DB2® ユニバーサル・データベース

管理 API 解説書

バージョン 7

SC88-8514-00
(英文原典：SC09-2947-00)

DB2® ユニバーサル・データベース

管理 API 解説書

バージョン 7

SC88-8514-00
(英文原典：SC09-2947-00)

ご注意!

本書、および本書がサポートする製品をご使用になる前に、 743ページの『特記事項』にある一般的な情報を必ずお読みください。

本書において、日本では発表されていない IBM 製品 (機械およびプログラム)、プログラミング、またはサービスについて言及または説明する場合があります。しかし、このことは、弊社がこのような IBM 製品、プログラミング、またはサービスを、日本で発表する意図があることを必ずしも示すものではありません。

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

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

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

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

(URL は、変更になる場合があります)

原 典:	SC09-2947-00 IBM® DB2® Universal Database Administrative API Reference Version 7
発 行:	日本アイ・ビー・エム株式会社
担 当:	ナショナル・ランゲージ・サポート

第1刷 2000.6

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

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

© Copyright International Business Machines Corporation 1993, 2000. All rights reserved.

Translation: © Copyright IBM Japan 2000

目次

© Copyright IBM Corp. 1993, 2000

本書について

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

- C
- COBOL
- FORTRAN
- REXX

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

本書の対象読者

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

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

本書の構成

本書では、管理アプリケーションを開発する際に必要となる参照情報を提供しています。

以下のトピックについて説明しています。

『第1章 アプリケーション・プログラミング・インターフェース』
すべてのデータベース・マネージャー API を説明しています。

『第2章 追加の REXX API』
REXX プログラミング言語でのみサポートされている DB2 API について説明しています。

© Copyright IBM Corp. 1993, 2000

『第3章 データ構造』

API の呼び出し時に使用されるデータ構造について説明しています。

『付録A. 命名規則』

データベースおよび表などのオブジェクトを命名する際の規則について説明しています。

『付録B. トランザクション API』

トランザクションおよびヒューリスティック API について説明しています。

『付録C. 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を可能にするため、 API の機能をどのように使用できるかについての情報を入手するのに、 IBM と連絡をとる方法について説明しています。

『付録D. ベンダー製品用のバックアップおよび復元 API』

DB2 と他のベンダー・ソフトウェアとのインターフェースを確立させるための API の機能および使用方法を説明しています。

『付録E. 並行アクセスを伴うスレッド化アプリケーション』

処理中のそれぞれのスレッドごとに、別個の環境またはコンテキストの割り振りを許可し、 DB2 データベースへの真の並行アクセスを可能にする API について説明しています。

『付録F. DB2 共通サーバー・ログ・レコード』

DB2 ログ・レコードの抽出と処理について説明しています。

『アプリケーション移行時の考慮事項』

アプリケーションを DB2 バージョン 6 に移行させるときに考慮すべき問題について説明しています。

第1章 アプリケーション・プログラミング・インターフェース

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

注: ディレクトリー・パスのスラッシュ (/) は、UNIX ベースのシステムに固有のものです。これは、OS/2 および Windows オペレーティング・システムのディレクトリー・パスで使われる円記号 (¥) と同等のものです。

DB2 API

以下に掲げる表では、API を機能別に分類して紹介しています。

表 1. DB2 API

API の説明	サンプル・コード [a]	INCLUDE ファイル [b]
データベース・マネージャーの制御		
273ページの『sqlepstart - データベース・マネージャーの始動』	makeapi、dbstart	sqlenv
276ページの『sqlepstp - データベース・マネージャーの停止』	makeapi、dbstop	sqlenv
331ページの『sqlfxsys - データベース・マネージャー構成の入手』	dbmconf	sqlutil
310ページの『sqlfdsys - データベース・マネージャー構成の省略時値の入手』	d_dbmcon	sqlutil
316ページの『sqlfrsys - データベース・マネージャー構成のリセット』	dbmconf	sqlutil
323ページの『sqlfusys - データベース・マネージャー構成の更新』	dbmconf	sqlutil
291ページの『sqlesdeg - 実行時処理度の設定』	setrundg	sqlenv
データベースの制御		
28ページの『db2DatabaseRestart - データベースの再始動』	n/a	db2ApiDf
186ページの『sqlecrea - データベースの作成』	dbconf	sqlenv

© Copyright IBM Corp. 1993, 2000

表 1. DB2 API (続き)

API の説明	サンプル・コード [a]	INCLUDE ファイル [b]
183ページの『sqlecran - ノードでのデータベースの作成』	n/a	sqlenv
222ページの『sqledrpd - データベースの消去』	dbconf	sqlenv
216ページの『sqledpan - ノードでのデータベースの消去』	n/a	sqlenv
262ページの『sqlemgdb - データベースの移行』	migrate	sqlenv
625ページの『sqlxphqr - 未確定トランザクションのリスト』	n/a	sqlxa
154ページの『sqle_activate_db - データベースの活動化』	n/a	sqlenv
158ページの『sqle_deactivate_db - データベースの非活動化』	n/a	sqlenv
152ページの『sqlcspqy - DRDA 未確定トランザクションのリスト』	n/a	sqlxa
データベース・ディレクトリーの管理		
174ページの『sqlecadb - データベースのカタログ』	dbcat	sqlenv
301ページの『sqleuncd - データベースのアンカタログ』	dbcat	sqlenv
238ページの『sqlegdad - DCS データベースのカタログ』	dcscat	sqlenv
244ページの『sqlegdel - DCS データベースのアンカタログ』	dcscat	sqlenv
202ページの『sqledcgd - データベース・コメントの変更』	dbcmt	sqlenv
212ページの『sqledosd - データベース・ディレクトリー走査のオープン』	dbcat	sqlenv
208ページの『sqledgne - データベース・ディレクトリーの次項目の入手』	dbcat	sqlenv
206ページの『sqledcls - データベース・ディレクトリー走査のクローズ』	dbcat	sqlenv
253ページの『sqlegdsc - DCS ディレクトリー走査のオープン』	dcscat	sqlenv
250ページの『sqlegdgt - DCS ディレクトリー項目の入手』	dcscat	sqlenv

表 1. DB2 API (続き)

API の説明	サンプル・コード [a]	INCLUDE ファイル [b]
242ページの『sqlegdcl - DCS ディレクトリー走査のクローズ』	dcscat	sqlenv
247ページの『sqlegdge - データベース用 DCS ディレクトリー項目の入手』	dcscat	sqlenv
クライアント / サーバーのディレクトリーの管理		
196ページの『sqlectnd - ノードのカタログ』	nodecat	sqlenv
304ページの『sqleuncn - ノードのアンカタログ』	nodecat	sqlenv
270ページの『sqlenops - ノード・ディレクトリー走査のオープン』	nodecat	sqlenv
267ページの『sqlengne - ノード・ディレクトリー次項目の入手』	nodecat	sqlenv
265ページの『sqlencls - ノード・ディレクトリー走査のクローズ』	nodecat	sqlenv
ネットワーク・サポート		
286ページの『sqleregs - 登録』	regder	sqlenv
219ページの『sqledreg - 登録解除』	regder	sqlenv
64ページの『db2LdapRegister』	n/a	db2ApiDf
72ページの『db2LdapUpdate』	n/a	db2ApiDf
62ページの『db2LdapDeregister』	n/a	db2ApiDf
60ページの『db2LdapCatalogNode』	n/a	db2ApiDf
70ページの『db2LdapUncatalogNode』	n/a	db2ApiDf
56ページの『db2LdapCatalogDatabase』	n/a	db2ApiDf
68ページの『db2LdapUncatalogDatabase』	n/a	db2ApiDf
データベースの構成		
327ページの『sqlfxdb - データベース構成の入手』	dbconf	sqlutil
307ページの『sqlfddb - データベース構成の省略時値の入手』	d_dbconf	sqlutil
313ページの『sqlfrdb - データベース構成のリセット』	dbconf	sqlutil
319ページの『sqlfudb - データベース構成の更新』	dbconf	sqlutil
回復		
345ページの『sqlubkp - データベースのバックアップ』	backrest	sqlutil
439ページの『sqlurcon - 調整』	n/a	sqlutil

表 1. DB2 API (続き)

API の説明	サンプル・コード [a]	INCLUDE ファイル [b]
447ページの『sqlurestore - データベースの復元』	backrest	sqlutil
465ページの『sqluroll - データベースのロールフォワード』	backrest	sqlutil
46ページの『db2HistoryOpenScan - 回復活動記録ファイルの走査のオープン』	n/a	db2ApiDf
42ページの『db2HistoryGetEntry - 回復活動記録ファイルの次項目の入手』	n/a	db2ApiDf
40ページの『db2HistoryCloseScan - 回復活動記録ファイルの走査のクローズ』	n/a	db2ApiDf
83ページの『db2Prune』	n/a	db2ApiDf
52ページの『db2HistoryUpdate - 回復活動記録ファイルの更新』	n/a	db2ApiDf
オペレーション・ユーティリティー		
233ページの『sqlefrce - アプリケーションの強制終了』	dbstop	sqlenv
442ページの『sqlureot - 表の再編成』	dbstat	sqlutil
477ページの『sqlustat - 統計の実行』	dbstat	sqlutil
データベース・モニター		
35ページの『db2GetSnapshotSize - db2GetSnapshot() 出力バッファーに必要なサイズの見積もり』	db2mon	sqlmon
80ページの『db2MonitorSwitches - モニター・スイッチの入手 / 更新』	db2mon	sqlmon
31ページの『db2GetSnapshot - スナップショットの入手』	n/a	db2ApiDf
89ページの『db2ResetMonitor - モニターのリセット』	db2mon	sqlmon
25ページの『db2ConvMonStream』	n/a	db2ApiDf
データ・ユーティリティー		
360ページの『sqluexpr - エクスポート』	impexp	sqlutil
381ページの『sqluimpr - インポート』	impexp	sqlutil
408ページの『sqluload - ロード』	tload	sqlutil
75ページの『db2LoadQuery - ロードの照会』	loadqry	db2ApiDf
一般的なアプリケーション・プログラミング		
20ページの『db2AutoConfig』	autoconf	db2AuCfg
24ページの『db2AutoConfigFreeMemory』	autoconf	db2AuCfg

表 1. DB2 API (続き)

API の説明	サンプル・コード [a]	INCLUDE ファイル [b]
103ページの『sqlaintp - エラー・メッセージの入手』	util、checkerr	sql
339ページの『sqlogstt - SQLSTATE メッセージの入手』	util、checkerr	sql
260ページの『sqleisig - 信号ハンドラーのインストール』	dbcmt	sqlenv
257ページの『sqleintr - 割り込み』	n/a	sqlenv
335ページの『sqlgdref - アドレスの参照解除』	n/a	sqlutil
337ページの『sqlgmcpy - メモリーのコピー』	n/a	sqlutil
231ページの『sqlefmem - メモリーの解放』	tspace	sqlenv
334ページの『sqlgaddr - アドレスの入手』	n/a	sqlutil
アプリケーションの準備		
107ページの『sqlaprep - プログラムのプリコンパイル』	makeapi	sql
97ページの『sqlabndx - バインド』	makeapi	sql
114ページの『sqlarbnd - 再バインド』	rebind	sql
リモート・サーバー・ユーティリティー		
169ページの『sqleatin - 接続』	dbinst	sqlenv
164ページの『sqleatcp - パスワードの接続と変更』	dbinst	sqlenv
229ページの『sqledtin - 切断』	dbinst	sqlenv
テーブル・スペースの管理		
149ページの『sqlbtcq - 表スペース・コンテナーの照会』	tabscont	sqlutil
135ページの『sqlbotcq - 表スペース・コンテナー照会のオープン』	tabscont	sqlutil
123ページの『sqlbftcq - 表スペース・コンテナー照会の取り出し』	tabscont	sqlutil
119ページの『sqlbctcq - 表スペース・コンテナー照会のクローズ』	tabscont	sqlutil
145ページの『sqlbstsc - 表スペース・コンテナーの設定』	backrest	sqlutil
132ページの『sqlbmtsq - 表スペースの照会』	tabspace	sqlutil
142ページの『sqlbstpq - 単一の表スペースの照会』	tabspace	sqlutil
138ページの『sqlbotsq - 表スペース照会のオープン』	tabspace	sqlutil

表 1. DB2 API (続き)

API の説明	サンプル・コード [a]	INCLUDE ファイル [b]
126ページの『sqlbftpq - 表スペース照会の取り出し』	tabspace	sqlutil
121ページの『sqlbctsq - 表スペース照会のクローズ』	tabspace	sqlutil
129ページの『sqlbgtss - 表スペース統計の入手』	tabspace	sqlutil
483ページの『sqluvqdp - 表の表スペースの静止』	tload	sqlutil
ノードの管理		
161ページの『sqleaddn - ノードの追加』	n/a	sqlenv
226ページの『sqledrpn - ノードのドロップの検査』	n/a	sqlenv
ノード・グループの管理		
355ページの『sqludrdt - ノード・グループの再配分』	n/a	sqlutil
追加の API		
342ページの『sqluadau - 許可の入手』	dbauth	sqlutil
255ページの『sqlegins - インスタンスの入手』	dbinst	sqlenv
280ページの『sqleqryc - クライアントの照会』	client	sqlenv
283ページの『sqleqryi - クライアント情報の照会』	cli_info	sqlenv
294ページの『sqlesetc - クライアントの設定』	client	sqlenv
298ページの『sqleseti - クライアント情報の設定』	cli_info	sqlenv
289ページの『sqlesact - 会計ストリングの設定』	setact	sqlenv
462ページの『sqlurlog - ログの非同期読み取り』	asynrlog	sqlutil
374ページの『sqlugrpn - 行の区分化番号の入手』	n/a	sqlutil
379ページの『sqlugtpi - 表の区分化情報の入手』	n/a	sqlutil
17ページの『db2AdminMsgWrite』	n/a	db2ApiDf

表 1. DB2 API (続き)

API の説明	サンプル・コード [a]	INCLUDE ファイル [b]

注:

[a] sqllib ディレクトリーの samples ディレクトリーにある各言語固有のディレクトリーに、サンプル・プログラムが収められています (たとえば、sqllib¥samples¥c には、C ソース・コードが含まれています)。コード例のファイル拡張子は、使用するプログラミング言語によって異なります。たとえば、サンプル・コードが C で書かれている場合、拡張子は .c または .sqc になります。どのプログラムでも、サポートされているプログラミング言語すべてで利用できるわけではありません。すべての API にサンプル・コードがあるとは限りません (その場合は n/a によって示されます)。

[b] INCLUDE ファイルのファイル拡張子は、使用するプログラミング言語によって異なります。たとえば、C 言語用の INCLUDE ファイルのファイル拡張子は .h になります。 INCLUDE ファイルはディレクトリー sqllib¥include に収められています (ディレクトリー区切り文字は、オペレーティング・システムによって異なります)。

DB2 サンプル・プログラム

以下に掲げる表では、API をサンプル・プログラム別に分類して紹介しています。 表2 では、組み込み SQL を含まないプログラムが呼び出す API をリストし、 11ページの表3 では、組み込み SQL を含まないプログラムが呼び出す API をリストします。

表 2. サンプル・プログラム別の DB2 API (組み込み SQL がないもの)

サンプル・コード	組み込まれた API
backrest	• sqlbftcq - 表スペース・コンテナー照会の取り出し • sqlbstsc - 表スペース・コンテナーの設定 • sqlfudb - データベース構成の更新 • sqlubkp - データベースのバックアップ • sqluroll - データベースのロールフォワード • sqlurst - データベースの復元
checkerr	• sqlaintp - エラー・メッセージの入手 • sqlogstt - SQLSTATE メッセージの入手

表 2. サンプル・プログラム別の DB2 API (組み込み SQL がないもの) (続き)

サンプル・コード	組み込まれた API
cli_info	• sqleqryi - クライアント情報の照会 • sqleseti - クライアント情報の設定
client	• sqleqryc - クライアントの照会 • sqlesetc - クライアントの設定
d_dbconf	• sqleatin - 接続 • sqledtin - 切断 • sqlfddb - データベース構成の省略時値の入手
d_dbmcon	• sqleatin - 接続 • sqledtin - 切断 • sqlfdsys - データベース・マネージャー構成の省略時値の入手
db_udcs	• sqleatin - 接続 • sqlecrea - データベースの作成 • sqledrpd - データベースの消去
db2mon	• sqleatin - 接続 • db2MonitorSwitches - モニター・スイッチの入手 / 更新 • db2GetSnapshot - スナップショットの入手 • db2GetSnapshotSize - db2GetSnapshot() 出力バッファーに必要なサイズの見積もり • db2ResetMonitorData - モニターのリセット
dbcat	• sqlecadb - データベースのカタログ • sqledcls - データベース・ディレクトリー走査のクローズ • sqledgne - データベース・ディレクトリーの次項目の入手 • sqledosd - データベース・ディレクトリー走査のオープン • sqleuncd - データベースのアンカタログ
dbcmt	• sqledcgd - データベース・コメントの変更 • sqledcls - データベース・ディレクトリー走査のクローズ • sqledgne - データベース・ディレクトリーの次項目の入手 • sqledosd - データベース・ディレクトリー走査のオープン • sqleisig - 信号ハンドラーのインストール

表 2. サンプル・プログラム別の DB2 API (組み込み SQL がないもの) (続き)

サンプル・コード	組み込まれた API
dbconf	• sqleatin - 接続 • sqlecrea - データベースの作成 • sqledrpd - データベースの消去 • sqlfrdb - データベース構成のリセット • sqlfudb - データベース構成の更新 • sqlfxdb - データベース構成の入手
dbinst	• sqleatcp - パスワードの接続と変更 • sqleatin - 接続 • sqledtin - 切断 • sqlegins - インスタンスの入手
dbmconf	• sqleatin - 接続 • sqledtin - 切断 • sqlfrsys - データベース・マネージャー構成のリセット • sqlfusys - データベース・マネージャー構成の更新 • sqlfxsys - データベース・マネージャー構成の入手
dbsnap	• sqleatin - 接続 • db2GetSnapshot - スナップショットの入手
dbstart	• sqlepstart - データベース・マネージャーの始動
dbstop	• sqlefrce - アプリケーションの強制終了 • sqlepstp - データベース・マネージャーの停止
dcscat	• sqlegdad - DCS データベースのカタログ • sqlegdcl - DCS ディレクトリー走査のクローズ • sqlegdel - DCS データベースのアンカタログ • sqlegdge - データベース用 DCS ディレクトリー項目の入手 • sqlegdgt - DCS ディレクトリー項目の入手 • sqlegdsc - DCS ディレクトリー走査のオープン
dmscont	• sqleatin - 接続 • sqlecrea - データベースの作成 • sqledrpd - データベースの消去

表 2. サンプル・プログラム別の DB2 API (組み込み SQL がないもの) (続き)

サンプル・コード	組み込まれた API
ebcdicdb	• sqleatin - 接続 • sqlecrea - データベースの作成 • sqledrpd - データベースの消去
migrate	• sqlemgdb - データベースの移行
monreset	• sqleatin - 接続 • sqlmrset - モニターのリセット
monsz	• sqleatin - 接続 • sqlmonss - スナップショットの入手 • sqlmonsz - sqlmonss() 出力バッファーに必要なサイズの見積もり
nodecat	• sqlectnd - ノードのカタログ • sqlencls - ノード・ディレクトリー走査のクローズ • sqlengne - ノード・ディレクトリー次項目の入手 • sqlenops - ノード・ディレクトリー走査のオープン • sqleuncn - ノードのアンカタログ
regder	• sqledreg - 登録解除 • sqleregs - 登録
RESTART	• sqlerstd - データベースの再始動
setact	• sqlesact - 会計ストリングの設定
setrundg	• sqlesdeg - ランタイム次数の設定
sws	• sqleatin - 接続 • sqlmon - モニター・スイッチの入手 / 更新
util	• sqlaintp - エラー・メッセージの入手 • sqlogstt - SQLSTATE メッセージの入手

注: [a] サンプル・プログラムは、 `sqllib` ディレクトリーの `samples` ディレクトリーにある各言語固有のディレクトリーに収められています (たとえば、 C ソース・コード用のサンプル・プログラムは `sqllib¥samples¥c` にあります)。コード例のファイル拡張子は、使用するプログラミング言語によって異なります。たとえば、サンプル・コードが C で書かれている場合、拡張子は `.c` または `.sqc` になります。どのプログラムでも、サポートされているプログラミング言語すべてで利用できるわけではありません。すべての API にサンプル・コードがあるとは限りません。

表 3. サンプル・プログラム別の DB2 API (組み込み SQL があるもの)

サンプル・コード	組み込まれた API
autoconf	• db2AutoConfig • db2AutoConfigFreeMemory
asynrlog	• sqlurlog - ログの非同期読み取り
bindfile	• sqlabndx - バインド
dbauth	• sqluadau - 許可の入手
dbstat	• sqlureot - 表の再編成 • sqlustat - 統計の実行
expsamp	• sqluexpr - エクスポート • sqluimpr - インポート
impexp	• sqluexpr - エクスポート • sqluimpr - インポート
loadqry	• db2LoadQuery - ロードの照会
makeapi	• sqlabndx - バインド • sqlaprep - プログラムのプリコンパイル • sqlepstp - データベース・マネージャーの停止 • sqlepstr - データベース・マネージャーの始動
rebind	• sqlarbnd - 再バインド
rechist	• sqlubkp - データベースのバックアップ • sqluhcls - 回復活動記録ファイルの走査のクローズ • sqluhgne - 回復活動記録ファイルの次項目の入手 • sqluhops - 回復活動記録ファイルの走査のオープン • sqluhprn - 回復活動記録ファイルの部分削除 • sqluhupd - 回復活動記録ファイルの更新
tabscont	• sqlbctcq - 表スペース・コンテナー照会のクローズ • sqlbftcq - 表スペース・コンテナー照会の取り出し • sqlbotcq - 表スペース・コンテナー照会のオープン • sqlbtcq - 表スペース・コンテナーの照会 • sqlefmem - メモリーの解放

表 3. サンプル・プログラム別の DB2 API (組み込み SQL があるもの) (続き)

サンプル・コード	組み込まれた API
tabspace	• sqlbctsq - 表スペース照会のクローズ • sqlbftpq - 表スペース照会の取り出し • sqlbgtss - 表スペース統計の入手 • sqlbmtsq - 表スペースの照会 • sqlbotsq - 表スペース照会のオープン • sqlbstpq - 単一の表スペースの照会 • sqlefmem - メモリーの解放
tload	• sqluexpr - エクスポート • sqluload - ロード • sqluvqdp - 表の表スペースの静止
tspace	• sqlbctcq - 表スペース・コンテナー照会のクローズ • sqlbctsq - 表スペース照会のクローズ • sqlbftcq - 表スペース・コンテナー照会の取り出し • sqlbftpq - 表スペース照会の取り出し • sqlbgtss - 表スペース統計の入手 • sqlbmtsq - 表スペースの照会 • sqlbotcq - 表スペース・コンテナー照会のオープン • sqlbotsq - 表スペース照会のオープン • sqlbstpq - 単一の表スペースの照会 • sqlbstsc - 表スペース・コンテナーの設定 • sqlbtcq - 表スペース・コンテナーの照会 • sqlefmem - メモリーの解放
注: [a] サンプル・プログラムは、 sqllib ディレクトリーの samples ディレクトリーにある各言語固有のディレクトリーに収められています (たとえば、 C ソース・コード用のサンプル・プログラムは sqllib¥samples¥c にあります)。コード例のファイル拡張子は、使用するプログラミング言語によって異なります。たとえば、サンプル・コードが C で書かれている場合、拡張子は .c または .sqc になります。どのプログラムでも、サポートされているプログラミング言語すべてで利用できるわけではありません。すべての API にサンプル・コードがあるとは限りません。	

API の説明の編成

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。

効力範囲

インスタンス内での API の操作の効力範囲です。単一ノードのシステムでは、効力範囲はその単一ノードのみに限られます。複数ノードのシステムでは、ノード構成ファイル db2nodes.cfg に定義された論理ノードすべてを集めたものが効力範囲です。

許可

API を正常に呼び出すのに必要な権限です。

必須接続

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。 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。データベース接続およびインスタンス接続機構については、 *管理の手引き* をご覧ください。

API 組み込みファイル

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

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ます。新しい API 定義の実装は、段階的に行われています。以下に、変更内容の概要を簡潔に示します。

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

  ```
  db2HistoryCloseScan
  db2HistoryGetEntry
  db2HistoryOpenScan
  db2HistoryUpdate
  ```

- 汎用 API の接頭部 ″db2g″ の後ろには、 C API 名と一致するストリングが続きます。汎用 API が使用するデータ構造の名前にも、接頭部 ″db2g″ があります。
- 関数 (*db2VersionNumber*) 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要素は、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   - 統合入力データ
  ioCamelCase  - 統合入出力データ
  oCamelCase   - 統合出力データ域
  ```

- 3 番目のパラメーターは SQLCA を指すポインターで、必須です。

汎用 API 構文

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

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

API パラメーター

それぞれの API パラメーターおよびその値についての説明です。事前定義された値は適切なシンボルを使用してリストされています。シンボルの実際の値は適切な言語組み込みファイルから得ることができます。 COBOL プログラマーの方は、すべての記号の中で下線 (_) の代わりにハイフン (-) を使用する必

要があります。各ホスト言語のパラメーター・データ・タイプに関する詳細については、サンプル・プログラムをご覧ください。

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

エラー情報は、 SQLCA 構造の SQLCODE フィールドと SQLSTATE フィールドに戻され、ほとんどの API 呼び出しが実行された後に更新されます。データベース・マネージャー API を呼び出すソース・ファイルは、 1 つまたは複数の SQLCA 構造 (名前は任意) を提供できます。 SQLCODE 値が 0 の場合は、正常完了 (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完了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・マネージャーで一貫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、移行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)。

新規のインターフェース、SQLDB2 が、 REXX からの API 呼び出しをサポートするために追加されました。 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。

- 複合 REXX 変数の SQLCA が設定されます。
- 省略時値では、すべての CLP 出力メッセージがオフにされます。

SQLDB2 インターフェースに関する詳細については、 *アプリケーション開発の手引き* を参照してください。

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)。

サンプル・プログラム

サンプル・プログラムの位置および名前です。サポートされている 1 つまたは複数の言語 (C、COBOL、 FORTRAN、および REXX) で API の使用例を示します。

使用上の注意

その他の情報です。

参照資料

関連のある情報を相互参照します。

db2AdminMsgWrite

ユーザーとレプリケーションが db2diag.log と Windows NT イベント・ログに情報を書き込むためのメカニズムを提供します。 DB2 サテライト・エディションの場合、メッセージのログは、 Windows NT のイベント・ログではなく通知ファイルに記録されます。

この API は、Windows NT、Windows 98、および Windows 95 でのみ使用可能です。

許可

ありません

必須接続

ありません

API 組み込みファイル

db2ApiDf.h

C API 構文

