InitialSessionAuthID パラメーター値のバイト単位の長さ。

username
出力。認証ユーザーと authid に対応するユーザー名。これは監査のためにのみ使用され、CONNECT ステートメントの監査記録内の「ユーザー ID」フィールドに記録されます。 API が username パラメーターに記入していない場合、DB2 データベース・マネージャーは userid からそれをコピーします。

usernamelen
出力。 username パラメーター値のバイト単位の長さ。

initsessionidtype
出力。 InitialSessionAuthid パラメーターがロールか authid かを示すセッション authid タイプ。 API は、以下の値のいずれか (db2secPlugin.h で定義された) を戻さなければなりません。

- DB2SEC_ID_TYPE_AUTHID (0)
- DB2SEC_ID_TYPE_ROLE (1)

errormsg
出力。 db2secGetAuthIDs API が正常に実行されない場合にこのパラメーターで戻されることのある、プラグインによって割り振られた ASCII エラー・メッセージ・ストリングのアドレスを指すポインター。

errormsglen
出力。 errormsg パラメーターのエラー・メッセージ・ストリングのバイト単位の長さを示す整数を指すポインター。

db2secGetDefaultLoginContext API - デフォルト・ログイン・コンテキストの取得

デフォルト・ログイン・コンテキストに関連したユーザーを判別します。すなわち、ユーザー ID を明示的に指定しない (データベースに対する暗黙的な認証か、ローカル許可) で DB2 コマンドを呼び出すユーザーの DB2 authid を判別します。この API は、authid とユーザー ID の両方を戻さなければなりません。

API とデータ構造構文

```
SQL_API_RC ( SQL_API_FN *db2secGetDefaultLoginContext)
                      ( char authid[DB2SEC_MAX_AUTHID_LENGTH],
                        db2int32 *authidlen,
                        char userid[DB2SEC_MAX_USERID_LENGTH],
                        db2int32 *useridlen,
                        db2int32 useridtype,
                        char usernamespace[DB2SEC_MAX_USERNAMESPACE_LENGTH],
                        db2int32 *usernamespacelen,
                        db2int32 *usernamespacetype,
                        const char *dbname,
                        db2int32 dbnamelen,
                        void      **token,
                        char      **errormsg,
                        db2int32 *errormsglen );
```

db2secGetDefaultLoginContext API パラメーター

authid 出力。 authid が戻されるパラメーター。戻り値は DB2 authid の命名規則に準拠していなければなりません。そうでなければ、ユーザーは要求されたアクションの実行を許可されません。

authidlen
出力。 authid パラメーター値のバイト単位の長さ。

userid 出力。デフォルト・ログイン・コンテキストに関連したユーザー ID を戻すパラメーター。

useridlen
出力。 userid パラメーター値のバイト単位の長さ。

useridtype
入力。プロセスの実ユーザー ID、または有効ユーザー ID が指定されているかどうかを示します。 Windowsの場合、実ユーザー ID のみ存在します。 UNIX および Linux では、アプリケーションの `uid` ユーザー ID がプロセスを実行しているユーザーの ID と異なる場合、実ユーザー ID と有効ユーザー ID が異なることがあります。 userid パラメーターの有効な値 (db2secPlugin.h で定義されている) は以下のとおりです。

DB2SEC_PLUGIN_REAL_USER_NAME
実ユーザー ID が指定されていることを示します。

DB2SEC_PLUGIN_EFFECTIVE_USER_NAME
有効ユーザー ID が指定されていることを示します。

注: 一部のプラグイン・インプリメンテーションによっては、実ユーザー ID と有効ユーザー ID を区別しないものがあります。特に、DB2 許可 ID を設定するためにユーザーの UNIX または Linux の ID を使用しないプラグインは、この区別を無視しても支障がありません。

usernamespace
出力。ユーザー ID のネーム・スペース。

usernamespacelen
出力。 usernamespace パラメーター値のバイト単位の長さ。
usernamespacetype パラメーターが値
DB2SEC_NAMESPACE_SAM_COMPATIBLE (db2secPlugin.h で定義されている) に設定されていなければならないという制限のもとでは、現在サポートされる最大長は 15 バイトになります。

usernamespacetype
出力。ネーム・スペース・タイプ値。現時点でサポートされる唯一のネーム・スペース・タイプ値、DB2SEC_NAMESPACE_SAM_COMPATIBLE です (domain¥myname などのユーザー名スタイルに相当します)。

dbname
入力。データベース接続のコンテキストでこの呼び出しが使用される場合に、接続先のデータベースの名前が入ります。ローカル許可アクションやインスタンス接続の場合、このパラメーターは NULL に設定されます。

dbnamelen
入力。dbname パラメーター値のバイト単位の長さ。

token 出力。これはプラグインが、そのプラグインでの後の認証呼び出しや、またはグループ検索プラグインに渡す、プラグインによって割り振られたデータを指すポインターです。このデータの構造は、プラグイン作成者によって決定されます。

errormsg
出力。 db2secGetDefaultLoginContext API が正常に実行されない場合にこのパラメーターで戻されることのある、プラグインによって割り振られた ASCII エラー・メッセージ・ストリングのアドレスを指すポインター。

errormsglen
出力。 errormsg パラメーターのエラー・メッセージ・ストリングのバイト単位の長さを示す整数を指すポインター。

db2secProcessServerPrincipalName API - サーバーから戻されたサービス・プリンシパル名の処理

サーバーから戻されたサービス・プリンシパル名を処理し、gss_init_sec_context API で使用される gss_name_t 内部形式のプリンシパル名を戻します。 db2secProcessServerPrincipalName API は、Kerberos 認証の使用時に、データベース・ディレクトリーでカタログされたサービス・プリンシパル名も処理します。通常、この変換では gss_import_name API が使用されます。コンテキストが確立されると、gss_name_t オブジェクトは gss_release_name API の呼び出しによって解放されます。 db2secProcessServerPrincipalName API は、gssName パラメーターが有効な GSS 名を指していれば値 DB2SEC_PLUGIN_OK を戻します。プリンシパル名が無効な場合は DB2SEC_PLUGIN_BAD_PRINCIPAL_NAME エラー・コードが戻されます。

API とデータ構造構文

```
SQL_API_RC ( SQL_API_FN *db2secProcessServerPrincipalName)
                      ( const char *name,
                        db2int32 namelen,
                        gss_name_t *gssName,
                        char       **errormsg,
                        db2int32 *errormsglen );
```

db2secProcessServerPrincipalName API パラメーター

name 入力。GSS_C_NT_USER_NAME 形式のサービス・プリンシパルのテキスト名 (例: service/host@REALM)。

namelen
入力。name パラメーター値のバイト単位の長さ。

gssName
出力。 GSS-API 内部形式の出力サービス・プリンシパル名を指すポインター。

errormsg
出力。プラグインによって割り振られた ASCII エラー・メッセージ・ストリングのアドレスを指すポインター。db2secProcessServerPrincipalName API が正常に実行されない場合にこのパラメーターに戻されることがあります。

errormsglen
出力。 errormsg パラメーターのエラー・メッセージ・ストリングのバイト単位の長さを示す整数を指すポインター。

db2secRemapUserid API - ユーザー ID およびパスワードの再マップ

この API は DB2 データベース・マネージャーによってクライアント・サイドで呼び出され、特定のユーザー ID およびパスワード (そしておそらく新規パスワードおよび usernamespace) を、接続時に指定された値とは異なる値に再マップします。DB2 データベース・マネージャーは、接続時にユーザー ID およびパスワードが指定されている場合にのみ、この API を呼び出します。これは、プラグインがユーザー ID を自らユーザー ID/パスワードのペアに再マップすることを防止します。この API はオプションであり、セキュリティー・プラグインによって提供あるいはインプリメントされていなければ呼び出されません。

API とデータ構造構文

```
SQL_API_RC ( SQL_API_FN *db2secRemapUserid)
                       ( char userid[DB2SEC_MAX_USERID_LENGTH],
                         db2int32 *useridlen,
                         char usernamespace[DB2SEC_MAX_USERNAMESPACE_LENGTH],
                         db2int32 *usernamespacelen,
                         db2int32 *usernamespacetype,
                         char password[DB2SEC_MAX_PASSWORD_LENGTH],
                         db2int32 *passwordlen,
                         char newpasswd[DB2SEC_MAX_PASSWORD_LENGTH],
                         db2int32 *newpasswdlen,
                         const char *dbname,
                         db2int32 dbnamelen,
                         char      **errormsg,
                         db2int32 *errormsglen);
```

db2secRemapUserid API パラメーター

userid 入力または出力。再マップされるユーザー ID。入力ユーザー ID 値がある場合は、API は出力ユーザー ID 値を提供しなければならず、それは入力ユーザー ID 値と同じか、あるいは異なる値になる可能性があります。入力ユーザー ID 値がない場合、API は出力ユーザー ID 値を戻すべきではありません。

useridlen
入力または出力。userid パラメーター値のバイト単位の長さ。

usernamespace
入力または出力。ユーザー ID のネーム・スペース。この値は、オプションで再マップすることができます。入力パラメーター値が指定されず、出力値が戻されている場合、usernamespace は CLIENT タイプ認証の場合にのみ DB2 データベース・マネージャーによって使用され、他の認証タイプでは無視されます。

usernamespacelen
入力または出力。usernamespace パラメーター値のバイト単位の長さ。usernamespacetype パラメーターが値 DB2SEC_NAMESPACE_SAM_COMPATIBLE (db2secPlugin.h で定義されて

いる) に設定されていなければならないという制限のもとでは、現在サポートされる最大長は 15 バイトになります。

usernamespacetype
入力または出力。namespacetype の古い値と新しい値。現時点でサポートされる唯一のネーム・スペース・タイプ値は、DB2SEC_NAMESPACE_SAM_COMPATIBLE です (domain¥myname などのユーザー名スタイルに相当します)。

password
入力または出力。入力の場合、これは再マップ対象のパスワードになります。出力の場合、これは再マップ済みパスワードになります。このパラメーターで入力値が指定されている場合、API は入力値とは異なる出力値を戻すことができなければなりません。入力値が指定されていない場合、API は出力 password 値を戻してはなりません。

passwordlen
入力または出力。password パラメーター値のバイト単位の長さ。

newpasswd
入力または出力。入力の場合、これは設定される新規パスワードになります。出力の場合、これは確認済みの新規パスワードになります。

注: これは、DB2 データベース・マネージャーがクライアント上またはサーバー上のどちらか (認証データベース・マネージャー構成パラメーターの値に応じる) の db2secValidatePassword API の newpassword パラメーターに渡す新規パスワードです。新規パスワードが入力として渡された場合は、API は出力値を戻すことができなければなりませんが、出力値が別の新規パスワードである可能性もあります。入力として渡された新規パスワードがない場合、API は出力の新規パスワードを戻すべきではありません。

newpasswdlen
入力または出力。newpasswd パラメーター値のバイト単位の長さ。

dbname
入力。クライアントの接続先のデータベースの名前。

dbnamelen
入力。dbname パラメーター値のバイト単位の長さ。

errormsg
出力。 db2secRemapUserid API が正常に実行されない場合にこのパラメーターで戻されることのある、プラグインによって割り振られた ASCII エラー・メッセージ・ストリングのアドレスを指すポインター。

errormsglen
出力。 errormsg パラメーターのエラー・メッセージ・ストリングのバイト単位の長さを示す整数を指すポインター。

db2secServerAuthPluginInit - サーバー認証プラグインの初期化

プラグインのロードの直後に DB2 データベース・マネージャーが呼び出す、サーバー認証プラグイン用の初期化 API。GSS-API の場合は、プラグインが、初期化時に gssapi_server_auth_functions 構造内部の serverPrincipalName パラメーターにサーバーのプリンシパル名を入れ、gssapi_server_auth_functions 構造内部の

serverCredHandle パラメーターにサーバーの証明書ハンドルを提供します。プリンシパル名および証明書ハンドルを保持するために割り振られているメモリーを解放するのは、gss_release_name および gss_release_cred API を呼び出すことによって、db2secServerAuthPluginTerm API が行うべき事柄です。

API とデータ構造構文

```
SQL_API_RC SQL_API_FN db2secServerAuthPluginInit
                              ( db2int32 version,
                                void *server_fns,
                                db2secGetConDetails *getConDetails_fn,
                                db2secLogMessage *logMessage_fn,
                                char               **errormsg,
                                db2int32 *errormsglen );
```

db2secServerAuthPluginInit API パラメーター

version

入力。DB2 データベース・マネージャーが現在サポートしている API の最大のバージョン番号。 DB2SEC_API_VERSION 値 (db2secPlugin.h 内) には、DB2 データベース・マネージャーが現在サポートしている API の最新のバージョン番号が含まれます。

server_fns

出力。 GSS-API 認証が使用される場合は、db2secGssapiServerAuthFunctions_<version_number> 構造 (gssapi_server_auth_functions_<version_number> としても知られる) のために DB2 データベース・マネージャーによって提供されたメモリーを指すポインター。ユーザー ID/パスワード認証が使用される場合は、db2secUseridPasswordServerAuthFunctions_<version_number> 構造 (userid_password_server_auth_functions_<version_number> としても知られる) のために DB2 データベース・マネージャーによって提供されたメモリーを指すポインター。 db2secGssapiServerAuthFunctions_<version_number> 構造は、GSS-API 認証プラグイン用にインプリメントされた API を指すポインターを含んでおり、db2secUseridPasswordServerAuthFunctions_<version_number> 構造は、ユーザー ID/パスワード認証プラグイン用にインプリメントされた API を指すポインターを含んでいます。

server_fns パラメーターは、プラグインがインプリメントしているバージョンに対応する gssapi_server_auth_functions_<version_number> 構造を指すポインターとしてキャストします。gssapi_server_auth_functions_<version_number> 構造または userid_password_server_auth_functions_<version_number> 構造の最初のパラメーターは、プラグインがインプリメントしている API のバージョンを DB2 データベース・マネージャーに知らせます。

注: DB2 のバージョンが、プラグインがインプリメントしている API のバージョンと同じかそれより大きい場合に限り、キャストが行われます。

gssapi_server_auth_functions_<version_number> または userid_password_server_auth_functions_<version_number> 構造内では、plugintype パスワードを DB2SEC_PLUGIN_TYPE_USERID_PASSWORD、DB2SEC_PLUGIN_TYPE_GSSAPI、または

DB2SEC_PLUGIN_TYPE_KERBEROS のいずれかに設定する必要があります。将来のバージョンの API では、他の値も定義される可能性があります。

getConDetails_fn

入力。DB2 によってインプリメントされる db2secGetConDetails API を指すポインター。 db2secServerAuthPluginInit API は、いずれかの他の認証 API で db2secGetConDetails API を呼び出して、データベース接続に関する詳細を取得することができます。これらの詳細には、接続に関連した通信メカニズム (TCP/IP の場合は IP アドレスなど) についての情報が含まれ、これは、プラグイン作成者が認証についての決定をする際に参照しなければならない可能性があります。例えば、プラグインは、特定のユーザーが特定の IP アドレスから接続しようとしているのでない場合、そのユーザーの接続を禁止できます。 db2secGetConDetails API の使用はオプションです。

データベース接続に関係しない状況で db2secGetConDetails API が呼び出された場合、これは値 DB2SEC_PLUGIN_NO_CON_DETAILS を戻し、それ以外の場合は、正常なら 0 を戻します。

db2secGetConDetails API は、db2sec_con_details_<version_number> 構造を指すポインターである pConDetails と、使用される db2sec_con_details 構造を示すバージョン番号である conDetailsVersion という 2 つの入力パラメーターをとります。可能な値は、db2sec_con_details1 が使用される場合は DB2SEC_CON_DETAILS_VERSION_1 で、db2sec_con_details2 が使用される場合は DB2SEC_CON_DETAILS_VERSION_2 です。使用するように推奨されているバージョン番号は DB2SEC_CON_DETAILS_VERSION_2 です。

正常に戻る場合、db2sec_con_details 構造 (db2sec_con_details1 または db2sec_con_details2) には以下の情報が含まれます。

- サーバーへの接続に使用されるプロトコル。プロトコル定義のリストは、ファイル sqlenv.h (include ディレクトリーにある) (SQL_PROTOCOL_*) にあります。この情報は clientProtocol パラメーターに書き込まれます。
- clientProtocol が SQL_PROTOCOL_TCPIP または SQL_PROTOCOL_TCPIP4 の場合、サーバーへのインバウンド接続の TCP/IP アドレス。この情報は clientIPAddress パラメーターに書き込まれます。
- クライアントが接続しようとしているデータベースの名前。これは、インスタンス接続の場合は設定されません。この情報は dbname および dbnameLen パラメーターに書き込まれます。
- db2secValidatePassword API の connection_details パラメーター内に記述されるのと同じ詳細を含む、接続情報のビットマップ。この情報は connect_info_bitmap パラメーターに書き込まれます。
- clientProtocol が SQL_PROTOCOL_TCPIP6 の場合、サーバーへのインバウンド接続の TCP/IP アドレス。この情報は clientIP6Address パラメーターに書き込まれ、db2secGetConDetails API 呼び出しに DB2SEC_CON_DETAILS_VERSION_2 が使用される場合にのみ使用できます。

logMessage_fn

入力。DB2 データベース・マネージャーによってインプリメントされる db2secLogMessage API を指すポインター。 db2secClientAuthPluginInit API は、db2secLogMessage API を呼び出して、デバッグまたは通知の目的でメッセージを db2diag.log に記録することができます。 db2secLogMessage API の最初のパラメーター (level) は、db2diag.log ファイルに記録される診断エラーのタイプを指定し、最後の 2 つのパラメーターはそれぞれメッセージ・ストリングとその長さです。 (db2secPlugin.h で定義された) dbesecLogMessage API の最初のパラメーターの有効な値は以下のとおりです。

DB2SEC_LOG_NONE (0)
ロギングなし

DB2SEC_LOG_CRITICAL (1)
重大エラーが検出された

DB2SEC_LOG_ERROR (2)
エラーが検出された

DB2SEC_LOG_WARNING (3)
警告

DB2SEC_LOG_INFO (4)
通知

メッセージ・テキストが db2diag.log に表示されるのは、db2secLogMessage API の 'level' パラメーターの値が 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。

そのため、例えば DB2SEC_LOG_INFO 値を使用する場合、メッセージ・テキストは diaglevel データベース・マネージャー構成パラメーターに 4 が設定されている場合にのみ db2diag.log に表示されます。

errormsg
出力。プラグインによって割り振られた ASCII エラー・メッセージ・ストリングのアドレスを指すポインター。db2secServerAuthPluginInit API が正常に実行されない場合にこのパラメーターに戻されることがあります。

errormsglen
出力。 errormsg パラメーターのエラー・メッセージ・ストリングのバイト単位の長さを示す整数を指すポインター。

db2secServerAuthPluginTerm API - サーバー認証プラグイン・リソースのクリーンアップ

サーバー認証プラグインによって使用されるリソースを解放します。この API は、DB2 データベース・マネージャーがサーバー認証プラグインをアンロードする直前に呼び出します。これは、プラグイン・ライブラリーが保持しているリソースの適切なクリーンアップを実行する、という方法でインプリメントされる必要があります。例えば、プラグインによって割り振られたメモリーを解放し、まだオープンしているファイルをクローズし、ネットワーク接続をクローズします。これらのリソースを解放するためにその記録を保持することは、プラグインが行います。この API は Windows プラットフォームでは呼び出されません。

API とデータ構造構文

```
SQL_API_RC ( SQL_API_FN *db2secServerAuthPluginTerm)
                     ( char     **errormsg,
                       db2int32 *errormsglen );
```

db2secServerAuthPluginTerm API パラメーター

errormsg
出力。 db2secServerAuthPluginTerm API が正常に実行されない場合にこのパラメーターで戻されることのある、プラグインによって割り振られた ASCII エラー・メッセージ・ストリングのアドレスを指すポインター。

errormsglen
出力。 errormsg パラメーターのエラー・メッセージ・ストリングのバイト単位の長さを示す整数を指すポインター。

db2secValidatePassword API - パスワードの検証

データベース接続の操作時に、ユーザー ID およびパスワードのスタイル認証を実行する方法について説明します。

注: API がクライアント・サイドで実行されている場合、API コードは、CONNECT ステートメントを実行するユーザーの特権で実行されます。この API は、認証構成パラメーターに CLIENT が設定されている場合にのみ、クライアント・サイドで呼び出されます。

API がサーバー・サイドで実行されている場合、API コードはインスタンス所有者の特権で実行されます。

認証に特別な特権 (UNIX 上のルート・レベルのシステム・アクセスなど) が必要な場合、プラグイン作成者は、上記の点を考慮する必要があります。

この API は、パスワードが正常な場合は値 DB2SEC_PLUGIN_OK (正常) を、また、パスワードが無効な場合は DB2SEC_PLUGIN_BADPWD などのエラー・コードを戻す必要があります。

API とデータ構造構文

```
SQL_API_RC ( SQL_API_FN *db2secValidatePassword)
                     ( const char *userid,
                       db2int32 useridlen,
                       const char *usernamespace,
                       db2int32 usernamespacelen,
                       db2int32 usernamespacetype,
                       const char *password,
                       db2int32 passwordlen,
                       const char *newpasswd,
                       db2int32 newpasswdlen,
                       const char *dbname,
                       db2int32 dbnamelen,
                       db2Uint32 connection_details,
                       void      **token,
                       char      **errormsg,
                       db2int32 *errormsglen );
```

db2secValidatePassword API パラメーター

userid 入力。パスワードが検証されるユーザー ID。

useridlen
入力。userid パラメーター値のバイト単位の長さ。

usernamespace
入力。取得されたユーザー ID が属するネーム・スペース。

usernamespacelen
入力。usernamespace パラメーター値のバイト単位の長さ。

usernamespacetype
入力。ネーム・スペースのタイプ。 usernamespacetype パラメーターの有効な値 (db2secPlugin.h で定義されている) は以下のとおりです。

- DB2SEC_NAMESPACE_SAM_COMPATIBLE は domain¥myname などのユーザー名スタイルに対応します。
- DB2SEC_NAMESPACE_USER_PRINCIPAL は myname@domain.ibm.com などのユーザー名スタイルに対応します。