```
/* File: db2ApiDf.h */
/* API: db2AdminMsgWrite */
/* ... */
SQL_API_RC  SQL_API_FN
  db2AdminMsgWrite (
    db2Uint32 versionNumber,
    void * pParmStruct,
    struct sqlca * pSqlca);
typedef struct
{
  db2Uint32 iMsgType;
  db2Uint32 iComponent;
  db2Uint32 iFunction;
  db2Uint32 iProbeID;
  char * piData_title;
  void * piData;
  db2Uint32 iDataLen;
  db2Uint32 iError_type;
} db2AdminMsgWriteStruct;
/* ... */
```

API パラメーター

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

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

pSqlca
出力。 *sqlca* 構造を指すポインターです。 この構造に関する詳細は、520ページの『SQLCA』を参照してください。

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 (2 進データの場合)、および STRING_MSG (ストリング・データの場合) です。

iComponent
入力。 0 を指定してください。

iFunction
入力。 0 を指定してください。

iProbeID
入力。数値のプローブ・ポイントを指定してください。

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

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

iDataLen
入力。ログ記録に使用する 2 進データのバイト数 (*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 が通知ファイルまたは Windows NT のイベント・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。

db2AutoConfig

アプリケーション・プログラムが、コントロール・センターでパフォーマンス構成ウィザードにアクセスできるようにします。このウィザードに関する詳細は、コントロール・センター内のオンライン・ヘルプ機能によって提供されます。

許可

sysadm

必須接続

データベース

API 組み込みファイル

db2AuCfg.h

C API 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];
    char  iDbAlias[SQL_ALIAS_SZ];
    db2int32  iApply;
    db2AutoConfigInput  iParams;
    db2AutoConfigOutput  oResult;
} db2AutoConfigInterface;
typedef struct {
   db2int32  token;
   db2int32  value;
} db2AutoConfigElement;

typedef struct {
   db2Uint32  numElements;
   db2AutoConfigElement *  pElements;
} db2AutoConfigArray;
typedef db2AutoConfigArray  db2AutoConfigInput;
typedef db2AutoConfigArray  db2AutoConfigDiags;
typedef struct {
   db2Uint32  numElements;
   struct sqlfupd *  pConfigs;
   void * pDataArea;
} db2ConfigValues;
typedef struct {
   db2ConfigValues  oOldDbValues;
```

```
   db2ConfigValues  oOldDbmValues;
   db2ConfigValues  oNewDbValues;
   db2ConfigValues  oNewDbmValues
   db2AutoConfigDiags  oDiagnostics;
} db2AutoConfigOutput;
```

API パラメーター

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

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

pSqlca
出力。 *sqlca* 構造を指すポインターです。 この構造に関する詳細は、520ページの『SQLCA』を参照してください。

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

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

iDbAlias
入力。データベース別名を指定するストリングです。

iApply
入力。構成を自動的に更新します。

iParams
入力。ウィザードにパラメーターを渡します。

oResult
出力。ウィザードからの結果がすべて含まれます。

Token 入力パラメーターと出力診断の両方に関する構成値を指定します。

Value トークンによって指定されたデータを保持します。

numElements
配列要素の数を示します。

pElements
要素配列を指すポインター。

db2AutoConfigDiags
診断と問題判別を行うためのトークンと値を返します。トークンは問題

を識別し、値は適切な勧告 (ある場合) を示します。トークンと値のリストについては、 API 組み込みファイル *db2AuCfg.h* を参照してください。

pConfigs
SQLFUPD 構造を指すポインター。この構造に関する詳細は、 572ページの『SQLFUPD』を参照してください。

pDataArea
構成の値が含まれるデータ域を指すポインター。

oOldDbValues
出力。 *iApply* の値が真であるとき、この値は、ウィザードが使用される前のデータベースの構成値を表します。値が偽である場合は、現在の値を表します。

oOldDbmValues
出力。 *iApply* の値が真であるとき、この値は、ウィザード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。値が偽である場合は、現在の値を表します。

oNewDbValues
出力。 *iApply* の値が真であるとき、この値は、現在のデータベース構成値を表します。値が偽である場合は、ウィザードに対する推奨値を示します。

oNewDbmValues
出力。 *iApply* の値が真であるとき、この値は、現在のデータベース・マネージャーの構成値を表します。値が偽である場合は、ウィザードに対する推奨値を示します。

oDiagnostics
出力。ウィザードからの診断が含まれます。

サンプル・プログラム

C ¥sqllib¥samples¥cpp¥autoconf.sqc

使用上の注意

db2AutoConfg によって割り当てられたメモリーを解放するには、 24ページの『db2AutoConfigFreeMemory』 を呼び出します。

参照資料

24ページの『db2AutoConfigFreeMemory』

319ページの『sqlfudb - データベース構成の更新』

323ページの『sqlfusys - データベース・マネージャー構成の更新』

db2AutoConfigFreeMemory

db2AutoConfig によって割り当てられたメモリーを解放します。

許可

sysadm

必須接続

データベース

API 組み込みファイル

db2AuCfg.h

C API 構文

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

API パラメーター

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

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

pSqlca
出力。 *sqlca* 構造を指すポインターです。 この構造に関する詳細は、520ページの『SQLCA』を参照してください。

サンプル・プログラム

C ¥sqllib¥samples¥cpp¥autoconf.sqc

db2ConvMonStream

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

許可

ありません

必須接続

ありません

API 組み込みファイル

db2ApiDf.h

C API 構文

```
/* File: db2ApiDf.h */
/* API: db2ConvMonStream */
/* ... */
int db2ConvMonStream (
  unsigned char version,
  db2ConvMonStreamData * data,
  struct sqlca * pSqlca);
typedef struct
{
  void * poTarget;
  sqlm_header_info * piSource;
  db2Uint32 iTargetType;
  db2Uint32 iTargetSize;
  db2Uint32 iSourceType
} db2ConvMonStreamData;
/* ... */
```

API パラメーター

version

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

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

pSqlca
出力。 *sqlca* 構造を指すポインターです。 この構造に関する詳細は、520ページの『SQLCA』を参照してください。

poTarget
出力。ターゲット・モニター出力構造 (sqlm_db2 など) を指すポインターです。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要素 (SQLM_ELM_DB2 など) を指すポインターです。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要素 (*sqlm_stmt* のステートメント・テキストなど) の場合は、 sqlm_stmt 統計サイズ要素、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_SZ 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 (事象モニター・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要素のリストを記載します。

スナップショット変数のデータ・ストリーム・タイプ	構造
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_subsectiion
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

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

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。

効力範囲

この API は、それが実行されるノードにのみ影響を与えます。

許可

ありません

必須接続

この API によってデータベース接続が確立されます。

API 組み込みファイル

db2ApiDf.h

C API 構文

```
/* File: db2ApiDf.h */
/* API: Restart Database */
/* ... */
SQL_API_RC SQL_API_FN
  db2DatabaseRestart (
    db2Uint32 versionNumber;
    void * pParamStruct;
    struct sqlca * pSqlca);
typedef struct
{
  char * piDatabaseName;
  char * piUserId;
  char * piPassword;
  char * piTablespaceNames;
} db2RestartDbStruct;
/* ... */
```

汎用 API 構文

```
/* File: db2ApiDf.h */
/* API: Restart Database */
/* ... */
SQL_API_RC SQL_API_FN
  db2DatabaseRestart (
    db2Uint32 versionNumber;
    void * pParamStruct;
    struct sqlca * pSqlca);
typedef struct
{
  char * piDatabaseName;
  char * piUserId;
  char * piPassword;
  char * piTablespaceNames;
} db2RestartDbStruct;
/* ... */
```

API パラメーター

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

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

pSqlca
出力。 *sqlca* 構造を指すポインターです。 この構造に関する詳細は、520ページの『SQLCA』を参照してください。

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です。

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です。ヌル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です。ヌルにすることもできます。

piTablespaceNames
入力。再始動操作時に消去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です。ヌルにすることもできます。

db2DatabaseRestart - データベースの再始動

REXX API 構文

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

REXX API パラメーター

database_alias
再始動するデータベースの別名です。

username
データベースの再始動に使用されるユーザー名。

password
ユーザー名の認証に使用されるパスワード。

使用上の注意

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

この API の完了時に、ユーザーに CONNECT 特権があれば、データベースへの共用接続は保たれます。未確定トランザクションが存在する場合には、SQL 警告を受け取ることになります。この場合、データベースはまだ使用可能ですが、未確定トランザクションが、データベースへの最終接続を消去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。未確定トランザクションのリストを生成するには、トランザクション API (619ページの『付録B. トランザクション API』 を参照) を使用してください。未確定トランザクションに関する詳細については、 *管理の手引き* を参照してください。

循環ログの場合、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、表スペースが消去保留状態になり、再始動操作を正常に完了できます。

参照資料

SQL 解説書 の CONNECT TO ステートメント。

db2GetSnapshot - スナップショットの入手

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

効力範囲

この API は、それが発行されたノードについての情報のみを戻します。

許可

以下のいずれかです。

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

必須接続

インスタンス。インスタンス接続が存在しない場合は、省略時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接続することが必要です。

API 組み込みファイル

db2ApiDf.h

C API 構文

```
int db2GetSnapshot( unsigned char version;
db2GetSnapshotData *data,
struct sqlca  *sqlca;
The parameters described in data are:
typedef struct db2GetSnapshotData{
  sqlma  *piSqlmaData;
  sqlm_collected  *poCollectedData
  void  *poBuffer;
  db2uint32 iVersion;
  db2int32 iBufferSize;
  db2uint8 iStoreResult;
db2uint16 iNodeNumber;
db2uint32 *poOutputFormat;
}db2GetSnapshotData;
```

API パラメーター

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

data 入力 / 出力。 *db2GetSnapshotData* 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です。 この構造に関する詳細は、520ページの『SQLCA』を参照してください。

piSqlmaData
入力。ユーザー割り当ての *sqlma* (モニター域) 構造を指すポインターです。この構造には、収集したいデータのタイプ (複数可) を指定します。詳細については、585ページの『SQLMA』を参照してください。

poCollectedData
出力。データベース・モニターが、要約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です。この構造に関する詳細については、580ページの『SQLM-COLLECTED』を参照してください。

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、以前のレベルのリモート・サーバーへのスナップショット呼び出しが実行される場合は、結果を処理するため、この構造を渡す必要があります。このため、常にこのパラメーターを渡すようお勧めします。

poBuffer
出力。スナップショット情報が戻される、ユーザー定義のデータ域を指すポインターです。このバッファーに戻されるデータを解釈する方法については、 *システム・モニター 手引きおよび解説書* を参照してください。

iVersion
入力。収集するデータベース・モニター・データのバージョン ID です。データベース・モニターは、要求されたバージョンについて使用可能なデータのみを戻します。このパラメーターは、以下のいずれかの記号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

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

iBufferSize

入力。データ・バッファーの長さ。このバッファーのサイズを見積もるには、35ページの『db2GetSnapshotSize - db2GetSnapshot() 出力バッファーに必要なサイズの見積もり』を使用してください。バッファーの大きさが十分でない場合には、割り当てられたバッファーに収まるだけの情報とともに、警告が戻されます。バッファーのサイズを変更し、API を再び呼び出すことが必要になる可能性があります。

iStoreResult

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

iNodeNumber

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

- SQLM_CURRENT_NODE
- SQLM_ALL_NODES
- *ノード値*

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

poOutputFormat

サーバーから返されるストリームの形式。次のいずれかです。

- SQLM_STREAM_STATIC_FORMAT
- SQLM_STREAM_DYNAMIC_FORMAT

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。

データベース・モニター API の使用法の詳細と、すべてのデータベース・モニター・データ要素およびモニター・グループの要約については、 システム・モニター *手引きおよび解説書* を参照してください。

参照資料

25ページの『db2ConvMonStream』

80ページの『db2MonitorSwitches - モニター・スイッチの入手 / 更新』

35ページの『db2GetSnapshotSize - db2GetSnapshot() 出力バッファーに必要なサイズの見積もり』

89ページの『db2ResetMonitor - モニターのリセット』

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

31ページの『db2GetSnapshot - スナップショットの入手』に必要なバッファー・サイズを見積もります。

効力範囲

この API は、呼び出し側のアプリケーションが接続されているインスタンスにのみ影響を与えます。

許可

以下のいずれかです。

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

必須接続

インスタンス。インスタンス接続が存在しない場合は、省略時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接続することが必要です。接続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 組み込みファイル

db2ApiDf.h

C API 構文

```
int db2GetSnapshotSize(db2Uint32 version,
    void*  pParamStruct,
    struct sqlca* sqlca);
typedef struct
{
        struct sqlma                    *piSqlmaData;
        sqluint32                       *poBufferSize;
        db2Uint32                       iVersion;
        db2int32                        iNodeNumber;
}db2GetSnapshotSizeData;
/* ...*/
```

API パラメーター

version
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。 *db2GetSnapshotSizeStruct* 構造を指すポインターです。

sqlca 出力。 *sqlca* 構造を指すポインターです。この構造の詳細については、 520ページの『SQLCA』を参照してください。

piSqlmaData
入力。ユーザー割り当ての *sqlma* (モニター域) 構造を指すポインターです。この構造は、収集されるスナップショット・データのタイプを指定し、 31ページの『db2GetSnapshot - スナップショットの入手』への入力として再利用できます。この構造の詳細については、 585ページの『SQLMA』を参照してください。

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です。

iVersion
入力。収集するデータベース・モニター・データのバージョン ID です。データベース・モニターは、要求されたバージョンについて使用可能なデータのみを戻します。このパラメーターは、以下のいずれかの記号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

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

iNodeNumber
入力。要求の送信先となるノードを示します。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
- *ノード値*

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

使用上の注意

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

データベース・システム・モニターは、活動中のデータベースまたはアプリケーションがないことを検出すると、ゼロのバッファー・サイズを戻す場合があります (たとえば、活動中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)。 31ページの『db2GetSnapshot - スナップショットの入手』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。

データベース・モニター API の使用法の詳細と、すべてのデータベース・モニター・データ要素およびモニター・グループの要約については、 システム・モニター *手引きおよび解説書* を参照してください。

参照資料

80ページの『db2MonitorSwitches - モニター・スイッチの入手 / 更新』

31ページの『db2GetSnapshot - スナップショットの入手』

89ページの『db2ResetMonitor - モニターのリセット』

db2GetSyncSession

サテライトの現行の同期セッション識別子を取得します。

許可

ありません

必須接続

ありません

API 組み込みファイル

db2ApiDf.h

C API 構文

```
/* File: db2ApiDf.h */
/* API: db2GetSyncSession */
/* ... */
SQL_API_RC SQL_API_FN
  db2db2GetSyncSession (
    db2Uint32 versionNumber,
    void * pParmStruct,
    struct sqlca *  pSqlca);
typedef struct
{
  char * poSyncSessionID;
} db2GetSyncSessionStruct;
/* ... */
```

API パラメーター

versionNumber

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

pParmStruct

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

pSqlca

出力。 *sqlca* 構造を指すポインターです。 この構造に関する詳細は、520ページの『SQLCA』を参照してください。

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識別子を指定します。

db2HistoryCloseScan - 回復活動記録ファイルの走査のクローズ

回復活動記録ファイルの走査を終了し、走査に必要だった DB2 リソースを解放します。この API は、46ページの『db2HistoryOpenScan - 回復活動記録ファイルの走査のオープン』 を正常に呼び出した後でなければ使用できません。

許可

ありません

必須接続

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

API 組み込みファイル

db2ApiDf.h

C API 構文

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

汎用 API 構文

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

API パラメーター

version

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

piHandle

入力。 46ページの『db2HistoryOpenScan - 回復活動記録ファイルの走査のオープン』によって戻された、走査アクセス用のハンドルを指定します。

pSqlca

出力。 *sqlca* 構造を指すポインターです。 この構造に関する詳細は、520ページの『SQLCA』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 から戻された走査識別子を含むホスト変数。

使用上の注意

回復活動記録ファイル API の使用に関する詳細は、 46ページの『db2HistoryOpenScan - 回復活動記録ファイルの走査のオープン』を参照してください。

参照資料

42ページの『db2HistoryGetEntry - 回復活動記録ファイルの次項目の入手』

46ページの『db2HistoryOpenScan - 回復活動記録ファイルの走査のオープン』

83ページの『db2Prune』

52ページの『db2HistoryUpdate - 回復活動記録ファイルの更新』

db2HistoryGetEntry - 回復活動記録ファイルの次項目の入手

回復活動記録ファイルの次項目を入手します。この API は、46ページの『db2HistoryOpenScan - 回復活動記録ファイルの走査のオープン』の正常呼び出しの後でなければ使用できません。

許可

ありません

必須接続

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

API 組み込みファイル

db2ApiDf.h

C API 構文

```
/* File: db2ApiDf.h */
/* API: Get Next Recovery History File Entry */
/* ... */
SQL_API_RC SQL_API_FN
  db2HistoryGetEntry (
    db2Uint32 version,
    void * pDB2HistoryGetEntryStruct,
    struct sqlca * pSqlca);
typedef struct
{
  db2Uint16 iHandle,
  db2Uint16 iCallerAction,
  struct db2HistData * pioHistData
} db2HistoryGetEntryStruct;
/* ... */
```

汎用 API 構文

```
/* File: db2ApiDf.h */
/* API: Get Next Recovery History File Entry */
/* ... */
SQL_API_RC SQL_API_FN
  db2GenHistoryGetEntry (
    db2Uint32 version,
    void * pDB2GenHistoryGetEntryStruct,
    struct sqlca * pSqlca);
typedef struct
{
  db2Uint16 iHandle,
  db2Uint16 iCallerAction,
  struct db2HistData * pioHistData
} db2GenHistoryGetEntryStruct;
/* ... */
```

API パラメーター

version

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

pDB2HistoryGetEntryStruct

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

pSqlca

出力。 *sqlca* 構造を指すポインターです。 この構造に関する詳細は、520ページの『SQLCA』を参照してください。

iHandle

入力。 46ページの『db2HistoryOpenScan - 回復活動記録ファイルの走査のオープン』によって戻された、走査アクセス用のハンドルが含まれます。

iCallerAction

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

DB2HISTORY_GET_ENTRY

次項目を入手しますが、コマンド・データはありません。

DB2HISTORY_GET_DDL

直前のフェッチからコマンド・データだけを入手します。

DB2HISTORY_GET_ALL
すべてのデータを含め、次項目を入手します。

pioHistData
入力。 *db2HistData* 構造を指すポインターです。この構造に関する詳細については、 493ページの『db2HistData』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 から戻された走査識別子を含むホスト変数。

value 回復活動記録ファイルの項目情報が戻される複合 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 レベル要素の数 (常に 15)

XXX.1 表スペース要素の数

XXX.2 使用された表スペース要素の数

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・スタンプおよび順序番号)

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 (バックアップ用の識別子)

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復元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復元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復元された表スペースの名前
XXX.13.3	以下同じ
XXX.14	LOCATION (バックアップまたはコピーが保管されている場所)
XXX.15	COMMENT (項目を記述するテキスト)

使用上の注意

戻されるレコードは、 46ページの『db2HistoryOpenScan - 回復活動記録ファイルの走査のオープン』への呼び出しで指定した値に基づいて選択されます。

回復活動記録ファイル API の使用に関する詳細は、 46ページの『db2HistoryOpenScan - 回復活動記録ファイルの走査のオープン』を参照してください。

参照資料

40ページの『db2HistoryCloseScan - 回復活動記録ファイルの走査のクローズ』

46ページの『db2HistoryOpenScan - 回復活動記録ファイルの走査のオープン』

83ページの『db2Prune』

52ページの『db2HistoryUpdate - 回復活動記録ファイルの更新』

db2HistoryOpenScan - 回復活動記録ファイルの走査のオープン

回復活動記録ファイルの走査を開始します。

許可

ありません

必須接続

インスタンス。この API を呼び出す前に、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には、リモート・ノードへのインスタンス接続が確立されます。

API 組み込みファイル

db2ApiDf.h

C API 構文

```
/* File: db2ApiDf.h */
/* API: Open Recovery History File Scan */
/* ... */
SQL_API_RC SQL_API_FN
  db2HistoryOpenScan (
    db2Uint32 version,
    void * pDB2HistoryOpenStruct,
    struct sqlca * pSqlca);
typedef struct
{
  char * piDatabaseAlias,
  char * piTimestamp,
  char * piObjectName,
  db2Uint32 oNumRows,
  db2Uint16 iCallerAction,
  db2Uint16 oHandle
} db2HistoryOpenStruct;
/* ... */
```

汎用 API 構文

```
/* File: db2ApiDf.h */
/* API: Open Recovery History File Scan */
/* ... */
SQL_API_RC SQL_API_FN
  db2GenHistoryOpenScan (
    db2Uint32 version,
    void * pDB2GenHistoryOpenStruct,
    struct sqlca * pSqlca);
typedef struct
{
  char * piDatabaseAlias,
  char * piTimestamp,
  char * piObjectName,
  db2Uint32 oNumRows,
  db2Uint16 iCallerAction,
  db2Uint16 oHandle
} db2GenHistoryOpenStruct;
/* ... */
```

API パラメーター

version

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

pDB2HistoryOpenStruct

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

pSqlca

出力。 *sqlca* 構造を指すポインターです。 この構造に関する詳細は、520ページの『SQLCA』を参照してください。

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です。

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です。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ヌルに設定するか、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です。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ヌルに設定するか、ゼロを指すようにすれば、オブジェクト名を用いての項目のフィルターを実行しないようにすることができます。

oNumRows
出力。 API からの戻り時に、このパラメーターには、一致した回復活動記録ファイルの項目の数が入れられます。

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

DB2HISTORY_LIST_HISTORY
他のフィルターを通過するすべてのレコード (バックアップ、復元、およびロード) を選択します。

DB2HISTORY_LIST_BACKUP
他のフィルターを通過するバックアップおよび復元レコードだけを選択します。

DB2HISTORY_LIST_ROLLFORWARD
他のフィルターを通過するロールフォワード・レコードだけを選択します。

DB2HISTORY_LIST_RUNSTATS
他のフィルターを通過する RUNSTATS レコードだけを選択します。

注: この値は、現在サポートされていません。

DB2HISTORY_LIST_REORG
他のフィルターを通過する表再編成レコードだけを選択します。

注: この値は、現在サポートされていません。

DB2HISTORY_LIST_ALT_TABLESPACE
他のフィルターを通過する ALTER TABLESPACE レコードだけを選択します。項目と関連する DDL フィールドは、戻されません。項目の DDL 情報を検索するには、この項目が取り出された直後に、呼び出しアクション DB2HISTORY_GET_DDL を指定して、 42ページの『db2HistoryGetEntry - 回復活動記録ファイルの次項目の入手』を呼び出す必要があります。

DB2HISTORY_LIST_DROPPED_TABLE
他のフィルターを通過する消去した表のレコードだけを選択します。項目と関連する DDL フィールドは、戻されません。項目の DDL 情報を検索するには、この項目が取り出された直後に、呼び出しアクション DB2HISTORY_GET_DDL を指定して、42ページの『db2HistoryGetEntry - 回復活動記録ファイルの次項目の入手』を呼び出す必要があります。

DB2HISTORY_LIST_LOAD
他のフィルターを通過するロード・レコードだけを選択します。

DB2HISTORY_LIST_REN_TABLESPACE
他のフィルターを通過する表スペース名前変更レコードだけを選択します。

oHandle
出力。 API からの戻り時に、このパラメーターには、走査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、42ページの『db2HistoryGetEntry - 回復活動記録ファイルの次項目の入手』、および 40ページの『db2HistoryCloseScan - 回復活動記録ファイルの走査のクローズ』で使用されます。

REXX API 構文

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

REXX API パラメーター

database_alias
活動記録ファイルがリストされる、データベースの別名です。

objname
レコードの選択に使用されるオブジェクト名を指定します。オブジェクトとして表または表スペースを使用できます。オブジェクトが表の場合、表の完全修飾名を指定する必要があります。このパラメーターをヌルに設定すれば、 *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ヌルに設定すれば、*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 回復活動記録ファイル情報が戻される複合 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 変数内の要素数 (常に 2)

XXX.1 将来の走査アクセスに使用される識別子 (ハンドル)

XXX.2 一致した回復活動記録ファイル項目の数

使用上の注意

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

オブジェクト名のフィルター処理の結果は、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。 (これが活動記録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、復元操作、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 つの活動記録ファイル走査が許可されています。

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

1. **db2HistoryOpenScan** を呼び出す。 *oNumRows* が戻されます。
2. *db2HistData* 構造に、 *n* 個の *oTablespace* フィールド用のスペースを割り振る。 *n* は任意の数値です。
3. *db2HistData* 構造の *iDB2NumTablespace* フィールドを *n* に設定する。
4. ループの中で、以下を実行してください。
   - **db2HistoryGetEntry** を呼び出して活動記録ファイルから取り出しを行います。

- **db2HistoryGetEntry** によって、 SQL_RC_OK という SQLCODE が戻されたら、 *db2HistData* 構造の *sqld* フィールドを使用して、戻された表スペース項目の数を判別します。
- **db2HistoryGetEntry** によって SQLUH_SQLUHINFO_VARS_WARNING の SQLCODE が戻された場合は、 DB2 が戻そうとしている表スペースのために十分なスペースが割り振られていません。この場合は、スペースをいったん解放し、*db2HistData* 構造に、 *oDB2UsedTablespace* 個の表スペース項目にとって十分なスペースを割り振り直し、 *iDB2NumTablespace* を *oDB2UsedTablespace* に設定してください。
- **db2HistoryGetEntry** によって SQLE_RC_NOMORE の SQLCODE が戻された場合は、すべての回復活動記録ファイル項目の検索が済んだことを意味します。
- 他の SQLCODE は、特定の問題が生じたことを示します。その指示に従ってください。

5. すべての情報の取り出しが終了したら、40ページの『db2HistoryCloseScan - 回復活動記録ファイルの走査のクローズ』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する。

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

参照資料

40ページの『db2HistoryCloseScan - 回復活動記録ファイルの走査のクローズ』

42ページの『db2HistoryGetEntry - 回復活動記録ファイルの次項目の入手』

83ページの『db2Prune』

52ページの『db2HistoryUpdate - 回復活動記録ファイルの更新』

db2HistoryUpdate - 回復活動記録ファイルの更新

活動記録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

以下のいずれかです。

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

必須接続

データベース。省略時データベース以外のデータベースの活動記録ファイル内にある項目を更新する場合は、この API を呼び出す前に、そのデータベースへの接続を確立しておく必要があります。

API 組み込みファイル

db2ApiDf.h

C API 構文

```
/* File: db2ApiDf.h */
/* API: Update Recovery History File */
/* ... */
SQL_API_RC SQL_API_FN
  db2HistoryUpdate (
    db2Uint32 version,
    void * pDB2HistoryUpdateStruct,
    struct sqlca * pSqlca);
typedef struct
{
  char * piNewLocation,
  char * piNewDeviceType,
  char * piNewComment,
  db2Uint32 iEID
} db2HistoryUpdateStruct;
/* ... */
```

汎用 API 構文

```
/* File: db2ApiDf.h */
/* API: Update Recovery History File */
/* ... */
SQL_API_RC SQL_API_FN
  db2GenHistoryUpdate (
    db2Uint32 version,
    void * pDB2GenHistoryUpdateStruct,
    struct sqlca * pSqlca);
typedef struct
{
  char * piNewLocation,
  char * piNewDeviceType,
  char * piNewComment,
  db2Uint32 iEID
} db2GenHistoryUpdateStruct;
/* ... */
```

API パラメーター

version

入力。 2 番目のパラメーター *pDB2HistoryUpdateStruct* として渡される構造のバージョンとリリースのレベルを指定します。

pDB2HistoryUpdateStruct

入力。 *db2HistoryUpdateStruct* 構造を指すポインターです。

pSqlca

出力。 *sqlca* 構造を指すポインターです。 この構造に関する詳細は、520ページの『SQLCA』を参照してください。

piNewLocation

入力。バックアップ、復元、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です。このパラメーターをヌルに設定するか、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、復元、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です。このパラメーターをヌルに設定するか、ゼロを指すようにすれば、値は変更されません。

piNewComment

入力。項目について説明する新規のコメントを指定するストリングを指

すポインターです。このパラメーターをヌルに設定するか、ゼロを指すようにすれば、コメントは変更されません。

iEID 入力。活動記録ファイルの特定の項目を更新するときに使用できる、固有の識別子です。

REXX API 構文

```
UPDATE RECOVERY HISTORY USING :value
```

REXX API パラメーター

value 回復活動記録ファイル項目の新規ロケーションに関する情報を含む、複合 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 変数内の要素数 (必ず 1 〜 4)

XXX.1 OBJECT_PART (タイム・スタンプと順序番号 001 〜 999)

XXX.2 バックアップまたはコピー・イメージの新規ロケーション (このパラメーターはオプションです)

XXX.3 バックアップまたはコピー・イメージの保管に使用される新規装置 (このパラメーターはオプションです)

XXX.4 新規コメント (このパラメーターはオプションです)

使用上の注意

この API は更新関数であり、変更前の情報はすべて新しい情報に置き換えられ、再作成することができなくなります。これらの変更は記録されません。

活動記録ファイルは、記録を保存する目的でのみ使用されます。復元またはロールフォワード関数によって、直接的に使用されることはありません。復元操作中は、バックアップ・イメージのロケーションを指定することができ、活動記録ファイルはこのロケーションを追跡するのに役立ちます。このファイルの情報は、後で 345ページの『sqlubkp - データベースのバックアップ』に提供することができます。同様に、ロード・コピー・イメージのロケーションが移動される場合には、ロールフォワード・ユーティリティーに新規のロケーションと記憶媒体のタイプを提供する必要があります。詳細については、*管理の手引き* および 465ページの『sqluroll - データベースのロールフォワード』を参照してください。

参照資料

40ページの『db2HistoryCloseScan - 回復活動記録ファイルの走査のクローズ』

42ページの『db2HistoryGetEntry - 回復活動記録ファイルの次項目の入手』

46ページの『db2HistoryOpenScan - 回復活動記録ファイルの走査のオープン』

83ページの『db2Prune』

db2LdapCatalogDatabase

LDAP (Lightweight Directory Access Protocol) のデータベース項目をカタログ化します。

この API は、Windows NT、Windows 98、Windows 95、および Windows 2000 でのみ使用可能です。

許可

ありません

必須接続

ありません

API 組み込みファイル

db2ApiDf.h

C API 構文

```
/* File: db2ApiDf.h */
/* API: db2LdapCatalogDatabase */
/* ... */
SQL_API_RC SQL_API_FN
  db2LdapCatalogDatabase(
    sqlint32 versionNumber,
    void * pParamStruct,
    struct sqlca * pSqlca);
typedef struct
{
 char * piAlias;
 char * piDatabaseName;
 char * piComment
 char * piNodeName;
 char * piGWNodeName;
 char * piParameters;
 char * piARLibrary;
 unsigned short iAuthentication;
 char * piDCEPrincipalName;
 char * piBindDN;
 char * piPassword;
} db2LdapCatalogDatabaseStruct;
/* ... */
```

API パラメーター

versionNumber

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

pParamStruct

入力。 *db2LdapCatalogDatabaseStruct* 構造を指すポインターです。

pSqlca

出力。 *sqlca* 構造を指すポインターです。 この構造に関する詳細は、520ページの『SQLCA』を参照してください。

piAlias

入力。カタログ化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。カタログ化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 コネクト・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 コネクトがインストールされていない場合、クライアントは DB2 コネクト・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。 DB2 コネクトが予期するこのストリングの形式については、 *DB2 コネクト 使用者の手引き* を参照してください。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。詳細については、*DB2 コネクト 使用者の手引き* を参照してください。

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。認証タイプに関する詳細は、*管理の手引き* を参照してください。

piDCEPrincipalName
入力。ターゲット・サーバーの完全修飾 DCE プリンシパル名を指定します。

piBindDN
入力。ユーザーの LDAP 識別名 (DN) を指定します。 LDAP ユーザー DN には、 LDAP ディレクトリーでオブジェクトを作成して更新するための十分な権限が必要です。ユーザーの LDAP DN が指定されていない場合は、現在のログオン・ユーザーの認証情報が使用されます。

piPassword
入力。アカウント・パスワードを示します。

使用上の注意

以下の場合は、データベースを手動で LDAP に登録またはカタログ化する必要があります。

- データベース・サーバーで LDAP がサポートされていない。この場合、LDAP をサポートするクライアントが、それぞれのクライアント・マシンでローカルにデータベースをカタログ化しなくても、データベースにアクセスできるようにするには、管理者がそれぞれのデータベースを手作業で LDAP に登録する必要があります。
- アプリケーションで、別の名前を使用して、データベースに接続する。この場合、管理者は別の別名を使用してデータベースをカタログ化する必要があります。
- CREATE DATABASE IN LDAP 時に、データベース名がすでに LDAP に存在している。データベースは依然としてローカル・マシン上で作成されています (ローカル・アプリケーションからアクセス可能) が、 LDAP にある既存の項目には、新規データベースの内容は反映されません。この場合、管理者は以下のようにすることができます。

– LDAP の既存のデータベース項目を削除し、 LDAP に新規のデータベースを手作業で登録する。
– 別の別名を使用して、LDAP に新規のデータベースを登録する。

db2LdapCatalogNode

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。

この API は、Windows NT、Windows 98、Windows 95、および Windows 2000 でのみ使用可能です。

許可

ありません

必須接続

ありません

API 組み込みファイル

db2ApiDf.h

C API 構文

```
/* File: db2ApiDf.h */
/* API: db2LdapCatalogNode */
/* ... */
SQL_API_RC SQL_API_FN
  db2LdapCatalogNode(
    sqlint32 versionNumber,
    void * pParamStruct,
    struct sqlca * pSqlca);
typedef struct
{
  char * piAlias;
  char * piNodeName;
  char * piBindDN;
  char * piPassword;
} db2LdapCatalogNodeStruct;
/* ... */
```

API パラメーター

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

pParamStruct

入力。 *db2LdapCatalogNodeStruct* 構造を指すポインター。

pSqlca

出力。 *sqlca* 構造を指すポインターです。 この構造に関する詳細は、520ページの『SQLCA』を参照してください。

piAlias

入力。ノード項目の代替名として使用する新規の別名を指定します。

piNodeName

入力。 LDAP にある DB2 サーバーを表すノード名を指定します。

piBindDN

入力。ユーザーの LDAP 識別名 (DN) を指定します。 LDAP ユーザー DN には、 LDAP ディレクトリーでオブジェクトを作成して更新するための十分な権限が必要です。ユーザーの LDAP DN が指定されていない場合は、現在のログオン・ユーザーの認証情報が使用されます。

piPassword

入力。アカウント・パスワードを示します。

db2LdapDeregister

LDAP (Lightweight Directory Access Protocol) から DB2 サーバーの登録を解除します。

この API は、Windows NT、Windows 98、Windows 95、および Windows 2000 でのみ使用可能です。

許可

ありません

必須接続

ありません

API 組み込みファイル

db2ApiDf.h

C API 構文

```
/* File: db2ApiDf.h */
/* API: db2LdapDeregister */
/* ... */
SQL_API_RC SQL_API_FN
  db2LdapDeregister (
    sqlint32 versionNumber,
    void * pParamStruct,
    struct sqlca * pSqlca);
typedef struct
{
  char * piNodeName;
  char * piBindDN;
  char * piPassword;
} db2LdapDeregisterStruct;
/* ... */
```

API パラメーター

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

pParamStruct
入力。 *db2LdapDeregisterStruct* 構造を指すポインターです。

pSqlca

出力。 *sqlca* 構造を指すポインターです。 この構造に関する詳細は、520ページの『SQLCA』を参照してください。

piNodeName

入力。 LDAP にある DB2 サーバーを表す短縮名を指定します。

piBindDN

入力。ユーザーの LDAP 識別名 (DN) を指定します。 LDAP ユーザー DN には、 LDAP ディレクトリーからオブジェクトを削除するための十分な権限が必要です。ユーザーの LDAP DN が指定されていない場合は、現在のログオン・ユーザーの認証情報が使用されます。

piPassword

入力。アカウント・パスワードを示します。

db2LdapRegister

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

この API は、Windows NT、Windows 98、Windows 95、および Windows 2000 でのみ使用可能です。

許可

ありません

必須接続

ありません

API 組み込みファイル

db2ApiDf.h

C API 構文

```
/* File: db2ApiDf.h */
/* API: db2LdapRegister */
/* ... */
SQL_API_RC SQL_API_FN
  db2LdapRegister (
    sqlint32 versionNumber,
    void * pParamStruct,
    struct sqlca * pSqlca);
typedef struct
{
  char * piNodeName;
  char * piComputer;
  char * piInstance;
  unsigned short iNodeType;
  db2LdapProtocolInfo iProtocol;
  char * piComment;
  char * piBindDN;
  char * piPassword;
} db2LdapRegisterStruct;
typedef struct
{
  char iType;
  char * piHostName;
  char * piServiceName;
  char * piNetbiosName;
  char * piNetworkId;
  char * piPartnerLU;
  char * piTPName;
  char * piMode;
  unsigned short iSecurityType;
```

```
  char * piLanAdapterAddress;
  char * piChangePasswordLU;
  char * piIpxAddress;
} db2LdapProtocolInfo;
/* ... */
```

API パラメーター

versionNumber

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

pParamStruct

入力。 *db2LdapRegisterStruct* 構造を指すポインターです。

pSqlca

出力。 *sqlca* 構造を指すポインターです。 この構造に関する詳細は、520ページの『SQLCA』を参照してください。

piNodeName

入力。 LDAP にある DB2 サーバーを表す短縮名 (8 文字未満) を指定します。

piComputer

入力。 DB2 サーバーが存在しているコンピューター・システムの名前を指定します。コンピューター名の値は、 LDAP にサーバー・マシンを追加するときの値と同じでなければなりません。 Windows NT の場合、この値は NT コンピューター名になります。 UNIX ベースのシステムの場合は、TCP/IP ホスト名です。 OS/2 の場合は、**DB2SYSTEM** レジストリー変数に指定されている値になります。ローカル・コンピューターに DB2 サーバーを登録する場合は、ヌル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、ヌルを指定します (**DB2SYSTEM** 環境変数に定義されているとおり)。

iNodeType

入力。データベース・サーバーのノード・タイプを指定します。有効な値は以下のとおりです。

```
SQLF_NT_SERVER
SQLF_NT_MPP
SQLF_NT_DCS
```

iProtocol
入力。 *db2LdapProtocolInfo* 構造内でプロトコル情報を指定します。

piComment
入力。 DB2 サーバーについて記述します。ネットワーク・ディレクトリーに登録されているサーバーについての記述を補足する、任意のコメントを入力できます。最大長は 30 文字です。復帰文字や改行文字は許可されません。

piBindDN
入力。ユーザーの LDAP 識別名 (DN) を指定します。 LDAP ユーザー DN には、 LDAP ディレクトリーでオブジェクトを作成して更新するための十分な権限が必要です。ユーザーの LDAP DN が指定されていない場合は、現在のログオン・ユーザーの認証情報が使用されます。

piPassword
入力。アカウント・パスワードを示します。

iType 入力。このサーバーがサポートするプロトコル・タイプを指定します。サーバーが 2 つ以上のプロトコルをサポートする場合は、複数の登録 (それぞれノード名とプロトコル・タイプが異なる) を行う必要があります。有効な値は以下のとおりです。

```
SQL_PROTOCOL_APPN   - APPC/APPN のサポート
SQL_PROTOCOL_NETB   - NetBIOS のサポート
SQL_PROTOCOL_TCPIP  - TCP/IP のサポート
SQL_PROTOCOL_SOCKS  - ソケット機密保護付きの TCP/IP
SQL_PROTOCOL_IPXSPX - IPX/SPX のサポート
SQL_PROTOCOL_NPIPE  - Windows NT 名前付きパイプのサポート
```

piHostName
入力。 TCP/IP ホスト名または IP アドレスを指定します。

piServiceName
入力。 TCP/IP サービス名またはポート番号を指定します。

piNetbiosName
入力。 NetBIOS ワークステーション名を指定します。 NetBIOS 名は、NetBIOS をサポートする場合に指定します。

piNetworkID
入力。ネットワーク ID を指定します。ネットワーク ID は、APPC/APPN をサポートする場合に指定します。

piPartnerLU
入力。 DB2 サーバー・マシンのパートナー LU 名を指定します。パートナー LU は、APPC/APPN をサポートする場合に指定します。

piTPName

入力。トランザクション・プログラム名を指定します。トランザクション・プログラム名は、 APPC/APPN をサポートする場合に指定します。

piMode

入力。モード名を指定します。モードは、APPC/APPN をサポートする場合に指定します。

iSecurityType

入力。 APPC 機密保護レベルを指定します。有効な値は以下のとおりです。

```
SQL_CPIC_SECURITY_NONE (default)
SQL_CPIC_SECURITY_SAME
SQL_CPIC_SECURITY_PROGRAM
```

piLanAdapterAddress

入力。ネットワーク・アダプター・アドレスを指定します。このパラメーターが必要なのは、APPC をサポートする場合だけです。 APPN の場合は、このパラメーターをヌルに設定できます。

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

固有の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 (Lightweight Directory Access Protocol) からデータベース項目を削除します。

この API は、Windows NT、Windows 98、Windows 95、および Windows 2000 でのみ使用可能です。

許可

ありません

必須接続

ありません

API 組み込みファイル

db2ApiDf.h

C API 構文

```
/* File: db2ApiDf.h */
/* API: db2LdapUncatalogDatabase */
/* ... */
SQL_API_RC SQL_API_FN
  db2LdapUncatalogDatabase(
    sqlint32 versionNumber,
    void * pParamStruct,
    struct sqlca * pSqlca);
typedef struct
{
  char * piAlias[SQL_ALIAS_SZ];
  char * piBindDN;
  char * piPassword;
} db2LdapUncatalogDatabaseStruct;
/* ... */
```

API パラメーター

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

pParamStruct
入力。 *db2LdapUncatalogDatabaseStruct* 構造を指すポインターです。

pSqlca

出力。 *sqlca* 構造を指すポインターです。 この構造に関する詳細は、520ページの『SQLCA』を参照してください。

piAlias

入力。データベース項目の別名を指定します。このパラメーターは、必須です。

piBindDN

入力。ユーザーの LDAP 識別名 (DN) を指定します。 LDAP ユーザー DN には、 LDAP ディレクトリーからオブジェクトを削除するための十分な権限が必要です。ユーザーの LDAP DN が指定されていない場合は、現在のログオン・ユーザーの認証情報が使用されます。

piPassword

入力。アカウント・パスワードを示します。

db2LdapUncatalogNode

LDAP (Lightweight Directory Access Protocol) からノード項目を削除します。

この API は、Windows NT、Windows 98、Windows 95、および Windows 2000 でのみ使用可能です。

許可

ありません

必須接続

ありません

API 組み込みファイル

db2ApiDf.h

C API 構文

```
/* File: db2ApiDf.h */
/* API: db2LdapUncatalogNode */
/* ... */
SQL_API_RC SQL_API_FN
  db2LdapUncatalogNode(
    sqlint32 versionNumber,
    void * pParamStruct,
    struct sqlca * pSqlca);
typedef struct
{
  char * piAlias;
  char * piBindDN;
  char * piPassword;
} db2LdapUncatalogNodeStruct;
/* ... */
```

API パラメーター

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

pParamStruct
入力。 *db2LdapUncatalogNodeStruct* 構造を指すポインターです。

pSqlca

出力。 *sqlca* 構造を指すポインターです。 この構造に関する詳細は、520ページの『SQLCA』を参照してください。

piAlias

入力。 LDAP からアンカタログするノードの別名を指定します。

piBindDN

入力。ユーザーの LDAP 識別名 (DN) を指定します。 LDAP ユーザー DN には、 LDAP ディレクトリーからオブジェクトを削除するための十分な権限が必要です。ユーザーの LDAP DN が指定されていない場合は、現在のログオン・ユーザーの認証情報が使用されます。

piPassword

入力。アカウント・パスワードを示します。

db2LdapUpdate

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。

この API は、Windows NT、Windows 98、Windows 95、および Windows 2000 でのみ使用可能です。

許可

ありません

必須接続

ありません

API 組み込みファイル

db2ApiDf.h

C API 構文

```
/* File: db2ApiDf.h */
/* API: db2LdapUpdate */
/* ... */
SQL_API_RC SQL_API_FN
  db2LdapUpdate (
    sqlint32 versionNumber,
    void * pParamStruct,
    struct sqlca * pSqlca);
typedef struct
{
  char * piNodeName;
  char * piComment;
  unsigned short iNodeType;
  db2LdapProtocolInfo iProtocol;
  char * piBindDN;
  char * piPassword;
} db2LdapUpdateStruct;
typedef struct
{
  char iType;
  char * piHostName;
  char * piServiceName;
  char * piNetbiosName;
  char * piNetworkId;
  char * piPartnerLU;
  char * piTPName;
  char * piMode;
  unsigned short iSecurityType;
  char * piLanAdapterAddress;
```

```
  char * piChangePasswordLU;
  char * piIpxAddress;
} db2LdapProtocolInfo;
/* ... */
```

API パラメーター

versionNumber

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

pParamStruct

入力。 *db2LdapUpdateStruct* 構造を指すポインターです。

pSqlca

出力。 *sqlca* 構造を指すポインターです。 この構造に関する詳細は、520ページの『SQLCA』を参照してください。

piNodeName

入力。 LDAP にある DB2 サーバーを表すノード名を指定します。

piComment

入力。 DB2 サーバーの新しい説明を指定します。最大長は 30 文字です。復帰文字や改行文字は許可されません。

iNodeType

入力。新規のノード・タイプを指定します。有効な値は以下のとおりです。

```
SQLF_NT_SERVER
SQLF_NT_MPP
SQLF_NT_DCS
SQL_PARM_UNCHANGE
```

iProtocol

入力。 *db2LdapProtocolInfo* 構造内で更新済みのプロトコル情報を指定します。

piBindDN

入力。ユーザーの LDAP 識別名 (DN) を指定します。 LDAP ユーザー DN には、 LDAP ディレクトリーでオブジェクトを作成して更新するための十分な権限が必要です。ユーザーの LDAP DN が指定されていない場合は、現在のログオン・ユーザーの認証情報が使用されます。

piPassword

入力。アカウント・パスワードを示します。

iType 入力。このサーバーがサポートするプロトコル・タイプを指定します。有効な値は以下のとおりです。

```
SQL_PROTOCOL_APPN   - APPC/APPN のサポート
SQL_PROTOCOL_NETB   - NetBIOS のサポート
SQL_PROTOCOL_TCPIP  - TCP/IP のサポート
SQL_PROTOCOL_SOCKS  - ソケット機密保護付きの TCP/IP
SQL_PROTOCOL_IPXSPX - IPX/SPX のサポート
SQL_PROTOCOL_NPIPE  - Windows NT 名前付きパイプのサポート
```

piHostName
入力。新規の TCP/IP ホスト名または IP アドレスを指定します。

piServiceName
入力。新規の TCP/IP サービス名またはポート番号を指定します。

piNetbiosName
入力。新規の NetBIOS ワークステーション名を指定します。

piNetworkID
入力。新規のネットワーク ID を指定します。

piPartnerLU
入力。 DB2 サーバー・マシンの新しいパートナー LU 名を指定します。

piTPName
入力。新規のトランザクション・プログラム名を指定します。

piMode
入力。新規のモード名を指定します。

iSecurityType
入力。新規の機密保護レベルを指定します。有効な値は以下のとおりです。

```
SQL_CPIC_SECURITY_NONE
SQL_CPIC_SECURITY_SAME
SQL_CPIC_SECURITY_PROGRAM
SQL_PARM_UNCHANGE
```

piLanAdapterAddress
入力。新規のネットワーク・アダプター・アドレスを指定します。

piChangePasswordLU
入力。ホスト・データベース・サーバーのパスワードを変更する際に使用するパートナー LU の新しい名前を指定します。

piIpxAddress
入力。新規の IPX アドレスを指定します。

db2LoadQuery - ロードの照会

処理中のロード操作の状況を検査します。

許可

ありません

必須接続

データベース

API 組み込みファイル

db2ApiDf.h

C API 構文

```
/* File: db2ApiDf.h */
/* API: Load Query */
/* ... */
SQL_API_RC SQL_API_FN
db2LoadQuery (
  db2Uint32 versionNumber,
  void * pParmStruct,
  struct sqlca *pSqlca);
typedef struct
{
  db2Uint32 iStringType;
  char * piString;
  db2Uint32 iShowLoadMessages;
  db2LoadQueryOutputStruct * poOutputStruct;
  char * piLocalMessageFile;
} db2LoadQueryStruct;
typedef struct
{
  db2Uint32 oRowsRead;
  db2Uint32 oRowsSkipped;
  db2Uint32 oRowsCommitted;
  db2Uint32 oRowsLoaded;
  db2Uint32 oRowsRejected;
  db2Uint32 oRowsDeleted;
  db2Uint32 oCurrentIndex;
  db2Uint32 oNumTotalIndexes;
  db2Uint32 oCurrentMPPNode;
  db2Uint32 oLoadRestarted;
  db2Uint32 oWhichPhase;
  db2Uint32 oWarningCount;
} db2LoadQueryOutputStruct;
/* ... */
```

汎用 API 構文

```
/* File: db2ApiDf.h */
/* API: Load Query */
/* ... */
SQL_API_RC SQL_API_FN
db2gLoadQuery (
  db2Uint32 versionNumber,
  void * pParmStruct,
  struct sqlca *pSqlca);
typedef struct
{
  db2Uint32 iStringType;
  db2Uint32 iStringLen;
  char * piString;
  db2Uint32 iShowLoadMessages;
  db2LoadQueryOutputStruct * poOutputStruct;
  db2Uint32 iLocalMessageFileLen;
  char * piLocalMessageFile
} db2gLoadQueryStruct;
typedef struct
{
  db2Uint32 oRowsRead;
  db2Uint32 oRowsSkipped;
  db2Uint32 oRowsCommitted;
  db2Uint32 oRowsLoaded;
  db2Uint32 oRowsRejected;
  db2Uint32 oRowsDeleted;
  db2Uint32 oCurrentIndex;
  db2Uint32 oNumTotalIndexes;
  db2Uint32 oCurrentMPPNode;
  db2Uint32 oLoadRestarted;
  db2Uint32 oWhichPhase;
  db2Uint32 oWarningCount;
} db2LoadQueryOutputStruct;
/* ... */
```

API パラメーター

versionNumber

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

pParmStruct

入力。 *db2LoadQueryStruct* 構造を指すポインターです。

pSqlca

出力。 *sqlca* 構造を指すポインターです。この構造に関する詳細については、 520ページの『SQLCA』を参照してください。

iStringType

入力。 *piString* のタイプを指定します。有効な値 (db2ApiDf.h で定義) は、以下のとおりです。

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

iStringLen

入力。 *piString* の長さ (バイト単位) を指定します。

piString

入力。 *iStringType* 値に応じて、一時ファイルのパス名または表名を指定します。

iShowLoadMessages

入力。ロード・ユーティリティーが戻すメッセージのレベルを指定します。有効な値 (db2ApiDf.h で定義) は、以下のとおりです。

DB2LOADQUERY_SHOW_ALL_MSGS
すべてのロード・メッセージを戻す。

DB2LOADQUERY_SHOW_NO_MSGS
ロード・メッセージを戻さない。

DB2LOADQUERY_SHOW_NEW_MSGS
この API を最後に呼び出した後で生成されたメッセージだけを戻す。

poOutputStruct

出力。ロード要約情報が含まれる、*db2LoadQueryOutputStruct* 構造を指すポインターです。要約が必要でない場合は、ヌルに設定してください。

iLocalMessageFileLen

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

piLocalMessageFile

入力。出力メッセージ用に使用されるローカル・ファイル名を指定します。

oRowsRead

出力。ロード・ユーティリティーがこれまでに読み取ったレコードの数を示します。

oRowsSkipped

出力。ロード操作が開始される前にスキップされたレコードの数を示します。

oRowsCommitted
出力。これまでにターゲット表にコミットされた行数を示します。

oRowsLoaded
出力。これまでにターゲット表にロードされた行の数を示します。

oRowsRejected
出力。これまでにターゲット表から拒否された行数を示します。

oRowsDeleted
出力。これまでにターゲット表から (削除フェーズで) 削除された行数を示します。

oCurrentIndex
出力。現在 (作成フェーズ時に) 作成中の索引を示します。

oCurrentMPPNode
出力。照会中のノード (MPP モード専用) を示します。

oLoadRestarted
出力。照会中のロード操作がロード再始動操作である場合に値が TRUE になるフラグを示します。

oWhichPhase
出力。照会中のロード操作の現在のフェーズを示します。有効な値 (db2ApiDf.h で定義) は、以下のとおりです。

DB2LOADQUERY_LOAD_PHASE
ロード・フェーズ。

DB2LOADQUERY_BUILD_PHASE
作成フェーズ。

DB2LOADQUERY_DELETE_PHASE
削除フェーズ。

oNumTotalIndexes
出力。 (作成フェーズで) 作成する索引の合計数を示します。

oWarningCount
出力。これまでに戻された警告の合計数を示します。

REXX API 構文

この API は SQLDB2 インターフェースを介して REXX から呼び出されます。 13ページの『API の説明の編成』、または *アプリケーション開発の手引き* を参照してください。構文の記述については、*コマンド解説書* を参照してください。

サンプル・プログラム

C	¥sqllib¥samples¥c¥loadqry.sqc
COBOL	¥sqllib¥samples¥cobol¥loadqry.sqb

使用上の注意

この API は、 *piString* で指定された表に対するロード操作の状況を読み取り、 *pLocalMsgFileName* で指定されたファイルに状況を書き込みます。

db2MonitorSwitches - モニター・スイッチの入手 / 更新

データベース・マネージャーによって収集されるモニター・データのグループについて、スイッチを選択的にオンまたはオフに切り替えます。呼び出しを発行しているアプリケーションについては、これらのスイッチの現行状態を戻します。

効力範囲

この API は、それが実行されるノードにのみ影響を与えます。

許可

以下のいずれかです。

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

必須接続

インスタンス。インスタンス接続が存在しない場合は、省略時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接続することが必要です。

API 組み込みファイル

db2ApiDf.h

C API 構文

```
int db2MonitorSwitches (db2Uint32 version,
                        void* pParamStruct,
                        struct sqlca* sqlca);
typedef struct
{
        struct sqlm_recording_group     *piGroupStates;
        void                            *poBuffer;
        db2Uint32                       iBufferSize;
        db2Uint32                       iReturnData;
        db2Uint32                       iVersion;
        db2int32                        iNodeNumber;
        db2Uint32                       *poOutputFormat;
}db2MonitorSwitchesData;
```

API パラメーター

version
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。 *db2MonitorSwitchesStruct* 構造を指すポインターです。

sqlca 出力。 *sqlca* 構造を指すポインターです。この構造の詳細については、 520ページの『SQLCA』を参照してください。

piGroupStates
入力。スイッチのリストが含まれている構造を指すポインターです。

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です。

iBufferSize
入力。出力バッファーのサイズを指定します。

iReturnData
入力。現在のスイッチの状態を *poBuffer* が指示するバッファーに書き込むかどうかを指定するフラグです。

iVersion
入力。収集するデータベース・モニター・データのバージョン ID です。データベース・モニターは、要求されたバージョンについて使用可能なデータのみを戻します。このパラメーターは、以下のいずれかの記号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

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

iNodeNumber
入力。要求の送信先となるノードを示します。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
- *ノード値*

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

poOutputFormat

サーバーから返されるストリームの形式。次のいずれかです。

SQLM_STREAM_STATIC_FORMAT

スイッチの状態が、バージョン 7 より前の静的スイッチ構造で戻されることを示します。

SQLM_STREAM_DYNAMIC_FORMAT

スイッチが、 **db2GetSnapshot** で戻されるのと同様の、自己記述形式で戻されることを示します。

注: データベース・モニター API の使用法の詳細と、すべてのデータベース・モニター・データ要素およびモニター・グループの要約については、 システム・モニター *手引きおよび解説書* を参照してください。

使用上の注意

データベース・マネージャー・レベルでのスイッチの状態を入手するには、*OBJ_TYPE* に SQMA_DB2 (データベース・マネージャーのスナップショットの入手) を指定して、 31ページの『db2GetSnapshot - スナップショットの入手』を呼び出してください。

データベース・モニター API の使用法の詳細と、すべてのデータベース・モニター・データ要素およびモニター・グループの要約については、 システム・モニター *手引きおよび解説書* を参照してください。