現在のところ、DB2 データベース・システムは値 DB2SEC_NAMESPACE_SAM_COMPATIBLE しかサポートしていません。ユーザー ID がない場合、usernamespacetype パラメーターの値は DB2SEC_USER_NAMESPACE_UNDEFINED (db2secPlugin.h で定義された) に設定されます。

password
入力。検証されるパスワード。

passwordlen
入力。password パラメーター値のバイト単位の長さ。

newpasswd
入力。パスワードが変更される場合の新規パスワード。変更が要求されない場合、このパラメーターは NULL に設定されます。このパラメーターが非ヌルの場合、この API は、旧パスワードを新規パスワードに変更する前に検証する必要があります。 API は、パスワードの変更要求を受け入れなくても構いませんが、受け入れない場合は、旧パスワードを検証せずに即時に戻り値 DB2SEC_PLUGIN_CHANGEPASSWORD_NOTSUPPORTED を戻す必要があります。

newpasswdlen
入力。newpasswd パラメーター値のバイト単位の長さ。

dbname
入力。接続先のデータベースの名前。この API は dbname パラメーターを無視しても差し支えありません。あるいは、特定のデータベースへのアクセスを、有効なパスワードを特別に持つユーザーのみに限定する方針をとっている場合は、この関数は値 DB2SEC_PLUGIN_CONNECTIONREFUSED を戻すことができます。このパラメーターは、NULL にすることができます。

dbnamelen
入力。dbname パラメーター値のバイト単位の長さ。 dbname パラメーターが NULL の場合、このパラメーターは 0 に設定されます。

connection_details
入力。32 ビットのパラメーター。そのうちの 3 ビットが、以下の情報を保管するために現在使用されています。

- 右端のビットは、ユーザー ID のソースが db2secGetDefaultLoginContext のデフォルトであるか、それとも接続時に明示的に指定されているかを示します。
- 右から 2 番目のビットは、接続がローカル (Inter Process Communication (IPC) を使用しているか、あるいはパーティション・データベース環境の db2nodes.cfg 内にあるノードのいずれかからの接続である) か、リモート (ネットワークまたはループバックを経由) かを示します。これによって、API は同一のマシン上のクライアントがパスワードなしで DB2 サーバーに接続できるかどうか判別できます。デフォルトのオペレーティング・システム・ベースのユーザー ID/パスワード・プラグインにより、同一のマシン上のクライアントからのパスワードなしのローカル接続が常時許可されます (ユーザーが接続特権を持つ場合)。
- 右から 3 番目のビットは、DB2 データベース・マネージャーがサーバー・サイドまたはクライアント・サイドのどちらで API を呼び出しているかを示します。

ビット値は、db2secPlugin.h 内で次のように定義されています。

- DB2SEC_USERID_FROM_OS は、ユーザー ID は OS から取得され、接続ステートメントで明示的には指定されていないことを示します。
- DB2SEC_CONNECTION_ISLOCAL はローカル接続を示します。
- DB2SEC_VALIDATING_ON_SERVER_SIDE は、 DB2 データベース・マネージャーがサーバー・サイドまたはクライアント・サイドのどちらからパスワードの検証を呼び出しているかを示します。このビット値が設定されている場合、DB2 データベース・マネージャーはサーバー・サイドから呼び出しています。それ以外の場合は、クライアント・サイドから呼び出しています。

暗黙的な認証での DB2 データベース・システムのデフォルト動作では、パスワード検証なしの接続が許可されます。しかし、プラグイン開発者には、DB2SEC_PLUGIN_BADPASSWORD エラーを戻すことによって暗黙的な認証を禁止するという選択肢もあります。

token 入力。現行接続中の後続の API 呼び出しに渡されるデータを指すポインター。呼び出される可能性のある API は、db2secGetAuthIDs API と db2secGetGroupsForUser API です。

errormsg
出力。 db2secValidatePassword API が正常に実行されない場合にこのパラメーターで戻されることのある、プラグインによって割り振られた ASCII エラー・メッセージ・ストリングのアドレスを指すポインター。

errormsglen
出力。 errormsg パラメーターのエラー・メッセージ・ストリングのバイト単位の長さを示す整数を指すポインター。

GSS-API 認証プラグインに必要な API および定義

以下に、DB2 セキュリティー・プラグイン・インターフェースに必要な GSS-API の完全なリストを提示します。

サポートされている API は、*Generic Security Service Application Program Interface, Version 2* (IETF RFC2743) および *Generic Security Service API Version 2: C-Bindings* (IETF RFC2744) の仕様に準拠しています。 GSS-API ベース・プラグインをインプリメントする前に、これらの仕様について完全に理解しておく必要があります。

表 12. GSS-API 認証プラグインに必要な API および定義

名前		説明
クライアント・サイド API	gss_init_sec_context	対等アプリケーションとのセキュリティー・コンテキストを開始します。
サーバー・サイド API	gss_accept_sec_context	対等アプリケーションによって開始されたセキュリティー・コンテキストを受け入れます。
サーバー・サイド API	gss_display_name	内部フォーマットの名前をテキストに変換します。
共通 API	gss_delete_sec_context	確立されたセキュリティー・コンテキストを削除します。
共通 API	gss_display_status	GSS-API 状況コードに関連したテキスト・エラー・メッセージを取得します。
共通 API	gss_release_buffer	バッファーを削除します。
共通 API	gss_release_cred	GSS-API 証明書に関連したローカル・データ構造を解放します。
共通 API	gss_release_name	内部フォーマットの名前を削除します。
必要な定義	GSS_C_DELEG_FLAG	委任を要求します。
必要な定義	GSS_C_EMPTY_BUFFER	gss_buffer_desc にデータが含まれていないことを意味します。
必要な定義	GSS_C_GSS_CODE	GSS メジャー状況コードを示します。
必要な定義	GSS_C_INDEFINITE	メカニズムがコンテキストの有効期限をサポートしていないことを示します。
必要な定義	GSS_C_MECH_CODE	GSS マイナー状況コードを示します。
必要な定義	GSS_C_MUTUAL_FLAG	相互認証が要求されました。
必要な定義	GSS_C_NO_BUFFER	gss_buffer_t 変数が有効な gss_buffer_desc 構造を指していないことを示します。
必要な定義	GSS_C_NO_CHANNEL_BINDINGS	通信チャネル・バインディングがありません。
必要な定義	GSS_C_NO_CONTEXT	gss_ctx_id_t 変数が有効なコンテキストを指していないことを示します。
必要な定義	GSS_C_NO_CREDENTIAL	gss_cred_id_t 変数が有効な証明書ハンドルを指していないことを示します。
必要な定義	GSS_C_NO_NAME	gss_name_t 変数が有効な内部名を指していないことを示します。
必要な定義	GSS_C_NO_OID	デフォルト認証メカニズムを使用します。
必要な定義	GSS_C_NULL_OID_SET	デフォルト・メカニズムを使用します。
必要な定義	GSS_S_COMPLETE	API が正常に完了しました。
必要な定義	GSS_S_CONTINUE_NEEDED	プロセスが完了しておらず、対等機能から受け取った応答トークンを指定して API をもう一度呼び出す必要があります。

GSS-API 認証プラグインに関する制約事項

以下は、GSS-API 認証プラグインに関する制約事項のリストです。

- デフォルト・セキュリティー・メカニズムが常時採用されるため、OID についての考慮事項はありません。
- `gss_init_sec_context()` で要求される GSS サービスは、相互認証および委任だけです。 DB2 データベース・マネージャーは常に委任のためのチケットを要求しますが、そのチケットを使用して新規チケットを生成することはありません。
- デフォルト・コンテキスト時刻のみが要求されます。
- `gss_delete_sec_context()` からのコンテキスト・トークンは、クライアントからサーバー、またはその逆には送信されません。
- 匿名はサポートされていません。
- チャネル・バインディングはサポートされていません。
- 初期証明書の有効期限が切れた場合、DB2 データベース・マネージャーはそれを自動的に更新しません。
- GSS-API 仕様は、`gss_init_sec_context()` または `gss_accept_sec_context()` が失敗しても、対等機能に送信するトークンがいずれかの関数によって戻されなければならないことを規定しています。しかし、DRDA の制限のため、DB2 データベース・マネージャーは `gss_init_sec_context()` が失敗し、かつ最初の呼び出しでトークンを生成した場合にのみトークンを送信できます。

セキュリティー・プラグインのサンプル

UNIX および Linux ディレクトリーの場合: 'C' サンプルは、sqllib/samples/security/plugins に置かれていて、JCC GSS-API プラグイン・サンプル (.java) は、sqllib/samples/java/jdbc に置かれています。

Windows ディレクトリーの場合: 'C' サンプルは sqllib¥samples¥security¥plugins に置かれていて、JCC GSS-API プラグイン・サンプル (.java) は sqllib¥samples¥java¥jdbc に置かれています。

表 13. セキュリティー・プラグインのサンプル・プログラム・ファイル

サンプル・プログラム名	プログラムの説明
combined.c	複合ユーザー ID/パスワード認証およびグループ検索のサンプル。
group_file.c	単純ファイル・ベースのグループ管理プラグインのサンプル。
gssapi_simple.c	基本 GSS-API 認証プラグインのサンプル (クライアントとサーバーの両方)。
IBMLDAPauthclient.c	LDAP ユーザー・レジストリーと相互作用する、クライアント・サイドの DB2 セキュリティー・プラグインをインプリメントします。
IBMLDAPauthserver.c	LDAP ユーザー・レジストリーと相互作用する、サーバー・サイドの DB2 セキュリティー・プラグインをインプリメントします。

表 13. セキュリティー・プラグインのサンプル・プログラム・ファイル (続き)

サンプル・プログラム名	プログラムの説明
IBMLDAPconfig.c	DB2 LDAP セキュリティー・プラグインの構成ファイルを検出して構文解析する関数が入っています。
IBMLDAPgroups.c	LDAP ベースのグループ検索用の DB2 セキュリティー・プラグインをインプリメントします。
IBMLDAPutils.c	DB2 LDAP セキュリティー・プラグインで使用されるユーティリティー関数が入っています。
IBMLDAPutils.h	LDAP セキュリティー・プラグインのヘッダー・ファイル。
JCCKerberosPlugin.java	IBM DB2 Universal Driver を使用して Kerberos 認証を行う GSS-API プラグインをインプリメントします。
JCCKerberosPluginTest.java	JCCKerberosPlugin を使用して、IBM DB2 Universal Driver を使用する DB2 接続を取得します。
JCCSimpleGSSPlugin.java	IBM DB2 Universal Driver を使用してユーザー ID とパスワードの検査を行う GSS-API プラグインをインプリメントします。
JCCSimpleGSSContext.java	JCCSimpleGSSPlugin で使用される GSSContext をインプリメントします。
JCCSimpleGSSCredential.java	JCCSimpleGSSPlugin で使用される GSSCredential をインプリメントします。
JCCSimpleGSSException.java	JCCSimpleGSSPlugin で使用される GSSException をインプリメントします。
JCCSimpleGSSName.java	JCCSimpleGSSPlugin で使用される GSSName をインプリメントします。
JCCSimpleGSSPluginTest.java	JCCSimpleGSSPlugin を使用して、IBM DB2 Universal Driver を使用する DB2 接続を取得します。

Storage Manager に対するバックアップおよびリストアの DB2 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。このインターフェースは、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を、この節では以降、ベンダー製品と呼びます。

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 API 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 API 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。 API 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 API 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。

以下は、バックアップおよびリストア・ベンダー・ストレージ・プラグイン API の説明に使用される用語の定義です。

バックアップおよびリストア・ベンダー・ストレージ・プラグイン
DB2 がベンダー製品向けの、ユーザー作成のバックアップおよびリストア API にアクセスするためにロードする、動的にロード可能なライブラリー。

入力 DB2 が、バックアップおよびリストア・ベンダー・ストレージ・プラグイン API パラメーターに値を入力することを示します。

出力 バックアップおよびリストア・ベンダー・ストレージ・プラグイン API が API パラメーターに値を入力することを示します。

db2VendorGetNextObj - 装置での次のオブジェクトの入手

この API は、検索条件に一致する次のオブジェクト (イメージまたはアーカイブ・ログ・ファイル) を入手するように、(sqluvint API 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

なし

必要な接続

データベース。

API インクルード・ファイル

db2VendorApi.h

API とデータ構造構文

```
int db2VendorGetNextObj ( void                          * vendorCB,
                          struct db2VendorQueryInfo     * queryInfo,
                          struct Return_code            * returnCode);

typedef struct db2VendorQueryInfo
{
   db2Uint64 sizeEstimate;
   db2Uint32 type;
   SQL_PDB_NODE_TYPE dbPartitionNum;
   db2Uint16 sequenceNum;
   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];
} db2VendorQueryInfo;
```

db2VendorGetNextObj 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を指すポインター。

queryInfo
出力。ベンダー・ライブラリーによって記入される db2VendorQueryInfo 構造を指すポインター。

returnCode
出力。 API 呼び出しからの戻りコード。

db2VendorQueryInfo data structure パラメーター

sizeEstimate
オブジェクトの見積もりサイズを指定します。

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。

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。

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。

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。

使用上の注意

すべてのパラメーターが、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パラメーターは、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。残りのパラメーターは、定義する特定のベンダー用に残されます。特定のパラメーターが関係しない場合、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。

戻りコード

以下の表に、この API によって戻される可能性のあるすべての戻りコードをリストします。

表 14. db2VendorGetNextObj API の戻りコード

番号	戻りコード	説明
2	SQLUV_COMM_ERROR	装置との通信エラー - 失敗。
4	SQLUV_INV_ACTION	無効なアクションが要求されたか、不可能な操作を生じさせる入力パラメーターが組み合わされた - 失敗。
5	SQLUV_NO_DEV_AVAIL	現在使用できる装置がない - 失敗。
6	SQLUV_OBJ_NOT_FOUND	削除するオブジェクトがない - 失敗。
12	SQLUV_INV_DEV_HANDLE	無効な装置ハンドル - 失敗。
14	SQLUV_END_OF_DATA	戻される照会オブジェクトはこれ以上ない - 成功。
18	SQLUV_DEV_ERROR	装置エラー - 失敗。
19	SQLUV_WARNING	警告、戻りコードを参照 - 成功。
21	SQLUV_MORE_DATA	さらに照会オブジェクトが戻される - 成功。
25	SQLUV_IO_ERROR	入出力エラー - 失敗。
30	SQLUV_UNEXPECTED_ERROR	重大エラーが検出された - 失敗。

db2VendorQueryApiVersion - ベンダー・ストレージ API のサポート・レベルの取得

バックアップおよびリストア・ベンダー・ストレージ・プラグインによってサポートされるベンダー・ストレージ API のレベルを判別します。指定されたベンダー・ストレージ・プラグインが DB2 と互換性がない場合、それは使用されません。

ベンダー・ストレージ・プラグインがこの API をログ用にインプリメントしていない場合、それは使用できず、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

なし

必要な接続

データベース。

API インクルード・ファイル

db2VendorApi.h

API とデータ構造構文

```
void db2VendorQueryApiVersion ( db2Uint32  * supportedVersion );
```

db2VendorQueryApiVersion 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・ストレージ API のバージョンを戻します。

使用上の注意

この API は、他のベンダー・ストレージ API の呼び出し前に呼び出されます。

sqluvdel - コミット済みセッションの削除

ベンダー装置からコミット済みセッションを削除します。

許可

なし

必要な接続

データベース

API インクルード・ファイル

sqluvend.h

API とデータ構造構文

```
int sqluvdel ( struct Init_input *in,
               struct Init_output *vendorDevData,
               struct Return_code *return_code);
```

sqluvdel API パラメーター

in　入力。Init_input および Return_code 用に割り振られたスペース。

vendorDevData
出力。ベンダー装置が戻した出力を含む構造。

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 API 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 API 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 15. sqluvdel についての有効な戻りコードと結果のデータベース・サーバー・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
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。

sqluvend - ベンダー装置のリンク解除およびリソースの解放

ベンダー装置のリンクを解除し、関連したリソースをすべて解放します。 sqluvend API 呼び出しが DB2 へ戻る前に、使用していないリソース (例えば、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

なし

必要な接続

データベース

API インクルード・ファイル

```
sqluvend.h
```

API とデータ構造構文

```
int sqluvend ( sqlint32             action,
               void *hdle,
               struct Init_output *in_out,
               struct Return_code *return_code);
```

sqluvend 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)

hdle 入力。Init_output 構造を指すポインター。

in_out 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 API 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。

Commit
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。

Abort 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 16. 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。	

sqluvget - ベンダー装置からのデータの読み取り

ベンダー装置の sqluvint API による初期化後、DB2 は、リストア操作の間、この API を呼び出して装置からデータを読み取ります。

許可

なし

必要な接続

データベース

API インクルード・ファイル

```
sqluvend.h
```

API とデータ構造構文

```
int sqluvget ( void * hdle,
               struct Data *data,
               struct Return_code *return_code);
```

sqluvget API パラメーター

hdle 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。

データ 入力または出力。Data 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。

使用上の注意

この API 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 17. 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 (後の注を参照)	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 (後の注を参照)	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 (後の注を参照)	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 (後の注を参照)	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 (後の注を参照)	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 あるいは 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 (後の注を参照)	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できる状態である。	sqluvget	
SQLUV_ENDOFMEDIA_NO_ DATA	メディアが終了し、0 バイトが読み取られた (例えば、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 (例えば、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 (後の注を参照)	セッションは終了します。

注: 次の呼び出し: 次の呼び出しが sqluvend、アクション = 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アクティブ・セッションは終了します。

sqluvint - ベンダー・デバイスの初期化、ベンダー・デバイスへのリンク

ベンダー・デバイスの初期化、DB2 とベンダー・デバイスの間の論理リンク確立についての情報を提供します。

許可

なし

必要な接続

データベース

API インクルード・ファイル

```
sqluvend.h
```

API とデータ構造構文

```
int sqluvint ( struct Init_input *in,
               struct Init_output *out,
               struct Return_code *return_code);
```

sqluvint API パラメーター

in 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構造。

out 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 API 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・ストレージ API 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 API 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、これらの各コードに対する DB2 の反応については、戻りコード表 (後に示す表を参照) 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 (例えば、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 API を呼び出して継続しますが、 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 18. 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 API (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 API (データのリストア) になります。
SQLUV_LINK_EXIST	セッションが既に活動化されている。	ありません。	セッションの初期化に失敗しました。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 API 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。	セッションの初期化に失敗しました。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 API 呼び出しは受信されません。

表 18. 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_INV_VERSION	DB2 とベンダー製品に互換性がない。	ありません。	セッションの初期化に失敗しました。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 API 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。	セッションの初期化に失敗しました。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 API 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。	セッションの初期化に失敗しました。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 API 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。	セッションの初期化に失敗しました。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 API 呼び出しは受信されません。
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。	セッションの初期化に失敗しました。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 API 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。	セッションの初期化に失敗しました。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 API 呼び出しは受信されません。

表 18. 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_INV_PASSWORD	提供されたパスワードが無効である。	ありません。	セッションの初期化に失敗しました。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 API 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。	セッションの初期化に失敗しました。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 API 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。	セッションの初期化に失敗しました。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 API 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。	セッションの初期化に失敗しました。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 API 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 API 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 API (データのリストア) になります。

表 18. 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_IO_ERROR	入出力エラー	ありません。	セッションの初期化に失敗しました。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 API 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。	セッションの初期化に失敗しました。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 API 呼び出しは受信されません。

sqluvput - ベンダー装置へのデータの書き込み

ベンダー装置の sqluvint API による初期化後、DB2 は、バックアップ操作の間、この API を呼び出して装置にデータを書き込みます。

許可

なし

必要な接続

データベース

API インクルード・ファイル

```
sqluvend.h
```

API とデータ構造構文

```
int sqluvput ( void *hdle,
               struct Data *data,
               struct Return_code *return_code);
```

sqluvput API パラメーター

hdle 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。

データ 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。

使用上の注意

この API 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 19. 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 (例えば、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 (後の注を参照)。	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 (後の注を参照)。	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 (後の注を参照)。	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 (後の注を参照)。	セッションは終了します。
SQLUV_ENDOFMEDIA	メディアが終了した (例えば、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送します。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 (後の注を参照)。	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 (後の注を参照)。	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 (後の注を参照)。	セッションは終了します。

注: 次の呼び出し: 次の呼び出しが sqluvend、アクション = 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アクティブ・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。

DB2_info

DB2 製品とバックアップ、リストア対象のデータベースに関する情報が含まれます。この構造は、ベンダー装置に DB2 に関する情報を伝えるため、また、DB2 とベンダー装置の間の特定のセッションについての情報を伝えるために使用されます。これは Init_input データ構造に含まれるかたちで、バックアップおよびリストア用のベンダー・ストレージ・プラグインに渡されます。

表 20. DB2_info 構造のフィールド

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固有の名前。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。

表 20. *DB2_info* 構造のフィールド (続き)

フィールド名	データ・タイプ	説明
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。次の値が設定可能です。アクションが SQLUV_WRITE の場合: 0 - フル・データベース・バックアップ。 3 - 表スペース・レベル・バックアップ。アクションが SQLUV_READ の場合: 0 - フル・リストア。 3 - オンライン表スペース・リストア。 4 - 表スペース・リストア。 5 - ヒストリー・ファイル・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。
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 0 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。

表 20. *DB2_info 構造のフィールド (続き)*

フィールド名	データ・タイプ	説明
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。

注: char データ・タイプ・フィールドはすべて、ヌル終了ストリングです。

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。

API とデータ構造構文

```
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;
  sqlint32 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

ベンダー装置がどのベンダーのものか、どのバージョンのものか、といった情報が収められ、Init_output 構造の中に入れられて DB2 に戻されます。

表 21. Vendor_info 構造のフィールド

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固有の名前。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 API 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。

注: char データ・タイプ・フィールドはすべて、ヌル終了ストリングです。

API とデータ構造構文

```
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

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。このデータ構造は、DB2 が、sqluvint、sqluvdel API を利用してバックアップおよびリストア・ベンダー・ストレージ・プラグインに情報を送信するのに使用されます。

表 22. Init_input 構造のフィールド

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。

表 22. *Init_input* 構造のフィールド (続き)

フィールド名	データ・タイプ	説明
オプション	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。このフィールドはヌル終了ストリングです。
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。

API とデータ構造構文

```
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

バックアップおよびリストア・ベンダー・ストレージ・プラグインによって DB2 に戻される、セッションおよび情報に対応する制御ブロックが含まれます。このデータ構造は、sqluvint、sqluvdel API によって使用されます。

表 23. *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。

API とデータ構造構文

```
typedef struct Init_output
{
   struct Vendor_info * vendor_session;
   void               * pVendorCB;
   void               * reserve;
} Init_output ;
```

Data

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。この構造は、sqluvput API によってデータがベンダー装置に書き込まれる際と、sqluvget API によってベンダー装置からデータが読み取られる際に使用されます。

表 24. *データ構造*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_buff_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。

API とデータ構造構文

```
typedef struct Data
{
   sqlint32 obj_num;
   sqlint32 buff_size;
   sqlint32 actual_buff_size;
   void *dataptr;
   void *reserve;
} Data;
```

Return_code

バックアップ、リストア・ベンダー・ストレージ・プラグインによって DB2 に戻されるエラーに対応する戻りコードと、エラーについての簡単な説明が含まれます。このデータ構造は、すべてのベンダー・ストレージ・プラグイン API によって使用されます。

表 25. 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 (下記の注を参照)	sqlint32	ベンダー API からの戻りコード。
説明	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。

注: これはベンダー固有の戻りコードで、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。

API とデータ構造構文