参照資料

31ページの『db2GetSnapshot - スナップショットの入手』

35ページの『db2GetSnapshotSize - db2GetSnapshot() 出力バッファーに必要なサイズの見積もり』

89ページの『db2ResetMonitor - モニターのリセット』

db2Prune

回復活動記録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

以下のいずれかです。

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

必須接続

データベース。省略時データベース以外のデータベースの回復活動記録ファイルから項目を削除する場合は、この API を呼び出す前に、そのデータベースへの接続を確立しておく必要があります。

API 組み込みファイル

db2ApiDf.h

C API 構文

```
/* File: db2ApiDf.h */
/* API: Prune Recovery History File */
/* ... */
SQL_API_RC SQL_API_FN
  db2Prune (
    db2Uint32 version,
    void * pDB2PruneStruct,
    struct sqlca * pSqlca);
typedef struct
{
  char * piString,
  db2Uint32 iEID,
  db2Uint32 iCallerAction,
  db2Uint32 iOptions
} db2PruneStruct;
/* ... */
```

汎用 API 構文

```
/* File: db2ApiDf.h */
/* API: Prune Recovery History File */
/* ... */
SQL_API_RC SQL_API_FN
  db2GenPrune (
    db2Uint32 version,
    void * pDB2GenPruneStruct,
    struct sqlca * pSqlca);
typedef struct
{
  db2Uint32 iStringLen;
  char * piString,
  db2Uint32 iEID,
  db2Uint32 iCallerAction,
  db2Uint32 iOptions
} db2GenPruneStruct;
/* ... */
```

API パラメーター

version

入力。 2 番目のパラメーター *pDB2PruneStruct* として渡される構造のバージョンとリリースのレベルを指定します。

pDB2PruneStruct

入力。 *db2PruneStruct* 構造を指すポインターです。

pSqlca

出力。 *sqlca* 構造を指すポインターです。 この構造に関する詳細は、520ページの『SQLCA』を参照してください。

iStringLen

入力。 *piString* の長さ (バイト単位) を指定します。

piString

入力。タイム・スタンプまたはログ順序番号 (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。ヌル・パラメーターを指定しても省略時の動作はありません。

このパラメーターは、LSN を渡すときにも使用できるので、非アクティブ・ログの枝取りが可能です。

iEID 入力。活動記録ファイルから単一の項目の枝取りをするときに使用できる、固有の識別子を示します。

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

DB2PRUNE_ACTION_HISTORY
活動記録ファイルの項目を削除します。

DB2PRUNE_ACTION_LOG
アクティブ・ログ・パスからログ・ファイルを削除します。

iOptions
入力。有効な値 (db2ApiDf で定義) は、以下のとおりです。

DB2PRUNE_OPTION_FORCE
最後のバックアップの削除を強制します。

DB2PRUNE_OPTION_LSNSTRING
piString の値を LSN に指定します。これは、呼び出し側アクション DB2PRUNE_ACTION_LOG が指定されている場合に使用します。

REXX API 構文

```
PRUNE RECOVERY HISTORY BEFORE :timestamp [WITH FORCE OPTION]
```

REXX API パラメーター

timestamp
タイム・スタンプを含むホスト変数を示します。指定されたタイム・スタンプと等しいかまたは小さいタイム・スタンプを持つすべての項目が、回復活動記録ファイルから削除されます。

WITH FORCE OPTION
指定した場合、最新の復元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・スタンプに従って回復活動記録ファイルの項目が削除されます。指定しない場合、入力時に指定したタイム・スタンプより小さいか等しい場合でも、最新の復元セットが保持されます。

使用上の注意

活動記録ファイルの項目を削除しても、実際のバックアップ、またはロード・ファイルは削除されません。それらのファイルを削除したい場合には、それを手作業で行って、それらのファイルが記憶媒体上で使用しているスペースを解放する必要があります。

考慮事項: 最新の全データベース・バックアップを媒体から削除する (さらに、活動記録ファイルから項目が削除される) 場合、すべての表スペース (カタログ表スペースおよびユーザー表スペースを含む) のバックアップを取るようにしてください。そのことを怠ると、データベースが回復不能になったり、データベース内のユーザー・データの一部が失われたりするおそれがあります。

参照資料

40ページの『db2HistoryCloseScan - 回復活動記録ファイルの走査のクローズ』

42ページの『db2HistoryGetEntry - 回復活動記録ファイルの次項目の入手』

46ページの『db2HistoryOpenScan - 回復活動記録ファイルの走査のオープン』

52ページの『db2HistoryUpdate - 回復活動記録ファイルの更新』

db2QuerySatelliteProgress

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

ありません

必須接続

ありません

API 組み込みファイル

db2ApiDf.h

C API 構文

```
/* File: db2ApiDf.h */
/* API: db2QuerySatelliteProgress */
/* ... */
SQL_API_RC SQL_API_FN
  db2QuerySatelliteProgress (
    db2Uint32 versionNumber,
    void * pParmStruct,
    struct sqlca *  pSqlca);
typedef struct
{
  db2int32 oStep;
  db2int32 oSubstep;
  db2int32 oNumSubsteps;
  db2int32 oScriptStep;
  db2int32 oNumScriptSteps;
  char * poDescription;
  char * poError;
  char * poProgressLog;
} db2QuerySatelliteProgressStruct;
/* ... */
```

API パラメーター

versionNumber

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

pParmStruct

入力。 *db2QuerySatelliteProgressStruct* 構造を指すポインターです。

pSqlca
出力。 *sqlca* 構造を指すポインターです。 この構造に関する詳細は、520ページの『SQLCA』を参照してください。

oStep 出力。同期セッション (`db2ApiDf.h` で定義されている) の現行ステップを示します。

oSubstep
出力。同期ステップ (*oStep*) をサブステップに分割できる場合、これは現行のサブステップになります。

oNumSubsteps
出力。同期セッションの現行ステップにサブステップ (*oSubstep*) が存在する場合、これは、同期ステップを構成するサブステップの合計数になります。

oScriptStep
出力。現行サブステップがスクリプトの実行である場合、このパラメーターはスクリプト実行の進行状況を報告します (可能な場合)。

oNumScriptSteps
出力。スクリプト・ステップが報告された場合、このパラメーターにはスクリプトの実行を構成するステップの合計数が入ります。

poDescription
出力。サテライトの同期セッションの状態の記述。

poError
出力。同期セッションでエラーが発生している場合、このパラメーターによってエラーの記述が渡されます。

poProgressLog
出力。このパラメーターはサテライトの同期セッションのログ全体を戻します。

db2ResetMonitor - モニターのリセット

呼び出しを発行しているアプリケーションについて、指定されたデータベースの、またはすべての活動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。

効力範囲

この API は、それが発行されたノードにのみ影響を与えます。

許可

以下のいずれかです。

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

必須接続

インスタンス。インスタンス接続が存在しない場合は、省略時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接続することが必要です。

API 組み込みファイル

db2ApiDf.h

C API 構文

```
int db2ResetMonitor  (db2Uint32 version,
                      void*           pParamStruct,
                      struct sqlca*   sqlca);
typedef struct
{
        db2Uint32                     iResetAll;
        char                          *piDbAlias;
        db2Uint32                     iVersion;
        db2int32                      iNodeNumber;
}db2ResetMonitorData;
```

API パラメーター

version
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。 *db2ResetMonitorData* 構造を指すポインターです。

sqlca 出力。 *sqlca* 構造を指すポインターです。この構造の詳細については、 520ページの『SQLCA』を参照してください。

iResetAll
入力。リセット・フラグです。

piDbAlias
入力。データベースの別名を指すポインターです。

iVersion
入力。収集するデータベース・モニター・データのバージョン ID です。データベース・モニターは、要求されたバージョンについて使用可能なデータのみを戻します。このパラメーターは、以下のいずれかの記号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

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

iNodeNumber
入力。要求の送信先となるノードを示します。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
- *ノード値*

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

使用上の注意

各プロセス (接続) 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省略時のスイッチ設定を継承します (331ページの『sqlfxsys - データベース・マネージャー構成の入手』を参照してください)。これらの設定は、80ページの『db2MonitorSwitches - モニター・スイッチの入手 / 更新』を用いて指定変更できます。

すべての活動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、80ページの『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。

データベース・モニター API の使用法の詳細と、すべてのデータベース・モニター・データ要素およびモニター・グループの要約については、 システム・モニター *手引きおよび解説書* を参照してください。

参照資料

80ページの『db2MonitorSwitches - モニター・スイッチの入手 / 更新』

31ページの『db2GetSnapshot - スナップショットの入手』

35ページの『db2GetSnapshotSize - db2GetSnapshot() 出力バッファーに必要なサイズの見積もり』

db2SetSyncSession

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

ありません

必須接続

ありません

API 組み込みファイル

db2ApiDf.h

C API 構文

```
/* File: db2ApiDf.h */
/* API: db2SetSyncSession */
/* ... */
SQL_API_RC SQL_API_FN
  db2db2SetSyncSession (
    db2Uint32 versionNumber,
    void * pParmStruct,
    struct sqlca *  pSqlca);
typedef struct
{
  char * piSyncSessionID;
} db2SetSyncSessionStruct;
/* ... */
```

API パラメーター

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

pParmStruct
入力。 *db2SetSyncSessionStruct* 構造を指すポインターです。

pSqlca

出力。 *sqlca* 構造を指すポインターです。 この構造に関する詳細は、520ページの『SQLCA』を参照してください。

piSyncSessionID

入力。サテライトが使用する同期セッションの識別子を指定します。指定された値は、サテライト制御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。

db2SyncSatellite

サテライトを同期化します。

許可

ありません

必須接続

ありません

API 組み込みファイル

db2ApiDf.h

C API 構文

```
/* File: db2ApiDf.h */
/* API: db2SyncSatellite */
/* ... */
SQL_API_RC SQL_API_FN
  db2SyncSatellite (
    db2Uint32 versionNumber,
    void * pParmStruct,
    struct sqlca *  pSqlca);
/* ... */
```

API パラメーター

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

pParmStruct
入力。ヌルに設定してください。

pSqlca
出力。 *sqlca* 構造を指すポインターです。 この構造に関する詳細は、520ページの『SQLCA』を参照してください。

db2SyncSatelliteStop

サテライトの現在活動中の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 94ページの『db2SyncSatellite』を呼び出して再開できるように停止されます。

許可

ありません

必須接続

ありません

API 組み込みファイル

db2ApiDf.h

C API 構文

```
/* File: db2ApiDf.h */
/* API: db2SyncSatelliteStop */
SQL_API_RC SQL_API_FN
  db2SyncSatelliteStop (
    db2Uint32 versionNumber,
    void * pParmStruct,
    struct sqlca *  pSqlca);
/* ... */
```

API パラメーター

versionNumber

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

pParmStruct

入力。ヌルに設定してください。

pSqlca

出力。 *sqlca* 構造を指すポインターです。 この構造に関する詳細は、520ページの『SQLCA』を参照してください。

db2SyncSatelliteTest

サテライトの同期化機能をテストします。

許可

ありません

必須接続

ありません

API 組み込みファイル

db2ApiDf.h

C API 構文

```
/* File: db2ApiDf.h */
/* API: db2SyncSatelliteTest */
/* ... */
SQL_API_RC SQL_API_FN
  db2SyncSatelliteTest (
    db2Uint32 versionNumber,
    void * pParmStruct,
    struct sqlca *  pSqlca);
/* ... */
```

API パラメーター

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

pParmStruct
入力。ヌルに設定してください。

pSqlca
出力。 *sqlca* 構造を指すポインターです。 この構造に関する詳細は、520ページの『SQLCA』を参照してください。

sqlabndx - バインド

バインド・ユーティリティーを呼び出し、プリコンパイラーによって生成されたバインド・ファイルに保管された SQL ステートメントを作成します。また、データベースに保管されるパッケージを作成します。

効力範囲

この API を db2nodes.cfg の任意のノードから呼び出すことができます。カタログ・ノードのデータベース・カタログを更新します。この影響はすべてのノードに反映されます。

許可

以下のいずれかです。

- *sysadm* または *dbadm* 権限。
- パッケージが存在していない場合、および以下のいずれかの場合には、BINDADD 特権。
  - パッケージのスキーマ名が存在していない場合、データベースに対する IMPLICIT_SCHEMA 権限。
  - パッケージのスキーマ名が存在している場合、そのスキーマに対する CREATEIN 特権。
- パッケージが存在している場合、そのスキーマに対する ALTERIN 特権
- パッケージが存在している場合、そのパッケージに対する BIND 特権。

アプリケーション内の静的 SQL ステートメントをコンパイルするために必要な特権もすべて必要です。グループに付与された特権が、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。

必須接続

データベース

API 組み込みファイル

sql.h

C API 構文

```
/* File: sql.h */
/* API: Bind */
/* ... */
SQL_API_RC SQL_API_FN
  sqlabndx (
    _SQLOLDCHAR * pBindFileName,
    _SQLOLDCHAR * pMsgFileName,
    struct sqlopt * pBindOptions,
    struct sqlca * pSqlca);
/* ... */
```

汎用 API 構文

```
/* File: sql.h */
/* API: Bind */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgbndx (
    unsigned short MsgFileNameLen,
    unsigned short BindFileNameLen,
    struct sqlca * pSqlca,
    struct sqlopt * pBindOptions,
    _SQLOLDCHAR * pMsgFileName,
    _SQLOLDCHAR * pBindFileName);
/* ... */
```

API パラメーター

MsgFileNameLen
入力。メッセージ・ファイル名の長さを示す 2 バイトの無符号整数 (バイト単位) です。

BindFileNameLen
入力。バインド・ファイル名の長さを示す 2 バイトの無符号整数 (バイト単位) です。

pSqlca
出力。 *sqlca* 構造を指すポインターです。 この構造に関する詳細は、520ページの『SQLCA』を参照してください。

pBindOptions

入力。 API へバインド・オプションを渡すのに使用される構造を示します。この構造に関する詳細については、589ページの『SQLOPT』を参照してください。

pMsgFileName

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

pBindFileName

入力。バインド・ファイル名、またはバインド・ファイル名のリストを含むファイルの名前を示すストリングです。バインド・ファイル名には、拡張子 .bnd を含めなければなりません。それらのファイルのパスで指定することができます。

バインド・リスト・ファイルの前には アットマーク (@) を付けます。たとえば、完全に修飾されたバインド・リスト・ファイル名は以下のようになります。

```
/u/user1/bnd/@all.lst
```

バインド・リスト・ファイルには 1 つまたは複数のバインド・ファイル名を含めてください。また、バインド・リスト・ファイルには拡張子 .lst を付けてください。

最初を除くすべてのバインド・ファイル名の前にプラス記号 (+) を付けます。このバインド・ファイル名は 2 行以上になることがあります。たとえば、バインド・リスト・ファイル all.lst には以下のものを含めることができます。

```
mybind1.bnd+mybind2.bnd+
mybind3.bnd+
mybind4.bnd
```

リスト・ファイルのバインド・ファイル名にパス指定を使用することもできます。パスが指定されていない場合、データベース・マネージャーはバインド・リスト・ファイルからパス情報を得ます。

REXX API 構文

この API は、SQLDB2 インターフェースを使って、REXX から呼び出すことができます。 13ページの『API の説明の編成』、または *アプリケーション開発の手引き* を参照してください。構文については、*コマンド解説書* を参照してください。

サンプル・プログラム

COBOL ¥sqllib¥samples¥cobol¥prepbind.sqb

使用上の注意

バインド処理は、アプリケーション・プログラムのソース・ファイルのプリコンパイル処理の一部として実行することもできますし、別のプロセスとして後から実行することもできます。バインドを別のプロセスとして実行するときは BIND を使用します。

パッケージを作成するのに使われた名前はバインド・ファイルに保管されます。その名前は、その名前が生成されたソース・ファイルの名前を基にして付けられます (既存のパスや拡張子は取り除かれます)。たとえば、プリコンパイルされた `myapp.sqc` というソース・ファイルは、 `myapp.bnd` という省略時バインド・ファイルと、 `MYAPP` という省略時パッケージ名を生成します。 (しかし、バインド・ファイル名およびパッケージ名は、 107ページの『sqlaprep - プログラムのプリコンパイル』 にある **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。データベースへのアプリケーション・プログラムのバインド処理に関する詳細については、 *アプリケーション開発の手引き* を参照してください。

次ページ以降の表には、バインド・オプション構造 (589ページの『SQLOPT』を参照) の *type* および *val* フィールドに有効な値がリストされています。さらに、対応する CLP オプションも掲載されています。バインド・オプション (省略時値を含む) の説明については、 *コマンド解説書* を参照してください。

表 4. BIND オプションのタイプおよび値

CLP オプション	オプションのタイプ	オプションの値
ACTION ADD	SQL_ACTION_OPT	SQL_ACTION_ADD
ACTION REPLACE	SQL_ACTION_OPT	SQL_ACTION_REPLACE
BLOCKING ALL	SQL_BLOCK_OPT	SQL_BL_ALL
BLOCKING NO	SQL_BLOCK_OPT	SQL_BL_NO
BLOCKING UNAMBIG	SQL_BLOCK_OPT	SQL_BL_UNAMBIG
CCSIDG	SQL_CCSIDG_OPT	sqlopt.sqloptions.val
CCSIDM	SQL_CCSIDM_OPT	sqlopt.sqloptions.val
CCSIDS	SQL_CCSIDS_OPT	sqlopt.sqloptions.val
CHARSUB BIT	SQL_CHARSUB_OPT	SQL_CHARSUB_BIT
CHARSUB DEFAULT	SQL_CHARSUB_OPT	SQL_CHARSUB_DEFAULT
CHARSUB MIXED	SQL_CHARSUB_OPT	SQL_CHARSUB_MIXED
CHARSUB SBCS	SQL_CHARSUB_OPT	SQL_CHARSUB_SBCS
CLIPKG	SQL_CLIPKG_OPT	3 ～ 30 の整数
CNULREQD NO	SQL_CNULREQD_OPT	SQL_CNULREQD_NO
CNULREQD YES	SQL_CNULREQD_OPT	SQL_CNULREQD_YES
COLLECTION	SQL_COLLECTION_OPT	sqlchar 構造
DATETIME DEF	SQL_DATETIME_OPT	SQL_DATETIME_DEF
DATETIME EUR	SQL_DATETIME_OPT	SQL_DATETIME_EUR
DATETIME ISO	SQL_DATETIME_OPT	SQL_DATETIME_ISO
DATETIME JIS	SQL_DATETIME_OPT	SQL_DATETIME_JIS
DATETIME LOC	SQL_DATETIME_OPT	SQL_DATETIME_LOC
DATETIME USA	SQL_DATETIME_OPT	SQL_DATETIME_USA
DECDEL COMMA	SQL_DECDEL_OPT	SQL_DECDEL_COMMA
DECDEL PERIOD	SQL_DECDEL_OPT	SQL_DECDEL_PERIOD
DEC 15	SQL_DEC_OPT	SQL_DEC_15
DEC 31	SQL_DEC_OPT	SQL_DEC_31
DEGREE 1	SQL_DEGREE_OPT	SQL_DEGREE_1
DEGREE ANY	SQL_DEGREE_OPT	SQL_DEGREE_ANY
DEGREE degree (度)	SQL_DEGREE_OPT	1 ～ 32767 の整数
DYNAMICRULES BIND	SQL_DYNAMICRULES_OPT	SQL_DYNAMICRULES_BIND
DYNAMICRULES RUN	SQL_DYNAMICRULES_OPT	SQL_DYNAMICRULES_RUN
DYNAMICRULES DEFINE	SQL_DYNAMICRULES_OPT	SQL_DYNAMICRULES_DEFINE
DYNAMICRULES INVOKE	SQL_DYNAMICRULES_OPT	SQL_DYNAMICRULES_INVOKE
EXPLAIN NO	SQL_EXPLAIN_OPT	SQL_EXPLAIN_NO
EXPLAIN YES	SQL_EXPLAIN_OPT	SQL_EXPLAIN_YES
EXPLAIN ALL	SQL_EXPLAIN_OPT	SQL_EXPLAIN_ALL

表 4. BIND オプションのタイプおよび値 (続き)

CLP オプション	オプションのタイプ	オプションの値
EXPLSNAP NO	SQL_EXPLSNAP_OPT	SQL_EXPLSNAP_NO
EXPLSNAP YES	SQL_EXPLSNAP_OPT	SQL_EXPLSNAP_YES
EXPLSNAP ALL	SQL_EXPLSNAP_OPT	SQL_EXPLSNAP_ALL
FUNCPATH	SQL_FUNCTION_PATH	sqlchar 構造
GENERIC	SQL_GENERIC_OPT	sqlchar 構造
GRANT	SQL_GRANT_OPT	sqlchar 構造
GRANT PUBLIC	SQL_GRANT_OPT	sqlchar 構造
GRANT TO USER	SQL_GRANT_USER_OPT	sqlchar 構造
GRANT TO GROUP	SQL_GRANT_GROUP_OPT	sqlchar 構造
INSERT BUF	SQL_INSERT_OPT	SQL_INSERT_BUF
INSERT DEF	SQL_INSERT_OPT	SQL_INSERT_DEF
ISOLATION RS	SQL_ISO_OPT	SQL_READ_STAB
ISOLATION NC	SQL_ISO_OPT	SQL_NO_COMMIT
ISOLATION CS	SQL_ISO_OPT	SQL_CURSOR_STAB
ISOLATION RR	SQL_ISO_OPT	SQL_REP_READ
ISOLATION UR	SQL_ISO_OPT	SQL_UNCOM_READ
OWNER	SQL_OWNER_OPT	sqlchar 構造
QUALIFIER	SQL_QUALIFIER_OPT	sqlchar 構造
QUERYOPT	SQL_QUERYOPT_OPT	SQL_QUERYOPT_0,1,2,3,5,7,9
RELEASE COMMIT	SQL_RELEASE_OPT	SQL_RELEASE_COMMIT
RELEASE DEALLOCATE	SQL_RELEASE_OPT	SQL_RELEASE_DEALLOCATE
REPLVER	SQL_REPLVER_OPT	sqlchar 構造
RETAIN NO	SQL_RETAIN_OPT	SQL_RETAIN_NO
RETAIN YES	SQL_RETAIN_OPT	SQL_RETAIN_YES
SQLERROR CHECK	SQL_SQLERROR_OPT	SQL_SQLERROR_CHECK
SQLERROR CONTINUE	SQL_SQLERROR_OPT	SQL_SQLERROR_CONTINUE
SQLERROR NOPACKAGE	SQL_SQLERROR_OPT	SQL_SQLERROR_NOPACKAGE
SQLWARN NO	SQL_SQLWARN_OPT	SQL_SQLWARN_NO
SQLWARN YES	SQL_SQLWARN_OPT	SQL_SQLWARN_YES
STRDEL APOSTROPHE	SQL_STRDEL_OPT	SQL_STRDEL_APOSTROPHE
STRDEL QUOTE	SQL_STRDEL_OPT	SQL_STRDEL_QUOTE
TEXT	SQL_TEXT_OPT	sqlchar 構造
TRANSFORM GROUP	SQL_TRANSFORMGROUP_OPT	sqlchar 構造
VALIDATE BIND	SQL_VALIDATE_OPT	SQL_VALIDATE_BIND
VALIDATE RUN	SQL_VALIDATE_OPT	SQL_VALIDATE_RUN

注: sqlchar 構造として示されているオプションの値には、 522ページの『SQLCHAR』を指すポインターを含む *val* フィールドがあります。この構造には、オプションの値を指定するための文字ストリングが含まれています。

参照資料

107ページの『sqlaprep - プログラムのプリコンパイル』

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

ありません

必須接続

ありません

API 組み込みファイル

sql.h

C API 構文

```
/* File: sql.h */
/* API: Get Error Message */
/* ... */
SQL_API_RC SQL_API_FN
  sqlaintp (
    char * pBuffer,
    short BufferSize,
    short LineWidth,
    struct sqlca * pSqlca);
/* ... */
```

汎用 API 構文

```
/* File: sql.h */
/* API: Get Error Message */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgintp (
    short BufferSize,
    short LineWidth,
    struct sqlca * pSqlca,
    _SQLOLDCHAR * pBuffer);
/* ... */
```

API パラメーター

BufferSize
入力。検索したメッセージ・テキストを保留するストリング・バッファーのサイズ (バイト単位) です。

LineWidth
入力。メッセージ・テキストの各行ごとの最大行幅を示します。ワード境界で改行されます。値ゼロは、メッセージ・テキストが改行されることなく検索されることを示します。

pSqlca
出力。 *sqlca* 構造を指すポインターです。 この構造に関する詳細は、520ページの『SQLCA』を参照してください。

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です。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、ヌル・ストリング終了文字で切り捨てることができます。

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。

サンプル・プログラム

C	¥sqllib¥samples¥c¥utilapi.c
COBOL	¥sqllib¥samples¥cobol¥checkerr.cbl
REXX	¥sqllib¥samples¥rexx¥dbcat.cmd

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。

戻りコード

コード メッセージ

+i 形式化メッセージのバイト数を示す正の整数です。呼び出し側が入力したバッファー・サイズよりもこの数の方が大きい場合、メッセージは切り捨てられます。

-1 メッセージ書式化サービスを機能させるには、利用可能なメモリーが不十分です。要求されたメッセージは、戻されません。

-2 エラーはありません。 *sqlca* に、エラー・コード (SQLCODE = 0) は含まれていませんでした。

-3 メッセージ・ファイルがアクセス不能または正しくありません。

-4 行幅が 0 未満です。

-5 無効 *sqlca*、不良バッファー・アドレス、または不良バッファー長を示します。

戻りコードが -1 または -3 の場合、メッセージ・バッファーにはその問題に関する、より詳細な情報が含まれています。

参照資料

339ページの『sqlogstt - SQLSTATE メッセージの入手』

sqlaprep - プログラムのプリコンパイル

組み込み SQL ステートメントを含むアプリケーション・プログラム・ソース・ファイルを処理します。 SQL ステートメントのホスト言語呼び出しが入った修正済みソース・ファイルが作成され、省略時にはデータベース内にパッケージが作成されます。

効力範囲

この API は、db2nodes.cfg の任意のノードから呼び出すことができ、カタログ・ノードのデータベース・カタログを更新します。この影響はすべてのノードに反映されます。

許可

以下のいずれかです。

- *sysadm* または *dbadm* 権限。
- パッケージが存在していない場合、および以下のいずれかの場合には、BINDADD 特権。
  - パッケージのスキーマ名が存在していない場合、データベースに対する IMPLICIT_SCHEMA 権限。
  - パッケージのスキーマ名が存在している場合、そのスキーマに対する CREATEIN 特権。
- パッケージが存在している場合、そのスキーマに対する ALTERIN 特権。
- パッケージが存在している場合、そのパッケージに対する BIND 特権。

アプリケーション内の静的 SQL ステートメントをコンパイルするために必要な特権もすべて必要です。グループに付与された特権が、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。

必須接続

データベース

API 組み込みファイル

sql.h

C API 構文

```
/* File: sql.h */
/* API: Precompile Program */
/* ... */
SQL_API_RC SQL_API_FN
  sqlaprep (
    _SQLOLDCHAR * pProgramName,
    _SQLOLDCHAR * pMsgFileName,
    struct sqlopt * pPrepOptions,
    struct sqlca * pSqlca);
/* ... */
```

汎用 API 構文

```
/* File: sql.h */
/* API: Precompile Program */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgprep (
    unsigned short MsgFileNameLen,
    unsigned short ProgramNameLen,
    struct sqlca * pSqlca,
    struct sqlopt * pPrepOptions,
    _SQLOLDCHAR * pMsgFileName,
    _SQLOLDCHAR * pProgramName);
/* ... */
```

API パラメーター

MsgFileNameLen

入力。メッセージ・ファイル名の長さを示す 2 バイトの無符号整数 (バイト単位) です。

ProgramNameLen

入力。プログラム名の長さを示す 2 バイトの無符号整数 (バイト単位) です。

pSqlca

出力。 *sqlca* 構造を指すポインターです。 この構造に関する詳細は、520ページの『SQLCA』を参照してください。

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。この構造に関する詳細については、589ページの『SQLOPT』を参照してください。

pMsgFileName

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

pProgramName

入力。プリコンパイルされるアプリケーションの名前を含むストリングです。以下の拡張子を使用してください。

- `.sqb` - COBOL アプリケーションの場合
- `.sqc` - C アプリケーションの場合
- `.sqC` - UNIX C++ アプリケーションの場合
- `.sqf` - FORTRAN アプリケーションの場合
- `.sqx` - C++ アプリケーションの場合

TARGET オプションを使用する場合は、入力ファイル名の拡張子をこの定義済みリストから入力する必要がありません。

UNIX ベース・システム上で組み込み SQL が含まれている C++ アプリケーションを使用する場合の優先拡張子は、 `sqC` です。ただし、UNIX ベース・システムでは、大文字小文字を区別しないシステム用に開発された `sqx` 表記も許容されています。

REXX API 構文

この API は、SQLDB2 インターフェースを使って、REXX から呼び出すことができます。 13ページの『API の説明の編成』、または *アプリケーション開発の手引き* を参照してください。構文については、*コマンド解説書* を参照してください。

サンプル・プログラム

C	¥sqllib¥samples¥c¥makeapi.sqc
COBOL	¥sqllib¥samples¥cobol¥prepbind.sqb

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。省略時には、接続がすでに確立されているデータベース内にパッケージが作成されます。パッケージ名は、プログラム・ファイル名 (拡張子を除く、大文字になります) と同じ最大 8 文字までです。

一度データベースに接続されると、 **sqlaprep** は開始済みのトランザクションの下で実行します。プリコンパイルの実行後、 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。

以下の表には、プリコンパイル・オプション構造 (589ページの『SQLOPT』を参照) の *type* および *val* フィールドに有効な値がリストされています。さらに、対応する CLP オプションも掲載されています。プリコンパイル・オプション (省略時値を含む) の説明については、 *コマンド解説書* を参照してください。

表 5. PRECOMPILE オプションのタイプおよび値

CLP オプション	API オプションのタイプ	API オプションの値
ACTION ADD	SQL_ACTION_OPT	SQL_ACTION_ADD
ACTION REPLACE	SQL_ACTION_OPT	SQL_ACTION_REPLACE
BINDFILE	SQL_BIND_OPT	Null (ヌル)
BINDFILE filename (ファイル名)	SQL_BIND_OPT	sqlchar 構造
BLOCKING ALL	SQL_BLOCK_OPT	SQL_BL_ALL
BLOCKING NO	SQL_BLOCK_OPT	SQL_BL_NO
BLOCKING UNAMBIG	SQL_BLOCK_OPT	SQL_BL_UNAMBIG
CCSIDG value (値)	SQL_CCSIDG_OPT	sqlopt.sqloptions.val
CCSIDM value (値)	SQL_CCSIDM_OPT	sqlopt.sqloptions.val
CCSIDS value (値)	SQL_CCSIDS_OPT	sqlopt.sqloptions.val
CHARSUB BIT	SQL_CHARSUB_OPT	SQL_CHARSUB_BIT
CHARSUB DEFAULT	SQL_CHARSUB_OPT	SQL_CHARSUB_DEFAULT
CHARSUB MIXED	SQL_CHARSUB_OPT	SQL_CHARSUB_MIXED
CHARSUB SBCS	SQL_CHARSUB_OPT	SQL_CHARSUB_SBCS
CNULREQD NO	SQL_CNULREQD_OPT	SQL_CNULREQD_NO
CNULREQD YES	SQL_CNULREQD_OPT	SQL_CNULREQD_YES
COLLECTION coll-id (収集 ID)	SQL_COLLECTION_OPT	sqlchar 構造

表 5. *PRECOMPILE* オプションのタイプおよび値 (続き)

CLP オプション	API オプションのタイプ	API オプションの値
CONNECT 1	SQL_CONNECT_OPT	SQL_CONNECT_1
CONNECT 2	SQL_CONNECT_OPT	SQL_CONNECT_2
DATETIME DEF	SQL_DATETIME_OPT	SQL_DATETIME_DEF
DATETIME EUR	SQL_DATETIME_OPT	SQL_DATETIME_EUR
DATETIME ISO	SQL_DATETIME_OPT	SQL_DATETIME_ISO
DATETIME JIS	SQL_DATETIME_OPT	SQL_DATETIME_JIS
DATETIME LOC	SQL_DATETIME_OPT	SQL_DATETIME_LOC
DATETIME USA	SQL_DATETIME_OPT	SQL_DATETIME_USA
DECDEL COMMA	SQL_DECDEL_OPT	SQL_DECDEL_COMMA
DECDEL PERIOD	SQL_DECDEL_OPT	SQL_DECDEL_PERIOD
DEC 15	SQL_DEC_OPT	SQL_DEC_15
DEC 31	SQL_DEC_OPT	SQL_DEC_31
DEFERRED_PREPARE ALL	SQL_DEFERRED_PREPARE_OPT	SQL_DEFERRED_PREPARE_ALL
DEFERRED_PREPARE NO	SQL_DEFERRED_PREPARE_OPT	SQL_DEFERRED_PREPARE_NO
DEFERRED_PREPARE YES	SQL_DEFERRED_PREPARE_OPT	SQL_DEFERRED_PREPARE_YES
DEGREE 1	SQL_DEGREE_OPT	SQL_DEGREE_1
DEGREE ANY	SQL_DEGREE_OPT	SQL_DEGREE_ANY
DEGREE degree (度)	SQL_DEGREE_OPT	1 ～ 32767 の整数
DISCONNECT EXPLICIT	SQL_DISCONNECT_OPT	SQL_DISCONNECT_EXPL
DISCONNECT CONDITIONAL	SQL_DISCONNECT_OPT	SQL_DISCONNECT_COND
DISCONNECT AUTOMATIC	SQL_DISCONNECT_OPT	SQL_DISCONNECT_AUTO
DYNAMICRULES BIND	SQL_DYNAMICRULES_OPT	SQL_DYNAMICRULES_BIND
DYNAMICRULES RUN	SQL_DYNAMICRULES_OPT	SQL_DYNAMICRULES_RUN
DYNAMICRULES DEFINE	SQL_DYNAMICRULES_OPT	SQL_DYNAMICRULES_DEFINE
DYNAMICRULES INVOKE	SQL_DYNAMICRULES_OPT	SQL_DYNAMICRULES_INVOKE
EXPLAIN NO	SQL_EXPLAIN_OPT	SQL_EXPLAIN_NO
EXPLAIN YES	SQL_EXPLAIN_OPT	SQL_EXPLAIN_YES
EXPLAIN ALL	SQL_EXPLAIN_OPT	SQL_EXPLAIN_ALL DRDA ではサポートされていません。
EXPLSNAP NO	SQL_EXPLSNAP_OPT	SQL_EXPLSNAP_NO
EXPLSNAP YES	SQL_EXPLSNAP_OPT	SQL_EXPLSNAP_YES
EXPLSNAP ALL	SQL_EXPLSNAP_OPT	SQL_EXPLSNAP_ALL
FUNCPATH	SQL_FUNCTION_PATH	sqlchar 構造
GENERIC	SQL_GENERIC_OPT	sqlchar 構造
INSERT BUF	SQL_INSERT_OPT	SQL_INSERT_BUF
INSERT DEF	SQL_INSERT_OPT	SQL_INSERT_DEF
ISOLATION RS	SQL_ISO_OPT	SQL_READ_STAB
ISOLATION NC	SQL_ISO_OPT	SQL_NO_COMMIT
ISOLATION CS	SQL_ISO_OPT	SQL_CURSOR_STAB
ISOLATION RR	SQL_ISO_OPT	SQL_REP_READ

表 5. *PRECOMPILE* オプションのタイプおよび値 *(続き)*

CLP オプション	API オプションのタイプ	API オプションの値
ISOLATION UR	SQL_ISO_OPT	SQL_UNCOM_READ
LANGLEVEL SAA1	SQL_STANDARDS_OPT	SQL_SAA_COMP
LANGLEVEL MIA	SQL_STANDARDS_OPT	SQL_MIA_COMP
LANGLEVEL SQL92E	SQL_STANDARDS_OPT	SQL_SQL92E_COMP
LEVEL levelname (レベル名)	SQL_LEVEL_OPT	sqlchar 構造
LONGERROR NO	SQL_LONGERROR_OPT	SQL_LONGERROR_NO
LONGERROR YES	SQL_LONGERROR_OPT	SQL_LONGERROR_YES
NOLINEMACRO	SQL_LINEMACRO_OPT	SQL_NO_LINE_MACROS
(省略時値)	SQL_LINEMACRO_OPT	SQL_LINE_MACROS
OPTLEVEL 0	SQL_OPTIM_OPT	SQL_DONT_OPTIMIZE
OPTLEVEL 1	SQL_OPTIM_OPT	SQL_OPTIMIZE
OUTPUT ファイル名	SQL_PREP_OUTPUT_OPT	sqlchar 構造
OWNER	SQL_OWNER_OPT	sqlchar 構造
PACKAGE	SQL_PKG_OPT	Null (ヌル)
PACKAGE pkgname (パッケージ名)	SQL_PKG_OPT	sqlchar 構造
PREPROCESSOR ″preprocessor-command (プリプロセッサー・コマンド)″	SQL_PREPROCESSOR_OPT	sqlchar 構造
QUALIFIER	SQL_QUALIFIER_OPT	sqlchar 構造
QUERYOPT	SQL_QUERYOPT_OPT	SQL_QUERYOPT_0,1,2,3,5,7,9
RELEASE COMMIT	SQL_RELEASE_OPT	SQL_RELEASE_COMMIT
RELEASE DEALLOCATE	SQL_RELEASE_OPT	SQL_RELEASE_DEALLOCATE
REPLVER versn-str (バージョン・ストリング)	SQL_REPLVER_OPT	sqlchar 構造
RETAIN NO	SQL_RETAIN_OPT	SQL_RETAIN_NO
RETAIN YES	SQL_RETAIN_OPT	SQL_RETAIN_YES
SQLCA SAA	SQL_SAA_OPT	SQL_SAA_YES
SQLCA NONE	SQL_SAA_OPT	SQL_SAA_NO
SQLERROR CHECK	SQL_SQLERROR_OPT	SQL_SQLERROR_CHECK
SQLERROR CONTINUE	SQL_SQLERROR_OPT	SQL_SQLERROR_CONTINUE
SQLERROR NOPACKAGE	SQL_SQLERROR_OPT	SQL_SQLERROR_NOPACKAGE
SQLFLAG SQL92E SYNTAX	SQL_FLAG_OPT	SQL_SQL92E_SYNTAX
SQLFLAG MVSDB2V23 SYNTAX	SQL_FLAG_OPT	SQL_MVSDB2V23_SYNTAX
SQLFLAG MVSDB2V31 SYNTAX	SQL_FLAG_OPT	SQL_MVSDB2V31_SYNTAX
SQLFLAG MVSDB2V41 SYNTAX	SQL_FLAG_OPT	SQL_MVSDB2V41_SYNTAX
SQLRULES DB2	SQL_RULES_OPT	SQL_RULES_DB2
SQLRULES STD	SQL_RULES_OPT	SQL_RULES_STD
SQLWARN NO	SQL_SQLWARN_OPT	SQL_SQLWARN_NO
SQLWARN YES	SQL_SQLWARN_OPT	SQL_SQLWARN_YES
STRDEL APOSTROPHE	SQL_STRDEL_OPT	SQL_STRDEL_APOSTROPHE
STRDEL QUOTE	SQL_STRDEL_OPT	SQL_STRDEL_QUOTE

表 5. *PRECOMPILE* オプションのタイプおよび値 *(続き)*

CLP オプション	API オプションのタイプ	API オプションの値
SYNCPOINT ONEPHASE	SQL_SYNCPOINT_OPT	SQL_SYNC_ONEPHASE
SYNCPOINT TWOPHASE	SQL_SYNCPOINT_OPT	SQL_SYNC_TWOPHASE
SYNCPOINT NONE	SQL_SYNCPOINT_OPT	SQL_SYNC_NONE
SYNTAX	SQL_SYNTAX_OPT	SQL_SYNTAX_CHECK
(省略時値)	SQL_SYNTAX_OPT	SQL_NO_SYNTAX_CHECK
TARGET コンパイラー	SQL_TARGET_OPT	sqlchar 構造
TEXT text-str (テキスト・ストリング)	SQL_TEXT_OPT	sqlchar 構造
TRANSFORM GROUP	SQL_TRANSFORMGROUP_OPT	sqlchar 構造
VALIDATE BIND	SQL_VALIDATE_OPT	SQL_VALIDATE_BIND
VALIDATE RUN	SQL_VALIDATE_OPT	SQL_VALIDATE_RUN
VERSION versn-str (バージョン・ストリング)	SQL_VERSION_OPT	sqlchar 構造
WCHARTYPE CONVERT	SQL_WCHAR_OPT	SQL_WCHAR_CONVERT
WCHARTYPE NOCONVERT	SQL_WCHAR_OPT	SQL_WCHAR_NOCONVERT
(ありません)	SQL_NO_OPT	(ありません)

参照資料

97ページの『sqlabndx - バインド』

sqlarbnd - 再バインド

バインド・ファイルを用いずに、データベースに保管されているパッケージを再作成できるようにします。

許可

以下のいずれかです。

- *sysadm* または *dbadm* 権限。
- スキーマに対する ALTERIN 特権。
- パッケージに対する BIND 特権。

SYSCAT.PACKAGES システム・カタログ表の BOUNDBY 列に記録された許可 ID (もっとも最近にパッケージをバインドした人の ID) が、再バインドを実行する際のバインド者の許可 ID として使用されます。また、その ID は省略時の *schema* としてパッケージ内の表参照にも使用されます。この省略時の修飾子は、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バインド・オプションを使用します。

必須接続

データベース

API 組み込みファイル

sql.h

C API 構文

```
/* File: sql.h */
/* API: Rebind */
/* ... */
SQL_API_RC SQL_API_FN
  sqlarbnd (
    char * pPackageName,
    struct sqlca * pSqlca,
    struct sqlopt * pRebindOptions);
/* ... */
```

汎用 API 構文

```
/* File: sql.h */
/* API: Rebind */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgrbnd (
    unsigned short PackageNameLen,
    char * pPackageName,
    struct sqlca * pSqlca,
    struct sqlopt * pRebindOptions);
/* ... */
```

API パラメーター

PackageNameLen
入力。パッケージ名の長さを示す 2 バイトの無符号整数 (バイト単位) です。

pPackageName
入力。再バインドするパッケージを指定する、修飾子付きまたは修飾子なしの名前を含むストリングです。修飾子のないパッケージ名には、現行の許可 ID によって暗黙的に修飾子が付けられます。

pSqlca
出力。 *sqlca* 構造を指すポインターです。 この構造に関する詳細は、520ページの『SQLCA』を参照してください。

pRebindOptions
入力。 sqlopt 構造を指すポインターです。 API に再バインド・オプションを渡すのに使用します。この構造の詳細については、SQLOPT を参照してください。

REXX API 構文

この API は、SQLDB2 インターフェースを使って、REXX から呼び出すことができます。 13ページの『API の説明の編成』、または *アプリケーション開発の手引き* を参照してください。構文については、*コマンド解説書* を参照してください。

サンプル・プログラム

C	¥sqllib¥samples¥c¥rebind.sqc
COBOL	¥sqllib¥samples¥cobol¥rebind.sqb

使用上の注意

再バインドが正常に行われても、 REBIND がトランザクションを自動的にコミットすることはありません。ですから、ユーザー自身がトランザクションを明示的にコミットする必要があります。しかし、このことにより「what if」分析が可能になります。つまり、特定の統計を更新した後、変更した内容を見るためにパッケージの再バインドを試行できるようになります。さらに、1 作業単位内で複数の再バインドを実行することも可能になります。

この API の特徴を、以下にいくつか示します。

- パッケージを短時間で再作成できます。この API を使用することにより、元のバインド・ファイルがなくても、システム内の変更を利用できるようになります。たとえば、特定の SQL ステートメントが新しく作成した索引を利用できそうな場合、 REBIND によってパッケージを再作成できます。 REBIND によって、 477ページの『sqlustat - 統計の実行』 の実行後にパッケージを再作成することもできます。その結果、新規の統計を利用できるようになります。
- 作動不能パッケージを再作成できます。作動不能パッケージは、バインド・ユーティリティーまたは再バインド・ユーティリティーのどちらかを呼び出すことにより、明示的に再バインドしなければなりません。あるパッケージが依存する機能インスタンスが消去されると、そのパッケージは作動不能としてマークされます (SYSCAT.PACKAGES システム・カタログの VALID 列が X に設定されます)。
- 無効パッケージの再バインドに関する制御がユーザーに与えられます。無効パッケージは、その実行時にデータベース・マネージャーによって自動的に (暗黙的に) 再バインドされます。それにより、無効パッケージに関する最初の SQL 要求を実行するのに著しい遅延が生じることがあります。その場合、初期遅延をなくし、予期しない SQL エラー・メッセージが戻されることのないようにするためには (暗黙的な再バインドが失敗すると戻されることがある)、無効なパッケージをシステムに自動的に再バインドさせるのではなく、ユーザー自身が無効パッケージを明示的に再バインドするようにしてください。たとえば、移行が行われると、データベースに保管されているすべてのパッケージが DB2 バージョン 5 の移行処理によって無効にされます。これに多数のパッケージが関係している場合、一度にすべての無効パッケージを明示的に再バインドしてください。この明示的な再バインドは BIND、REBIND、または **db2rbind** ツール (*コマンド解説書* の 「db2rbind - すべてのパッケージの再バインド」を参照してください) 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かぎり、REBIND を使用するようにしてください。それは、REBIND の方が BIND よりもパフォーマンスの点で非常に優れているためです。ただし、以下の場合には必ず BIND を使用してください。

- プログラムに修正が加えられている場合 (たとえば、 SQL ステートメントが追加または削除された場合、またはパッケージがそのプログラムの実行可能モジュールと一致しない場合など)。
- 再バインドにおいてバインド・オプションのいずれかを変更したい場合。 REBIND はバインド・オプションをサポートしていません。たとえば、バインド処理の過程として、付与されたパッケージに対する特権を入手したい場合には、 BIND を使用しなければなりません。 BIND には **SQL_GRANT_OPT** オプションがあるからです。
- パッケージが現在ではデータベース内に存在していない場合。
- すべての バインド・エラーを検出することが必要な場合。 REBIND の場合、検出された最初のエラーだけが戻され、その後終了してしまいます。それに対し、BIND コマンドはバインド処理中に発生した最初の 100 エラーを戻してきます。

REBIND は DB2 コネクトによってサポートされています。

他のユーザーが使用中のパッケージ上で REBIND を実行しても、そのユーザーの論理作業単位が終了するまでは、再バインドは行われません。再バインド中、排他ロックが SYSCAT.PACKAGES システム・カタログ表にあるそのパッケージのレコードに保留されているからです。

REBIND を実行すると、データベース・マネージャーによって、 SYSCAT.STATEMENTS システム・カタログ表に保管されている SQL ステートメントを基にしたパッケージが再作成されます。

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。

次ページ以降の表には、再バインド・オプション構造 (589ページの『SQLOPT』を参照) の *type*、および *val* フィールドに有効な値がリストされています。さらに、対応する CLP オプションも掲載されています。再バインド・オプション (省略時値を含む) については、 *コマンド解説書* を参照してください。

表 6. REBIND オプションのタイプおよび値

CLP オプション	オプションのタイプ	オプションの値
RESOLVE ANY	SQL_RESOLVE_OPT	SQL_RESOLVE_ANY
RESOLVE CONSERVATIVE	SQL_RESOLVE_OPT	SQL_RESOLVE_CONSERVATIVE

参照資料

97ページの『sqlabndx - バインド』

477ページの『sqlustat - 統計の実行』

sqlbctcq - 表スペース・コンテナー照会のクローズ

表スペース・コンテナーの照会要求を終了し、関連したリソースを解放します。

許可

以下のいずれかです。

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

必須接続

データベース

API 組み込みファイル

sqlutil.h

C API 構文

```
/* File: sqlutil.h */
/* API: Close Tablespace Container Query */
/* ... */
SQL_API_RC SQL_API_FN
  sqlbctcq (
    struct sqlca * pSqlca);
/* ... */
```

汎用 API 構文

```
/* File: sqlutil.h */
/* API: Close Tablespace Container Query */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgctcq (
    struct sqlca * pSqlca);
/* ... */
```

API パラメーター

pSqlca
出力。 *sqlca* 構造を指すポインターです。 この構造に関する詳細は、520ページの『SQLCA』を参照してください。

サンプル・プログラム

C	¥sqllib¥samples¥c¥tabscont.sqc
COBOL	¥sqllib¥samples¥cobol¥tabscont.sqb

参照資料

123ページの『sqlbftcq - 表スペース・コンテナー照会の取り出し』

135ページの『sqlbotcq - 表スペース・コンテナー照会のオープン』

145ページの『sqlbstsc - 表スペース・コンテナーの設定』

149ページの『sqlbtcq - 表スペース・コンテナーの照会』

sqlbctsq - 表スペース照会のクローズ

表スペースの照会要求を終了し、関連したリソースを解放します。

許可

以下のいずれかです。

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

必須接続

データベース

API 組み込みファイル

sqlutil.h

C API 構文

```
/* File: sqlutil.h */
/* API: Close Tablespace Query */
/* ... */
SQL_API_RC SQL_API_FN
  sqlbctsq (
    struct sqlca * pSqlca);
/* ... */
```

汎用 API 構文

```
/* File: sqlutil.h */
/* API: Close Tablespace Query */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgctsq (
    struct sqlca * pSqlca);
/* ... */
```

API パラメーター

pSqlca

出力。 *sqlca* 構造を指すポインターです。 この構造に関する詳細は、520ページの『SQLCA』を参照してください。

サンプル・プログラム

C	¥sqllib¥samples¥c¥tabspace.sqc
COBOL	¥sqllib¥samples¥cobol¥tabspace.sqb

参照資料

126ページの『sqlbftpq - 表スペース照会の取り出し』

129ページの『sqlbgtss - 表スペース統計の入手』

138ページの『sqlbotsq - 表スペース照会のオープン』

142ページの『sqlbstpq - 単一の表スペースの照会』

132ページの『sqlbmtsq - 表スペースの照会』

sqlbftcq - 表スペース・コンテナー照会の取り出し

指定された行数の表スペース・コンテナーの照会データを取り出します。各行はコンテナー用のデータで構成されます。

効力範囲

区分データベース・サーバー環境では、現行のノード上の表スペースだけがリストされます。

許可

以下のいずれかです。

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

必須接続

データベース

API 組み込みファイル

sqlutil.h

C API 構文

```
/* File: sqlutil.h */
/* API: Fetch Tablespace Container Query */
/* ... */
SQL_API_RC SQL_API_FN
  sqlbftcq (
    struct sqlca * pSqlca,
    sqluint32 MaxContainers,
    struct SQLB_TBSCONTQRY_DATA * pContainerData,
    sqluint32 * pNumContainers);
/* ... */
```

汎用 API 構文

```
/* File: sqlutil.h */
/* API: Fetch Tablespace Container Query */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgftcq (
    struct sqlca * pSqlca,
    sqluint32 MaxContainers,
    struct SQLB_TBSCONTQRY_DATA * pContainerData,
    sqluint32 * pNumContainers);
/* ... */
```

API パラメーター

pSqlca
出力。 *sqlca* 構造を指すポインターです。 この構造に関する詳細は、520ページの『SQLCA』を参照してください。

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です。この構造に関する詳細については、513ページの『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。

サンプル・プログラム

C	¥sqllib¥samples¥c¥tabscont.sqc
COBOL	¥sqllib¥samples¥cobol¥tabscont.sqb

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** が生成したリストを取り出すこともできます。

詳細については、135ページの『sqlbotcq - 表スペース・コンテナー照会のオープン』を参照してください。

参照資料

119ページの『sqlbctcq - 表スペース・コンテナー照会のクローズ』

135ページの『sqlbotcq - 表スペース・コンテナー照会のオープン』

145ページの『sqlbstsc - 表スペース・コンテナーの設定』

149ページの『sqlbtcq - 表スペース・コンテナーの照会』

sqlbftpq - 表スペース照会の取り出し

指定された行数の表スペースの照会データを取り出します。各行は表スペース用のデータで構成されます。

効力範囲

区分データベース・サーバー環境では、現行のノード上の表スペースだけがリストされます。

許可

以下のいずれかです。

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

必須接続

データベース

API 組み込みファイル

sqlutil.h

C API 構文

```
/* File: sqlutil.h */
/* API: Fetch Tablespace Query */
/* ... */
SQL_API_RC SQL_API_FN
  sqlbftpq (
    struct sqlca * pSqlca,
    sqluint32 MaxTablespaces,
    struct SQLB_TBSPQRY_DATA * pTablespaceData,
    sqluint32 * pNumTablespaces);
/* ... */
```

汎用 API 構文

```
/* File: sqlutil.h */
/* API: Fetch Tablespace Query */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgftpq (
    struct sqlca * pSqlca,
    sqluint32 MaxTablespaces,
    struct SQLB_TBSPQRY_DATA * pTablespaceData,
    sqluint32 * pNumTablespaces);
/* ... */
```

API パラメーター

pSqlca
出力。 *sqlca* 構造を指すポインターです。 この構造に関する詳細は、520ページの『SQLCA』を参照してください。

MaxTablespaces
入力。ユーザー割り当ての出力域 (*pTablespaceData* によって指される) が保留できるデータの最大行数です。

pTablespaceData
入力 / 出力。データを照会するための構造である、出力域を指すポインターです。この構造に関する詳細については、515ページの『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。

サンプル・プログラム

C	¥sqllib¥samples¥c¥tabspace.sqc
COBOL	¥sqllib¥samples¥cobol¥tabspace.sqb

使用上の注意

ユーザーには、 *pTablespaceData* パラメーターによって指されるメモリーを割り振ったり解放したりする責任があります。この API を使用できるのは、 **sqlbotsq** 呼び出しが正常に行われた後だけです。この API を繰り返し呼び出して、 **sqlbotsq** が生成するリストを取り出すこともできます。

詳細については、138ページの『sqlbotsq - 表スペース照会のオープン』を参照してください。

参照資料

121ページの『sqlbctsq - 表スペース照会のクローズ』

129ページの『sqlbgtss - 表スペース統計の入手』

138ページの『sqlbotsq - 表スペース照会のオープン』

142ページの『sqlbstpq - 単一の表スペースの照会』

132ページの『sqlbmtsq - 表スペースの照会』

sqlbgtss - 表スペース統計の入手

表スペースの使用状況に関する情報を提供します。

効力範囲

区分データベース・サーバー環境では、現行のノード上の表スペースだけがリストされます。

許可

以下のいずれかです。

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

必須接続

データベース

API 組み込みファイル

sqlutil.h

C API 構文

```
/* File: sqlutil.h */
/* API: Get Tablespace Statistics */
/* ... */
SQL_API_RC SQL_API_FN
  sqlbgtss (
    struct sqlca * pSqlca,
    sqluint32 TablespaceId,
    struct SQLB_TBS_STATS * pTablespaceStats);
/* ... */
```

sqlbgtss - 表スペース統計の入手

汎用 API 構文

```
/* File: sqlutil.h */
/* API: Get Tablespace Statistics */
/* ... */
SQL_API_RC SQL_API_FN
  sqlggtss (
    struct sqlca * pSqlca,
    sqluint32 TablespaceId,
    struct SQLB_TBS_STATS * pTablespaceStats);
/* ... */
```

API パラメーター

pSqlca
出力。 *sqlca* 構造を指すポインターです。 この構造に関する詳細は、520ページの『SQLCA』を参照してください。

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です。表スペースに関する情報は、この構造に戻されます。この構造に関する詳細については、511ページの『SQLB-TBS-STATS』を参照してください。