```
typedef struct Return_code
{
        sqlint32 return_code;
        char description[SQLUV_COMMENT_LEN];
        void *reserve;
} Return_code;
```

バックアップとリストアの操作に伴う圧縮に使用する DB2 API

DB2 には、サード・パーティーの圧縮製品によるバックアップ・イメージの圧縮、解凍に使用できる API が用意されています。このインターフェースは、DB2 の標準部分としてサポートされている圧縮ライブラリーに機能を追加するもの、あるいは代替となるものとして設計されています。圧縮プラグイン・インターフェースは、バックアップおよびリストア DB2 API、あるいはベンダー・ストレージ装置向けバックアップおよびリストア・プラグインとともに使用できます。

DB2 では、多くのベンダーが使用できる圧縮、解凍向け汎用インターフェースを提供する API プロトタイプのセットが定義されています。これらの API は、共用ライブラリー (Linux および UNIX 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。 API が DB2 によって呼び出されると、バックアップ、またはリストア・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 API が呼び出されます。

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、8 つの API が定義されています。

- InitCompression - 圧縮ライブラリーの初期設定

- GetSavedBlock - バックアップ・イメージのためのベンダー・ブロックの取得
- Compress - データ・ブロックの圧縮
- GetMaxCompressedSize - 最大限可能なバッファー・サイズの見積もり
- TermCompression - 圧縮ライブラリーの終了
- InitDecompression - 解凍ライブラリーの初期設定
- Decompress - データ・ブロックの解凍
- TermDecompression - 解凍ライブラリーの終了

DB2 には、COMPR_DB2INFO 構造の定義が用意されています。ベンダーは、バックアップとリストアに伴う圧縮に使用する、他の構造と API のそれぞれの定義を用意します。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 がこれらの API を呼び出したら、これらの API は共用ライブラリー (Linux および UNIX 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整合性を危険にさらす可能性があります。

呼び出しシーケンスの例

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。

```
InitCompression
```

続いて、0 個から 1 個の

```
GetMaxCompressedSize
圧縮
```

続いて、1 個の

```
TermCompress
```

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。

```
InitDecompression
```

続いて、1 個から n 個の

```
Decompress
```

続いて、1 個の

```
TermCompression
```

圧縮プラグイン・インターフェース戻りコード

以下は、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。

COMPR_CB

この構造は、プラグイン・ライブラリーによって、制御ブロックとして内部で使用されます。ここには、圧縮および解凍 API 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、構造のパラメーターの定義や構造のメモリー管理を含め、ライブラリーによって決められます。

API とデータ構造構文

```
struct COMPR_CB;
```

COMPR_DB2INFO

DB2 環境について記述します。 DB2 は、この構造を割り振り、定義して、パラメーターとして InitCompression および InitDecompression API に渡します。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。バックアップ・イメージの特定には、dbalias、instance、node、catnode、timestamp といったパラメーターが使用されます。

API とデータ構造構文

```
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 データ構造パラメーター

tag 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。現在のバージョンは 1 です。将来は、この構造にさらにパラメーター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
データベース別名。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。

instance
インスタンス名。

node ノード番号。

catnode
カタログ・ノード番号。

timestamp
バックアップまたはリストアするイメージのタイム・スタンプ。

bufferSize
転送バッファーのサイズを指定します (4K ページ単位)。

options
db2Backup API または db2Restore API で指定される iOptions パラメーター。

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 パラメーターを指定します。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、(インクルード・ディレクトリーに置かれる) 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 パラメーターの値を指定します。

savedBlockSize
savedBlock のサイズ (単位: バイト)。

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。

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 API 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。

API とデータ構造構文

```
struct COMPR_PIINFO {
   char tag[16];
   db2Uint32 version;
   db2Uint32 size;
   db2Uint32 useCRC;
```

```
   db2Uint32 useGran;
   db2Uint32 useAllBlocks;
   db2Uint32 savedBlockSize;
};
```

COMPR_PIINFO データ構造パラメーター

tag 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。現在のバージョンは 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整合性を検査できます。

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の増分で圧縮できる場合、ライブラリーは、このフィールドを 1 に設定します。圧縮ルーチンがバイト単位の増分でのみデータの圧縮を行う場合、ライブラリーは、このフィールドを 0 に設定します。このインディケーターの設定の詳細な意味については、Compress API の srcGran パラメーターについての記述を参照してください。

リストア操作の場合、このパラメーターは無視されます。

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、すべてのブロックについて圧縮済みバージョンのデータを保管する場合、ライブラリーはこの値を 1 に設定します。元のデータよりも小さい場合にだけ、DB2 が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、このパラメーターを、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、このパラメーターをゼロに設定します。リストア操作の場合、このパラメーターは無視されます。

Compress - データ・ブロックの圧縮

データ・ブロックを圧縮します。 src パラメーターは、サイズが srcLen バイトのデータ・ブロックを指します。 tgt パラメーターは、サイズが tgtSize バイトのバッファーを指します。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。

許可

なし

必要な接続

なし

API インクルード・ファイル

```
sqlucompr.h
```

API とデータ構造構文

```
int Compress(
      struct COMPR_CB *pCB,
      const char *src,
      db2int32 srcSize,
      db2Uint32 srcGran,
      char *tgt,
      db2int32 tgtSize,
      db2int32 *srcAct,
      db2int32 *tgtAct,
      db2Uint32 *tgtCRC);
```

Compress API パラメーター

pCB 入力。これは、InitCompression API 呼び出しによって戻された制御ブロックです。

src 入力。圧縮されるデータ・ブロックを指すポインター。

srcLen 入力。圧縮されるデータ・ブロックのサイズ (単位: バイト)。

srcGran
入力。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。 (例えば、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 piInfo->srcGran に 0 の値を戻した場合、細分度は 1 バイトです。

tgt 入出力。圧縮データのためのターゲット・バッファー。DB2 は、このターゲット・バッファーを提供し、プラグインは、src のデータを圧縮して、圧縮済みのデータをここに書き込みます。

tgtSize 入力。ターゲット・バッファーのサイズ (単位: バイト)。

srcAct 出力。圧縮された src の、圧縮しない状態での実際のデータの量 (単位: バイト)。

tgtAct 出力。 tgt に格納された圧縮データの実際の大きさ (単位: バイト)。

tgtCRC
出力。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。

Decompress - データ・ブロックの解凍

データ・ブロックを解凍します。 src パラメーターは、サイズが srcLen バイトのデータ・ブロックを指します。 tgt パラメーターは、サイズが tgtSize バイトのバッファーを指します。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。

許可

なし

必要な接続

なし

API インクルード・ファイル

```
sqlucompr.h
```

API とデータ構造構文

```
int Decompress(
      struct COMPR_CB *pCB,
      const char *src,
      db2int32 srcSize,
      char *tgt,
      db2int32 tgtSize,
      db2int32 *tgtLen,
      db2Uint32 *tgtCRC);
```

Decompress API パラメーター

pCB 入力。これは、InitDecompression API 呼び出しによって戻された制御ブロックです。

src 入力。解凍されるデータ・ブロックを指すポインター。

srcLen 入力。解凍されるデータ・ブロックのサイズ (単位: バイト)。

tgt 入出力。解凍データのためのターゲット・バッファー。DB2 は、このター

ゲット・バッファーを提供し、プラグインは、src のデータを解凍して、解凍済みのデータをここに書き込みます。

tgtSize 入力。ターゲット・バッファーのサイズ (単位: バイト)。

tgtLen 出力。 tgt に格納された解凍データの実際の大きさ (単位: バイト)。

tgtCRC
出力。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。

GetMaxCompressedSize - 最大限可能なバッファー・サイズの見積もり

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。 DB2 は、戻り値に関係なくバックアップを正しく作成します。

許可

なし

必要な接続

なし

API インクルード・ファイル

```
sqlucompr.h
```

API とデータ構造構文

```
int GetMaxCompressedSize(
      struct COMPR_CB *pCB,
      db2Uint32 srcLen);
```

GetMaxCompressedSize API パラメーター

pCB 入力。これは、InitCompression API 呼び出しによって戻された制御ブロックです。

srcLen 入力。圧縮されるデータ・ブロックのサイズ (単位: バイト)。

GetSavedBlock - バックアップ・イメージのデータ・ブロックのベンダーの取得

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 API は、バックアップの場合にのみ、最初の db2bm プロセスによって初期データ処理中に呼び出されます。 db2Backup API で 1 より大きい並列処理が指定される場合でも、この API は、バックアップごとに一回だけ呼び出されます。

許可

なし

必要な接続

なし

API インクルード・ファイル

```
sqlucompr.h
```

API とデータ構造構文

```
int GetSavedBlock(
      struct COMPR_CB *pCB,
      db2Uint32 blockSize,
      void *data);
```

GetSavedBlock API パラメーター

pCB 入力。これは、InitCompression API 呼び出しによって戻された制御ブロックです。

blocksize
入力。これは、InitCompression API 呼び出しにより、piInfo->savedBlockSize で戻されたブロックのサイズです。

データ 出力。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・ブロックです。

InitCompression - 圧縮ライブラリーの初期設定

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 の適切なパラメーターを記入し、pCB を割り当てて、割り振られたメモリーを指すポインターを戻します。

許可

なし

必要な接続

なし

API インクルード・ファイル

```
sqlucompr.h
```

API とデータ構造構文

```
int InitCompression(
      const struct COMPR_DB2INFO
                        *db2Info,
      struct COMPR_PIINFO
                        *piInfo,
      struct COMPR_CB  **pCB);
```

InitCompression API パラメーター

db2Info
入力。バックアップされるデータベースについて記述し、操作が行われる DB2 環境について詳細を示します。

piInfo 出力。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。 DB2 によって割り振られて初期設定され、主なパラメーターは、プラグイン・ライブラリーによって入力されます。

pCB 出力。これは、圧縮ライブラリーによって使用される制御ブロックです。プラグイン・ライブラリーは、構造のメモリー管理を行います。

InitDecompression - 解凍ライブラリーの初期設定

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を指すポインターを戻します。

許可

なし

必要な接続

なし

API インクルード・ファイル

```
sqlucompr.h
```

API とデータ構造構文

```
int InitDecompression(
      const struct COMPR_DB2INFO
                        *db2Info,
      struct COMPR_CB  **pCB);
```

InitDecompression API パラメーター

db2Info
入力。バックアップされるデータベースについて記述し、操作が行われる DB2 環境について詳細を示します。

pCB 出力。これは、解凍ライブラリーによって使用される制御ブロックです。プラグイン・ライブラリーは、構造のメモリー管理を行います。

TermCompression - 圧縮ライブラリーの停止

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。

許可

なし

必要な接続

なし

API インクルード・ファイル

```
sqlucompr.h
```

API とデータ構造構文

```
int TermCompression(
      struct COMPR_CB *pCB);
```

TermCompression API パラメーター

pCB 入力。これは、InitCompression API 呼び出しによって戻された制御ブロックです。

TermDecompression - 解凍ライブラリーの停止

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。圧縮 API によって内部で使用されるメモリーはすべて、ライブラリ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も管理します。 DB2 は、データ・バッファー (圧縮 API の src および tgt パラメーター) に使用するメモリーを管理します。

許可

なし

必要な接続

なし

API インクルード・ファイル

```
sqlucompr.h
```

API とデータ構造構文

```
int TermDecompression(
      struct COMPR_CB *pCB);
```

TermDecompression API パラメーター

pCB 入力。これは、InitDecompression API 呼び出しによって戻された制御ブロックです。

第 9 章 API で使用されるデータ構造

db2HistoryData

この構造は、db2HistoryGetEntry API への呼び出し後に情報を戻すために使用されます。

表 26. db2HistoryData 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、*yyyymmddhhmmss* 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 *yyyymmddhhmmss* の形式のタイム・スタンプ。
oFirstLog	db2Char	最初期のログ・ファイルの ID (範囲は S0000000 から S9999999 まで): • オンライン・バックアップにロールフォワード・リカバリーを適用する際に必要。 • オフライン・バックアップにロールフォワード・リカバリーを適用する際に必要。 • ロード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	直近のログ・ファイルの ID (範囲は S0000000 から S9999999 まで): • オンライン・バックアップにロールフォワード・リカバリーを適用する際に必要。 • オフライン・バックアップで現時点へのロールフォワード・リカバリーを適用する際に必要。 • ロード操作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・リカバリーを適用しない場合には、**oFirstLog** と同じになる)。

表 26. *db2HistoryData* 構造のフィールド (続き)

フィールド名	データ・タイプ	説明
oID	db2Char	固有のバックアップまたは表 ID。
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** パラメーター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** パラメーターの値によって解釈が変わります。 • ディスクあるいはディスケット (D または K) の場合、完全修飾ファイル名。 • 磁気テープ (T) の場合、ボリューム・ラベル。 • TSM (A および F) の場合、バックアップを実行したベンダー・ライブラリー名/パス。 • ユーザー出口その他 (U または O) の場合、フリー・フォーム・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	固有の項目 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。
iNumTablespaces	db2Uint32	db2HistoryGetEntry API によって使用できる **poTablespace** リスト中の項目の数。
oNumTablespaces	db2Uint32	db2HistoryGetEntry API によって使用された、**poTablespace** リスト中の項目の数。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	539 ページの表 27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)、T (表)、R (範囲パーティション)、および I (索引)。
oOptype	char	539 ページの表 28を参照してください。

表 26. db2HistoryData 構造のフィールド (続き)

フィールド名	データ・タイプ	説明
oStatus	char	項目の状況: A = アクティブ; I = 非アクティブ; E = 有効期限切れ; D = 削除; X = 削除対象外 (do not delete)。
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)、F (スナップショット・バックアップ)、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 27. db2HistoryData 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	load	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
X	ログのアーカイブ	DB2HISTORY_OP_ARCHIVE_LOG	DB2HIST_OP_ARCHIVE_LOG

表 28. db2HistData 構造の有効な **oOptype** *値*

oOperation	oOptype	説明	C/COBOL/FORTRAN 定義
B	F N I O D E	オフライン、オンライン、増分オフライン、増分オンライン、デルタ・オフライン、デルタ・オンライン	DB2HISTORY_OPTYPE_OFFLINE、DB2HISTORY_OPTYPE_ONLINE、DB2HISTORY_OPTYPE_INCR_OFFLINE、DB2HISTORY_OPTYPE_INCR_ONLINE、DB2HISTORY_OPTYPE_DELTA_OFFLINE、DB2HISTORY_OPTYPE_DELTA_ONLINE
F	E P	ログの終わり、ポイント・イン・タイム	DB2HISTORY_OPTYPE_EOL、DB2HISTORY_OPTYPE_PIT

表 28. *db2HistData* 構造の有効な **oOptype** 値 (続き)

oOperation	oOptype	説明	C/COBOL/FORTRAN 定義
G	F N	オフライン、オンライン	DB2HISTORY_OPTYPE_OFFLINE、DB2HISTORY_OPTYPE_ONLINE
L	I R	挿入、置換	DB2HISTORY_OPTYPE_INSERT、DB2HISTORY_OPTYPE_REPLACE
Q	S U X Z	静止共用、静止更新、静止排他、静止リセット	DB2HISTORY_OPTYPE_SHARE、DB2HISTORY_OPTYPE_UPDATE、DB2HISTORY_OPTYPE_EXCL、DB2HISTORY_OPTYPE_RESET
R	F N I O R	オフライン、オンライン、増分オフライン、増分オンライン、再作成	DB2HISTORY_OPTYPE_OFFLINE、DB2HISTORY_OPTYPE_ONLINE、DB2HISTORY_OPTYPE_INCR_OFFLINE、DB2HISTORY_OPTYPE_INCR_ONLINE、DB2HISTORY_OPTYPE_REBUILD
T	C R	コンテナーの追加、リバランス	DB2HISTORY_OPTYPE_ADD_CONT、DB2HISTORY_OPTYPE_REB
X	N P M F 1 2	アーカイブ・ログ・コマンド、1 次ログ・パス、ミラー・ログ・パス、アーカイブ障害パス、ログ・アーカイブ方式 1、ログ・アーカイブ方式 2	DB2HISTORY_OPTYPE_ARCHIVE_CMD、DB2HISTORY_OPTYPE_PRIMARY、DB2HISTORY_OPTYPE_MIRROR、DB2HISTORY_OPTYPE_ARCHFAIL、DB2HISTORY_OPTYPE_ARCH1、DB2HISTORY_OPTYPE_ARCH2

表 29. *db2HistoryEID* 構造のフィールド

フィールド名	データ・タイプ	説明
ioNode ioHID	SQL_PDB_NODE_TYPE db2Uint32	ノード番号。ローカル履歴ファイルの項目 ID。

API とデータ構造構文

```
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;
   struct db2Char *poTablespace;
   db2Uint32 iNumTablespaces;
   db2Uint32 oNumTablespaces;
   char oOperation;
   char oObject;
```

```
   char oOptype;
   char oStatus;
   char oDeviceType;
} db2HistoryData;

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

typedef SQL_STRUCTURE db2HistoryEID
{
   SQL_PDB_NODE_TYPE ioNode;
   db2Uint32 ioHID;
} db2HistoryEID;
```

db2Char データ構造パラメーター

pioData
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
入力。**pioData** バッファーのサイズ。

oLength
出力。 **pioData** バッファー内にあるデータの有効文字の数。

db2HistoryEID データ構造パラメーター

ioNode 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。ノード番号を示します。

ioHID 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。ローカル履歴ファイルの項目 ID を示します。

sql_authorizations

注: この構造は使用されなくなり、sqluadau() API の後方互換性のためにのみ使用されます。

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 30. 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。

表 30. SQL_AUTHORIZATIONS 構造のフィールド (続き)

フィールド名	説明
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。
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。
SQL_CREATE_EXT_RTN_AUTH	CREATE_EXTERNAL_ROUTINE 権限。
SQL_CREATE_EXT_RTN_GRP_AUTH	CREATE_EXTERNAL_ROUTINE 権限を保持するグループに所属するユーザー。
SQL_QUIESCE_CONNECT_AUTH	QUIESCE CONNECT 権限。
SQL_QUIESCE_CONNECT_GRP_AUTH	QUIESCE CONNECT 権限を保持するグループに所属するユーザー。
SQL_SECURITY_ADMIN_AUTH	SECADM 権限。
SQL_SECURITY_ADMIN_GRP_AUTH	SECADM 権限を保持するグループに所属するユーザー。
SQL_SYSMON_AUTH	SYSMON 権限。
SQL_SYSMON_GRP_AUTH	SYSMON 権限を保持するグループに所属するユーザー。

注: SYSADM、SYSMAINT、SYSMON、および SYSCTRL は間接権限にすぎず、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できます。

API とデータ構造構文

```
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;
   short sql_create_ext_rtn_auth;
   short sql_create_ext_rtn_grp_auth;
   short sql_quiesce_connect_auth;
   short sql_quiesce_connect_grp_auth;
   short sql_security_admin_auth;
   short sql_security_admin_grp_auth;
   short sql_library_admin_auth;
   short sql_library_admin_grp_auth;
   short sql_sysmon_auth;
   short sql_sysmon_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.
*
```

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 31. 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 RELEASE CODEPAGE COMMENT LDB TDB AR PARM	SMALLINT SMALLINT SMALLINT CHAR(30) CHAR(8) CHAR(18) CHAR(32) CHAR(512)	構造 ID。 SQL_DCS_STR_ID (sqlenv で定義) に設定してください。(API が割り当てた) リリース・バージョン。コメント用コード・ページ。データベースのオプション・コメント。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。データベースの実名。アプリケーション・クライアントの名前。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。

API とデータ構造構文

```
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).
*
```

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 32. 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。

注: この値は減少することがあるので、実際の「最高水準点」ではなく「現在の水準点」です。 SMS には適用されません。

表スペースのバランス調整中、使用できるページの数には、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、バランス調整が完了するまではフリー・ページの数に反映されません。表スペースのバランス調整が実行されていないときは、使用中のページ数とフリー・ページの数の合計が使用できるページの数になります。

API とデータ構造構文

```
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 33. 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	常に 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)。

API とデータ構造構文

```
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 34. 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。
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)。

表 34. 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
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。

表 34. 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
QUIESCER	SQLB_QUIESCER_DATA 構造の配列	配列の各項目は静止オブジェクトの静止データから構成されます。
FSCACHING	UNSIGNED CHAR	直接 I/O をサポートするファイル・システム・キャッシュ・ポリシー。これは 31 ビット・フィールドです。
RESERVED	CHAR(31)	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
すべてのタイプの永続データ。 REGULAR 表スペース。

SQLB_TBS_LONG
すべてのタイプの永続データ。 LARGE 表スペース。

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
使用不可ペンディング

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
ストレージを定義する必要があります

SQLB_RESTORE_IN_PROGRESS
リストア進行中

SQLB_STORDEF_ALLOWED
ストレージを定義可能

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。

API とデータ構造構文

```
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];
   unsigned char   fsCaching;
   char reserved[31];
};

SQL_STRUCTURE SQLB_QUIESCER_DATA
{
   sqluint32 quiesceId;
   sqluint32 quiesceObject;
};
```

SQLB_QUIESCER_DATA データ構造パラメーター

pad 予約済み。構造体の配置のために使用される。ユーザー・データは設定しない。

pad1 予約済み。構造体の配置のために使用される。ユーザー・データは設定しない。

quiesceId
入力。静止オブジェクトが作成された表スペースの ID。

quiesceObject
入力。静止オブジェクトのオブジェクト ID。

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-FSCACHING          PIC X(1).
    05 SQL-RESERVED           PIC X(31).
*
```

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).
*
```

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 35. 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。

API とデータ構造構文

```
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、COBOL で構造を定義する方法のわかる例を示します。

```
* 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.
*
```

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 36. 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。

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 (例えば、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する可能性があります。

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。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。

表 37. 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 38. 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 39. 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。

表 39. SQLLOCPAIR 構造のフィールド (続き)

フィールド名	データ・タイプ	説明
END_LOC	SMALLINT	外部ファイル内の列データの終了位置。

API とデータ構造構文

```
SQL_STRUCTURE sqldcol
{
   short dcolmeth;
   short dcolnum;
   struct sqldcoln dcolname[1];
};

SQL_STRUCTURE sqldcoln
{
   short dcolnlen;
   char *dcolnptr;
};

SQL_STRUCTURE sqlloctab
{
   struct sqllocpair locpair[1];
};