サンプル・プログラム

C	¥sqllib¥samples¥c¥tabspace.sqc
COBOL	¥sqllib¥samples¥cobol¥tabspace.sqb

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、
511ページの『SQLB-TBS-STATS』を参照してください。

参照資料

121ページの『sqlbctsq - 表スペース照会のクローズ』

126ページの『sqlbftpq - 表スペース照会の取り出し』

138ページの『sqlbotsq - 表スペース照会のオープン』

142ページの『sqlbstpq - 単一の表スペースの照会』

132ページの『sqlbmtsq - 表スペースの照会』

sqlbmtsq - 表スペースの照会

表スペース照会データへの 1 回呼び出しインターフェースを提供します。データベース内のすべての表スペースの照会データが、 1 つの配列に戻されます。

効力範囲

区分データベース・サーバー環境では、現行のノード上の表スペースだけがリストされます。

許可

以下のいずれかです。

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

必須接続

データベース

API 組み込みファイル

sqlutil.h

C API 構文

```
/* File: sqlutil.h */
/* API: Tablespace Query */
/* ... */
SQL_API_RC SQL_API_FN
  sqlbmtsq (
    struct sqlca * pSqlca,
    sqluint32 * pNumTablespaces,
    struct SQLB_TBSPQRY_DATA *** pppTablespaceData,
    sqluint32 reserved1,
    sqluint32 reserved2);
/* ... */
```

汎用 API 構文

```
/* File: sqlutil.h */
/* API: Tablespace Query */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgmtsq (
    struct sqlca * pSqlca,
    sqluint32 * pNumTablespaces,
    struct SQLB_TBSPQRY_DATA *** pppTablespaceData,
    sqluint32 reserved1,
    sqluint32 reserved2);
/* ... */
```

API パラメーター

pSqlca
出力。 *sqlca* 構造を指すポインターです。 この構造に関する詳細は、520ページの『SQLCA』を参照してください。

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。呼び出しの戻りで、*SQLB_TBSPQRY_DATA* の配列を指すポインターが表スペース照合データの完全な集合を指します。

reserved1
入力。常に SQLB_RESERVED1 です。

reserved2
入力。常に SQLB_RESERVED2 です。

サンプル・プログラム

C	¥sqllib¥samples¥c¥tabspace.sqc
COBOL	¥sqllib¥samples¥cobol¥tabspace.sqb

使用上の注意

この API は、より低レベルなサービス、つまり以下の 3 つの API を利用します。

- 138ページの『sqlbotsq - 表スペース照会のオープン』
- 126ページの『sqlbftpq - 表スペース照会の取り出し』
- 121ページの『sqlbctsq - 表スペース照会のクローズ』

それにより、表スペース照会データのすべてを一度に入手できます。

十分な量のメモリーが利用可能であれば、この関数は表スペースの数を戻すとともに、表スペース照会データのメモリー・ロケーションを指すポインターも戻します。 **sqlefmem** (231ページの『sqlefmem - メモリーの解放』 を参照) 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、138ページの『sqlbotsq - 表スペース照会のオープン』、 126ページの『sqlbftpq - 表スペース照会の取り出し』、および 121ページの『sqlbctsq - 表スペース照会のクローズ』を使用して、完全ではないそのリストをとりあえず取り出してください。

参照資料

121ページの『sqlbctsq - 表スペース照会のクローズ』

126ページの『sqlbftpq - 表スペース照会の取り出し』

129ページの『sqlbgtss - 表スペース統計の入手』

138ページの『sqlbotsq - 表スペース照会のオープン』

142ページの『sqlbstpq - 単一の表スペースの照会』

sqlbotcq - 表スペース・コンテナー照会のオープン

表スペース・コンテナーの照会操作を実行し、現在、表スペースにあるコンテナーの数を戻します。

許可

以下のいずれかです。

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

必須接続

データベース

API 組み込みファイル

sqlutil.h

C API 構文

```
/* File: sqlutil.h */
/* API: Open Tablespace Container Query */
/* ... */
SQL_API_RC SQL_API_FN
  sqlbotcq (
    struct sqlca * pSqlca,
    sqluint32 TablespaceId,
    sqluint32 * pNumContainers);
/* ... */
```

sqlbotcq - 表スペース・コンテナー照会のオープン

汎用 API 構文

```
/* File: sqlutil.h */
/* API: Open Tablespace Container Query */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgotcq (
    struct sqlca * pSqlca,
    sqluint32 TablespaceId,
    sqluint32 * pNumContainers);
/* ... */
```

API パラメーター

pSqlca
出力。 *sqlca* 構造を指すポインターです。 この構造に関する詳細は、520ページの『SQLCA』を参照してください。

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。特殊な識別子 SQLB_ALL_TABLESPACES (sqlutil 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。

サンプル・プログラム

C	¥sqllib¥samples¥c¥tabscont.sqc
COBOL	¥sqllib¥samples¥cobol¥tabscont.sqb

使用上の注意

通常、この API の後には 123ページの『sqlbftcq - 表スペース・コンテナー照会の取り出し』 が 1 回かそれ以上呼び出され、その後 119ページの『sqlbctcq - 表スペース・コンテナー照会のクローズ』が 1 回呼び出されます。

アプリケーションは以下に挙げる API を呼び出して、表スペースで使用しているコンテナーに関する情報を取り出すことができます。

- 149ページの『sqlbtcq - 表スペース・コンテナーの照会』
  コンテナー情報の完全なリストを取り出します。この API は、全コンテナーの情報を保留するのに必要なスペースを割り振り、この情報を指すポインターを戻します。この API を使用して、コンテナーのリストを走査し、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 231ページの『sqlefmem - メモリーの解放』を呼び出して、メモリーを解放するようにしてください。

- 135ページの『sqlbotcq - 表スペース・コンテナー照会のオープン』
- 123ページの『sqlbftcq - 表スペース・コンテナー照会の取り出し』
- 119ページの『sqlbctcq - 表スペース・コンテナー照会のクローズ』

前述の 3 つの API は、SQL カーソルと同様の働きをします。つまり、OPEN/FETCH/CLOSE パラダイムを使用します。呼び出し元は、取り出しのための出力域を提供する必要があります。 SQL カーソルの場合とは異なり、一度に活動状態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走査し、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 (149ページの『sqlbtcq - 表スペース・コンテナーの照会』 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。

ロック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。

参照資料

119ページの『sqlbctcq - 表スペース・コンテナー照会のクローズ』

123ページの『sqlbftcq - 表スペース・コンテナー照会の取り出し』

145ページの『sqlbstsc - 表スペース・コンテナーの設定』

149ページの『sqlbtcq - 表スペース・コンテナーの照会』

sqlbotsq - 表スペース照会のオープン

表スペースの照会操作を実行し、現在データベースにある表スペースの数を戻します。

許可

以下のいずれかです。

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

必須接続

データベース

API 組み込みファイル

sqlutil.h

C API 構文

```
/* File: sqlutil.h */
/* API: Open Tablespace Query */
/* ... */
SQL_API_RC SQL_API_FN
  sqlbotsq (
    struct sqlca * pSqlca,
    sqluint32 TablespaceQueryOptions,
    sqluint32 * pNumTablespaces);
/* ... */
```

汎用 API 構文

```
/* File: sqlutil.h */
/* API: Open Tablespace Query */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgotsq (
    struct sqlca * pSqlca,
    sqluint32 TablespaceQueryOptions,
    sqluint32 * pNumTablespaces);
/* ... */
```

API パラメーター

pSqlca
出力。 *sqlca* 構造を指すポインターです。 この構造に関する詳細は、520ページの『SQLCA』を参照してください。

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復元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。

サンプル・プログラム

C	¥sqllib¥samples¥c¥tabspace.sqc
COBOL	¥sqllib¥samples¥cobol¥tabspace.sqb

使用上の注意

通常、この API の後には 126ページの『sqlbftpq - 表スペース照会の取り出し』が 1 回以上呼び出され、その後 121ページの『sqlbctsq - 表スペース照会のクローズ』が 1 回呼び出されます。

アプリケーションは以下に挙げる API を呼び出して、現在定義されている表スペースに関する情報を取り出すことができます。

- 142ページの『sqlbstpq - 単一の表スペースの照会』

  指定された表スペースに関する情報を取り出します。ただ 1 つの表スペース項目だけが (呼び出し側で提供したスペースに) 戻されます。表スペースの識別子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。
- 132ページの『sqlbmtsq - 表スペースの照会』

  すべての表スペースに関する情報を取り出します。この API は、全表スペースの情報を保留するのに必要なスペースを割り振り、この情報を指すポインターを戻します。この API を使用して、特定の情報の探索時に、表スペースのリストを走査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 231ページの『sqlefmem - メモリーの解放』を呼び出して、メモリーを解放するようにしてください。
- 138ページの『sqlbotsq - 表スペース照会のオープン』
- 126ページの『sqlbftpq - 表スペース照会の取り出し』
- 121ページの『sqlbctsq - 表スペース照会のクローズ』

  前述の 3 つの API は、SQL カーソルと同様の働きをします。つまり、OPEN/FETCH/CLOSE パラダイムを使用します。呼び出し元は、取り出しのための出力域を提供する必要があります。 SQL カーソルの場合とは異なり、一度に活動状態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走査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 (132ページの『sqlbmtsq - 表スペースの照会』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 (**sqlbtsq** 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に置き換えられます。

ロック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。

参照資料

121ページの『sqlbctsq - 表スペース照会のクローズ』

126ページの『sqlbftpq - 表スペース照会の取り出し』

142ページの『sqlbstpq - 単一の表スペースの照会』

132ページの『sqlbmtsq - 表スペースの照会』

sqlbstpq - 単一の表スペースの照会

現在定義されている単一の表スペースに関する情報を検索します。

効力範囲

区分データベース・サーバー環境では、現行のノード上の表スペースだけがリストされます。

許可

以下のいずれかです。

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

必須接続

データベース

API 組み込みファイル

sqlutil.h

C API 構文

```
/* File: sqlutil.h */
/* API: Single Tablespace Query */
/* ... */
SQL_API_RC SQL_API_FN
  sqlbstpq (
    struct sqlca * pSqlca,
    sqluint32 TablespaceId,
    struct SQLB_TBSPQRY_DATA * pTablespaceData,
    sqluint32 reserved);
/* ... */
```

汎用 API 構文

```
/* File: sqlutil.h */
/* API: Single Tablespace Query */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgstpq (
    struct sqlca * pSqlca,
    sqluint32 TablespaceId,
    struct SQLB_TBSPQRY_DATA * pTablespaceData,
    sqluint32reserved);
/* ... */
```

API パラメーター

pSqlca
出力。 *sqlca* 構造を指すポインターです。 この構造に関する詳細は、520ページの『SQLCA』を参照してください。

TablespaceId
入力。照会する表スペースの識別子です。

pTablespaceData
入力 / 出力。表スペース情報が戻される、ユーザー提供の *SQLB_TBSPQRY_DATA* 構造を指すポインターです。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。

サンプル・プログラム

C	¥sqllib¥samples¥c¥tabspace.sqc
COBOL	¥sqllib¥samples¥cobol¥tabspace.sqb

使用上の注意

この API は、照会したい表スペースの識別子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識別子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、その表スペースを走査し

ます。表スペース ID は、システム・カタログから探し出すことができます。戻される項目が 1 つだけなので、エージェントのスナップショットは取られません。その項目が直接戻されます。

詳細については、138ページの『sqlbotsq - 表スペース照会のオープン』を参照してください。

参照資料

121ページの『sqlbctsq - 表スペース照会のクローズ』

126ページの『sqlbftpq - 表スペース照会の取り出し』

129ページの『sqlbgtss - 表スペース統計の入手』

138ページの『sqlbotsq - 表スペース照会のオープン』

132ページの『sqlbmtsq - 表スペースの照会』

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復元するよう宛先変更された 復元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集合が必要になります。

表スペースが*記憶域定義保留中* 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。これらの状態は復元操作中、データベース・ページを復元する直前に起こり得るものです。

許可

以下のいずれかです。

- *sysadm*
- *sysctrl*

必須接続

データベース

API 組み込みファイル

sqlutil.h

C API 構文

```
/* File: sqlutil.h */
/* API: Set Tablespace Containers */
/* ... */
SQL_API_RC SQL_API_FN
  sqlbstsc (
    struct sqlca * pSqlca,
    sqluint32 SetContainerOptions,
    sqluint32 TablespaceId,
    sqluint32 NumContainers,
    struct SQLB_TBSCONTQRY_DATA * pContainerData);
/* ... */
```

汎用 API 構文

```
/* File: sqlutil.h */
/* API: Set Tablespace Containers */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgstsc (
    struct sqlca * pSqlca,
    sqluint32 SetContainerOptions,
    sqluint32 TablespaceId,
    sqluint32 NumContainers,
    struct SQLB_TBSCONTQRY_DATA * pContainerData);
/* ... */
```

API パラメーター

pSqlca

出力。 *sqlca* 構造を指すポインターです。 この構造に関する詳細は、520ページの『SQLCA』を参照してください。

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識別子です。

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。

サンプル・プログラム

C ¥sqllib¥samples¥c¥backrest.c

COBOL ¥sqllib¥samples¥cobol¥backrest.cbl

使用上の注意

この API は、447ページの『sqlurestore - データベースの復元』とともに使用されます。

データベース、または 1 つまたは複数の表スペースのバックアップを取ることにより、バックアップしようとする表スペースが使用しているすべての表スペース・コンテナーのレコードを保持できます。復元中、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、復元は失敗してしまいます。そのような場合であっても復元を行えるようにするため、復元中は表スペース・コンテナーの宛先変更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。詳細については、*管理の手引き* を参照してください。

通常、この API は以下のような順序のアクションで使用します。

1. *CallerAction* を SQLUD_RESTORE_STORDEF に設定して、 447ページの『sqlurestore - データベースの復元』を呼び出す。

   復元ユーティリティーによって、コンテナーの一部がアクセス不能であることを示す *sqlcode* が戻されます。
2. **sqlbstsc** を呼び出して、表スペース・コンテナーの定義を設定する。 *SetContainerOptions* パラメーターは SQLB_SET_CONT_FINAL_STATE に設定します。
3. *CallerAction* を SQLUD_CONTINUE に設定して、もう 1 度 **sqlurst** を呼び出す。

上記の手順により、新規の表スペース・コンテナーの定義を復元に使えるようになります。また、復元の完了後に 465ページの『sqluroll - データベースのロールフォワード』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、復元またはロールフォワードが元のデータをすべてそれら新規のコンテナーに置き換えられるよう十分なディスク・スペースが必要であることに注意してください。十分

なスペースがないと、それらの表スペースは、十分なディスク・スペースが利用可能になるまで、 *recovery pending* 状態のままになります。賢明なデータベース管理者であれば、ディスク使用状況のレコードを保持しているはずです。その後、復元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。

参照資料

345ページの『sqlubkp - データベースのバックアップ』

465ページの『sqluroll - データベースのロールフォワード』

447ページの『sqlurestore - データベースの復元』

sqlbtcq - 表スペース・コンテナーの照会

表スペース・コンテナー照会データへの 1 回呼び出しインターフェースを提供します。特定の表スペース内のすべてのコンテナー、またはすべての表スペース内のすべてのコンテナーの照会データが、 1 つの配列に戻されます。

効力範囲

区分データベース・サーバー環境では、現行のノード上の表スペースだけがリストされます。

許可

以下のいずれかです。

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

必須接続

データベース

API 組み込みファイル

sqlutil.h

C API 構文

```
/* File: sqlutil.h */
/* API: Tablespace Container Query */
/* ... */
SQL_API_RC SQL_API_FN
  sqlbtcq (
    struct sqlca * pSqlca,
    sqluint32 TablespaceId,
    sqluint32 * pNumContainers,
    struct SQLB_TBSCONTQRY_DATA ** ppContainerData);
/* ... */
```

汎用 API 構文

```
/* File: sqlutil.h */
/* API: Tablespace Container Query */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgtcq (
    struct sqlca * pSqlca,
    sqluint32 TablespaceId,
    sqluint32 * pNumContainers,
    struct SQLB_TBSCONTQRY_DATA ** ppContainerData);
/* ... */
```

API パラメーター

pSqlca

出力。 *sqlca* 構造を指すポインターです。 この構造に関する詳細は、520ページの『SQLCA』を参照してください。

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。

サンプル・プログラム

C	¥sqllib¥samples¥c¥tabscont.sqc
COBOL	¥sqllib¥samples¥cobol¥tabscont.sqb

使用上の注意

この API は、より低レベルなサービス、つまり以下の 3 つの API を利用します。

- 135ページの『sqlbotcq - 表スペース・コンテナー照会のオープン』
- 123ページの『sqlbftcq - 表スペース・コンテナー照会の取り出し』
- 119ページの『sqlbctcq - 表スペース・コンテナー照会のクローズ』

それにより、表スペース・コンテナー照会データのすべてを一度に入手できます。

十分な量のメモリーが利用可能であれば、この関数はコンテナーの数を戻すとともに、表スペース・コンテナー照会データのメモリー・ロケーションを指すポインターも戻します。 **sqlefmem** (231ページの『sqlefmem - メモリーの解放』 を参照) 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、135ページの『sqlbotcq - 表スペース・コンテナー照会のオープン』、 123ページの『sqlbftcq - 表スペース・コンテナー照会の取り出し』、および 119ページの『sqlbctcq - 表スペース・コンテナー照会のクローズ』を使用して、完全ではないそのリストをとりあえず取り出してください。

参照資料

119ページの『sqlbctcq - 表スペース・コンテナー照会のクローズ』

123ページの『sqlbftcq - 表スペース・コンテナー照会の取り出し』

135ページの『sqlbotcq - 表スペース・コンテナー照会のオープン』

145ページの『sqlbstsc - 表スペース・コンテナーの設定』

149ページの『sqlbtcq - 表スペース・コンテナーの照会』

sqlcspqy - DRDA 未確定トランザクションのリスト

LU 6.2 プロトコルで接続されたパートナー LU 間で、未確定のトランザクションのリストを提供します。

許可

sysadm

必須接続

インスタンス。

API 組み込みファイル

sqlxa.h

C API 構文

```
/* File: sqlxa.h */
/* API: List DRDA Indoubt Transactions */
/* ... */
extern int SQL_API_FN sqlcspqy(SQLCSPQY_INDOUBT     **indoubt_data,
                               sqlint32             *indoubt_count,
                               struct sqlca          *sqlca);
/* ... */
```

API パラメーター

indoubt_data
出力。戻された配列を指すポインターです。

indoubt_count
出力。戻された配列内の要素数を示します。

pSqlca
出力。 *sqlca* 構造を指すポインターです。 この構造に関する詳細は、520ページの『SQLCA』を参照してください。

使用上の注意

分散作業単位内で調整プログラムと参加プログラムの間に通信がなくなるときに、 DRDA 未確定トランザクションが発生します。

分散作業単位により、ユーザーまたはアプリケーションは、単一作業単位内の複数のロケーションにあるデータを読み取ったり更新したりすることができます。ただし、そのような作業には、2 フェーズ・コミットが必要です。

最初のフェーズで、すべての参加プログラムがコミットの準備をするよう要求します。次のフェーズで、トランザクションのコミットまたはロールバックを行います。最初のフェーズの後に調整プログラムか参加プログラムが使用できなくなると、分散トランザクションが未確定になります。

LIST DRDA INDOUBT TRANSACTIONS を発行する前に、アプリケーション処理は同期点マネージャー (SPM) インスタンスに接続されていなければなりません。 SPM_NAME を CONNECT ステートメント上の *dbalias* として使用します (CONNECT の使用についての詳細は、 *SQL 解説書* を参照してください)。 SPM_NAME は、データベース・マネージャーの構成パラメーターです。

sqle_activate_db - データベースの活動化

指定されたデータベースを活動化し、必要なデータベースのサービスをすべて開始します。こうして、データベースの接続が使用可能になり、任意のアプリケーションが使用できるようになります。

効力範囲

この API は、システム内のすべてのノードで、指定されたデータベースを活動化します。データベースの活動化中に 1 つまたは複数のノード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ノード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調整プログラム・ノードかカタログ・ノードの場合、API は否定の *sqlcode* を戻し、どのノード上でもデータベースは活動化されません。

許可

以下のいずれかです。

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

必須接続

ありません。 ACTIVATE DATABASE を呼び出しているアプリケーションには、既存のデータベース接続は全くありません。

API 組み込みファイル

sqlenv.h

C API 構文

```
/* File: sqlenv.h */
/* API: Activate Database */
/* ... */
SQL_API_RC SQL_API_FN
  sqle_activate_db (
    char * pDbAlias,
    char * pUserName,
    char * pPassword,
    void * pReserved,
    struct sqlca * pSqlca);
/* ... */
```

汎用 API 構文

```
/* File: sqlenv.h */
/* API: Activate Database */
/* ... */
SQL_API_RC SQL_API_FN
  sqlg_activate_db (
    unsigned short DbAliasLen,
    unsigned short UserNameLen,
    unsigned short PasswordLen,
    char * pDbAlias,
    char * pUserName,
    char * pPassword,
    void * pReserved,
    struct sqlca * pSqlca);
/* ... */
```

API パラメーター

DbAliasLen

入力。データベース別名の長さを示す 2 バイトの無符号整数 (バイト単位) です。

UserNameLen

入力。ユーザー名の長さを示す 2 バイトの無符号整数 (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 バイトの無符号整数 (バイト単位) です。パスワードが提供されていない場合は、ゼロに設定してください。

pDbAlias
入力。データベースの別名を指すポインターです。

pUserName
入力。データベースを開始させるユーザー ID を指すポインターです。ヌル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です。ヌル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
将来の使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です。 この構造に関する詳細は、520ページの『SQLCA』を参照してください。

REXX API 構文

この API は、SQLDB2 インターフェースを使って、REXX から呼び出すことができます。 13ページの『API の説明の編成』、または *アプリケーション開発の手引き* を参照してください。構文については、*コマンド解説書* を参照してください。

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、 158ページの『sqle_deactivate_db - データベースの非活動化』または 276ページの『sqlepstp - データベース・マネージャーの停止』を使用しないと遮断できません。活動化されたデータベースのリストを入手するには、 31ページの『db2GetSnapshot - スナップショットの入手』を呼び出してください。

データベースが DB2 CONNECT TO (または暗黙接続) で開始され、続いてその同じデータベースに ACTIVATE DATABASE が発行される場合、そのデータベースを遮断するには DEACTIVATE DATABASE を使用しなければなりません。

再始動させる必要があるデータベース (たとえば、矛盾状態にあるデータベース) での処理時には、 ACTIVATE DATABASE は DB2 CONNECT TO (または暗黙接続) と同じような働きをします。 ACTIVATE DATABASE で初期設定される前に、データベースは再始動します。

参照資料

158ページの『sqle_deactivate_db - データベースの非活動化』

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。

効力範囲

MPP システムでは、この API がシステム内のすべてのノードにある、指定されたデータベースを非活動化します。 1 つまたは複数のノードでエラーが起きた場合、警告が戻されます。一部のノードではデータベースの非活動化が正常に行われても、エラーが起きたノード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調整ノードかカタログ・ノードの場合、 API は否定の *sqlcode* を戻し、データベースが非活動化されたノード上での再活動化は行われません。

許可

以下のいずれかです。

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

必須接続

ありません。 DEACTIVATE DATABASE を呼び出しているアプリケーションには、既存のデータベース接続は全くありません。

API 組み込みファイル

sqlenv.h

C API 構文

```
/* File: sqlenv.h */
/* API: Deactivate Database */
/* ... */
SQL_API_RC SQL_API_FN
  sqle_deactivate_db (
    char * pDbAlias,
    char * pUserName,
    char * pPassword,
    void * pReserved,
    struct sqlca * pSqlca);
/* ... */
```

汎用 API 構文

```
/* File: sqlenv.h */
/* API: Deactivate Database */
/* ... */
SQL_API_RC SQL_API_FN
  sqlg_deactivate_db (
    unsigned short DbAliasLen,
    unsigned short UserNameLen,
    unsigned short PasswordLen,
    char * pDbAlias,
    char * pUserName,
    char * pPassword,
    void * pReserved,
    struct sqlca * pSqlca);
/* ... */
```

API パラメーター

DbAliasLen

入力。データベース別名の長さを示す 2 バイトの無符号整数 (バイト単位) です。

UserNameLen

入力。ユーザー名の長さを示す 2 バイトの無符号整数 (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 バイトの無符号整数 (バイト単位) です。パスワードが提供されていない場合は、ゼロに設定してください。

pDbAlias
入力。データベースの別名を指すポインターです。

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 を指すポインターです。ヌル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です。ヌル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
将来の使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です。 この構造に関する詳細は、520ページの『SQLCA』を参照してください。

REXX API 構文

この API は、SQLDB2 インターフェースを使って、REXX から呼び出すことができます。 13ページの『API の説明の編成』、または *アプリケーション開発の手引き* を参照してください。構文については、*コマンド解説書* を参照してください。

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 を使用しないと遮断できません。データベース・マネージャーが停止する前に、276ページの『sqlepstp - データベース・マネージャーの停止』によって、活動化されたデータベースがすべて自動的に停止されます。データベースが ACTIVATE DATABASE で初期設定されたのであれば、 DB2 CONNECT RESET ステートメント (0 に等しいカウンター) では遮断しないので、 DEACTIVATE DATABASE を使用しなければなりません。

参照資料

154ページの『sqle_activate_db - データベースの活動化』

sqleaddn - ノードの追加

並列データベース・システムに新規ノードを追加します。この API は、 MPP サーバー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、新規ノード上にデータベース区画を作成します。ユーザーは、任意のシステム一時表スペース用のソース・ノードをデータベースとともに作成するか、またはシステム一時表スペースを作成しないかを指定することができます。 API は追加するノードから発行し、必ず MPP サーバー上で行ってください。

効力範囲

この API は、それが実行されるノードにのみ影響を与えます。

許可

以下のいずれかです。

- *sysadm*
- *sysctrl*

必須接続

ありません。

API 組み込みファイル

sqlenv.h

C API 構文

```
/* File: sqlenv.h */
/* API: Add Node */
/* ... */
SQL_API_RC SQL_API_FN
  sqleaddn (
    void * pAddNodeOptions,
    struct sqlca * pSqlca);
/* ... */
```

汎用 API 構文

```
/* File: sqlenv.h */
/* API: Add Node */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgaddn (
    unsigned short addnOptionsLen,
    struct sqlca * pSqlca,
    void * pAddNodeOptions);
/* ... */
```

API パラメーター

addnOptionsLen
入力。オプション *sqle_addn_options* 構造の長さを示す 2 バイトの無符号整数 (バイト単位) です。

pSqlca
出力。 *sqlca* 構造を指すポインターです。 この構造に関する詳細は、520ページの『SQLCA』を参照してください。

pAddNodeOptions
入力。オプション *sqle_addn_options* 構造を指すポインターです。システム一時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・ノードを指定します。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区画に関するものです。何も指定されていない場合 (つまり、ヌル・ポインターが指定されている場合)、システム一時表スペース定義はカタログ・ノードの定義と同じになります。この構造に関する詳細については、 530ページの『SQLE-ADDN-OPTIONS』を参照してください。

REXX API 構文

この API は、SQLDB2 インターフェースを使って、REXX から呼び出すことができます。 13ページの『API の説明の編成』、または *アプリケーション開発の手引き* を参照してください。構文については、*コマンド解説書* を参照してください。

使用上の注意

新規のノードを追加する前に、十分な記憶域を確保しておいてください。この記憶域には、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、新規ノード上に、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区画が作成されます。新規のデータベース区画用の構成パラメーターは省略時値に設定されます。

データベース区画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に除去されます。これは、追加されたノード (つまり、ローカル・ノード) からだけデータベース区画が除去されるということです。その他のノード上に存在しているデータベース区画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 NODEGROUP ステートメントを使用してノードをノード・グループに追加してからでなければ、新規のノード上にデータベース区画にユーザー・データを含めることはできません。詳細については、*SQL 解説書* を参照してください。

データベース作成操作またはデータベースの消去操作が進行中の場合、この API は正常に実行されません。操作が一度完了してしまうと、この API をもう一度呼び出すことができます。

データベース区画とともにシステム一時表スペースが作成された場合、表スペース定義を検索するために、 **sqleaddn** は MPP システム内の他のノードと通信する必要があります。 *start_stop_time* データベース・マネージャー構成パラメーターを使用して時間 (分) を指定します。 MPP システム内の他のノードはこの時間内で表スペース定義に答える必要があります。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。

参照資料

186ページの『sqlecrea - データベースの作成』

226ページの『sqledrpn - ノードのドロップの検査』

273ページの『sqlepstart - データベース・マネージャーの始動』

sqleatcp - パスワードの接続と変更

インスタンス・レベルの関数 (たとえば、 CREATE DATABASE や FORCE APPLICATION) が実行されるノードを、アプリケーションが指定できるようにします。このノードには、現行のインスタンス (**DB2INSTANCE** 環境変数の値で定義された) はもちろん、同一ワークステーション上の別のインスタンスを指定することもできますし、リモート・ワークステーション上のインスタンスでもかまいません。指定されたノードへの論理的なインスタンスの接続が確立されますが、物理的な接続がまだ存在していない場合には、そのノードへの物理的な通信接続が開始されます。

注: この API は、接続中のインスタンスのユーザー・パスワードのオプションを変更できるようにして、 169ページの『sqleatin - 接続』の機能を拡張します。

許可

ありません。

必須接続

この API はインスタンスの接続を確立します。

API 組み込みファイル

sqlenv.h

C API 構文

```
/* File: sqlenv.h */
/* API: Attach and Change Password */
/* ... */
SQL_API_RC SQL_API_FN
  sqleatcp (
    char * pNodeName,
    char * pUserName,
    char * pPassword,
    char * pNewPassword,
    struct sqlca * pSqlca);
/* ... */
```

汎用 API 構文

```
/* File: sqlenv.h */
/* API: Attach and Change Password */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgatcp (
    unsigned short NewPasswordLen,
    unsigned short PasswordLen,
    unsigned short UserNameLen,
    unsigned short NodeNameLen,
    struct sqlca * pSqlca,
    char * pNewPassword,
    char * pPassword,
    char * pUserName,
    char * pNodeName);
/* ... */
```

API パラメーター

NewPasswordLen

入力。新規パスワードの長さを示す 2 バイトの無符号整数 (バイト単位) です。新規パスワードが提供されていない場合は、ゼロに設定してください。

PasswordLen

入力。パスワードの長さを示す 2 バイトの無符号整数 (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 バイトの無符号整数 (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 バイトの無符号整数 (バイト単位) です。ノード名が提供されていない場合は、ゼロに設定してください。

pSqlca

出力。 *sqlca* 構造を指すポインターです。 この構造に関する詳細は、520ページの『SQLCA』を参照してください。

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、ヌルに設定してください。

pPassword
入力。指定されたユーザー名のパスワードを含むストリングを示します。ヌルにすることもできます。

pUserName
入力。接続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。ヌルにすることもできます。

pNodeName
入力。接続したいインスタンスの別名を含むストリングを指定します。このインスタンスには、ローカル・ノード・ディレクトリーに一致する項目がなければなりません。唯一の例外は、ローカル・インスタンス (**DB2INSTANCE** 環境変数で指定された) です。このローカル・インスタンスは、接続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。ヌルにすることもできます。

REXX API 構文

この API を REXX から直接呼び出すことはできません。ただし、REXX プログラマーは、 DB2 コマンド行プロセッサーを呼び出して ATTACH コマンドを実行することにより、この関数を利用できます。詳細については、*アプリケーション開発の手引き* を参照してください。

サンプル・プログラム

C	¥sqllib¥samples¥c¥dbinst.c
COBOL	¥sqllib¥samples¥cobol¥dbinst.cbl

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みなすことができます。

接続要求が成功すると、 *sqlca* の *sqlerrmc* フィールドには、 16 進数 FF で区切られた 9 つのトークンが含まれるようになります (CONNECT 要求が正常に実行されたときに戻されるトークンと同様です)。

1. アプリケーション・サーバーの国別コード

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 (*SQL 解説書* を参照してください。)
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. サーバーが区分データベース・サーバーの場合は区分の数

ノード名が長さゼロのストリングまたはヌルの場合、現行の接続状態に関する情報が戻されます。接続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、その接続に関する情報が *sqlca* の *sqlerrmc* フィールドに戻されます (上述のとおりです)。

接続が実行されなかった場合、インスタンス・レベルの API が **DB2INSTANCE** 環境変数で指定された現行のインスタンスに対して実行されます。

特定の関数 (**db2start**、**db2stop**、およびすべてのディレクトリー・サービスなど) は、リモートには実行されません。つまり、それらの関数が影響を与えるのは、 **DB2INSTANCE** 環境変数の値で定義されたローカル・インスタンス環境に対してのみです。

接続が存在し、かつ API がノード名を指定して発行された場合、現行の接続は消去され、新規ノードへの接続が試行されます。

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。認証タイプに関する詳細については、*管理の手引き* を参照してください。

接続が確立されたノードは、 294ページの『sqlesetc - クライアントの設定』を呼び出すことによっても指定できます (535ページの『SQLE-CONN-SETTING』の SQL_ATTACH_NODE オプションを参照してください)。

参照資料

169ページの『sqleatin - 接続』

229ページの『sqledtin - 切断』

294ページの『sqlesetc - クライアントの設定』

sqleatin - 接続

インスタンス・レベルの関数 (たとえば、 CREATE DATABASE や FORCE APPLICATION) が実行されるノードを、アプリケーションが指定できるようにします。このノードには、現行のインスタンス (**DB2INSTANCE** 環境変数の値で定義された) はもちろん、同一ワークステーション上の別のインスタンスを指定することもできますし、リモート・ワークステーション上のインスタンスでもかまいません。指定されたノードへの論理的なインスタンスの接続が確立されますが、物理的な接続がまだ存在していない場合には、そのノードへの物理的な通信接続が開始されます。

注: パスワード変更が必要な場合は、 **sqleatin** ではなく 164ページの『sqleatcp - パスワードの接続と変更』を使用してください。

許可

ありません。

必須接続

この API はインスタンスの接続を確立します。

API 組み込みファイル

sqlenv.h

C API 構文

```
/* File: sqlenv.h */
/* API: Attach */
/* ... */
SQL_API_RC SQL_API_FN
  sqleatin (
    char * pNodeName,
    char * pUserName,
    char * pPassword,
    struct sqlca * pSqlca);
/* ... */
```

汎用 API 構文

```
/* File: sqlenv.h */
/* API: Attach */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgatin (
    unsigned short PasswordLen,
    unsigned short UserNameLen,
    unsigned short NodeNameLen,
    struct sqlca * pSqlca,
    char * pPassword,
    char * pUserName,
    char * pNodeName);
/* ... */
```

API パラメーター

PasswordLen
入力。パスワードの長さを示す 2 バイトの無符号整数 (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 バイトの無符号整数 (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 バイトの無符号整数 (バイト単位) です。ノード名が提供されていない場合は、ゼロに設定してください。

pSqlca
出力。 *sqlca* 構造を指すポインターです。 この構造に関する詳細は、520ページの『SQLCA』を参照してください。

pPassword
入力。指定されたユーザー名のパスワードを含むストリングを示します。ヌルにすることもできます。

pUserName
入力。接続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。ヌルにすることもできます。

pNodeName
入力。接続したいインスタンスの別名を含むストリングを指定します。

このインスタンスには、ローカル・ノード・ディレクトリーに一致する項目がなければなりません。唯一の例外は、ローカル・インスタンス (**DB2INSTANCE** 環境変数で指定された) です。このローカル・インスタンスは、接続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。ヌルにすることもできます。

REXX API 構文

```
ATTACH [TO nodename [USER username USING password]]
```

REXX API パラメーター

nodename
ユーザーが接続したいインスタンスの別名。このインスタンスには、ローカル・ノード・ディレクトリーに一致する項目がなければなりません。唯一の例外は、ローカル・インスタンス (**DB2INSTANCE** 環境変数で指定された) です。このローカル・インスタンスは、接続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
インスタンスに接続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。

サンプル・プログラム

C	¥sqllib¥samples¥c¥dbinst.c
COBOL	¥sqllib¥samples¥cobol¥dbinst.cbl
REXX	¥sqllib¥samples¥rexx¥dbinst.cmd

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みなすことができます。

接続要求が成功すると、 *sqlca* の *sqlerrmc* フィールドには、 16 進数 FF で区切られた 9 つのトークンが含まれるようになります (CONNECT 要求が正常に実行されたときに戻されるトークンと同様です)。

1. アプリケーション・サーバーの国別コード
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 (*SQL 解説書* を参照してください。)
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. サーバーが区分データベース・サーバーの場合は区分の数

ノード名が長さゼロのストリングまたはヌルの場合、現行の接続状態に関する情報が戻されます。接続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、その接続に関する情報が *sqlca* の *sqlerrmc* フィールドに戻されます (上述のとおりです)。

接続が実行されなかった場合、インスタンス・レベルの API が **DB2INSTANCE** 環境変数で指定された現行のインスタンスに対して実行されます。

特定の関数 (**db2start**、**db2stop**、およびすべてのディレクトリー・サービスなど) は、リモートには実行されません。つまり、それらの関数が影響を与えるのは、 **DB2INSTANCE** 環境変数の値で定義されたローカル・インスタンス環境に対してのみです。

接続が存在し、かつ API がノード名を指定して発行された場合、現行の接続は消去され、新規ノードへの接続が試行されます。

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。認証タイプに関する詳細については、*管理の手引き* を参照してください。

接続が確立されたノードは、 294ページの『sqlesetc - クライアントの設定』を呼び出すことによっても指定できます (535ページの『SQLE-CONN-SETTING』の SQL_ATTACH_NODE オプションを参照してください)。

参照資料

164ページの『sqleatcp - パスワードの接続と変更』

229ページの『sqledtin - 切断』

294ページの『sqlesetc - クライアントの設定』

sqlecadb - データベースのカタログ

システム・データベース・ディレクトリーに、データベースのロケーション情報を保管します。データベースは、ローカル・ワークステーションかリモート・ノードのどちらかに位置付けることができます。

効力範囲

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。区分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サーバー上のノードからこの API を呼び出す必要があります。

許可

以下のいずれかです。

- *sysadm*
- *sysctrl*

必須接続

ありません。

API 組み込みファイル

sqlenv.h

C API 構文

```
/* File: sqlenv.h */
/* API: Catalog Database */
/* ... */
SQL_API_RC SQL_API_FN
  sqlecadb (
    _SQLOLDCHAR * pDbName,
    _SQLOLDCHAR * pDbAlias,
    unsigned char Type,
    _SQLOLDCHAR * pNodeName,
    _SQLOLDCHAR * pPath,
    _SQLOLDCHAR * pComment,
    unsigned short Authentication,
    _SQLOLDCHAR * pPrincipal,
    struct sqlca * pSqlca);
/* ... */
```

汎用 API 構文

```
/* File: sqlenv.h */
/* API: Catalog Database */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgcadb (
    unsigned short PrinLen,
    unsigned short CommentLen,
    unsigned short PathLen,
    unsigned short NodeNameLen,
    unsigned short DbAliasLen,
    unsigned short DbNameLen,
    struct sqlca * pSqlca,
    _SQLOLDCHAR * pPrinName,
    unsigned short Authentication,
    _SQLOLDCHAR * pComment,
    _SQLOLDCHAR * pPath,
    _SQLOLDCHAR * pNodeName,
    unsigned char Type,
    _SQLOLDCHAR * pDbAlias,
    _SQLOLDCHAR * pDbName);
/* ... */
```

API パラメーター

PrinLen
入力。プリンシパル名の長さを示す 2 バイトの無符号整数 (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。この値は、認証が SQL_AUTHENTICATION_DCE または SQL_AUTHENTICATION_KERBEROS として指定されている場合に限り、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 バイトの無符号整数 (バイト単位) です。コメントが提供されていない場合は、ゼロに設定してください。

PathLen
入力。ローカル・データベース・ディレクトリーのパスの長さを示す 2 バイトの無符号整数 (バイト単位) です。パスが提供されていない場合は、ゼロに設定してください。

NodeNameLen
入力。ノード名の長さを示す 2 バイトの無符号整数 (バイト単位) です。ノード名が提供されていない場合には、ゼロに設定してください。

DbAliasLen
入力。データベース別名の長さを示す 2 バイトの無符号整数 (バイト単位) です。

DbNameLen
入力。データベース名の長さを示す 2 バイトの無符号整数 (バイト単位) です。

pSqlca
出力。 *sqlca* 構造を指すポインターです。 この構造に関する詳細は、520ページの『SQLCA』を参照してください。

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。この値は、認証が SQL_AUTHENTICATION_DCE または SQL_AUTHENTICATION_KERBEROS の場合に限って指定してください。DCE の場合、プリンシパルは、サーバーのキータブ・ファイルに保管されているものと同じ値にしてください。

Authentication
入力。データベース用に指定される認証タイプが入ります。認証はユーザーが本人かどうかを検査するプロセスです。データベース・オブジェクトへのアクセスはユーザーの認証によって決まります。有効な値 (sqlenv から) は以下のとおりです。

SQL_AUTHENTICATION_SERVER
認証が、宛先データベースを含むノードで行われるということを指定します。

SQL_AUTHENTICATION_CLIENT
認証が、アプリケーションを呼び出すノードで行われるということを指定します。

SQL_AUTHENTICATION_DCS
認証が、宛先データベースを含むノードで行われるということを指定します。ただし、DB2 コネクトの使用時と、認証を DRDA AS で行うように指定した場合を除きます。

SQL_AUTHENTICATION_DCE
認証が、DCE 機密保護サービスを使用するということを指定します。

SQL_AUTHENTICATION_KERBEROS
認証が、kerberos セキュリティー・メカニズムを使用するということを指定します。

SQL_AUTHENTICATION_NOT_SPECIFIED
認証は指定されません。

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

SQL_AUTHENTICATION_DCS_ENCRYPT
認証が、DB2 コネクトの使用時 (このとき認証は DRDA AS で行われる) を除いて、宛先データベースを含むノードで行われることを指定します。また、認証のパスワードは暗号化されます。

このパラメーターは SQL_AUTHENTICATION_NOT_SPECIFIED に設定することができます。ただし、DB2 バージョン 1 サーバーに存在するデータベースをカタログする場合は設定できません。

データベース・カタログに認証タイプを指定すると、接続中のパフォーマンスが向上するという結果になります。

認証タイプに関する詳細は、*管理の手引き* を参照してください。

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。ヌル・ストリングはコメントがないことを示します。コメント・ストリングの最大長は 30 文字です。

pPath 入力。 UNIX ベースのシステムの場合は、カタログするデータベースが存在するパス名を指定するストリングを指定します。最大長は 215 文字です。

OS/2 または Windows オペレーティング・システムの場合は、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。

ヌル・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* で指定された省略時データベースが指定されたものとみなされます。

pNodeName
入力。データベースが位置するノードの名前を含むストリングを指定します。ヌル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みなされます。

Type 入力。データベースが間接であるか、リモートであるか、それともDCE を通してカタログするかを指定する単一の文字。有効な値 (sqlenv で定義) は、以下のとおりです。

SQL_INDIRECT
データベースが今のインスタンスに存在するよう指定します。

SQL_REMOTE
データベースが別のインスタンスに存在するよう指定します。

SQL_DCE
データベースが DCE を通してカタログされるよう指定します。

pDbAlias
入力。データベースの別名を含むストリングを指定します。

pDbName
入力。データベース名を含むストリングを指定します。

REXX API 構文

```
CATALOG DATABASE dbname [AS alias] [ON path|AT NODE nodename]
[AUTHENTICATION authentication] [WITH "comment"]
```

REXX API パラメーター

dbname
カタログするデータベースの名前を指定します。

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みなされます。

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
認証が宛先データベースを含むノードで行われます。これが省略時の値です。

CLIENT
認証がアプリケーションを呼び出すノードで行われます。

DCS DB2 コネクトを使用してアクセスされたデータベースの認証がどのように行われるかを指定します。認証タイプが SERVER の場合を除き、タイプ SERVER の場合の動作と同じです。つまり、DB2 コネクトは認証をゲートウェイで強制的に行います。認証タイプが DCS の場合、認証はホストで行われるものとみなされます。

DCE SERVER PRINCIPAL *dce_principal_name*
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。この値は、ターゲット・サーバーのキータブ・ファイルにも記録されます。

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。

REXX API 構文

```
CATALOG GLOBAL DATABASE db_global_name AS alias
USING DIRECTORY {DCE} [WITH comment]
```

REXX API パラメーター

db_global_name
DCE 名前スペース内のデータベースを固有に識別する完全修飾名を示します。

alias データベースの代替名を指定します。

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。

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。

例

```
call SQLDBS 'CATALOG GLOBAL DATABASE /.../cell1/subsys/database/DB3
AS dbtest USING DIRECTORY DCE WITH "Sample Database"'
```

サンプル・プログラム

C	¥sqllib¥samples¥c¥dbcat.c
COBOL	¥sqllib¥samples¥cobol¥dbcat.cbl
REXX	¥sqllib¥samples¥rexx¥dbcat.cmd

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。

現行インスタンス (**DB2INSTANCE** 環境変数の値で定義された) で作成されたデータベースは、*間接* としてカタログされます。その他のインスタンスで作成されたデータベースは、 (物理的には同一のマシンに存在している場合であっても) *リモート* としてカタログされます。

システム・データベース・ディレクトリーが存在しない場合、CATALOG DATABASE は自動的にそれを作成します。システム・データベース・ディレクトリーは、使用中のデータベース・マネージャー・インスタンスを含むパス

に保管されます。システム・データベース・ディレクトリーは、データベースの外部で保守されます。ディレクトリーに含まれる項目は、次のとおりです。

- 別名
- 認証タイプ
- コメント
- データベース
- 項目タイプ
- ローカル・データベース・ディレクトリー (ローカル・データベースをカタログするとき)
- ノード名 (リモート・データベースをカタログするとき)
- リリース情報

タイプ・パラメーターを SQL_INDIRECT に設定してデータベースをカタログすると、入力された認証パラメーターの値は無視され、ディレクトリーの認証は SQL_AUTHENTICATION_NOT_SPECIFIED に設定されます。

システム・データベース・ディレクトリーの内容は、212ページの『sqledosd - データベース・ディレクトリー走査のオープン』、208ページの『sqledgne - データベース・ディレクトリーの次項目の入手』、および 206ページの『sqledcls - データベース・ディレクトリー走査のクローズ』を使用してリストします。

ディレクトリーのキャッシュが使用可能にされている場合 (331ページの『sqlfxsys - データベース・マネージャー構成の入手』の構成パラメーター *dir_cache* を参照)、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。

参照資料

206ページの『sqledcls - データベース・ディレクトリー走査のクローズ』

208ページの『sqledgne - データベース・ディレクトリーの次項目の入手』

212ページの『sqledosd - データベース・ディレクトリー走査のオープン』

301ページの『sqleuncd - データベースのアンカタログ』

sqlecran - ノードでのデータベースの作成

API を呼び出すノード上だけでデータベースを作成します。この API は汎用目的ではありません。たとえば、あるノードのデータベース区画が損傷して再作成が必要な場合に、 447ページの『sqlurestore - データベースの復元』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) 除去されたデータベース区画を再作成するために使用した場合、このノードのデータベースは復元保留中状態になります。データベース区画を再作成したら、ただちにデータベースをこのノードに復元する必要があります。

効力範囲

この API は、それが呼び出されたノードにのみ影響を与えます。

許可

以下のいずれかです。

- *sysadm*
- *sysctrl*

必須接続

インスタンス。データベースを別のノードで作成する場合、まずそのノードに接続することが必要になります。データベース接続は、この API によって処理中に一時的に確立されます。

API 組み込みファイル

sqlenv.h

C API 構文

```
/* File: sqlenv.h */
/* API: Create Database at Node */
/* ... */
SQL_API_RC SQL_API_FN
  sqlecran (
    char * pDbName,
    void * pReserved,
    struct sqlca * pSqlca);
/* ... */
```

汎用 API 構文

```
/* File: sqlenv.h */
/* API: Create Database at Node */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgcran (
    unsigned short reservedLen,
    unsigned short dbNameLen,
    struct sqlca * pSqlca,
    void * pReserved,
    char * pDbName);
/* ... */
```

API パラメーター

reservedLen
入力。 *pReserved* の長さのために予約されています。

dbNameLen
入力。データベース名の長さを示す 2 バイトの無符号整数 (バイト単位) です。

pSqlca
出力。 *sqlca* 構造を指すポインターです。 この構造に関する詳細は、520ページの『SQLCA』を参照してください。

pReserved
入力。ヌル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・ポインターです。将来の使用のために予約されています。

pDbName
入力。作成されるデータベースの名前を含むストリングを指定します。ヌルにはしないでください。

REXX API 構文

この API は、SQLDB2 インターフェースを使って、REXX から呼び出すことができます。 13ページの『API の説明の編成』、または *アプリケーション開発の手引き* を参照してください。構文については、*コマンド解説書* を参照してください。

使用上の注意

データベースが正常に作成されると、復元保留中の状態になります。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ノード上に復元する必要があります。

参照資料

186ページの『sqlecrea - データベースの作成』

216ページの『sqledpan - ノードでのデータベースの消去』

sqlecrea - データベースの作成

任意選択でユーザー定義の照合順序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、さらには回復ログを割り当てます。

効力範囲

複数ノード環境では、この API は `$HOME/sqllib/db2nodes.cfg` ファイルにリストされているすべてのノードに影響を与えます。

この API の呼び出し側のノードは、新規のデータベース用のカタログ・ノードになります。

許可

以下のいずれかです。

- *sysadm*
- *sysctrl*

必須接続

インスタンス。データベースを別の (リモート) ノードで作成すると、まずそのノードに接続することが必要になります。データベース接続は、この API によって処理中に一時的に確立されます。

API 組み込みファイル

sqlenv.h

C API 構文

```
/* File: sqlenv.h */
/* API: Create Database */
/* ... */
SQL_API_RC SQL_API_FN
  sqlecrea (
    char * pDbName,
    char * pLocalDbAlias,
    char * pPath,
    struct sqledbdesc * pDbDescriptor,
    struct sqledbcountryinfo * pCountryInfo,
    char Reserved2,
    void * pReserved1,
    struct sqlca * pSqlca);
/* ... */
```

汎用 API 構文

```
/* File: sqlenv.h */
/* API: Create Database */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgcrea (
    unsigned short PathLen,
    unsigned short LocalDbAliasLen,
    unsigned short DbNameLen,
    struct sqlca * pSqlca,
    void * pReserved1,
    unsigned short Reserved2,
    struct sqledbcountryinfo * pCountryInfo,
    struct sqledbdesc * pDbDescriptor,
    char * pPath,
    char * pLocalDbAlias,
    char * pDbName);
/* ... */
```

API パラメーター

PathLen

入力。パスの長さを示す 2 バイトの無符号整数 (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 バイトの無符号整数 (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 バイトの無符号整数 (バイト単位) です。

pSqlca
出力。 *sqlca* 構造を指すポインターです。 この構造に関する詳細は、520ページの『SQLCA』を参照してください。

pReserved1
入力。ヌル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・ポインターです。

Reserved2
入力。将来の使用のために予約されています。

pCountryInfo
入力。 *sqledbcountryinfo* 構造を指すポインターです。データベースのロケールおよびコード・セットが含まれます。この構造に関する詳細については、 556ページの『SQLEDBCOUNTRYINFO』を参照してください。有効なロケールおよびコード・セット値のリストについては、*概説およびインストール* の 1 つを参照してください。ヌルにすることもできます。

pDbDescriptor
入力。データベースの作成時に使用されるデータベース説明ブロックを指すポインターです。データベース説明ブロックは、照合順序のように、永久にデータベースの構成ファイルに保管される値を提供するために使用することができます。その構造は 557ページの『SQLEDBDESC』で説明されます。ヌルにすることもできます。

pPath 入力。 UNIX ベースのシステムの場合は、データベースを作成するパスを指定します。パスが指定されない場合、データベースは、データベース・マネージャー構成ファイルに指定されている省略時のデータベース・パス (*dftdbpath* パラメーター) に作成されます。 OS/2 または 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。ヌルにすることもできます。

注: MPP システムの場合は、データベースを NFS がマウントされたディレクトリーには作成しないでください。パスが指定されていな

い場合、 NFS がマウントされたパスに *dftdbpath* データベース・マネージャー構成パラメーターが指定されないようにしてください。 (たとえば、UNIX ベースのシステムの場合には、インスタンスの所有者の $HOME ディレクトリーを指定しないでください。) MPP システムでこの API に指定されたパスを相対パスにすることはできません。

pLocalDbAlias

入力。クライアントのシステム・データベース・ディレクトリーに置かれる別名を含むストリングを指定します。ヌル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、データベース名は省略時のものになります。

pDbName

入力。データベース名を含むストリングを指定します。これはシステム・データベース・ディレクトリーにカタログされるデータベース名です。データベースは、サーバーのシステム・データベース・ディレクトリーに正常に作成されると、データベース名と同じデータベース別名を持つシステム・データベース・ディレクトリーに自動的にカタログされます。ヌルにはしないでください。

REXX API 構文

```
CREATE DATABASE dbname [ON path] [ALIAS dbalias]
[USING CODESET codeset TERRITORY territory]
[COLLATE USING {SYSTEM | IDENTITY | USER :udcs}]
[NUMSEGS numsegs] [DFT_EXTENT_SZ dft_extentsize]
[CATALOG TABLESPACE <tablespace_definition>]
[USER TABLESPACE <tablespace_definition>]
[TEMPORARY TABLESPACE <tablespace_definition>]
[WITH comment]

<tablespace_definition> は以下の事柄を表します。
MANAGED BY {
SYSTEM USING :SMS_string |
DATABASE USING :DMS_string }
[ EXTENTSIZE number_of_pages ]
[ PREFETCHSIZE number_of_pages ]
[ OVERHEAD number_of_milliseconds ]
[ TRANSFERRATE number_of_milliseconds ]
```

REXX API パラメーター

dbname
データベースの名前。

dbalias
データベースの別名。

path データベースを作成するパス。

パスが指定されない場合、データベースは、データベース・マネージャー構成ファイルに指定されている省略時のデータベース・パス (*dftdbpath* 構成パラメーター) に作成されます。

注: MPP システムの場合は、データベースを NFS がマウントされたディレクトリーには作成しないでください。パスが指定されていない場合、 NFS がマウントされたパスに *dftdbpath* データベース・マネージャー構成パラメーターが指定されないようにしてください。 (たとえば、UNIX ベースのシステムの場合には、インスタンスの所有者の `$HOME` ディレクトリーを指定しないでください。) MPP システムでこの 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
現行の国別コードに基づくオペレーティング・システムの照合順序を使用します。

IDENTITY
照合順序は ID 順序で、ストリングはバイトごとに比較されます。左端のバイトから比較されていきます。

USER udcs
照合順序は、アプリケーションを呼び出すことにより、照合順序を定義する 256 バイトのストリングを含むホスト変数に指定されます。

numsegs
DAT、IDX、および LF ファイルを保管するために作成および使用されるセグメント・ディレクトリーの数を表します。

dft_extentsize
データベースの表スペースの、省略時 *extentsize* を指定します。

SMS_string
表スペースに属する予定の 1 つまたは複数のコンテナーを識別する複合 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。おのおののディレクトリー名は 254 バイトを超えることはできないことに注意してください。

XXX.0 指定されたディレクトリーの数。

XXX.1 SMS 表スペースの最初のディレクトリー名。

XXX.2 SMS 表スペースの 2 番目のディレクトリー名。

XXX.3 以降、3 番目、4 番目 ... と続きます。

DMS_string
表スペースに属する予定の 1 つまたは複数のコンテナーを識別する複合 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。おのおののコンテナー名は 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 REXX ホスト変数内のストリングの数 (最初のレベル要素の数)。