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.
*
```

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 40. SQLE-ADDN-OPTIONS 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。

表 40. SQLE-ADDN-OPTIONS 構造のフィールド (続き)

フィールド名	データ・タイプ	説明
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。注: このオプションは、自動ストレージを使用するよう定義された SYSTEM TEMPORARY 表スペース (CREATE TABLESPACE ステートメントの MANAGED BY AUTOMATIC STORAGE 節によって作成されたか、あるいは、MANAGED BY 節がまったく指定されなかった SYSTEM TEMPORARY 表スペース) については無視されます。この種の表スペースに関しては、コンテナー作成を先に延ばすこと、あるいは、他のパーティションに定義されるようなコンテナーのセットの作成を選択することはできません。コンテナーは、データベース・マネージャーにより、データベースに関連するストレージ・パスを基に自動的に割り当てられます。
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
SYSTEM TEMPORARY 表スペースを作成しません。

SQLE_TABLESPACES_LIKE_NODE
SYSTEM 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
SYSTEM 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。

API とデータ構造構文

```
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).
*
```

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クライアント・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 41. 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 42. 接続設定

Type	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。例えば、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。例えば、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。例えば、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_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 Universal Database 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。例えば、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。

API とデータ構造構文

```
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.
*
```

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 43. 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 VALUE	SMALLINT SMALLINT	設定のタイプ。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 44. 接続設定

Type	値	説明
SQL_CONNECT_TYPE	SQL_CONNECT_1 SQL_CONNECT_2	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。 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 SQL_RULES_STD	現行接続を、確立済みの (休止) 接続に切り替える SQL CONNECT ステートメントを有効にします。 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 44. 接続設定 (続き)

Type	値	説明
SQL_DISCONNECT	SQL_DISCONNECT_EXPL SQL_DISCONNECT_COND SQL_DISCONNECT_AUTO	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。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。コミット時にすべての接続を切断します。
SQL_DEFERRED_PREPARE	SQL_DEFERRED_PREPARE_NO SQL_DEFERRED_PREPARE_YES SQL_DEFERRED_PREPARE_ALL	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。 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 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。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 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。

表 44. 接続設定 (続き)

Type	値	説明
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。例えば、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。例えば、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。

API とデータ構造構文

```
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.
*
```

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 45. 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。

API とデータ構造構文

```
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.
*
```

sqle_node_npipe

この構造は、sqlectnd API の Named PIPE ノードをカタログするために使用されます。

表 46. 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。

API とデータ構造構文

```
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.
*
```

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。

注: NetBIOS は、現在サポートされていません。 SNA、およびその API の APPC、APPN、および CPI-C も、サポートされなくなりました。それらのプロトコルを使用している場合は、TCP/IP などのサポートされているプロトコルを使用してノードとデータベースを再カタログしてください。それらのプロトコルに言及している箇所がある場合、それらは無視してください。

表 47. 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 の有効な値 (sqlenv で定義) は、以下のとおりです。

- SQL_PROTOCOL_APPC
- SQL_PROTOCOL_APPN
- SQL_PROTOCOL_CPIC
- SQL_PROTOCOL_LOCAL
- SQL_PROTOCOL_NETB
- SQL_PROTOCOL_NPIPE
- SQL_PROTOCOL_SOCKS
- SQL_PROTOCOL_TCPIP

API とデータ構造構文

```
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).
*
```

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、TCP/IPv4、TCP/IPv6 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 構造で、それぞれ SQL_PROTOCOL_TCPIP、SQL_PROTOCOL_TCPIP4、SQL_PROTOCOL_TCPIP6 に設定してください。TCP/IP または TCP/IPv4 の 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 構造で、それぞれ SQL_PROTOCOL_SOCKS、SQL_PROTOCOL_SOCKS4 に設定してください。SOCKS は IPv6 ではサポートされません。例えば、IPv6 アドレスとともに SQL_PROTOCOL_SOCKS を使用することはできません。

表 48. 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ホスト名または IP アドレス。受け入れられる IP アドレスのタイプは、選択されるプロトコルによって変わります。
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。

API とデータ構造構文

```
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.
*
```

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 49. 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、バージョン管理を目的に妥当性が検査されます。
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。

表 49. *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
SQLDBUDC	CHAR(256)	SQLDBCSS を SQL_CS_USER に設定すると、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入ります。 SQLDBCSS を SQL_CS_UNICODE に設定すると、このフィールドには言語認識またはロケールを区別する UCA ベースの照合名 (長さ最大 128 バイトまでの NULL 終了ストリング) が含まれます。 SQLDBCSS が SQL_CS_USER または SQL_CS_UNICODE と異なる場合、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。

表 49. 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
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。
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 50. 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、バージョン管理を目的に妥当性が検査されます。
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は dft_extent_sz 構成パラメーターの現行値になります。

表 50. SQLETSDESC 構造のフィールド (続き)

フィールド名	データ・タイプ	説明
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は dft_prefetch_sz 構成パラメーターの現行値になります。
SQLFSCACHING	UNSIGNED CHAR	ファイル・システム・キャッシュ。 1 の値を指定すると、ファイル・システム・キャッシュは現在の表スペースに対してオフになります。 0 の値を指定すると、ファイル・システム・キャッシュは現在の表スペースに対してオンになります。デフォルト設定にする場合には、2 を指定します。この場合、AIX、 Linux、 Solaris、 Windows(AIX JFS を除く)、 Linux on System z®、SMS TEMPORARY 表スペース・ファイル用および SMS ラージ・オブジェクト・ファイルまたはラージ・ファイル用の VxFS 以外の Solaris では、ファイル・システム・キャッシングはオフになります。その他のすべてプラットフォームでは、ファイル・システム・キャッシングはオンになります。
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、デフォルトでは、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、デフォルトでは、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。

表 50. SQLETSDESC 構造のフィールド (続き)

フィールド名	データ・タイプ	説明
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 51. 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。
SQLCONTR	CHAR(256)	コンテナー・ストリング。

SQLDBCSS の有効な値 (sqlenv で定義) は、以下のとおりです。

SQL_CS_SYSTEM
Unicode 以外のデータベースの場合、これはデフォルト・オプションであり、その照合シーケンスは、データベース・テリトリーをベースにします。 Unicode データベースの場合、このオプションは IDENTITY オプションと同等です。 NULL ポインターを渡すと、オペレーティング・システムの照合シーケンス (現行のロケール・コードとコード・ページに基づいて) が使用されます。これは、SQL_CS_SYSTEM (0) と等しい SQLDBCSS を指定するのと同じです。

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
ストリングがバイト単位で比較される一致照合順序。 Unicode データベースの場合は、これがデフォルトです。

SQL_CS_COMPATABILITY
前のバージョン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 4.0.0 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 の詳細は、Unicode Technical Standard #10 にあります。この仕様は Unicode Consortium Web サイト (www.unicode.org) で入手できます。 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_LSK
Unicode Standard バージョン 4.0.0 に基づく UCA (Unicode Collation Algorithm) 照合順序。ただし、スロバキア語文字はすべて適切な順にソートされます。 UCA の詳細は、Unicode Technical Standard #10 にあります。この仕様は Unicode Consortium Web サイト (www.unicode.org) で入手できます。 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 4.0.0 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 の詳細は、Unicode Technical Standard #10 にあります。この仕様は Unicode Consortium Web サイト (www.unicode.org) で入手できます。 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_UNICODE

Unicode データベースの場合、照合シーケンスは言語ベースです。SQLDBUDC フィールドに特定の照合名を指定しますが、照合名は 0x00 バイトで終了させる必要があります。この照合名は、『Unicode データ用の言語認識照合』で定義されている言語認識照合、または『Unicode 照合アルゴリズムに基づく照合』に示されているロケールを区別する UCA ベースの照合を識別できます。

例えば、コード・ページ 819 にある US English に相当する照合を使用するには、SQLDBCSS を SQL_CS_UNICODE に設定し、 SQLDBUDC を SYSTEM_819_US に設定します。

注: CREATE DATABASE をバージョン 9.5 より前のサーバーに実行する場合には、このオプションは使用できません。デフォルトでは、そうしたサーバー上の Unicode データベースは SYSTEM 照合を使用して作成されます。

SQLTSTYP の有効な値 (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 の有効な値 (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。

API とデータ構造構文

```
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;
};

SQL_STRUCTURE SQLETSDESC
{
        char sqltsdid[8];
        sqlint32 sqlextnt;
        sqlint32 sqlprftc;
        double sqlpovhd;
        double sqltrfrt;
        char sqltstyp;
        unsigned char  sqlfscaching;
        short sqlccnt;
        struct SQLETSCDESC containr[1];
};

SQL_STRUCTURE SQLETSCDESC
{
        char sqlctype;
        char pad1[3];
        sqlint32 sqlcsize;
```

```
        short sqlclen;
        char sqlcontr[SQLB_MAX_CONTAIN_NAME_SZ];
        char pad2[2];
};
```

sqledbdesc 構造パラメーター

pad1 予約済み。構造体の配置のために使用しますが、これにはユーザー・データを取り込んではなりません。

pad2 予約済み。構造体の配置のために使用しますが、これにはユーザー・データを取り込んではなりません。

SQLETSCDESC 構造パラメーター

pad1 予約済み。構造体の配置のために使用しますが、これにはユーザー・データを取り込んではなりません。

pad2 予約済み。構造体の配置のために使用しますが、これにはユーザー・データを取り込んではなりません。

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).
*
```

sqledbdescext

拡張データベース記述ブロック (sqledbdescext) 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されます。拡張データベース記述ブロックは、データベースの自動ストレージを有効にし、データベースのデフォルト・ページ・サイズを選択し、導入されている新規表スペース属性の値を指定します。この構造は、データベース記述ブロック (sqledbdesc) 構造に追加するものとして使用され、それに置き換わるものではありません。

この構造が sqlecrea API に渡されない場合、以下の動作となります。

- データベースの自動ストレージは有効
- データベースのデフォルト・ページ・サイズは 4096 バイト (4 KB)
- 該当する場合、DB2 データベース・システムは拡張表スペース属性の値を自動的に決定する

API とデータ構造構文

```
SQL_STRUCTURE sqledbdescext
{
        sqluint32 sqlPageSize;
        struct sqleAutoStorageCfg *sqlAutoStorage;
        struct SQLETSDESCEXT *sqlcattsext;
        struct SQLETSDESCEXT *sqlusrtsext;
        struct SQLETSDESCEXT *sqltmptsext;
        void *reserved;
};

SQL_STRUCTURE sqleAutoStorageCfg
{
        char sqlEnableAutoStorage;
        char pad[3];
        sqluint32 sqlNumStoragePaths;
        char **sqlStoragePaths;
};

SQL_STRUCTURE SQLETSDESCEXT
{
        sqlint64 sqlInitSize;
        sqlint64 sqlIncreaseSize;
        sqlint64 sqlMaximumSize;
        char sqlAutoResize;
        char sqlInitSizeUnit;
        char sqlIncreaseSizeUnit;
        char sqlMaximumSizeUnit;
};

SQL_STRUCTURE sqledboptions
{
        void *piAutoConfigInterface;
        sqlint32 restrictive;
        void *reserved;
};
```

sqledbdescext data structure パラメーター

表 52. sqledbdescext 構造のフィールド

フィールド名	データ・タイプ	説明
SQLPAGESIZE	sqluint32	データベースの作成時の初期表スペース (SYSCATSPACE、 TEMPSPACE1、 USERSPACE1) に加え、デフォルトのバッファー・プールのページ・サイズを指定します。指定される値は、将来のすべての CREATE BUFFERPOOL および CREATE TABLESPACE ステートメントのデフォルトのページ・サイズも表します。この表に続く、値についての情報を参照してください。
SQLAUTOSTORAGE	ポインター	自動ストレージ構成構造を指すポインター。このポインターは、データベースの自動ストレージを有効または無効にします。ポインターを指定すると、自動ストレージが有効または無効になります。 NULL の場合、自動ストレージが有効になり、単一のストレージ・パスが、パスされる dbpath か、またはデータベース・マネージャー構成パラメーターの dftdbpath によって決定された値を使用して想定されます。
SQLCATTSEXT	ポインター	システム・カタログ表スペースの拡張表スペース記述制御ブロック (SQLETSDESCEXT) を指すポインター。これは SQLETSDESC にある属性の、追加の属性を定義します。 NULL の場合、データベース・マネージャーはそれらの属性の値を自動的に決定します (該当する場合)。
SQLUSRTSEXT	ポインター	ユーザー表スペースの拡張表スペース記述制御ブロック (SQLETSDESCEXT) を指すポインター。これは SQLETSDESC にある属性の、追加の属性を定義します。 NULL の場合、データベース・マネージャーはそれらの属性の値を自動的に決定します (該当する場合)。
SQLTMPTSEXT	ポインター	SYSTEM TEMPORARY 表スペースの拡張表スペース記述制御ブロック (SQLETSDESCEXT) を指すポインター。これは SQLETSDESC にある属性の、追加の属性を定義します。 NULL の場合、データベース・マネージャーはそれらの属性の値を自動的に決定します (該当する場合)。
RESERVED	ポインター	データベース・オプション制御ブロック (sqledboptions) を指すポインター。

SQLPAGESIZE の有効な値 (sqlenv で定義) は、以下のとおりです。

SQL_PAGESIZE_4K
データベースのデフォルト・ページ・サイズは 4 096 バイトです。

SQL_PAGESIZE_8K
データベースのデフォルト・ページ・サイズは 8 192 バイトです。

SQL_PAGESIZE_16K
データベースのデフォルト・ページ・サイズは 16 384 バイトです。

SQL_PAGESIZE_32K
データベースのデフォルト・ページ・サイズは 32 768 バイトです。

自動ストレージ構成 (sqleAutoStorageCfg) データ構造パラメーター

自動ストレージ構成 (sqleAutoStorageCfg) 構造は、sqlecrea API への呼び出し中に使用できます。これは sqledbdescext 構造のエレメントであり、自動ストレージをデータベースに対して有効にするかどうかを指定します。

表 53. sqleAutoStorageCfg 構造のフィールド

フィールド名	データ・タイプ	説明
SQLENABLEAUTOSTORAGE	CHAR(1)	自動ストレージをデータベースに対して有効にするかどうかを指定します。この表に続く、値についての情報を参照してください。
SQLNUMSTORAGEPATHS	sqluint32	SQLSTORAGEPATHS 配列によって指されているストレージ・パスの数を示す値。値が 0 の場合、SQLSTORAGEPATHS ポインターは NULL でなければなりません。ストレージ・パスの最大数 (SQL_MAX_STORAGE_PATHS) は 128 です。
SQLSTORAGEPATHS	ポインター	ストレージ・パスを指すストリング・ポインターの配列。配列内のポインターの数は SQLNUMSTORAGEPATHS によって反映されます。ストレージ・パスが指定されていない場合は、SQLSTORAGEPATHS を NULL に設定します (この場合、SQLNUMSTORAGEPATHS は 0 に設定する必要があります)。それぞれのパスの最大長は 175 文字です。

SQLENABLEAUTOSTORAGE の有効な値 (sqlenv で定義) は、以下のとおりです。

SQL_AUTOMATIC_STORAGE_NO
データベースの自動ストレージは無効です。この値を使用する場合、SQLNUMSTORAGEPATHS は 0 に設定し、SQLSTORAGEPATHS は NULL に設定する必要があります。

SQL_AUTOMATIC_STORAGE_YES
データベースの自動ストレージは有効です。自動ストレージに使用されるストレージ・パスは、SQLSTORAGEPATHS ポインターを使用して指定されます。このポインターが NULL の場合、単一のストレージ・パスが、データベース・マネージャー構成パラメーターの dftdbpath によって決定された値を使用して想定されます。

SQL_AUTOMATIC_STORAGE_DFT
データベース・マネージャーが、自動ストレージを有効にするかどうかを決定します。現在のところ、この選択は SQLSTORAGEPATHS ポインターに基づいて行われます。このポインターが NULL の場合、自動ストレージは無効になり、そうでなければ有効になります。デフォルト値は SQL_AUTOMATIC_STORAGE_YES に相当します。

拡張表スペース記述ブロック (SQLETSDESCEXT) 構造パラメーター

拡張表スペース記述ブロック (SQLETSDESCEXT) 構造は、3 つの初期表スペースの属性を指定するために使用されます。この構造は、表スペース記述ブロック (SQLETSDESC) 構造に追加するものとして使用され、それに置き換わるものではありません。

表 54. SQLETSDESCEXT 構造のフィールド

フィールド名	データ・タイプ	説明
SQLINITSIZE	sqlint64	自動ストレージを使用する各表スペースの初期サイズを定義します。このフィールドは、標準または大規模な自動ストレージ表スペースだけに関係します。それ以外の表スペースのタイプの場合、または DB2 に自動的に初期サイズを決定させることを意図している場合は、SQL_TBS_AUTOMATIC_INITSIZE の値を使用します。注: データベース・マネージャーによって使用される実際の値は、指定されたものよりいくらか大小の相違がある可能性があります。指定された値では表スペース内のコンテナー同士のサイズの一貫性が保てない場合に、この処置が取られて一貫性を保ちます。
SQLINCREASESIZE	sqlint64	表スペースが満杯になったときに、データベース・マネージャーが表スペースを増やすサイズの単位を定義します。このフィールドは、自動サイズ変更が可能な表スペースだけに関係します。自動サイズ変更が無効の場合、またはデータベース・マネージャーに自動的にサイズの増加を決定させることを意図している場合は、SQL_TBS_AUTOMATIC_INCSIZE の値を使用します。注: データベース・マネージャーによって使用される実際の値は、指定されたものよりいくらか大小の相違がある可能性があります。指定された値では表スペース内のコンテナー同士のサイズの一貫性が保てない場合に、この処置が取られて一貫性を保ちます。

表 54. SQLETSDESCEXT 構造のフィールド (続き)

フィールド名	データ・タイプ	説明
SQLMAXIMUMSIZE	sqlint64	データベース・マネージャーが自動的に表スペースを増やす場合の最大サイズを定義します。別の方法として、SQL_TBS_NO_MAXSIZE の値を使用して、unlimited の最大サイズを指定できます。この場合、表スペースは、表スペースの設計上の限界まで、または filesystem full 条件が検出されるまでが最大サイズになります。このフィールドは、自動サイズ変更が可能な表スペースだけに関係します。自動サイズ変更が無効の場合、またはデータベース・マネージャーに自動的に最大サイズを決定させることを意図している場合は、SQL_TBS_AUTOMATIC_MAXSIZE の値を使用します。注: データベース・マネージャーによって使用される実際の値は、指定されたものよりいくらか大小の相違がある可能性があります。指定された値では表スペース内のコンテナー同士のサイズの一貫性が保てない場合に、この処置が取られて一貫性を保ちます。
SQLAUTORESIZE	CHAR(1)	自動サイズ変更が表スペースについて有効かどうかを指定します。この表に続く、値についての情報を参照してください。
SQLINITSIZEUNIT	CHAR(1)	該当する場合、SQLINITSIZE をバイト、キロバイト、メガバイト、またはギガバイトのどの単位で指定するかを示します。この表に続く、値についての情報を参照してください。
SQLINCREASESIZEUNIT	CHAR(1)	該当する場合、SQLINCREASESIZE をバイト、キロバイト、メガバイト、ギガバイトのどの単位で指定するか、またはパーセンテージとして指定するかを示します。この表に続く、値についての情報を参照してください。
SQLMAXIMUMSIZEUNIT	CHAR(1)	該当する場合、SQLMAXIMUMSIZE をバイト、キロバイト、メガバイト、またはギガバイトのどの単位で指定するかを示します。この表に続く、値についての情報を参照してください。

SQLAUTORESIZE の有効な値 (sqlenv で定義) は、以下のとおりです。

SQL_TBS_AUTORESIZE_NO
自動サイズ変更は表スペースには無効です。この値は、データベース管理スペース (DMS) 表スペースまたは自動ストレージ表スペースに対してのみ指定できます。

SQL_TBS_AUTORESIZE_YES
自動サイズ変更は表スペースについて有効です。この値は、データベース管理スペース (DMS) 表スペースまたは自動ストレージ表スペースに対してのみ指定できます。

SQL_TBS_AUTORESIZE_DFT
データベース・マネージャーは、自動サイズ変更が有効かどうかを、表スペースのタイプに基づいて判別します。つまり自動サイズ変更を、データベース管理スペース (DMS) 表スペースの場合はオフに、自動ストレージ表スペースの場合はオンにします。自動サイズ変更は、システム管理スペース (SMS) 表スペース・タイプの表スペースには適用されないので、それにはこの値を使用してください。

SQLINITSIZEUNIT、SQLINCREASESIZEUNIT、および SQLMAXIMUMSIZEUNIT の有効な値 (sqlenv で定義) は、以下のとおりです。

SQL_TBS_STORAGE_UNIT_BYTES
対応するサイズ・フィールドで指定される値は、バイト単位です。

SQL_TBS_STORAGE_UNIT_KILOBYTES
対応するサイズ・フィールドで指定される値は、キロバイト単位です(1 キロバイト = 1 024 バイト)。

SQL_TBS_STORAGE_UNIT_MEGABYTES
対応するサイズ・フィールドで指定される値は、メガバイト単位です (1 メガバイト = 1 048 576 バイト)。

SQL_TBS_STORAGE_UNIT_GIGABYTES
対応するサイズ・フィールドで指定される値は、ギガバイト単位です (1 ギガバイト = 1 073 741 824 バイト)。

SQL_TBS_STORAGE_UNIT_PERCENT
対応するサイズ・フィールドで指定される値は、パーセンテージです (有効な範囲は 1 から 100)。この値は、SQLINCREASESIZEUNIT の場合のみ有効です。

sqledboptions データ構造パラメーター

piAutoConfigInterface
入力。構成アドバイザー用の入力としての役割を果たす情報を含む db2AutoConfigInterface 構造を指すポインター。

restrictive
制限フィールドの設定は RESTRICT_ACCESS データベース構成パラメーターに保管され、このデータベースの今後のマイグレーションすべてに反映されます。つまり、データベースが DB2 の以降のリリースにマイグレーションされるときに、マイグレーション・ユーティリティーは RESTRICT_ACCESS データベース構成パラメーターの設定を調べて、デフォルト・アクションの制限セットを新規の DB2 リリースに導入される新規オブジェクト (例えば、新規のシステム・カタログ表) に適用する必要があるかどうかを判別します。

このパラメーターの有効な値は以下のとおりです (インクルード・ディレクトリーの sqlenv ヘッダー・ファイルで定義される)。

SQL_DB_RESTRICT_ACCESS_NO or SQL_DB_RESTRICT_ACCESS_DFT
データベースがデフォルト・アクションの制限セットを使用しないで作成されることを示します。この設定により、以下の特権が PUBLIC に付与されます。

- CREATETAB 特権
- BINDADD 特権
- CONNECT 特権
- IMPLSCHEMA 特権
- スキーマ SQLJ 内のすべてのプロシージャーに対する GRANT 特権付きの EXECUTE
- スキーマ SYSPROC 内のすべての関数およびプロシージャーに対する GRANT 特権付きの EXECUTE
- NULLID スキーマで作成されたすべてのパッケージに対する BIND 特権
- NULLID スキーマで作成されたすべてのパッケージに対する EXECUTE 特権
- スキーマ SQLJ に対する CREATEIN 特権
- スキーマ NULLID に対する CREATEIN 特権
- 表スペース USERSPACE1 に対する USE 特権
- SYSIBM カタログ表に対する SELECT 特権
- SYSCAT カタログ・ビューに対する SELECT 特権
- SYSSTAT カタログ・ビューに対する SELECT 特権
- SYSSTAT カタログ・ビューに対する UPDATE 特権

SQL_DB_RESTRICT_ACCESS_YES
データベースがデフォルト・アクションの制限セットを使用して作成されることを示します。 SQL_DB_RESTRICT_ACCESS_NO の下の、上記でリストされた付与アクションが起こらないことを意味します。

reserved
将来の利用のために予約されています。

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 55. 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。

API とデータ構造構文

```
SQL_STRUCTURE sqledbcountryinfo
{
        char sqldbcodeset[SQL_CODESET_LEN + 1];
        char sqldblocale[SQL_LOCALE_LEN + 1];
};
typedef SQL_STRUCTURE sqledbcountryinfo SQLEDBTERRITORYINFO;
```

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.
*
```

sqleninfo

この構造は、sqlengne API への呼び出し後に情報を戻します。

注: NetBIOS は、現在サポートされていません。 SNA、およびその API の APPC、APPN、および CPI-C も、サポートされなくなりました。それらのプロトコルを使用している場合は、TCP/IP などのサポートされているプロトコルを使用してノードとデータベースを再カタログしてください。それらのプロトコルに言及している箇所がある場合、それらは無視してください。

表 56. 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。

表 56. 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
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、あるいは IPv4、IPv6 アドレス。
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。
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

API とデータ構造構文

```
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;
        _SQLOLDCHAR chgpwd_lu[SQL_RMTLU_SZ];
        _SQLOLDCHAR transpn[SQL_TPNAME_SZ];
        _SQLOLDCHAR lanaddr[SQL_LANADDRESS_SZ];
};
```

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.
*
```

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 57. 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 によって指定されるデータを入れ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 バイト整数

Sint32 符号付き、4 バイト整数

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 58. 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
alt_collate	SQLF_DBTN_ALT_COLLATE	809	Uint32
app_ctl_heap_sz	SQLF_DBTN_APP_CTL_HEAP_SZ	500	Uint16
appgroup_mem_sz	SQLF_DBTN_APPGROUP_MEM_SZ	800	Uint32
applheapsz	SQLF_DBTN_APPLHEAPSZ	51	Uint16
archretrydelay	SQLF_DBTN_ARCHRETRYDELAY	828	Uint16

表 58. 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
• auto_maint • auto_db_backup • auto_tbl_maint • auto_runstats • auto_stats_prof • auto_prof_upd • auto_reorg	• SQLF_DBTN_AUTO_MAINT • SQLF_DBTN_AUTO_DB_BACKUP • SQLF_DBTN_AUTO_TBL_MAINT • SQLF_DBTN_AUTO_RUNSTATS • SQLF_DBTN_AUTO_STATS_PROF • SQLF_DBTN_AUTO_PROF_UPD • SQLF_DBTN_AUTO_REORG	• 831 • 833 • 835 • 837 • 839 • 844 • 841	Uint16
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
database_memory	SQLF_DBTN_DATABASE_MEMORY	803	Uint64
dbheap	SQLF_DBTN_DB_HEAP	58	Uint64
db_mem_thresh	SQLF_DBTN_DB_MEM_THRESH	849	Uint16
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_mttb_types	SQLF_DBTN_DFT_MTTB_TYPES	843	Uint32
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
discover	SQLF_DBTN_DISCOVER	308	Uint16
dlchktime	SQLF_DBTN_DLCHKTIME	9	Uint32
dyn_query_mgmt	SQLF_DBTN_DYN_QUERY_MGMT	604	Uint16
failarchpath	SQLF_DBTN_FAILARCHPATH	826	char(243)
groupheap_ratio	SQLF_DBTN_GROUPHEAP_RATIO	801	Uint16
hadr_local_host	SQLF_DBTN_HADR_LOCAL_HOST	811	char(256)
hadr_local_svc	SQLF_DBTN_HADR_LOCAL_SVC	812	char(41)
hadr_remote_host	SQLF_DBTN_HADR_REMOTE_HOST	813	char(256)
hadr_remote_inst	SQLF_DBTN_HADR_REMOTE_INST	815	char(9)
hadr_remote_svc	SQLF_DBTN_HADR_REMOTE_SVC	814	char(41)
hadr_syncmode	SQLF_DBTN_HADR_SYNCMODE	817	Uint32
hadr_timeout	SQLF_DBTN_HADR_TIMEOUT	816	Sint32
indexrec	SQLF_DBTN_INDEXREC	30	Uint16
locklist	SQLF_DBTN_LOCK_LIST	704	Uint64
locktimeout	SQLF_DBTN_LOCKTIMEOUT	34	Sint16
logarchmeth1	SQLF_DBTN_LOGARCHMETH1	822	Uint16

表 58. 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
logarchmeth2	SQLF_DBTN_LOGARCHMETH2	823	Uint16
logarchopt1	SQLF_DBTN_LOGARCHOPT1	824	char(243)
logarchopt2	SQLF_DBTN_LOGARCHOPT2	825	char(243)
logbufsz	SQLF_DBTN_LOGBUFSZ	33	Uint16
logfilsiz	SQLF_DBTN_LOGFIL_SIZ	92	Uint32
logindexbuild	SQLF_DBTN_LOGINDEXBUILD	818	Uint32
logprimary	SQLF_DBTN_LOGPRIMARY	16	Uint16
logretain	SQLF_DBTN_LOG_RETAIN	23	Uint16
logsecond	SQLF_DBTN_LOGSECOND	17	Uint16
max_log	SQLF_DBTN_MAX_LOG	807	Uint16
maxappls	SQLF_DBTN_MAXAPPLS	6	Uint16
maxfilop	SQLF_DBTN_MAXFILOP	3	Uint16
maxlocks	SQLF_DBTN_MAXLOCKS	15	Uint16
max_log	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_freqvalues	SQLF_DBTN_NUM_FREQVALUES	36	Uint16
num_iocleaners	SQLF_DBTN_NUM_IOCLEANERS	37	Uint16
num_ioservers	SQLF_DBTN_NUM_IOSERVERS	39	Uint16
num_log_span	SQLF_DBTN_NUM_LOG_SPAN	808	Uint16
num_quantiles	SQLF_DBTN_NUM_QUANTILES	48	Uint16
numarchretry	SQLF_DBTN_NUMARCHRETRY	827	Uint16
overflowlogpath	SQLF_DBTN_OVERFLOWLOGPATH	805	char(242)
pckcachesz	SQLF_DBTN_PCKCACHE_SZ	505	Uint32
rec_his_retentn	SQLF_DBTN_REC_HIS_RETENTN	43	Sint16
self_tuning_mem	SQLF_DBTN_SELF_TUNING_MEM	848	Uint16
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)

表 58. 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
userexit	SQLF_DBTN_USER_EXIT	24	Uint16
util_heap_sz	SQLF_DBTN_UTIL_HEAP_SZ	55	Uint32
vendoropt	SQLF_DBTN_VENDOROPT	829	char(242)

SQLF_DBTN_AUTONOMIC_SWITCHES の各ビットは、いくつかの自動保守構成パラメーターのデフォルト設定値を示します。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。

```
デフォルト => ビット 1 がオン  (xxxx xxxx xxxx xxx1): auto_maint
           ビット 2 がオフ (xxxx xxxx xxxx xx0x): auto_db_backup
           ビット 3 がオン (xxxx xxxx xxxx x1xx): auto_tbl_maint
           ビット 4 がオン (xxxx xxxx xxxx 1xxx): auto_runstats
           ビット 5 がオフ (xxxx xxxx xxx0 xxxx): auto_stats_prof
           ビット 6 がオフ (xxxx xxxx xx0x xxxx): auto_prof_upd
           ビット 7 がオフ (xxxx xxxx x0xx xxxx): auto_reorg
           ビット 8 がオフ (xxxx xxxx 0xxx xxxx): auto_storage
           ビット 9 がオフ (xxxx xxx0 xxxx xxxx): auto_stmt_stats
                       0    0    0    D

最大 => ビット 1 がオン (xxxx xxxx xxxx xxx1): auto_maint
           ビット 2 がオフ (xxxx xxxx xxxx xx1x): auto_db_backup
           ビット 3 がオン (xxxx xxxx xxxx x1xx): auto_tbl_maint
           ビット 4 がオン (xxxx xxxx xxxx 1xxx): auto_runstats
           ビット 5 がオフ (xxxx xxxx xxx1 xxxx): auto_stats_prof
           ビット 6 がオフ (xxxx xxxx xx1x xxxx): auto_prof_upd
           ビット 7 がオフ (xxxx xxxx x1xx xxxx): auto_reorg
           ビット 8 がオフ (xxxx xxxx 1xxx xxxx): auto_storage
           ビット 9 がオフ (xxxx xxx1 xxxx xxxx): auto_stmt_stats
                       0    1    F    F
```

indexrec の有効な値 (sqlutil.h で定義されているもの):

- SQLF_INX_REC_SYSTEM (0)
- SQLF_INX_REC_REFERENCE (1)
- SQLF_INX_REC_RESTART (2)

logretain の有効な値 (sqlutil.h で定義されているもの):

- SQLF_LOGRETAIN_NO (0)
- SQLF_LOGRETAIN_RECOVERY (1)
- SQLF_LOGRETAIN_CAPTURE (2)

表 59. 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) (下記の注 1 を参照)
collate_info	SQLF_DBTN_COLLATE_INFO	44	char(260)

表 59. 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
country/region	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) (下記の注 2 を参照)
user_exit_status	SQLF_DBTN_USER_EXIT_STATUS	115	Uint16

注:

1. HP-UX、Solaris、および Linux オペレーティング・システムの場合は char(17)。
2. HP-UX、Solaris、および Linux オペレーティング・システムの場合は 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 60. 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	SQLF_KTN_AUTHENTICATION	78	Uint16
catalog_noauth	SQLF_KTN_CATALOG_NOAUTH	314	Uint16
clnt_krb_plugin	SQLF_KTN_CLNT_KRB_PLUGIN	812	char(33)
clnt_pw_plugin	SQLF_KTN_CLNT_PW_PLUGIN	811	char(33)
comm_bandwidth	SQLF_KTN_COMM_BANDWIDTH	307	float
conn_elapse	SQLF_KTN_CONN_ELAPSE	508	Uint16
cpuspeed	SQLF_KTN_CPUSPEED	42	float
dft_account_str	SQLF_KTN_DFT_ACCOUNT_STR	28	char(25)
dft_monswitches	SQLF_KTN_DFT_MONSWITCHES	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

表 60. 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
dft_mon_stmt	SQLF_KTN_DFT_MON_STMT	31	Uint16
dft_mon_table	SQLF_KTN_DFT_MON_TABLE	32	Uint16
dft_mon_timestamp	SQLF_KTN_DFT_MON_ TIMESTAMP	36	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	SQLF_KTN_DISCOVER	304	Uint16
discover_inst	SQLF_KTN_DISCOVER_INST	308	Uint16
fcm_num_buffers	SQLF_KTN_FCM_NUM_BUFFERS	503	Uint32
fcm_num_channels	SQLF_KTN_FCM_NUM_CHANNELS	902	Uint32
fed_noauth	SQLF_KTN_FED_NOAUTH	806	Uint16
federated	SQLF_KTN_FEDERATED	604	Sint16
federated_async	SQLF_KTN_FEDERATED_ASYNC	849	Sint32
fenced_pool	SQLF_KTN_FENCED_POOL	80	Sint32
group_plugin	SQLF_KTN_GROUP_PLUGIN	810	char(33)
health_mon	SQLF_KTN_HEALTH_MON	804	Uint16
indexrec	SQLF_KTN_INDEXREC	20	Uint16
instance_memory	SQLF_KTN_INSTANCE_MEMORY	803	Uint64
intra_parallel	SQLF_KTN_INTRA_PARALLEL	306	Sint16
java_heap_sz	SQLF_KTN_JAVA_HEAP_SZ	310	Sint32
jdk_path	SQLF_KTN_JDK_PATH	311	char(255)
keepfenced	SQLF_KTN_KEEPFENCED	81	Uint16
local_gssplugin	SQLF_KTN_LOCAL_GSSPLUGIN	816	char(33)
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
mon_heap_sz	SQLF_KTN_MON_HEAP_SZ	79	Uint16
notifylevel	SQLF_KTN_NOTIFYLEVEL	605	Sint16
num_initagents	SQLF_KTN_NUM_INITAGENTS	500	Uint32
num_initfenced	SQLF_KTN_NUM_INITFENCED	601	Sint32
num_poolagents	SQLF_KTN_NUM_POOLAGENTS	502	Sint32
numdb	SQLF_KTN_NUMDB	6	Uint16
query_heap_sz	SQLF_KTN_QUERY_HEAP_SZ	49	Sint32
resync_interval	SQLF_KTN_RESYNC_INTERVAL	68	Uint16
rqrioblk	SQLF_KTN_RQRIOBLK	1	Uint16

表 60. 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
sheapthres	SQLF_KTN_SHEAPTHRES	21	Uint32
spm_log_file_sz	SQLF_KTN_SPM_LOG_FILE_SZ	90	Sint32
spm_log_path	SQLF_KTN_SPM_LOG_PATH	313	char(226)
spm_max_resync	SQLF_KTN_SPM_MAX_RESYNC	91	Sint32
spm_name	SQLF_KTN_SPM_NAME	92	char(8)
srvcon_auth	SQLF_KTN_SRVCON_AUTH	815	Uint16
srvcon_gssplugin_list	SQLF_KTN_SRVCON_GSSPLUGIN_ LIST	814	char(256)
srv_plugin_mode	SQLF_KTN_SRV_PLUGIN_MODE	809	Uint16
srvcon_pw_plugin	SQLF_KTN_SRVCON_PW_PLUGIN	813	char(33)
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)
sysmon_group	SQLF_KTN_SYSMON_GROUP	808	char(30)
tm_database	SQLF_KTN_TM_DATABASE	67	char(8)
tp_mon_name	SQLF_KTN_TP_MON_NAME	66	char(19)
trust_allclnts	SQLF_KTN_TRUST_ALLCLNTS	301	Uint16
trust_clntauth	SQLF_KTN_TRUST_CLNTAUTH	302	Uint16
util_impact_lim	SQLF_KTN_UTIL_IMPACT_LIM	807	Uint32

注: 構成パラメーター `maxagents`と `maxcagents` は使用すべきではありません。今後のリリースでは、これらの構成パラメーターは完全に除去される予定です。

authentication の有効な値 (sqlenv.h で定義されているもの):

- 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_GSSPLUGIN (9)
- SQL_AUTHENTICATION_GSS_SVR_ENC (10)
- SQL_AUTHENTICATION_DATAENC (11)
- SQL_AUTHENTICATION_DATAENC_CMP (12)
- SQL_AUTHENTICATION_NOT_SPEC (255)

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。

- ビット 1 (xxxx xxx1): dft_mon_uow
- ビット 2 (xxxx xx1x): dft_mon_stmt
- ビット 3 (xxxx x1xx): dft_mon_table
- ビット 4 (xxxx 1xxx): dft_mon_buffpool
- ビット 5 (xxx1 xxxx): dft_mon_lock
- ビット 6 (xx1x xxxx): dft_mon_sort
- ビット 7 (x1xx xxxx): dft_mon_timestamp

discover の有効な値 (sqlutil.h で定義されているもの):

- SQLF_DSCVR_KNOWN (1)
- SQLF_DSCVR_SEARCH (2)

indexrec の有効な値 (sqlutil.h で定義されているもの):

- SQLF_INX_REC_SYSTEM (0)
- SQLF_INX_REC_REFERENCE (1)
- SQLF_INX_REC_RESTART (2)

trust_allclnts の有効な値 (sqlutil.h で定義されているもの):

- SQLF_TRUST_ALLCLNTS_NO (0)
- SQLF_TRUST_ALLCLNTS_YES (1)
- SQLF_TRUST_ALLCLNTS_DRDAONLY (2)

表 61. 更新できないデータベース・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	SQLF_KTN_NODETYPE	100	Uint16
release	SQLF_KTN_RELEASE	101	Uint16

nodetype の有効な値 (sqlutil.h で定義されているもの):

- 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)

API とデータ構造構文

```
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.
*
```

sqllob

この構造は、ホスト・プログラミング言語で LOB データ・タイプを表現するのに使用されます。

表 62. sqllob 構造のフィールド

フィールド名	データ・タイプ	説明
length	sqluint32	data パラメーターの長さ (単位: バイト)。
データ	char(1)	渡されるデータ。

API とデータ構造構文

```
SQL_STRUCTURE sqllob
{
        sqluint32 length;
        char data[1];
};
```

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 63. 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 64. 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(128)	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 の有効な値は以下のとおりです (include ディレクトリーの sqlmon ヘッダー・ファイルで定義される)。

SQLMA_DB2
インスタンス関連情報。

SQLMA_DBASE
特定データベースのデータベース関連情報。SQLMA_DBASE 値を使用する場合は、sqlm_obj_struct 構造のオブジェクト・パラメーターのデータベース名を指定する必要があります。

SQLMA_APPL
アプリケーション ID が指定と一致するアプリケーションに関するアプリケーション情報。 SQLMA_APPL 値を使用する場合は、sqlm_obj_struct 構造のオブジェクト・パラメーターのアプリケーション ID を指定する必要があります。

SQLMA_AGENT_ID
エージェント ID が指定と一致するアプリケーションに関するアプリケーション情報。 SQLMA_AGENT_ID 値を使用する場合は、sqlm_obj_struct 構造の agent_id パラメーターのエージェント ID を指定する必要があります。

SQLMA_DBASE_TABLES
特定のデータベースの表情報。 SQLMA_DBASE_TABLES 値を使用する場合は、sqlm_obj_struct 構造のオブジェクト・パラメーターのデータベース名を指定する必要があります。

SQLMA_DBASE_APPLS
特定のデータベースに接続されたアプリケーションすべてに関するアプリケーション情報。 SQLMA_DBASE_APPLS 値を使用する場合は、sqlm_obj_struct 構造のオブジェクト・パラメーターのデータベース名を指定する必要があります。

SQLMA_DBASE_APPLINFO
特定のデータベースへの接続に関する、サマリー・アプリケーション情報。SQLMA_DBASE_APPLINFO 値を使用する場合は、sqlm_obj_struct 構造のオブジェクト・パラメーターのデータベース名を指定する必要があります。

SQLMA_DBASE_LOCKS
特定のデータベースに保持されているロックのリスト。

SQLMA_DBASE_LOCKS 値を使用する場合は、sqlm_obj_struct 構造のオブジェクト・パラメーターのデータベース名を指定する必要があります。

SQLMA_APPL_LOCKS
指定と一致するアプリケーション ID を持つアプリケーションによって保持されているロックのリスト。SQLMA_APPL_LOCKS 値を使用する場合は、sqlm_obj_struct 構造のオブジェクト・パラメーターのアプリケーション ID を指定する必要があります。

SQLMA_APPL_LOCKS_AGENT_ID
指定と一致するエージェント ID を持つアプリケーションによって保持されているロックのリスト。SQLMA_APPL_LOCKS_AGENT_ID 値を使用する場合は、sqlm_obj_struct 構造の agent_id パラメーターのエージェント ID を設定します。

SQLMA_DBASE_ALL
インスタンス内の全アクティブ・データベースに関するデータベース情報。

SQLMA_APPL_ALL
インスタンス内の全データベース接続に関するアプリケーション情報。

SQLMA_APPLINFO_ALL
インスタンスへの接続すべてに関する、サマリー・アプリケーション情報

SQLMA_DCS_APPLINFO_ALL
インスタンスへのデータベース接続サービス (DCS) 接続のリスト。

SQLMA_DYNAMIC_SQL
特定のデータベースに関する動的 SQL ステートメント情報。
SQLMA_DYNAMIC_SQL 値を使用する場合は、sqlm_obj_struct 構造のオブジェクト・パラメーターのデータベース名を指定する必要があります。

SQLMA_DCS_DBASE
特定のデータベース接続サービス (DCS) データベースに関する情報。
SQLMA_DCS_DBASE 値を使用する場合は、sqlm_obj_struct 構造のオブジェクト・パラメーターのデータベース名を指定する必要があります。

SQLMA_DCS_DBASE_ALL
すべてのアクティブなデータベース接続サービス (DCS) データベースに関する情報。

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 (DCS) アプリケーション情報。

SQLMA_DCS_APPL
アプリケーション ID が指定と一致するアプリケーションに関するデータベース接続サービス (DCS) アプリケーション情報。 SQLMA_DCS_APPL 値を使用する場合は、sqlm_obj_struct 構造のオブジェクト・パラメーターのアプリケーション ID を設定します。

SQLMA_DCS_APPL_HANDLE
エージェント ID が指定と一致するアプリケーションに関するデータベース接続サービス (DCS) アプリケーション情報。
SQLMA_DCS_APPL_HANDLE 値を使用する場合は、sqlm_obj_struct 構造の agent_id パラメーターのエージェント ID を設定します。

SQLMA_DCS_DBASE_APPLS
特定のデータベースへのすべてのアクティブな接続に関するデータベース接続サービス (DCS) アプリケーション情報。SQLMA_DCS_DBASE_APPLS 値を使用する場合は、sqlm_obj_struct 構造のオブジェクト・パラメーターのデータベース名を指定する必要があります。

SQLMA_DBASE_TABLESPACES
特定のデータベースの表スペース情報。SQLMA_DBASE_TABLESPACES 値を使用する場合は、sqlm_obj_struct 構造のオブジェクト・パラメーターのデータベース名を指定する必要があります。

SQLMA_DBASE_BUFFERPOOLS
特定のデータベースに関するバッファー・プール情報。
SQLMA_DBASE_BUFFERPOOLS 値を使用する場合は、sqlm_obj_struct 構造のオブジェクト・パラメーターのデータベース名を指定する必要があります。

SQLMA_BUFFERPOOLS_ALL
すべてのバッファー・プールに関する情報。

SQLMA_DBASE_REMOTE
特定のフェデレーテッド・データベースに関するリモート・アクセス情報。SQLMA_DBASE_REMOTE 値を使用する場合は、sqlm_obj_struct 構造のオブジェクト・パラメーターのデータベース名を指定する必要があります。

SQLMA_DBASE_REMOTE_ALL
すべてのフェデレーテッド・データベースに関するリモート・アクセス情報。

SQLMA_DBASE_APPLS_REMOTE
特定のフェデレーテッド・データベースに接続されたアプリケーションに関するリモート・アクセス情報。SQLMA_DBASE_APPLS_REMOTE 値を使用する場合は、sqlm_obj_struct 構造のオブジェクト・パラメーターのデータベース名を指定する必要があります。

SQLMA_APPL_REMOTE_ALL
すべてのアプリケーションに関するリモート・アクセス情報。

API とデータ構造構文

```
typedef struct sqlma
{
   sqluint32 obj_num;
   sqlm_obj_struct obj_var[1];
}sqlma;

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(128).
*
```

sqlopt

この構造は、sqlabndx API にバインド・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 に再バインド・オプションを渡すために使用されます。

表 65. 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 66. 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 67. SQLOPTIONS 構造のフィールド

フィールド名	データ・タイプ	説明
TYPE VAL	INTEGER INTEGER	バインド/プリコンパイル/再バインド・オプション・タイプ。 バインド/プリコンパイル/再バインド・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、または再バインド・オプションについて繰り返されます。

API とデータ構造構文

```
SQL_STRUCTURE sqlopt
{
   SQL_STRUCTURE sqloptheader header;
   SQL_STRUCTURE sqloptions   option[1];
};

SQL_STRUCTURE sqloptheader
{
   sqluint32 allocated;
   sqluint32 used;
};
```

```
SQL_STRUCTURE sqloptions
{
   sqluint32 type;
   sqluintptr 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
*
```

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。 LSN は、データベース・ログの始まりからのログ・レコードのバイト・オフセットを表します。

表 68. 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。

API とデータ構造構文

```
typedef union SQLU_LSN
{
   unsigned char   lsnChar[6];
   unsigned short  lsnWord[3];
} SQLU_LSN;
```

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 69. 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。

表 69. SQLU-MEDIA-LIST 構造のフィールド (続き)

フィールド名	データ・タイプ	説明
SESSIONS	INTEGER	この構造の target フィールドが指す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。指す構造のタイプは、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。
FILLER	CHAR(3)	メモリー内のデータ構造を正しく配置するために使用される充てん文字。

表 70. 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。
PREMOTEFETCH	ポインター	sqlu_remotefetch_entry 構造を指すポインター。

表 71. 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 MEDIA_ENTRY	INTEGER CHAR(215)	media_entry フィールドの長さ。 C 以外の言語用。パ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 72. 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。

表 72. SQLU-VENDOR 構造のフィールド (続き)

フィールド名	データ・タイプ	説明
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 73. 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 74. 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。

表 75. SQLU-REMOTEFETCH-ENTRY 構造のフィールド

フィールド名	データ・タイプ	説明
pDatabaseName	ポインター	ソース・データベース名
iDatabaseNameLen	INTEGER	ソース・データベース名の長さ
pUserID	ポインター	ユーザー ID を指すポインター
iUserIDLen	INTEGER	ユーザー ID の長さ
pPassword	ポインター	パスワードを指すポインター
iPasswordLen	INTEGER	パスワードの長さ
pTableSchema	ポインター	ソース表のスキーマを指すポインター
iTableSchemaLen	INTEGER	スキーマの長さ
pTableName	ポインター	ソース表の名前を指すポインター
iTableNameLen	INTEGER	ソース表名の長さ
pStatement	ポインター	ステートメントの名前を指すポインター
iStatementLen	INTEGER	ステートメントの長さ
pIsolationLevel	ポインター	分離レベルを指すポインター (デフォルトは CS)

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・デバイス (ファイルまたは Named PIPE)。 piSourceList パラメーターまたは piLobFileList パラメーターの場合のみ指定できます。

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_XBSA_MEDIA
XBSA

SQLU_OTHER_MEDIA
ベンダー・ライブラリー

SQLU_REMOTEFETCH
リモート・フェッチ・メディア (ロードのみ)。 piSourceList パラメーターにのみ指定できます。

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_NULL_MEDIA
NULL (DB2 データベースによって内部的に生成されます)。

SQLU_TAPE_MEDIA
テープ (ベンダー API のみ)

SQLU_PIPE_MEDIA
Named PIPE (ベンダー API のみ)

API とデータ構造構文

```
typedef SQL_STRUCTURE sqlu_media_list
{
   char media_type;
   char filler[3];
   sqlint32 sessions;
   union sqlu_media_list_targets target;
} sqlu_media_list;

union sqlu_media_list_targets
{
   struct sqlu_media_entry *media;
   struct sqlu_vendor *vendor;
   struct sqlu_location_entry *location;
   struct sqlu_statement_entry *pStatement;
   struct sqlu_remotefetch_entry *pRemoteFetch;
};

typedef SQL_STRUCTURE sqlu_media_entry
```

```
{
   sqluint32 reserve_len;
   char media_entry[SQLU_DB_DIR_LEN+1];
} sqlu_media_entry;

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;

typedef SQL_STRUCTURE sqlu_location_entry
{
   sqluint32 reserve_len;
   char location_entry[SQLU_MEDIA_LOCATION_LEN+1];
} sqlu_location_entry;

typedef SQL_STRUCTURE sqlu_statement_entry
{
   sqluint32 length;
   char *pEntry;
} sqlu_statement_entry;

typedef SQL_STRUCTURE sqlu_remotefetch_entry
{
   char *pDatabaseName;
   sqluint32 iDatabaseNameLen;
   char *pUserID;
   sqluint32 iUserIDLen;
   char *pPassword;
   sqluint32 iPasswordLen;
   char *pTableSchema;
   sqluint32 iTableSchemaLen;
   char *pTableName;
   sqluint32 iTableNameLen;
   char *pStatement;
   sqluint32 iStatementLen;
   sqlint32 *pIsolationLevel;
   sqluint32 *piEnableParallelism;
} sqlu_remotefetch_entry;
```

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 API 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 76. 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。

API とデータ構造構文

```
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;
```

sqlupi

この構造は、表の分散マップや分散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 77. 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	分散マップの長さ (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	分散マップ。
SQLD	INTEGER	使用された SQLPARTKEY エレメントの数。つまり、分散キー内のキー・パーツの数。
SQLPARTKEY	構造体	分散キー内の分散列の記述。分散列の最大数は SQL_MAX_NUM_PART_KEYS です。

以下の表に、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 列は、項目のデータ・タイプを表す数値を指定します。

表 78. SQLUPI 構造の SQL データ・タイプおよび長さ

データ・タイプ	SQLTYPE (Null 不可)	SQLTYPE (Null 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

表 78. SQLUPI 構造の SQL データ・タイプおよび長さ (続き)

データ・タイプ	SQLTYPE (Null 不可)	SQLTYPE (Null 可能)	SQLLEN	AIX
10 進数	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
ロング GRAPHIC ストリング	472	473	無視	いいえ

sqlpartkey データ構造パラメーターの記述

sqltype
入力。分散キーのデータ・タイプ。

sqllen 入力。分散キーのデータ長。

API とデータ構造構文

```
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];
};

SQL_STRUCTURE sqlpartkey
{
   unsigned short  sqltype;
   unsigned short  sqllen;
};
```

SQLXA_XID

この構造は、XA トランザクションを識別するために、トランザクション API によって使用されます。トランザクション API グループは、sqlxhfrg、sqlxphcm、sqlxphrl、sqlcspqy、および db2XaListIndTrans の各 API で構成されます。これらの API は、未確定トランザクションの管理に使用されます。

表 79. 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。

表 79. SQLXA-XID 構造のフィールド (続き)

フィールド名	データ・タイプ	説明
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。

API とデータ構造構文

```
struct sqlxa_xid_t {
  sqlint32 formatID;
  sqlint32 gtrid_length;
  sqlint32 bqual_length;
  char data[SQLXA_XIDDATASIZE];
  } ;
typedef struct sqlxa_xid_t SQLXA_XID;
```

付録 A. プリコンパイラーのカスタマイズ API

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。例えば 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 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/support/manualsv9.html

付録 B. DB2 ログ・レコード

DB2 ログ・レコード

このセクションでは、iFilterOption 入力値 DB2READLOG_FILTER_ON が指定されている場合に、db2ReadLog API によって戻される DB2 ログ・レコードの構造を記述します。この値が使用されている場合には、伝搬されたログ・レコードのみ戻されます。伝搬されたログ・レコードのみ説明されています。他のすべてのログ・レコードについては IBM の内部使用専用であるため、詳述しません。

すべての DB2 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、612 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できます。固有トランザクション 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 80. DB2 ログ・レコード

Type	レコード名	説明
データ・マネージャー	629 ページの『表の初期設定のログ・レコード』	新しい永続表の作成。
データ・マネージャー	631 ページの『インポート置換 (切り捨て) ログ・レコード』	インポート置換活動。
データ・マネージャー	631 ページの『NOT LOGGED INITIALLY のアクティブ化ログ・レコード』	ACTIVATE NOT LOGGED INITIALLY 節を含んだ表の変更の活動。
データ・マネージャー	632 ページの『挿入のロールバック・ログ・レコード』	ロールバック行の挿入。
データ・マネージャー	632 ページの『表の再編成のログ・レコード』	REORG のコミット。

表 80. DB2 ログ・レコード (続き)

Type	レコード名	説明
データ・マネージャー	632 ページの『索引の作成、索引のドロップのログ・レコード』	索引活動。
データ・マネージャー	633 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のログ・レコード』	表活動。
データ・マネージャー	633 ページの『表属性変更のログ・レコード』	伝搬、チェック・ペンディング、および付加モード活動。
データ・マネージャー	634 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のログ・レコード』	既存の表への列の追加。
データ・マネージャー	635 ページの『列属性変更のログ・レコード』	列活動。
データ・マネージャー	635 ページの『列属性変更取り消しのログ・レコード』	列活動。
データ・マネージャー	635 ページの『レコードの挿入、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のログ・レコード』	表レコード活動。
データ・マネージャー	639 ページの『空ページへのレコードの挿入、ページからの最後のレコードの削除、ページからの最後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のログ・レコード』	マルチディメンション・クラスタリング (MDC) 表活動。
データ・マネージャー	640 ページの『レコードの更新のログ・レコード』	記憶場所が変更されない場合の行更新。
データ・マネージャー	641 ページの『表またはスキーマの名前変更のログ・レコード』	表またはスキーマ名の活動。
データ・マネージャー	641 ページの『表またはスキーマの名前変更取り消しのログ・レコード』	表またはスキーマ名の活動。
長フィールド・マネージャー	623 ページの『長フィールド・レコードの追加/削除/非更新のログ・レコード』	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	614 ページの『通常コミットのログ・レコード』	トランザクションのコミット。
トランザクション・マネージャー	615 ページの『ヒューリスティック・コミットのログ・レコード』	未確定トランザクションのコミット。
トランザクション・マネージャー	615 ページの『MPP コーディネーター・コミットのログ・レコード』	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。

表 80. DB2 ログ・レコード (続き)

Type	レコード名	説明
トランザクション・マネージャー	616 ページの『MPP 従属ノード・コミットのログ・レコード』	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。
トランザクション・マネージャー	616 ページの『通常打ち切りのログ・レコード』	トランザクションの打ち切り。
トランザクション・マネージャー	617 ページの『ヒューリスティック打ち切りのログ・レコード』	未確定トランザクションの打ち切り。
トランザクション・マネージャー	617 ページの『ローカル・ペンディング・リストのログ・レコード』	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。
トランザクション・マネージャー	618 ページの『グローバル・ペンディング・リストのログ・レコード』	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)。
トランザクション・マネージャー	618 ページの『XA 準備のログ・レコード』	2 フェーズ・コミット環境での XA トランザクションの準備。
トランザクション・マネージャー	619 ページの『MPP 従属ノード準備のログ・レコード』	2 フェーズ・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。
トランザクション・マネージャー	620 ページの『TM 準備のログ・レコード』	2 フェーズ・コミットの一環としての整合トランザクションの準備。ここでデータベースは TM データベースとして動作します。
トランザクション・マネージャー	620 ページの『バックアウト解放のログ・レコード』	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。
トランザクション・マネージャー	621 ページの『アプリケーション情報のログ・レコード』	トランザクションを開始したアプリケーションに関する情報。
トランザクション・マネージャー	621 ページの『フェデレーテッド準備のログ・レコード』	トランザクションに関与するフェデレーテッド・リソース・マネージャーに関する情報。
ユーティリティー・マネージャー	624 ページの『マイグレーションの開始ログ・レコード』	カタログ・マイグレーションの開始。
ユーティリティー・マネージャー	624 ページの『マイグレーションの終了のログ・レコード』	カタログ・マイグレーションの完了。
ユーティリティー・マネージャー	625 ページの『ロードの開始のログ・レコード』	表ロードの開始。
ユーティリティー・マネージャー	625 ページの『バックアップの終了のログ・レコード』	バックアップ活動の完了。
ユーティリティー・マネージャー	626 ページの『表スペースのロールフォワードのログ・レコード』	表スペース・ロールフォワードの完了。

表 80. DB2 ログ・レコード (続き)

Type	レコード名	説明
ユーティリティー・マネージャー	626 ページの『特定時点への表スペース・ロールフォワード開始のログ・レコード』	特定の時点への表スペース・ロールフォワードの開始。
ユーティリティー・マネージャー	626 ページの『特定時点への表スペース・ロールフォワード終了のログ・レコード』	特定の時点への表スペース・ロールフォワードの終了。

ログ・マネージャー・ヘッダー

すべての DB2 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 81. ログ・マネージャーのログ・レコード・ヘッダー

説明	Type	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 (613 ページの表 82 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)
固有トランザクション ID	SQLU_TID[3]	14(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
   一時表                                        0x0004
   表スペース・ロールフォワードの取り消し        0x0008
   単一トランザクション (コミットなし/ロールバック) 0x0010
   条件付きリカバリー可能                        0x0080
   チェック制約処理時の表スペース・ロールフォワード 0x0100
   ```

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 { unsigned char  [6] ;
                     unsigned 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 { unsigned char  [6] ;
                     unsigned short [3] ;
                     }
   ```

4. レコード ID (RID)

   レコードの位置を示す固有の数値。

   ```
   RID: Page number  char [4];
   slot number  char [2];
   ```

表 82. 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
0x0041	通常打ち切り
0x0042	バックアウト解放
0x0043	適合
0x0049	ヒューリスティック打ち切り
0x004A	ロードの開始
0x004E	通常ログ・レコード
0x004F	バックアップの終了
0x0051	グローバル・ペンディング・リスト
0x0052	再実行
0x0055	取り消し
0x0056	マイグレーションの開始
0x0057	マイグレーションの終了
0x0069	通知のみ
0x006F	バックアップ開始
0x0071	特定時点への表スペース・ロールフォワードの終了
0x007B	MPP の準備
0x007C	XA 準備
0x007D	トランザクション・マネージャー (TM) の準備

表 82. 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
0x0084	通常コミット
0x0085	MPP 従属ノード・コミット
0x0086	MPP コーディネーター・ノード・コミット
0x0087	ヒューリスティック・コミット
0x0089	特定時点への表スペース・ロールフォワードの開始
0x008A	ローカル・ペンディング・リスト
0x008B	アプリケーション情報

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 (例えば、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 (UTC) 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。

通常コミットのログ・レコード

このログ・レコードは、単一ノード環境のトランザクションについて、あるいは複数ノード環境のトランザクションについて (そのトランザクションの影響を受けるのは 1 つのノードのみ) 書き込まれます。このログ・レコードは、以下のイベントのいずれか 1 つに続く、トランザクションのコミット時に書き込まれます。

1. ユーザーが COMMIT を発行した
2. CONNECT RESET 中に暗黙のコミットが行われた

表 83. 通常コミットのログ・レコードの構造

説明	Type	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] (ログ・レコードが伝搬可能とマークされている場合)	unsigned short	28 (2)
アプリケーション[1]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 [ ]	30 (可変 [2])
全長: 30 バイト + 伝搬可能な可変長部分 (伝搬不可能な 28 バイト)		

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

ヒューリスティック・コミットのログ・レコード

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 84. ヒューリスティック・コミットのログ・レコードの構造

説明	Type	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] (ログ・レコードが伝搬可能とマークされている場合)	unsigned short	28 (2)
アプリケーション[1]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 [ ]	30 (可変 [2])
全長: 30 バイト + 伝搬可能な可変長部分 (伝搬不可能な 28 バイト)		

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

MPP コーディネーター・コミットのログ・レコード

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 85. MPP コーディネーター・ノード・コミットのログ・レコードの構造

説明	Type	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)
TNL	unsigned char [ ]	50(最大ノード番号/8 + 1)
許可 ID の長さ [1] (ログ・レコードが伝搬可能とマークされている場合)	unsigned short	可変 (2)
アプリケーション[1]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 [2])
全長: 可変		

注:

1. トランザクションに関係していたコーディネーター・ノードを除き、ノードは TNL により定義されます。
2. 許可 ID の長さに基づく変数

MPP 従属ノード・コミットのログ・レコード

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 86. MPP 従属ノード・コミットのログ・レコードの構造

説明	Type	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 (2)
許可 ID の長さ [1] (ログ・レコードが伝搬可能とマークされている場合)	unsigned short	50 (2)
アプリケーション [2] 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 [ ]	52 (可変 [3])
全長: 48 バイト + 可変長部分		

注:

1. トランザクションが単一のデータベース・パーティションのみで行われている場合、これは現在のデータベース・パーティション番号です。そうでない場合は、コーディネーター・パーティション番号です。
2. ログ・レコードが伝搬可能としてマークされている場合
3. 許可 ID の長さに基づく変数

通常打ち切りのログ・レコード

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 87. 通常打ち切りのログ・レコードの構造

説明	Type	オフセット (バイト)
ログ・ヘッダー	LogManagerLogRecordHeader	0 (20)
許可 ID の長さ [1] (ログ・レコードが伝搬可能とマークされている場合)	unsigned short	20 (2)

表 87. 通常打ち切りのログ・レコードの構造 (続き)

説明	Type	オフセット (バイト)
アプリケーション[1]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 [ ]	22 (可変 [2])
全長: 22 バイト + 伝搬可能な可変長部分 (伝搬不可能な 20 バイト)		

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

ヒューリスティック打ち切りのログ・レコード

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 88. ヒューリスティック打ち切りのログ・レコードの構造

説明	Type	オフセット (バイト)
ログ・ヘッダー	LogManagerLogRecordHeader	0 (20)
許可 ID の長さ [1] (ログ・レコードが伝搬可能とマークされている場合)	unsigned short	20 (2)
アプリケーション[1]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 [ ]	22 (可変 [2])
全長: 22 バイト + 伝搬可能な可変長部分 (伝搬不可能な 20 バイト)		

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

ローカル・ペンディング・リストのログ・レコード

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 89. ローカル・ペンディング・リストのログ・レコードの構造

説明	Type	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)

表 89. ローカル・ペンディング・リストのログ・レコードの構造 (続き)

説明	Type	オフセット (バイト)
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・リストのログ・レコード

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 90. グローバル・ペンディング・リストのログ・レコードの構造

説明	Type	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 準備のログ・レコード

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 準備のログ・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 91. XA 準備のログ・レコードの構造

説明	Type	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)
予約済み	sqluint32	可変 (2)
トランザクションの XA ID	SQLXA_XID[1]	可変 (140)
同期ログ情報	variable	可変 (可変)
全長: 182 バイト + 可変長部分		

注: 1. SQLXA_XID ログ・レコードのタイプに関する詳細は、 605 ページの『SQLXA_XID』を参照してください。

MPP 従属ノード準備のログ・レコード

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 従属ノード準備のログ・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 92. MPP 従属ノード準備のログ・レコードの構造

説明	Type	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)
トランザクションの MPP ID	SQLP_GXID[1]	44 (20)
全長: 64 バイト + 可変長部分		

注: 1. SQLP-GXID ログ・レコードは、MPP 環境でトランザクションを識別するために使用されます。

表 93. SQLP-GXID 構造のフィールド

フィールド名	データ・タイプ	説明
FORMATID	INTEGER	GXID 形式 ID
GXID_LENGTH	INTEGER	GXID の長さ
BQAL_LENGTH	INTEGER	ブランチ ID の長さ
DATA	CHAR(8)	最初の 2 バイトはノード番号、残りはトランザクション ID

TM 準備のログ・レコード

このログ・レコードは、単一パーティション・データベース環境、または MPP のコーディネーター・パーティション上の DB2 整合トランザクションに関して書き込まれます。ここでデータベースは TM データベースとして動作します。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。

表 94. TM 準備のログ・レコードの構造

説明	Type	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)
予約済み	sqluint32	可変 (2)
トランザクションの XA ID	SQLXA_XID	可変 (140)
同期ログ情報	variable	可変 (可変)
全長: 182 バイト + 可変長部分		

バックアウト解放のログ・レコード

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) が含まれます。特定のシナリオでは、バックアウト解放のログ・レコードにもログ・データが入ります。これはオフセット 26 で始まり、対応するデータ・マネージャー・ログ・レコードにログ記録されたデータと同じです。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 95. バックアウト解放のログ・レコードの構造

説明	Type	オフセット (バイト)
ログ・ヘッダー	LogManagerLogRecordHeader	0 (20)
Complsn	SQLP_LSN	20 (6)
ログ・データ[1]	variable	変数
全長: 26 バイト + 可変長部分		

注: 1. 特定のシナリオでのみ適用され、使用される場合、ログ・ヘッダー内のログ・レコード全体の長さは 26 バイトより長くなります。

アプリケーション情報のログ・レコード

このログ・レコードには、このトランザクションを開始したアプリケーションに関する情報が含まれています。

表 96. アプリケーション情報のログ・レコードの構造

説明	Type	オフセット (バイト)
ログ・ヘッダー	LogManagerLogRecordHeader	0 (20)
トランザクションの開始時刻	sqluint32	20 (4)
予約済み	char[ ]	24 (16)
コード・ページ	sqluint32	40 (4)
アプリケーション名の長さ	sqluint32	44 (4)
Application Name	char [ ]	48 (可変)
アプリケーション ID の長さ	sqluint32	可変 (4)
アプリケーション ID	char [ ]	可変 (可変)
シーケンス番号の長さ	sqluint32	可変 (4)
シーケンス番号	char [ ]	可変 (可変)
クライアントによって使用されたデータベース別名の長さ	sqluint32	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 [ ]	可変 (可変)
許可 ID の長さ	sqluint32	可変 (4)
許可 ID	char [ ]	可変 (可変)
全長: 64 バイト + 可変長部分		

フェデレーテッド準備のログ・レコード

このログ・レコードには、トランザクションに関与したフェデレーテッド・リソース・マネージャーに関する情報が含まれています。

表 97. フェデレーテッド準備のログ・レコードの構造

説明	Type	オフセット (バイト)
ログ・ヘッダー	LogManagerLogRecordHeader	0 (20)

表 97. フェデレーテッド準備のログ・レコードの構造 (続き)

説明	Type	オフセット (バイト)
リソース・マネージャーの数	sqluint32	20 (4)
許可 ID の長さ	sqluint16	24 (2)
暗号化されたパスワードの長さ	sqluint16	26 (2)
許可 ID	char [128]	28 (128)
暗号化されたパスワード	char [255]	156 (255)
リソース・マネージャー項目	variable	411 (可変)
全長: 411 バイト + 可変長部分		

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 が有効な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。

注: LOB マネージャーのログ・レコードは伝搬できないため、ドキュメント化されません。

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 ステートメントが指定された) 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。長フィールド・レコードの削除/追加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 98. 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	Type	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 (表 99 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)
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 99. 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/非更新のログ・レコード

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 100. 長フィールド・レコードの追加/削除/非更新のログ・レコードの構造

説明	Type	オフセット (バイト)
ログ・ヘッダー	LongFieldLogRecordHeader	0 (10)
内部	内部	10 (1)
元の操作タイプ [1]	char	11 (1)
列 ID [2]	unsigned short	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. 長フィールドのデータ長を 512 バイトのセクターを単位として示したもの (実際のデータ長は、続く挿入/削除/更新ログ・レコードの中に書式設定済みユーザー・データ・レコードの一部としてログ記録される、長フィールド記述子 (LF 記述子) の最初の 4 バイトとして記録されます)。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・データが挿入、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。

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 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。

- マイグレーション
- 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。これらのユーティリティーに関して伝搬可能なログ・レコードのみ説明されています。

マイグレーションの開始ログ・レコード

このログ・レコードは、カタログ・マイグレーションの開始と関連しています。

表 101. マイグレーションの開始ログ・レコードの構造

説明	Type	オフセット (バイト)
ログ・ヘッダー	LogManagerLogRecordHeader	0 (20)
マイグレーションの開始時刻	char[ ]	20 (10)
マイグレーション元のリリース	unsigned short	30 (2)
マイグレーション先のリリース	unsigned short	32 (2)
全長: 34 バイト		

マイグレーションの終了のログ・レコード

このログ・レコードは、カタログ・マイグレーションの正常終了と関連しています。

表 102. マイグレーションの終了のログ・レコードの構造

説明	Type	オフセット (バイト)
ログ・ヘッダー	LogManagerLogRecordHeader	0 (20)

表 102. マイグレーションの終了のログ・レコードの構造 (続き)

説明	Type	オフセット (バイト)
マイグレーションの終了時刻	char[ ]	20 (10)
マイグレーション先のリリース	unsigned short	30 (2)
全長: 32 バイト		

ロードの開始のログ・レコード

このログ・レコードは、ロードの開始と関連しています。

伝搬可能なのはロードのログ・レコードだけです。それは、ロード・フェーズの開始時点で書き込まれます。このログ・レコードを、伝搬可能ではないセット・フェーズの開始時に書き込まれるロード開始レコードのその他のタイプと混同しないようにしてください。

ログ・レコードの伝搬のため、ログ開始のログ・レコードを読んだなら、その後、その特定の表のログ・レコードに関しては、ターゲット表への伝搬を継続しないことをお勧めします。ロード開始のログ・レコードの後、コールド・リスタートなどが発生するまでの間は、トランザクション境界に関係なく、ロード対象の表に属するログ・レコードのうち伝搬可能なものはすべて無視できます。ソース表とターゲット表を同期するには、コールド・リスタートが必要です。

表 103. ロードの開始のログ・レコードの構造

説明	Type	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	sqluint32	28 (4)
オブジェクト・プールのリスト	variable	32 (可変)
全長: 32 バイト + 可変長部分		

バックアップの終了のログ・レコード

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 104. バックアップの終了のログ・レコードの構造

説明	Type	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のログ・レコード

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 105. 表スペースのロールフォワードのログ・レコードの構造

説明	Type	オフセット (バイト)
ログ・ヘッダー	LogManagerLogRecordHeader	0 (20)
表スペース ID	sqluint32	20 (4)
全長: 24 バイト		

特定時点への表スペース・ロールフォワード開始のログ・レコード

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、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 106. 特定時点への表スペース・ロールフォワード開始のログ・レコードの構造

説明	Type	オフセット (バイト)
ログ・ヘッダー	LogManagerLogRecordHeader	0 (20)
このログ・レコードのタイム・スタンプ	sqluint64	20 (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	sqluint32	28 (4)
ロールフォワードされるプールの数	sqluint32	32 (4)
全長: 36 バイト		

特定時点への表スペース・ロールフォワード終了のログ・レコード

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、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 107. 特定時点への表スペース・ロールフォワード終了のログ・レコードの構造

説明	Type	オフセット (バイト)
ログ・ヘッダー	LogManagerLogRecordHeader	0 (20)
このログ・レコードのタイム・スタンプ	sqluint64	20 (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	28 (4)

表 107. 特定時点への表スペース・ロールフォワード終了のログ・レコードの構造 (続き)

説明	Type	オフセット (バイト)
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	sqluint32	32 (4)
全長: 36 バイト		

イベント・ログのイベント・タイミングを相互に区別できるだけの十分な精度を確保するため、2 つのタイム・スタンプ・フィールドが必要です。最初のタイム・スタンプでは、ログの書き込み時刻を秒単位の精度で示すために 8 バイトが使用されます。そのタイム・スタンプの最初の 4 バイトが秒を示します。 1 秒の間であっても多くのアクションが発生する可能性があるため、イベントの発生順序を確定するためには、もっと高い精度が必要になります。 2 番目のタイム・スタンプ・フィールドの 4 バイトは、ナノ秒単位の値を記録するために使用されます。 2 つのログ・レコードのログ・レコード・タイム・スタンプが同一だった場合には、それに関連するログ・イベントの発生順序を決定するために、さらに 4 バイトのタイム・スタンプ・フィールドを使用できます。

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 108. DMS ログ・レコード・ヘッダーの構造 (DMSLogRecordHeader)

説明	Type	オフセット (バイト)
コンポーネント ID (=1)	unsigned char	0(1)
関数 ID (表 109を参照)	unsigned char	1(1)
表 ID 表スペース ID 表 ID	 unsigned short unsigned short	 2(2) 4(2)
全長: 6 バイト		

表 109. 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

表 109. 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 (続き)

値	定義
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	列を変更する
115	列の変更を取り消す
122	スキーマまたは表を名前変更する
123	スキーマまたは表の名前変更を取り消す
124	表属性を変更する
128	表を初期設定する
131	空のページへのレコードの挿入を取り消す
161	レコードを削除する
162	レコードを挿入する
163	レコードを更新する
164	レコードを削除して空ページにする
165	空ページへレコードを挿入する
166	レコードを削除して空ページにする操作の取り消し
167	複数レコードの挿入
168	複数レコードの挿入の取り消し

表 110. DOM ログ・レコード・ヘッダーの構造 (DOMLogRecordHeader)

説明	Type	オフセット (バイト)
コンポーネント ID (=4)	unsigned char	0(1)
関数 ID (表 111を参照)	unsigned char	1(1)
オブジェクト ID 表スペース ID オブジェクト ID	 unsigned short unsigned short	 2(2) 4(2)
表 ID 表スペース ID 表 ID	 unsigned short unsigned short	 6(2) 8(2)
オブジェクト・タイプ	unsigned char	10(1)
Flags	unsigned char	11(1)
全長: 12 バイト		

表 111. 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

表 111. 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 (続き)

値	定義
4	表のドロップ
5	表のドロップを取り消す
11	表を切り捨てる (インポート置換)
12	NOT LOGGED INITIALLY のアクティブ化
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。

関数 ID の短縮名が 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

表の初期設定のログ・レコード

表の初期設定のログ・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 およびブロック・マップ記憶オブジェクトを作成するログ・レコードの後、LF および LOB 記憶オブジェクトを作成するログ・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。関数 ID は 128 です。

表 112. 表の初期設定ログ・レコードの構造

説明	Type	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(8)
内部	内部	14(74)
表記述の長さ	sqluint32	88(4)
表記述レコード	variable	92 (可変)
全長: 92 バイト + 表記述レコードの長さ		

表 113. 表記述レコード

説明	Type	オフセット (バイト)
レコード・タイプ	unsigned char	0(1)
内部	内部	1(1)

表 113. 表記述レコード (続き)

説明	Type	オフセット (バイト)
列の数	unsigned short	2(2)
列記述子の配列	variable long	変数
全長: 4 バイト + 列記述子配列の長さ		

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
DECFLOAT64   0x0006
DECFLOAT128  0x0007
CHAR         0x0100
VARCHAR      0x0101
LONG VARCHAR 0x0104
DATE         0x0105
TIME         0x0106
TIMESTAMP    0x0107
BLOB         0x0108
CLOB         0x0109
STRUCT       0x010D
XMLTYPE      0x0112
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 0 は unsigned char、精度 (全長)、バイト 1 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、生成、またはデフォルト入力の短縮

```
ISNULL                 0x0001
NONULLS                0x0002
TYPE_DEFAULT           0x0004
USER_DEFAULT           0x0008
GENERATED              0x0040
COMPRESS_SYSTEM_DEFAULT 0x0080
```

- フィールド・オフセット (unsigned short、2 バイト)。これは、ユーザー・レコードの固定長部分の開始位置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 列記述子配列

(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) ログ・レコード

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・ヘッダー内の表 ID により、切り捨ての対象である表が識別されます (IMPORT REPLACE)。これは通常ログ・レコードです。関数 ID は 11 です。

表 114. インポート置換 (切り捨て) ログ・レコードの構造

説明	Type	オフセット (バイト)
ログ・ヘッダー	DOMLogRecordHeader	0(12)
内部	内部	12 (可変)
全長: 12 バイト + 可変長部分		

NOT LOGGED INITIALLY のアクティブ化ログ・レコード

NOT LOGGED INITIALLY のアクティブ化ログ・レコードは、ユーザーが ACTIVATE NOT LOGGED INITIALLY 節を含む ALTER TABLE ステートメントを発行した場合に書き込まれます。これは通常ログ・レコードです。これは、関数 ID 12 です。

表 115. NOT LOGGED INITIALLY のアクティブ化ログ・レコードの構造

説明	Type	オフセット (バイト)
ログ・ヘッダー	DOMLogRecordHeader	0(12)
内部	内部	12(4)
長い表スペース ID*	unsigned short	16(2)
索引表スペース ID*	unsigned short	18(2)
索引オブジェクト ID	unsigned short	20(2)
LF オブジェクト ID	unsigned short	22(2)
LOB オブジェクト ID	unsigned short	24(2)
XML オブジェクト ID	unsigned short	26(2)
全長: 28 バイト		

** DOM ヘッダー内の表スペース ID と同じ。これは、データベース内で定義されている各表スペースの固有 ID です。*

挿入のロールバック・ログ・レコード

挿入のロールバックのログ・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。関数 ID は 110 です。

表 116. 挿入のロールバック・ログ・レコードの構造

説明	Type	オフセット (バイト)
ログ・ヘッダー	DMSLogRecordHeader	0(6)
内部	内部	6(2)
レコード長	unsigned short	8(2)
フリー・スペース	unsigned short	10(2)
RID	char[]	12(6)
全長: 16 バイト		

表の再編成のログ・レコード

表の再編成のログ・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。関数 ID は 35 です。

表 117. 表の REORG のログ・レコードの構造

説明	Type	オフセット (バイト)
ログ・ヘッダー	DOMLogRecordHeader	0(12)
内部	variable	12(476)
索引トークン [1]	unsigned short	488(2)
TEMPORARY 表スペース ID[2]	unsigned short	490(2)
LONG TEMPORARY 表スペース ID	unsigned short	492(2)
全長: 494 バイト		

注:

1. 索引トークンの値が 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. TEMPORARY 表スペース ID の値が 0 でない場合、これは再編成された表の作成に使用された SYSTEM TEMPORARY 表スペースです。

索引の作成、索引のドロップのログ・レコード

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 の場合、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。

これは通常ログ・レコードです。この関数 ID は 2 (索引の作成) または 3 (索引のドロップ) のいずれかです。

表 118. 索引の作成、索引のドロップのログ・レコードの構造

説明	Type	オフセット (バイト)
ログ・ヘッダー	DOMLogRecordHeader	0(12)
内部	内部	12(2)
索引トークン	unsigned short	14(2)
索引ルート・ページ	sqluint32	16(4)
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のログ・レコード

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。 MDC 表の作成の場合、ブロック・マップ・オブジェクトの作成での表作成のログ・レコードもあります。 DATA オブジェクト (および該当する場合、ブロック・マップ・オブジェクト) は、CREATE TABLE 操作中に、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。関数 ID は、101 (表の作成)、4 (表のドロップ)、 130 (表作成のロールバック)、または 5 (表ドロップのロールバック) のいずれかです。

表 119. 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のログ・レコードの構造

説明	Type	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		

表属性変更のログ・レコード

表属性変更のログ・レコードは、表の状態が ALTER TABLE ステートメントを使用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。これは通常ログ・レコードまたは補正ログ・レコードです。関数 ID は 124 です。

表 120. 表属性変更、表属性変更取り消し

説明	Type	オフセット (バイト)
ログ・ヘッダー	DMSLogRecordHeader	0(6)
ビット (属性) 変更マスク	sqluint64	6(8)
ビット (属性) 変更値	sqluint64	14(8)
全長: 22 バイト		

属性ビット

```
0x00000001   Propagation
0x00000002   Check Pending
0x00000010   Value Compression
0x00010000   Append Mode
0x00200000   LF Propagation
```

他のすべてのビットは内部使用されます。

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のログ・レコード

表の変更による列の追加のログ・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。
  - 各列に 1 つの、8 バイトのエレメント。
  - LOB 列がある場合は、各 LOB 列に 1 つの、12 バイトのエレメント。これは 8 バイトのエレメントの配列に続きます。

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。関数 ID は、102 (列の追加) または 104 (列追加の取り消し) です。

表 121. 表の変更による列の追加、列追加のロールバックのログ・レコードの構造

説明	Type	オフセット (バイト)
ログ・ヘッダー	DMSLogRecordheader	0(6)
内部	内部	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	variable	可変 (可変)
全長: 16 バイト + 2 セットの平行配列。		

配列エレメント:

1. この配列内のエレメントの長さは、次のように定義されています。
   - このエレメントが列記述子なら、エレメントの長さは 8 バイトです。
   - このエレメントが LOB 列記述子なら、エレメントの長さは 12 バイトです。

列記述子の配列または LOB 列記述子の配列については、629 ページの表 113 の後の説明を参照してください。

列属性変更のログ・レコード

関数 ID は 113 です。

表 122. 列属性変更のログ・レコードの構造

説明	Type	オフセット (バイト)
ログ・ヘッダー	DMSLogRecordheader	0(6)
列 ID	unsigned short	6(2)
古い列定義	列記述子[1]	8(8)
新しい列定義	列記述子[1]	16(8)
全長: 24 バイト + レコード長。		

[1]列記述子の配列については、 629 ページの表 113 の後の説明を参照してください。

列属性変更取り消しのログ・レコード

関数 ID は 115 です。

表 123. 列属性変更取り消しのログ・レコードの構造

説明	Type	オフセット (バイト)
ログ・ヘッダー	DMSLogRecordHeader	0(6)
列 ID	unsigned short	6(2)
古い列定義	列記述子[1]	8(8)
新しい列定義	列記述子[1]	16(8)
全長: 24 バイト + レコード長。		

[1]列記述子の配列については、 629 ページの表 113 の後の説明を参照してください。

レコードの挿入、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のログ・レコード

これらのログ・レコードは、行が表に挿入されるとき、あるいは削除または更新がロールバック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のログ・レコードも、更新中に、修正されたレコード・データに合わせて更新対象レコードの位置を変更しなければならない場合に生成できます。レコード挿入ログ・レコードは、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。関数 ID は 162 (挿入)、111 (削除のロールバック)、または 112 (更新のロールバック) です。

表 124. レコードの挿入、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のログ・レコードの構造

説明	Type	オフセット (バイト)
ログ・ヘッダー	DMSLogRecordHeader	0(6)
内部	内部	6(2)
レコード長	unsigned short	8(2)
フリー・スペース	unsigned short	10(2)

表 124. レコードの挿入、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のログ・レコードの構造 (続き)

説明	Type	オフセット (バイト)
RID	char[]	12(6)
レコード・オフセット	unsigned short	18(2)
レコードのヘッダーとデータ	variable	20 (可変)
全長: 20 バイト + レコード長		

以下は、レコードのヘッダーとデータに関する詳細です。

レコード・ヘッダー

- 4 バイト
- レコード・タイプ (unsigned char、1 バイト)。
- 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)。
- 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。
- 次のフィールドは、レコード・タイプに 1 ビットが設定されたユーザー・データ・レコードに適用されます。
  - 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。
  - 書式レコード (すべての固定長列の後に可変長列が続く)。
- 次のフィールドは、レコード・タイプに 2 ビットが設定されたユーザー・データ・レコードに適用されます。
  - 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。 638 ページの『VALUE COMPRESSION を使用した表の定様式のユーザー・データ・レコード』を参照してください。

    注: オフセット配列には、1 + 列数が含まれます。
  - 書式レコード (オフセット配列の後にデータ列が続く)。

ユーザー・レコードは、次の特性によって完全に指定されます。

1. 外部レコード・タイプが 0、または
2. 外部レコード・タイプが 0x10、または
3. 外部レコード・タイプに 0x04 ビットが設定されており、さらに
1. 内部レコード・タイプに 0x01 ビットが設定されているか、または
2. 内部レコード・タイプに 0x02 ビットが設定されている。

注: 行圧縮とデータ・キャプチャーには、互換性がありません。

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 オプションを使用しない形式のレコードの場合、すべてのフィールドに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
- 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。このフィールドの長さは、(精度 + 2)/2 です。符号ニブルは正 (+) の場合に xC で、負 (-) の場合に 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。例えば、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。例えば、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 | マイクロ秒) の形式の 10 バイトのパック 10 進数です。

- VARCHAR LONG VARCHAR 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。

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 125. 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 • length • NULL フラグ • フィールド・オフセット	LOB 情報

注: 詳しくは、 629 ページの表 112 の後の説明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 126. 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: 詳しくは、 635 ページの表 124 の後の説明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のユーザー・データ・レコード

VALUE COMPRESSION オプションを使用した形式のレコードは、オフセット配列とデータ部分で構成されます。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 (例えば、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 (0x8000) 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。 (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、オフセット配列に列データが続きます。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 127. 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: 詳しくは、 635 ページの表 124 の後の説明を参照してください。

空ページへのレコードの挿入、ページからの最後のレコードの削除、ページからの最後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のログ・レコード

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。空ページへのレコード挿入のログ・レコードは、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り、かつそのページがブロックの先頭ページではない場合に書き込まれます。このログ・レコードでは、ページへの挿入に加えて、そのページが空ではないことを示すようブロックの先頭ページのビットが更新されたことも記録されます。ページからの最後のレコードの削除のログ・レコードは、ブロックの先頭ページではないページから最後のレコードが削除されたときに書き込まれます。このログ・レコードでは、ページからの削除に加えて、そのページが空であることを示すようブロックの先頭ページのビットが更新されたことも記録されます。空ページへのレコード挿入のログ・レコード、およびページからの最後のレコードの削除のログ・レコードは、通常ログ・レコードです。レコード削除のロールバックのログ・レコードおよびレコード挿入のロールバックのログ・レコードは、補正ログ・レコードです。関数 ID は、165 (空ページへのレコードの挿入)、164 (ページからの最後のレコードの削除)、166 (ページからの最後のレコードの削除のロールバック)、または 131 (空ページへのレコードの挿入のロールバック) です。

表 128. 空ページへのレコードの挿入のロールバック

説明	Type	オフセット (バイト)
ログ・ヘッダー	DMSLogRecordHeader	0(6)
内部	内部	6(2)
レコード長	unsigned short	8(2)
フリー・スペース	unsigned short	10(2)
RID	char[]	12(6)

表 128. 空ページへのレコードの挿入のロールバック (続き)

説明	Type	オフセット (バイト)
内部	内部	18(2)
ブロックの先頭ページ	sqluint32	20(4)
全長: 24 バイト		

表 129. 空ページへのレコードの挿入、ページからの最後のレコードの削除のロールバック、ページからの最後のレコードの削除

説明	Type	オフセット (バイト)
ログ・ヘッダー	DMSLogRecordHeader	0(6)
内部	内部	6(2)
レコード長	unsigned short	8(2)
フリー・スペース	unsigned short	10(2)
RID	char[]	12(6)
内部	内部	18(2)
ブロックの先頭ページ	sqluint32	20(4)
レコード・オフセット	unsigned short	24(2)
レコードのヘッダーとデータ	variable	26(可変)
全長: 26 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、 635 ページの表 124 の後の説明を参照してください。

レコードの更新のログ・レコード

レコードの更新のログ・レコードは、行が更新され、その保管場所が同じ場合に書き込まれます。レコードの形式は 2 つあり、それらはレコードの挿入 (およびログ・レコードの削除) のログ・レコードと同じです (635 ページの『レコードの挿入、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のログ・レコード』 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。関数 ID は 163 です。

表 130. レコードの更新のログ・レコードの構造

説明	Type	オフセット (バイト)
ログ・ヘッダー	DMSLogRecordHeader	0(6)
内部	内部	6(2)
新しいレコード長	unsigned short	8(2)
フリー・スペース	unsigned short	10(2)
RID	char[]	12(6)
レコード・オフセット	unsigned short	18(2)
古いレコードのヘッダーとデータ	variable	20 (可変)
ログ・ヘッダー	DMSLogRecordHeader	可変 (6)
内部	内部	可変 (2)

表 130. レコードの更新のログ・レコードの構造 (続き)

説明	Type	オフセット (バイト)
古いレコード長	unsigned short	可変 (2)
フリー・スペース	unsigned short	可変 (2)
RID	char[]	可変 (6)
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)
全長: 40 バイト + 2 つのレコード長		

表またはスキーマの名前変更のログ・レコード

表スキーマ名前変更のログ・レコードは、表またはスキーマの名前が変更された場合に書き込まれます。これは、関数 ID 122 です。

表 131. 表またはスキーマの名前変更のログ・レコードの構造

説明	Type	オフセット (バイト)
ログ・ヘッダー	DMSLogRecordHeader	0(6)
全長: 6 バイト		

表またはスキーマの名前変更のログ・レコードに、表またはスキーマ・オブジェクトの古い名前と新しい名前に関する情報は含まれていません。表またはスキーマの名前が変更されると、システム・カタログ表に対するさまざまな操作に関連した挿入、更新、および削除のログ・レコードが別個に生成されます。

表またはスキーマの名前変更取り消しのログ・レコード

表スキーマ名前変更取り消しのログ・レコードは、表またはスキーマの名前変更がロールバックされた場合に書き込まれます。これは、関数 ID 123 です。

表 132. 表またはスキーマ名前変更取り消しのログ・レコードの構造

説明	Type	オフセット (バイト)
ログ・ヘッダー	DMSLogRecordHeader	0(6)
全長: 6 バイト		

表またはスキーマの名前変更のログ・レコードに、表またはスキーマ・オブジェクトの古い名前と新しい名前に関する情報は含まれていません。表またはスキーマの名前が変更されると、システム・カタログ表に対するさまざまな操作に関連した挿入、更新、および削除のログ・レコードが別個に生成されます。

複数レコードの挿入、複数レコードの挿入の取り消し

これらのログ・レコードは、複数の行が表の同じページに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。関数 ID は 167 と 168 です。

表 133. 複数レコードの挿入の構造

説明	Type	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)
可変部分の長さ	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 134 と 表 135 を参照してください。	variable	20 (可変)
全長: 20 バイト + レコード長		

表 134. レコード記述 (レコードごとに 1 つ)

説明	Type	オフセット (バイト)
RID	unsigned char[6]	0(6)
レコード・オフセット	unsigned short	6(2)
レコードのヘッダーとデータ	variable	8 (可変)
全長: 8 バイト + レコード長		

表 135. ロールバック記述 (レコードごとに 1 つ)

説明	Type	オフセット (バイト)
RID	unsigned char[6]	0(6)
レコード・オフセット	unsigned short	6(2)
全長: 8 バイト		

レコードのヘッダーとデータの詳細については、 635 ページの表 124 の後の説明を参照してください。

付録 C. DB2 技術情報の概説

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 (タスク、概念、およびリファレンス・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
- DB2 資料
  - PDF ファイル (ダウンロード可能)
  - PDF ファイル (DB2 PDF DVD に含まれる)
  - 印刷資料
- コマンド行ヘルプ
  - コマンド・ヘルプ
  - メッセージ・ヘルプ

注: *DB2 インフォメーション・センター* のトピックは、PDF やハードコピー資料よりも頻繁に更新されます。最新の情報を入手するには、資料の更新が発行されたときにそれをインストールするか、ibm.com® にある *DB2 インフォメーション・センター* を参照してください。

技術資料、ホワイト・ペーパー、IBM Redbooks® 資料などのその他の DB2 技術情報には、オンライン (ibm.com) 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 (http://www.ibm.com/software/data/sw-library/) にアクセスしてください。

資料についてのフィードバック

DB2 の資料についてのお客様からの貴重なご意見をお待ちしています。DB2 の資料を改善するための提案については、db2docs@ca.ibm.com まで E メールを送信してください。DB2 の資料チームは、お客様からのフィードバックすべてに目を通しますが、直接お客様に返答することはありません。お客様が関心をお持ちの内容について、可能な限り具体的な例を提供してください。特定のトピックまたはヘルプ・ファイルについてのフィードバックを提供する場合は、そのトピック・タイトルおよび URL を含めてください。

DB2 お客様サポートに連絡する場合には、この E メール・アドレスを使用しないでください。資料を参照しても、DB2 の技術的な問題が解決しない場合は、お近くの IBM サービス・センターにお問い合わせください。

IBM Information Management 製品の使用をより容易にするために IBM にご協力いただける場合は、コンシューマビリティーに関する次のアンケートにご回答ください。http://www.ibm.com/software/data/info/consumability-survey/

DB2 テクニカル・ライブラリー (ハードコピーまたは PDF 形式)

以下の表は、DB2 ライブラリーについて説明しています。DB2 ライブラリー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。英語の DB2 バージョン 9.5 のマニュアル (PDF 形式) とその翻訳版は、 www.ibm.com/support/docview.wss?rs=71&uid=swg2700947 からダウンロードできます。

この表には印刷資料が入手可能かどうかが示されていますが、国または地域によっては入手できない場合があります。

資料番号は、資料が更新される度に大きくなります。資料を参照する際は、以下にリストされている最新版であることを確認してください。

注: *DB2 インフォメーション・センター* は、PDF やハードコピー資料よりも頻繁に更新されます。

表 136. DB2 の技術情報

資料名	資料番号	印刷資料が入手可能かどうか	最終更新
管理 API リファレンス	SC88-4431-02	入手可能	2009 年 4 月
管理ルーチンおよびビュー	SC88-4435-02	入手不可	2009 年 4 月
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	SC88-4433-02	入手可能	2009 年 4 月
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	SC88-4434-02	入手可能	2009 年 4 月
コマンド・リファレンス	SC88-4432-02	入手可能	2009 年 4 月
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	SC88-4421-02	入手可能	2009 年 4 月
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	SC88-4423-02	入手可能	2009 年 4 月
データ・サーバー、データベース、およびデータベース・オブジェクトのガイド	SC88-4259-02	入手可能	2009 年 4 月
データベース・セキュリティー・ガイド	SC88-4418-02	入手可能	2009 年 4 月
ADO.NET および OLE DB アプリケーションの開発	SC88-4425-02	入手可能	2009 年 4 月

表 136. DB2 の技術情報 (続き)

資料名	資料番号	印刷資料が入手可能かどうか	最終更新
組み込み SQL アプリケーションの開発	SC88-4426-02	入手可能	2009 年 4 月
Java™ アプリケーションの開発	SC88-4427-02	入手可能	2009 年 4 月
Perl および PHP アプリケーションの開発	SC88-4428-02	入手不可	2009 年 4 月
SQL および外部ルーチンの開発	SC88-4429-02	入手可能	2009 年 4 月
データベース・アプリケーション開発の基礎	GC88-4430-02	入手可能	2009 年 4 月
DB2 インストールおよび管理 概説 (Linux および Windows 版)	GC88-4439-02	入手可能	2009 年 4 月
国際化対応ガイド	SC88-4420-02	入手可能	2009 年 4 月
メッセージ・リファレンス 第 1 巻	GI88-4109-01	入手不可	2009 年 4 月
メッセージ・リファレンス 第 2 巻	GI88-4110-01	入手不可	2009 年 4 月
マイグレーション・ガイド	GC88-4438-02	入手可能	2009 年 4 月
Net Search Extender 管理およびユーザーズ・ガイド	SC88-4630-02	入手可能	2009 年 4 月
パーティションおよびクラスタリングのガイド	SC88-4419-02	入手可能	2009 年 4 月
Query Patroller 管理およびユーザーズ・ガイド	SC88-4611-01	入手可能	2009 年 4 月
IBM データ・サーバー・クライアント機能 概説およびインストール	GC88-4441-02	入手不可	2009 年 4 月
DB2 サーバー機能 概説およびインストール	GC88-4440-02	入手可能	2009 年 4 月
Spatial Extender および Geodetic Data Management Feature ユーザーズ・ガイドおよびリファレンス	SC88-4629-02	入手可能	2009 年 4 月
SQL リファレンス 第 1 巻	SC88-4436-02	入手可能	2009 年 4 月
SQL リファレンス 第 2 巻	SC88-4437-02	入手可能	2009 年 4 月

表 136. DB2 の技術情報 (続き)

資料名	資料番号	印刷資料が入手可能かどうか	最終更新
システム・モニター ガイドおよびリファレンス	SC88-4422-02	入手可能	2009 年 4 月
Text Search ガイド	SC88-4424-01	入手可能	2009 年 4 月
問題判別ガイド	GI88-4108-02	入手不可	2009 年 4 月
データベース・パフォーマンスのチューニング	SC88-4417-02	入手可能	2009 年 4 月
Visual Explain チュートリアル	SC88-4449-00	入手不可	
新機能	SC88-4445-02	入手可能	2009 年 4 月
ワークロード・マネージャー ガイドおよびリファレンス	SC88-4446-02	入手可能	2009 年 4 月
pureXML ガイド	SC88-4447-02	入手可能	2009 年 4 月
XQuery リファレンス	SC88-4448-02	入手不可	2009 年 4 月

表 137. DB2 Connect 固有の技術情報

資料名	資料番号	印刷資料が入手可能かどうか	最終更新
DB2 Connect Personal Edition 概説およびインストール	GC88-4443-02	入手可能	2009 年 4 月
DB2 Connect サーバー機能 概説およびインストール	GC88-4444-02	入手可能	2009 年 4 月
DB2 Connect ユーザーズ・ガイド	SC88-4442-02	入手可能	2009 年 4 月

表 138. Information Integration の技術情報

資料名	資料番号	印刷資料が入手可能かどうか	最終更新
Information Integration: フェデレーテッド・システム 管理ガイド	SC88-4166-01	入手可能	2008 年 3 月
Information Integration: レプリケーションおよびイベント・パブリッシングのための ASNCLP プログラム・リファレンス	SC88-4167-02	入手可能	2008 年 3 月

表 138. *Information Integration* の技術情報 (続き)

資料名	資料番号	印刷資料が入手可能かどうか	最終更新
Information Integration: フェデレーテッド・データ・ソース 構成ガイド	SC88-4185-01	入手不可	
Information Integration: SQL レプリケーション ガイドおよびリファレンス	SC88-4168-01	入手可能	2008 年 3 月
Information Integration: レプリケーションとイベント・パブリッシング 概説	GC88-4187-01	入手可能	2008 年 3 月

DB2 の印刷資料の注文方法

DB2 の印刷資料が必要な場合、オンラインで購入することができますが、すべての国および地域で購入できるわけではありません。 DB2 の印刷資料については、IBM 営業担当員にお問い合わせください。 *DB2 PDF* ドキュメンテーション DVD の一部のソフトコピー・ブックは、印刷資料では入手できないことに留意してください。例えば、「*DB2 メッセージ・リファレンス*」はどちらの巻も印刷資料としては入手できません。

DB2 PDF ドキュメンテーション DVD で利用できる DB2 の印刷資料の大半は、IBM に有償で注文することができます。国または地域によっては、資料を IBM Publications Center からオンラインで注文することもできます。お客様の国または地域でオンライン注文が利用できない場合、DB2 の印刷資料については、IBM 営業担当員にお問い合わせください。 DB2 PDF ドキュメンテーション DVD に収録されている資料の中には、印刷資料として提供されていないものもあります。

注: 最新で完全な DB2 資料は、DB2 インフォメーション・センター (http://publib.boulder.ibm.com/infocenter/db2luw/v9r5) で参照することができます。

DB2 の印刷資料は以下の方法で注文することができます。

- 日本 IBM 発行のマニュアルはインターネット経由でご購入いただけます。詳しくは http://www.ibm.com/shop/publications/order の「ご注文について」をご覧ください。資料の注文情報にアクセスするには、お客様の国、地域、または言語を選択してください。その後、各ロケーションにおける注文についての指示に従ってください。
- DB2 の印刷資料を IBM 営業担当員に注文するには、以下のようにします。
  1. 以下の Web サイトのいずれかから、営業担当員の連絡先情報を見つけてください。
     - IBM Directory of world wide contacts (www.ibm.com/planetwide)
     - IBM Publications Web サイト (http://www.ibm.com/shop/publications/order) 国、地域、または言語を選択し、お客様の所在地に該当する Publications ホ

ーム・ページにアクセスしてください。このページから、「このサイトについて」のリンクにアクセスしてください。

2. 電話をご利用の場合は、DB2 資料の注文であることをご指定ください。
3. 担当者に、注文する資料のタイトルと資料番号をお伝えください。タイトルと資料番号は、 644 ページの『DB2 テクニカル・ライブラリー (ハードコピーまたは PDF 形式)』でご確認いただけます。

コマンド行プロセッサーから SQL 状態ヘルプを表示する

DB2 は、SQL ステートメントの結果の原因に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。

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 or ? 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。
例えば、`? 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。

異なるバージョンの DB2 インフォメーション・センターへのアクセス

DB2 バージョン 9.5 のトピックを扱っている DB2 インフォメーション・センターの URL は、http://publib.boulder.ibm.com/infocenter/db2luw/v9r5/です。

DB2 バージョン 9 のトピックを扱っている DB2 インフォメーション・センターの URL は http://publib.boulder.ibm.com/infocenter/db2luw/v9/です。

DB2 バージョン 8 のトピックについては、バージョン 8 のインフォメーション・センターの URL http://publib.boulder.ibm.com/infocenter/db2luw/v8/にアクセスしてください。

DB2 インフォメーション・センターでの希望する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。

- 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「**上に移動**」ボタンをクリックします。

3. ブラウザー・キャッシュを消去してから、ページを最新表示します。希望する言語で DB2 インフォメーション・センターが表示されます。

- Firefox または Mozilla Web ブラウザーの場合に、希望する言語でトピックを表示するには、以下のようにします。
  1. 「**ツール**」 -> 「**オプション**」 -> 「**詳細**」 ダイアログの「**言語**」セクションにあるボタンを選択します。「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. ブラウザー・キャッシュを消去してから、ページを最新表示します。希望する言語で DB2 インフォメーション・センターが表示されます。

ブラウザーとオペレーティング・システムの組み合わせによっては、オペレーティング・システムの地域の設定も希望のロケールと言語に変更しなければならない場合があります。

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された DB2 インフォメーション・センターの更新

DB2 インフォメーション・センターをローカルにインストールしている場合は、IBM から資料の更新を入手してインストールすることができます。

ローカルにインストールされた *DB2 インフォメーション・センター* を更新するには、以下のことを行う必要があります。

1. コンピューター上の *DB2 インフォメーション・センター* を停止し、インフォメーション・センターをスタンドアロン・モードで再始動します。インフォメーション・センターをスタンドアロン・モードで実行すると、ネットワーク上の他のユーザーがそのインフォメーション・センターにアクセスできなくなります。これで、更新を適用できるようになります。非管理者および非 root の *DB2 インフォメーション・センター* は常にスタンドアロン・モードで実行されます。を参照してください。
2. 更新機能を使用することにより、どんな更新が利用できるかを確認します。インストールする更新がある場合は、更新機能を使用してそれを入手およびインストールできます。

   注: ご使用の環境において、インターネットに接続されていないマシンに *DB2 インフォメーション・センター* の更新をインストールする必要がある場合は、

インターネットに接続されていて *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マシンを使用して、更新サイトをローカル・ファイル・システムにミラーリングする必要があります。ネットワーク上の多数のユーザーが資料の更新をインストールする場合にも、更新サイトをローカルにミラーリングして、更新サイト用のプロキシーを作成することにより、個々のユーザーが更新を実行するのに要する時間を短縮できます。
更新パッケージが入手可能な場合、更新機能を使用してパッケージを入手します。ただし、更新機能は、スタンドアロン・モードでのみ使用できます。

3. スタンドアロンのインフォメーション・センターを停止し、コンピューター上の *DB2* インフォメーション・センター を再始動します。

注: Windows Vista の場合、下記のコマンドは管理者として実行する必要があります。完全な管理者特権でコマンド・プロンプトまたはグラフィカル・ツールを起動するには、ショートカットを右クリックしてから、 「**管理者として実行**」を選択します。

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されている *DB2* インフォメーション・センター を更新するには、以下のようにします。

1. *DB2* インフォメーション・センター を停止します。
   - Windows では、「**スタート**」 → 「**コントロール パネル**」 → 「**管理ツール**」 → 「**サービス**」をクリックします。次に、「**DB2 インフォメーション・センター**」サービスを右クリックして「**停止**」を選択します。
   - Linux では、以下のコマンドを入力します。

     ```
     /etc/init.d/db2icdv95 stop
     ```
2. インフォメーション・センターをスタンドアロン・モードで開始します。
   - Windows の場合:
     a. コマンド・ウィンドウを開きます。
     b. 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パスにナビゲートします。*DB2* インフォメーション・センター は、デフォルトで *Program_files*¥IBM¥DB2 Information Center¥Version 9.5 ディレクトリーにインストールされます。ここで、*Program_files* は Program Files ディレクトリーのロケーションを表します。
     c. インストール・ディレクトリーから doc¥bin ディレクトリーにナビゲートします。
     d. 次のように help_start.bat ファイルを実行します。

        ```
        help_start.bat
        ```
   - Linux の場合:
     a. 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パスにナビゲートします。デフォルトでは、*DB2* インフォメーション・センター は /opt/ibm/db2ic/V9.5 ディレクトリーにインストールされています。
     b. インストール・ディレクトリーから doc/bin ディレクトリーにナビゲートします。
     c. 次のように help_start スクリプトを実行します。

        ```
        help_start
        ```

システムのデフォルト Web ブラウザーが起動し、スタンドアロンのインフォメーション・センターが表示されます。

3. 「**更新**」ボタン () をクリックします。インフォメーション・センターの右側のパネルで、「**更新の検索 (Find Updates)**」をクリックします。 既存の文書に対する更新のリストが表示されます。
4. インストール・プロセスを開始するには、インストールする更新をチェックして選択し、「**更新のインストール**」をクリックします。
5. インストール・プロセスが完了したら、「**完了**」をクリックします。
6. 次のようにして、スタンドアロンのインフォメーション・センターを停止します。
   - Windows の場合は、インストール・ディレクトリーの doc¥bin ディレクトリーにナビゲートしてから、次のように help_end.bat ファイルを実行します。

     ```
     help_end.bat
     ```

     注: help_end バッチ・ファイルには、help_start バッチ・ファイルを使用して開始したプロセスを安全に終了するのに必要なコマンドが含まれています。help_start.bat は、Ctrl-C や他の方法を使用して終了しないでください。
   - Linux の場合は、インストール・ディレクトリーの doc/bin ディレクトリーにナビゲートしてから、次のように help_end スクリプトを実行します。

     ```
     help_end
     ```

     注: help_end スクリプトには、help_start スクリプトを使用して開始したプロセスを安全に終了するのに必要なコマンドが含まれています。他の方法を使用して、help_start スクリプトを終了しないでください。
7. *DB2 インフォメーション・センター* を再始動します。
   - Windows では、「**スタート**」 → 「**コントロール パネル**」 → 「**管理ツール**」 → 「**サービス**」をクリックします。次に、「**DB2 インフォメーション・センター**」サービスを右クリックして「**開始**」を選択します。
   - Linux では、以下のコマンドを入力します。

     ```
     /etc/init.d/db2icdv95 start
     ```

更新された *DB2 インフォメーション・センター* に、更新された新しいトピックが表示されます。

DB2 チュートリアル

DB2 チュートリアルは、DB2 製品のさまざまな機能について学習するのを支援します。この演習をとおして段階的に学習することができます。

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。

演習の中で、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、チュートリアルを参照してください。

DB2 チュートリアル

チュートリアルを表示するには、タイトルをクリックします。

「*pureXML ガイド* 」の『**pureXML™**』
XML データを保管し、ネイティブ XML データ・ストアに対して基本的な操作を実行できるように、DB2 データベースをセットアップします。

「*Visual Explain チュートリアル*」の『**Visual Explain**』
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 データベース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 問題判別ガイド」、またはDB2 インフォメーション・センターの『データベースの基本』セクションにあります。ここには、DB2 診断ツールおよびユーティリティーを使用して、問題を切り分けて識別する方法、最も頻繁に起こる幾つかの問題に対するソリューションについての情報、および DB2 データベース製品を使用する際に発生する可能性のある問題の解決方法についての他のアドバイスがあります。

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 資料、TechNotes、プログラム診断依頼書 (APAR またはバグ修正)、フィックスパック、およびその他のリソースへのリンク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/support/db2_9/) にアクセスしてください。

ご利用条件

これらの資料は、以下の条件に同意していただける場合に限りご使用いただけます。

個人使用: これらの資料は、すべての著作権表示その他の所有権表示をしていただくことを条件に、非商業的な個人による使用目的に限り複製することができます。ただし、IBM の明示的な承諾をえずに、これらの資料またはその一部について、二次的著作物を作成したり、配布 (頒布、送信を含む) または表示 (上映を含む) することはできません。

商業的使用: これらの資料は、すべての著作権表示その他の所有権表示をしていただくことを条件に、お客様の企業内に限り、複製、配布、および表示することができます。 ただし、IBM の明示的な承諾をえずにこれらの資料の二次的著作物を作成したり、お客様の企業外で資料またはその一部を複製、配布、または表示することはできません。

ここで明示的に許可されているもの以外に、資料や資料内に含まれる情報、データ、ソフトウェア、またはその他の知的所有権に対するいかなる許可、ライセンス、または権利を明示的にも黙示的にも付与するものではありません。

資料の使用が IBM の利益を損なうと判断された場合や、上記の条件が適切に守られていないと判断された場合、IBM はいつでも自らの判断により、ここで与えた許可を撤回できるものとさせていただきます。

お客様がこの情報をダウンロード、輸出、または再輸出する際には、米国のすべての輸出入関連法規を含む、すべての関連法規を遵守するものとします。

IBM は、これらの資料の内容についていかなる保証もしません。これらの資料は、特定物として現存するままの状態で提供され、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なしで提供されます。

付録 D. 特記事項

本書は米国 IBM が提供する製品およびサービスについて作成したものです。

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。

〒106-8711
東京都港区六本木 3-2-12
日本アイ・ビー・エム株式会社
法務・知的財産
知的財産権ライセンス渉外

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。

本書は、IBM 以外の Web サイトおよびリソースへのリンクまたは参照を含む場合があります。 IBM は、本書より参照もしくはアクセスできる、または本書からリンクされた IBM 以外の Web サイトもしくは第三者のリソースに対して一切の責任を負いません。 IBM 以外の Web サイトにリンクが張られていることにより IBM が当該 Web サイトを推奨するものではなく、またその内容、使用もしくはサイトの所有者について IBM が責任を負うことを意味するものではありません。 また、IBM は、お客様が IBM Web サイトから第三者の存在を知ることになった場合にも (もしくは、IBM Web サイトから第三者へのリンクを使用した場合にも)、お客様と第三者との間のいかなる取引に対しても一切責任を負いません。 従って、お客様は、IBM が上記の外部サイトまたはリソースの利用について責任を負うものではなく、また、外部サイトまたはリソースからアクセス可能なコンテンツ、サービス、

製品、またはその他の資料一切に対して IBM が責任を負うものではないことを承諾し、同意するものとします。 第三者により提供されるソフトウェアには、そのソフトウェアと共に提供される固有の使用条件が適用されます。

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

IBM、IBM ロゴ、ibm.com は、International Business Machines Corporation 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。現時点での IBM の商標リストについては、www.ibm.com/legal/copytrade.shtml の「Copyright and trademark information」をご覧ください。

以下は、それぞれ各社の商標または登録商標です。

- Linux は、Linus Torvalds 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。
- Intel、Intel (ロゴ)、Intel Inside、Intel Inside (ロゴ)、Intel Centrino、Intel Centrino (ロゴ)、Celeron、Intel Xeon、Intel SpeedStep、Itanium、Pentium は、Intel Corporation または子会社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。
- 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。

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。

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

R

S

T

U

V

X

Printed in Japan

SC88-4431-02

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

Spine information:

DB2 Version 9.5 for Linux, UNIX, and Windows

管理 API リファレンス