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
データの事前取り出しを実行する際に、表スペースから読み取られることになる 4KB ページの数。

OVERHEAD number_of_milliseconds
入出力制御装置のオーバーヘッド、ディスク・シーク、および待ち時間を指定する数 (ミリ秒単位)。

TRANSFERRATE number_of_milliseconds
1 つの 4KB ページをメモリーに読み取る時間を指定する数 (ミリ秒単位)。

comment
システム・ディレクトリー内のデータベースまたはデータベース項目のコメント。コメント中に復帰文字または改行文字を使用しないでください。コメント文は必ず二重引用符で囲んでください。コメントは最大 30 文字まで記述できます。

サンプル・プログラム

C	¥sqllib¥samples¥c¥dbconf.c
COBOL	¥sqllib¥samples¥cobol¥dbconf.cbl
REXX	¥sqllib¥samples¥rexx¥dbconf.cmd

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。 MPP システムでは、 `db2nodes.cfg` にリストされたすべてのノードを作成し、それぞれのノードにある指定されたサブディレクトリーの下に `$DB2INSTANCE/NODE`*xxxx* ディレクトリーを作成します。 *xxxx* はローカル・ノード番号です。 MPP 以外のシステムでは、指定されたサブディレクトリーの下に `$DB2INSTANCE/NODE0000` ディレクトリーを作成します。
- システム・カタログ表および回復ログを作成します。
- 以下のディレクトリーのデータベースをカタログします。

– *pPath* によって示されたパスにあるサーバーのローカル・データベース・ディレクトリー。または、パスが指定されていない場合は、データベース・マネージャーのシステム構成ファイルで定義された省略時データベース・パスが使用されます。ローカル・データベース・ディレクトリーは、データベースを含む各ファイル・システムに存在します。
– 付加されたインスタンス用のサーバーのシステム・データベース・ディレクトリー。作成されるディレクトリー項目にはデータベース名やデータベース別名が含まれます。

  API がリモート・クライアントから呼び出された場合は、クライアントのシステム・データベース・ディレクトリーもデータベース名やデータベース別名で更新されます。

システム・データベース・ディレクトリーとローカル・データベース・ディレクトリーがどちらも存在しない場合、そのどちらかを作成します。指定されていれば、コメントおよびコード設定値は両方のディレクトリーに入れられます。

- 指定されたコード設定、区域、および照合順序を保管します。照合順序が固有のウェイトから成っているか、または ID 順序である場合、フラグがデータベース構成ファイルで設定されます。
- SYSCAT、SYSFUN、SYSIBM、および SYSIBM を所有者とする SYSSTAT という各スキーマを作成します。この API が呼び出されるサーバー・ノードは、新規のデータベース用のカタログ・ノードになります。 2 つのノード・グループ (IBMDEFAULTGROUP と IBMCATGROUP) が自動的に作成されます。詳細については、*SQL 解説書* を参照してください。
- 事前に定義されたデータベース・マネージャー・バインド・ファイルのデータベースにバインドします (これらは `db2ubind.lst` にリストされています)。これらの 1 つまたは複数のファイルのバインドが正常に行われた場合、 **sqlecrea** は SQLCA に警告を戻します。また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、ユーザーは訂正の処置をとり、手動で失敗したファイルをバインドできます。データベースはどのような場合にでも作成されます。 PUBLIC を許可した CREATEIN 特権を伴った、バインドを行った際 NULLID と呼ばれているスキーマが、暗黙的に作成されます。
- SYSCATSPACE、TEMPSPACE1、および USERSPACE1 表スペースを作成します。 SYSCATSPACE 表スペースはカタログ・ノードにのみ作成されます。すべてのノードには同じ表スペース定義があります。
- 以下の権限を付与します。

- データベース作成者に対する DBADM 権限と、CONNECT、CREATETAB、BINDADD、CREATE_NOT_FENCED、IMPLICIT_SCHEMA、および LOAD の各種特権。
- PUBLIC に対する CONNECT、CREATETAB、BINDADD、IMPLICIT_SCHEMA の各種特権。
- PUBLIC に対する USERSPACE1 表スペース上の USE 特権。
- PUBLIC に対する各システム・カタログ上の SELECT 特権。
- 正常実行したバインド・ユーティリティーごとに、 PUBLIC に対する BIND および EXECUTE 特権。

dbadm 権限を使用すると、これらの権限を他のユーザーまたは PUBLIC に付与 (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。

MPP 環境では、データベース・マネージャーはすべてのノード上の指定されたパスまたは省略時パスの下に、サブディレクトリー、$DB2INSTANCE/NODE*xxxx* を作成します。 *xxxx* は db2nodes.cfg ファイルで定義されたノード番号です (つまり、ノード 0 は NODE0000 になります)。 SQL00001 から SQL*nnnnn* のサブディレクトリーはこのパスにあります。これにより、別々のノードと関連があるデータベース・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省略時パスの下にあるサブディレクトリー $DB2INSTANCE をすべてのノードが共有している場合でも、そのようになります)。

アプリケーションがすでにデータベースに接続されている場合、 CREATE DATABASE は失敗します。

データベース説明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 (557ページの『SQLEDBDESC』を参照してください)。

データベース説明ブロックの "目印" は、記号値 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順序の例は、ホスト言語組み込みファイルで使用できます。

sqle819a データベースのコード・ページが 819 (ISO Latin/1) である場合、この順序により、ホスト CCSID 500 (EBCDIC 国際標準) に従って分類が行われます。

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) に従って分類が行われます。

sqle850a データベースのコード・ページが 850 (ASCII Latin/1) である場合、この順序により、ホスト CCSID 500 (EBCDIC 国際標準) に従って分類が行われます。

sqle850b データベースのコード・ページが 850 (ASCII Latin/1) である場合、この順序により、ホスト CCSID 037 (EBCDIC 米国英語) に従って分類が行われます。

sqle932a データベースのコード・ページが 932 (ASCII 日本語) である場合、この順序により、ホスト CCSID 5035 (EBCDIC 日本語) に従って分類が行われます。

sqle932b データベースのコード・ページが 932 (ASCII 日本語) である場合、この順序により、ホスト CCSID 5026 (EBCDIC 日本語) に従って分類が行われます。

CREATE DATABASE 中に指定された照合順序は後で変更することはできません。データベースのすべての文字比較は指定された照合順序を使用します。これは索引の構造と照会の結果に影響します。

新規のデータベースに異なる別名を定義するには、 **sqlecadb** を使用してください。

参照資料

97ページの『sqlabndx - バインド』

174ページの『sqlecadb - データベースのカタログ』

183ページの『sqlecran - ノードでのデータベースの作成』

216ページの『sqledpan - ノードでのデータベースの消去』

222ページの『sqledrpd - データベースの消去』

sqlectnd - ノードのカタログ

DB2 サーバーのインスタンスへのアクセスに使用する通信プロトコルに基づいて、そのインスタンスの位置に関する情報をノード・ディレクトリーに保管します。この情報は、アプリケーションとサーバー・インスタンスの間でデータベース接続を確立するのに必要です。

許可

以下のいずれかです。

- *sysadm*
- *sysctrl*

必須接続

ありません。

API 組み込みファイル

sqlenv.h

C API 構文

```
/* File: sqlenv.h */
/* API: Catalog Node */
/* ... */
SQL_API_RC SQL_API_FN
  sqlectnd (
    struct sqle_node_struct * pNodeInfo,
    void * pProtocolInfo,
    struct sqlca * pSqlca);
/* ... */
```

汎用 API 構文

```
/* File: sqlenv.h */
/* API: Catalog Node */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgctnd (
    struct sqlca * pSqlca,
    struct sqle_node_struct * pNodeInfo,
    void * pProtocolInfo);
/* ... */
```

API パラメーター

pNodeInfo
入力。ノード・ディレクトリー構造を指すポインターです。この構造に関する詳細については、547ページの『SQLE-NODE-STRUCT』を参照してください。

pProtocolInfo
入力。プロトコル構造を指すポインターです。この構造に関する詳細については、以下の項目を参照してください。

- 542ページの『SQLE-NODE-CPIC』
- 543ページの『SQLE-NODE-IPXSPX』
- 544ページの『SQLE-NODE-LOCAL』
- 545ページの『SQLE-NODE-NETB』
- 546ページの『SQLE-NODE-NPIPE』
- 549ページの『SQLE-NODE-TCPIP』

pSqlca
出力。 *sqlca* 構造を指すポインターです。 この構造に関する詳細は、520ページの『SQLCA』を参照してください。

REXX API 構文

```
CATALOG APPC NODE nodename DESTINATION symbolic_destination_name
[SECURITY {NONE|SAME|PROGRAM}]
[WITH comment]
```

REXX API パラメーター

nodename
カタログするノードの別名。

symbolic_destination_name
リモート・パートナー・ノードの記号宛先名。

comment
そのノード・ディレクトリー項目と関連したオプション・コメント。コメントには CR/LF 文字を含めないようにしてください。最大長は 30 文字です。コメント・テキストは必ず二重引用符で囲んでください。

REXX API 構文

```
CATALOG IPXSPX NODE nodename REMOTE file_server SERVER objectname
[WITH comment]
```

REXX API パラメーター

nodename
カタログするノードの別名。

file_server
データベース・マネージャー・インスタンスのインターネットワーク・アドレスを登録する、 NetWare ファイル・サーバーの名前。インターネットワーク・アドレスは、 NetWare ファイル・サーバーのバインダリーに保管され、 *objectname* を使用してアクセスされます。

objectname
データベース・マネージャー・サーバーのインスタンスは、Netware ファイル・サーバーで、オブジェクトとして、*objectname* と表示されます。サーバーの IPX/SPX インターネットワーク・アドレスはこのオブジェクトに保管され、このオブジェクトから検索されます。

comment
そのノード・ディレクトリー項目と関連したオプション・コメント。コメントには CR/LF 文字を含めないようにしてください。最大長は 30 文字です。コメント・テキストは必ず二重引用符で囲んでください。

REXX API 構文

```
CATALOG LOCAL NODE nodename INSTANCE instance_name [WITH comment]
```

REXX API パラメーター

nodename
カタログするノードの別名。

instance_name
カタログするインスタンスの名前。

comment
そのノード・ディレクトリー項目と関連したオプション・コメント。コ

メントには CR/LF 文字を含めないようにしてください。最大長は 30 文字です。コメント・テキストは必ず二重引用符で囲んでください。

REXX API 構文

```
CATALOG NETBIOS NODE nodename REMOTE server_nname ADAPTER adapternum
[WITH comment]
```

REXX API パラメーター

nodename
カタログするノードの別名。

server_nname
リモート・ワークステーションの名前。サーバー・インスタンスのデータベース・マネージャー構成ファイルに検出されたワークステーション名 (*nname*) です。

adapternum
ローカル LAN アダプター番号。

comment
そのノード・ディレクトリー項目と関連したオプション・コメント。コメントには CR/LF 文字を含めないようにしてください。最大長は 30 文字です。コメント・テキストは必ず二重引用符で囲んでください。

REXX API 構文

```
CATALOG NPIPE NODE nodename REMOTE computer_name INSTANCE instance_name
```

REXX API パラメーター

nodename
カタログするノードの別名。

computer_name
宛先データベースが存在するノードのコンピューター名。

instance_name
カタログするインスタンスの名前。

REXX API 構文

```
CATALOG TCPIP NODE nodename REMOTE hostname SERVER servicename
[WITH comment]
```

REXX API パラメーター

nodename
カタログするノードの別名。

hostname
宛先データベースが存在するノードのホスト名。

servicename
リモート・ノード上にあるデータベース・マネージャー・インスタンスのサービス名か、このサービス名と関連があるポート番号のどちらかです。

comment
そのノード・ディレクトリー項目と関連したオプション・コメント。コメントには CR/LF 文字を含めないようにしてください。最大長は 30 文字です。コメント・テキストは必ず二重引用符で囲んでください。

サンプル・プログラム

C	¥sqllib¥samples¥c¥nodecat.c
COBOL	¥sqllib¥samples¥cobol¥nodecat.cbl
REXX	¥sqllib¥samples¥rexx¥nodecat.cmd

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。 OS/2 または Windows オペレーティング・システムの場合、そのノード・ディレクトリーは使用中のインスタンスのディレクトリーに格納されます。 UNIX ベースのシステムの場合、DB2 インストール・ディレクトリー (たとえば、 `sqllib`) に格納されます。

ディレクトリーのキャッシュが使用可能にされている場合 (331ページの『sqlfxsys - データベース・マネージャー構成の入手』の構成パラメーター *dir_cache* を参照)、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。

ノード・ディレクトリーの内容をリストするには、270ページの『sqlenops - ノード・ディレクトリー走査のオープン』、 267ページの『sqlengne - ノード・ディレクトリー次項目の入手』、および 265ページの『sqlencls - ノード・ディレクトリー走査のクローズ』を使用します。

参照資料

265ページの『sqlencls - ノード・ディレクトリー走査のクローズ』

267ページの『sqlengne - ノード・ディレクトリー次項目の入手』

270ページの『sqlenops - ノード・ディレクトリー走査のオープン』

304ページの『sqleuncn - ノードのアンカタログ』

sqledcgd - データベース・コメントの変更

システム・データベース・ディレクトリーまたはローカル・データベース・ディレクトリー内のデータベース・コメントを変更します。現行のコメント関連テキストは、新規のコメント・テキストと置き換えることができます。

効力範囲

この API は、それが発行されたノードにのみ影響を与えます。

許可

以下のいずれかです。

- *sysadm*
- *sysctrl*

必須接続

ありません。

API 組み込みファイル

sqlenv.h

C API 構文

```
/* File: sqlenv.h */
/* API: Change Database Comment */
/* ... */
SQL_API_RC SQL_API_FN
  sqledcgd (
    _SQLOLDCHAR * pDbAlias,
    _SQLOLDCHAR * pPath,
    _SQLOLDCHAR * pComment,
    struct sqlca * pSqlca);
/* ... */
```

汎用 API 構文

```
/* File: sqlenv.h */
/* API: Change Database Comment */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgdcgd (
    unsigned short CommentLen,
    unsigned short PathLen,
    unsigned short DbAliasLen,
    struct sqlca * pSqlca,
    _SQLOLDCHAR * pComment,
    _SQLOLDCHAR * pPath,
    _SQLOLDCHAR * pDbAlias);
/* ... */
```

API パラメーター

CommentLen
入力。コメントの長さを示す 2 バイトの無符号整数 (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 バイトの無符号整数 (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 バイトの無符号整数 (バイト単位) です。

pSqlca
出力。 *sqlca* 構造を指すポインターです。 この構造に関する詳細は、520ページの『SQLCA』を参照してください。

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。ヌル・ストリングはコメントがないことを示します。既存のデータベース・コメントに変更が加えられないことを示す場合もあります。

pPath 入力。ローカル・データベース・ディレクトリーが存在するパスを含むストリングを指定します。指定されたパスがヌル・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 が実行されるノードで、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・ディレクトリーの変更

だけが行われます。すべてのノードのデータベース・コメントを変更するには、ノードごとに API を実行します。

pDbAlias
入力。データベース別名を含むストリングを指定します。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名前です。パスが指定されていない場合には、ローカル・データベース・ディレクトリーにカタログされる名前です。

REXX API 構文

```
CHANGE DATABASE database_alias COMMENT [ON path] WITH comment
```

REXX API パラメーター

database_alias
コメントが変更されるデータベースの別名を指定します。

システム・データベース・ディレクトリーでコメントを変更するには、データベース別名を指定する必要があります。

データベースが存在するパスを (*path* パラメーターで) 指定した場合、データベースの名前 (別名ではなく) を入力してください。ローカル・データベース・ディレクトリーでコメントを変更するには、この方式を使用してください。

path データベースが存在するパス。

comment
システム・データベース・ディレクトリーまたはローカル・データベース・ディレクトリーの項目を記述します。カタログしたデータベースについての記述を補足する、あらゆるコメントを入力することができます。コメント・ストリングの最大長は 30 文字です。復帰文字や改行文字は許可されません。コメント・テキストは必ず二重引用符で囲んでください。

サンプル・プログラム

C	¥sqllib¥samples¥c¥dbcmt.c
COBOL	¥sqllib¥samples¥cobol¥dbcmt.cbl
REXX	¥sqllib¥samples¥rexx¥dbcmt.cmd

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。

既存のコメントを修正するには、以下の手順に従ってください。

1. 212ページの『sqledosd - データベース・ディレクトリー走査のオープン』を呼び出す。
2. 208ページの『sqledgne - データベース・ディレクトリーの次項目の入手』を呼び出して、元のコメントを検索する。
3. 検索したコメントを修正する。
4. 206ページの『sqledcls - データベース・ディレクトリー走査のクローズ』を呼び出す。
5. ″sqledcgd - データベース・コメントの変更″ を呼び出して、元のテキストを修正したテキストに置き換える。

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。

参照資料

186ページの『sqlecrea - データベースの作成』

174ページの『sqlecadb - データベースのカタログ』

sqledcls - データベース・ディレクトリー走査のクローズ

212ページの『sqledosd - データベース・ディレクトリー走査のオープン』によって割り振られたリソースを解放します。

許可

ありません。

必須接続

ありません。

API 組み込みファイル

sqlenv.h

C API 構文

```
/* File: sqlenv.h */
/* API: Close Database Directory Scan */
/* ... */
SQL_API_RC SQL_API_FN
  sqledcls (
    unsigned short Handle,
    struct sqlca * pSqlca);
/* ... */
```

汎用 API 構文

```
/* File: sqlenv.h */
/* API: Close Database Directory Scan */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgdcls (
    unsigned short Handle,
    struct sqlca * pSqlca);
/* ... */
```

API パラメーター

Handle

入力。関連する OPEN DATABASE DIRECTORY SCAN API から戻される識別子です。

pSqlca

出力。 *sqlca* 構造を指すポインターです。 この構造に関する詳細は、520ページの『SQLCA』を参照してください。

REXX API 構文

```
CLOSE DATABASE DIRECTORY scanid
```

REXX API パラメーター

scanid

OPEN DATABASE DIRECTORY SCAN API によって戻された *scanid* を含むホスト変数。

サンプル・プログラム

C	¥sqllib¥samples¥c¥dbcat.c
COBOL	¥sqllib¥samples¥cobol¥dbcat.cbl
REXX	¥sqllib¥samples¥rexx¥dbcat.cmd

参照資料

208ページの『sqledgne - データベース・ディレクトリーの次項目の入手』

212ページの『sqledosd - データベース・ディレクトリー走査のオープン』

sqledgne - データベース・ディレクトリーの次項目の入手

212ページの『sqledosd - データベース・ディレクトリー走査のオープン』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

ありません。

必須接続

ありません。

API 組み込みファイル

sqlenv.h

C API 構文

```
/* File: sqlenv.h */
/* API: Get Next Database Directory Entry */
/* ... */
SQL_API_RC SQL_API_FN
  sqledgne (
    unsigned short Handle,
    struct sqledinfo ** ppDbDirEntry,
    struct sqlca * pSqlca);
/* ... */
```

汎用 API 構文

```
/* File: sqlenv.h */
/* API: Get Next Database Directory Entry */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgdgne (
    unsigned short Handle,
    struct sqledinfo ** ppDbDirEntry,
    struct sqlca * pSqlca);
/* ... */
```

API パラメーター

Handle

入力。関連する OPEN DATABASE DIRECTORY SCAN API から戻される識別子です。

ppDbDirEntry

出力。呼び出し側が API を、 *sqledinfo* 構造を指すポインターのアドレスで提供します。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。 206ページの『sqledcls - データベース・ディレクトリー走査のクローズ』を呼び出せば、割り振られたスペースを解放できます。バッファーに戻される情報については、565ページの『SQLEDINFO』に説明されています。

pSqlca

出力。 *sqlca* 構造を指すポインターです。 この構造に関する詳細は、520ページの『SQLCA』を参照してください。

REXX API 構文

```
GET DATABASE DIRECTORY ENTRY :scanid [USING :value]
```

REXX API パラメーター

scanid

OPEN DATABASE DIRECTORY SCAN API によって戻された識別子を含む REXX ホスト変数。

value データベース項目情報が戻される複合 REXX ホスト変数。名前が指定されなかった場合、名 SQLDINFO 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 変数内の要素数 (通常 12)

XXX.1 ALIAS (データベースの別名)

XXX.2 DBNAME (データベースの名前)

XXX.3 DRIVE/PATH (ローカル・データベース・ディレクトリーのパス名)

XXX.3.1	NODE NUMBER (ローカル・データベース・ディレクトリーのみに有効)
XXX.4	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)
XXX.5	NODENAME (データベースが位置するノードの名前)
XXX.6	DBTYPE (製品名およびリリース番号)
XXX.7	COMMENT (データベースに関するコメント)
XXX.8	予約されています。
XXX.9	TYPE (項目タイプ)
XXX.10	AUTHENTICATION (認証タイプ)
XXX.10.1	DCE プリンシパル
XXX.11	GLBDBNAME (グローバル・データベース名)
XXX.12	CATALOG NODE NUMBER

サンプル・プログラム

C	¥sqllib¥samples¥c¥dbcat.c
COBOL	¥sqllib¥samples¥cobol¥dbcat.cbl
REXX	¥sqllib¥samples¥rexx¥dbcat.cmd

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。

以降の GET NEXT DATABASE DIRECTORY ENTRY は、現行の項目に続く項目を入手します。

GET NEXT DATABASE DIRECTORY ENTRY が呼び出されるとき、走査する項目がもはや存在していないならば、 *sqlca* の *sqlcode* 値は 1014 に設定されています。

GET NEXT DATABASE DIRECTORY ENTRY を呼び出すことにより、OPEN DATABASE DIRECTORY SCAN API によって戻されたカウント値を使用して、走査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に走査できます。

参照資料

206ページの『sqledcls - データベース・ディレクトリー走査のクローズ』

212ページの『sqledosd - データベース・ディレクトリー走査のオープン』

sqledosd - データベース・ディレクトリー走査のオープン

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

データベース・ディレクトリーの中で次々にデータベース項目に関する情報を調べていくには、 208ページの『sqledgne - データベース・ディレクトリーの次項目の入手』を使用してください。走査をクローズするには、206ページの『sqledcls - データベース・ディレクトリー走査のクローズ』を使用してください。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

ありません

必須接続

ありません

API 組み込みファイル

sqlenv.h

C API 構文

```
/* File: sqlenv.h */
/* API: Open Database Directory Scan */
/* ... */
SQL_API_RC SQL_API_FN
  sqledosd (
    _SQLOLDCHAR * pPath,
    unsigned short * pHandle,
    unsigned short * pNumEntries,
    struct sqlca * pSqlca);
/* ... */
```

汎用 API 構文

```
/* File: sqlenv.h */
/* API: Open Database Directory Scan */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgdosd (
    unsigned short PathLen,
    struct sqlca * pSqlca,
    unsigned short * pNumEntries,
    unsigned short * pHandle,
    _SQLOLDCHAR * pPath);
/* ... */
```

API パラメーター

PathLen
入力。パス・パラメーターの長さを示す 2 バイトの無符号整数 (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。

pSqlca
出力。 *sqlca* 構造を指すポインターです。 この構造に関する詳細は、520ページの『SQLCA』を参照してください。

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。

pHandle
出力。戻された識別子の 2 バイト域のアドレスを示します。データベース項目を走査するには、この識別子を 208ページの『sqledgne - データベース・ディレクトリーの次項目の入手』に渡す必要があります。リソースを解放するには、この識別子を 206ページの『sqledcls - データベース・ディレクトリー走査のクローズ』に渡す必要があります。

pPath 入力。ローカル・データベース・ディレクトリーが存在しているパスの名前を指定します。指定されたパスがヌル・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。

REXX API 構文

```
OPEN DATABASE DIRECTORY [ON path_name] USING :value
```

REXX API パラメーター

path_name
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。パスが指定されなかった場合、システム・データベース・ディレクトリーが使用されます。

value データベース・ディレクトリー情報が戻される複合 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 変数内の要素数 (常に 2)。

XXX.1 将来の走査アクセスに使用される識別子 (ハンドル)。

XXX.2 ディレクトリー内に含まれている項目の数。

サンプル・プログラム

C	¥sqllib¥samples¥c¥dbcat.c
COBOL	¥sqllib¥samples¥cobol¥dbcat.cbl
REXX	¥sqllib¥samples¥rexx¥dbcat.cmd

使用上の注意

この API が割り振った記憶域は、206ページの『sqledcls - データベース・ディレクトリー走査のクローズ』により解放されます。

同一のディレクトリーに対して、複数の OPEN DATABASE DIRECTORY SCAN API を発行できます。とはいえ、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・ディレクトリー走査をオープンすることができます。

参照資料

206ページの『sqledcls - データベース・ディレクトリー走査のクローズ』

208ページの『sqledgne - データベース・ディレクトリーの次項目の入手』

sqledpan - ノードでのデータベースの消去

指定されたノードでデータベースを消去します。 MPP サーバーでのみ実行可能です。

効力範囲

この API は、それが呼び出されたノードにのみ影響を与えます。

許可

以下のいずれかです。

- *sysadm*
- *sysctrl*

必須接続

ありません。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 組み込みファイル

sqlenv.h

C API 構文

```
/* File: sqlenv.h */
/* API: Drop Database at Node */
/* ... */
SQL_API_RC SQL_API_FN
  sqledpan (
    char * pDbAlias,
    void * pReserved,
    struct sqlca * pSqlca);
/* ... */
```

汎用 API 構文

```
/* File: sqlenv.h */
/* API: Drop Database at Node */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgdpan (
    unsigned short Reserved1,
    unsigned short DbAliasLen,
    struct sqlca * pSqlca,
    void * pReserved2,
    char * pDbAlias);
/* ... */
```

API パラメーター

Reserved1
将来の使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 バイトの無符号整数 (バイト単位) です。

pSqlca
出力。 *sqlca* 構造を指すポインターです。 この構造に関する詳細は、520ページの『SQLCA』を参照してください。

pReserved2
ヌル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・ポインターです。将来の使用のために予約されています。

pDbAlias
入力。消去されるデータベースの別名を含むストリングを指定します。これは、システム・データベース・ディレクトリーにある実際のデータベース名を参照するための名前です。

REXX API 構文

この API は、SQLDB2 インターフェースを使って、REXX から呼び出すことができます。 13ページの『API の説明の編成』、または *アプリケーション開発の手引き* を参照してください。構文については、*コマンド解説書* を参照してください。

使用上の注意

この API は DB2 ユニバーサル・データベース エンタープライズ拡張エディションに付属するユーティリティーによって使用されるもので、汎用目的では使用しません。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。

参照資料

183ページの『sqlecran - ノードでのデータベースの作成』

222ページの『sqledrpd - データベースの消去』

sqledreg - 登録解除

ネットワーク・ファイル・サーバーから DB2 サーバーの登録を解除します。DB2 サーバーのネットワーク・アドレスが、ファイル・サーバーの指定されたディレクトリーから除去されます。

許可

ありません

必須接続

ありません

API 組み込みファイル

sqlenv.h

C API 構文

```
/* File: sqlenv.h */
/* API: Deregister */
/* ... */
SQL_API_RC SQL_API_FN
  sqledreg (
    unsigned short Registry,
    void * pRegisterInfo,
    struct sqlca * pSqlca);
/* ... */
```

汎用 API 構文

```
/* File: sqlenv.h */
/* API: Deregister */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgdreg (
    unsigned short Registry,
    void * pRegisterInfo,
    struct sqlca * pSqlca);
/* ... */
```

API パラメーター

Registry

入力。ネットワーク・サーバー上で DB2 サーバーの登録を解除する位置を示します。このリリースでは、 SQL_NWBINDERY ディレクトリー (sqlenv で定義された NetWare ファイル・サーバー・バインダリー) だけがサポートされています。

pRegisterInfo

入力。 *sqle_reg_nwbindery* 構造を指すポインターです。この構造では、呼び出し側はネットワーク・ファイル・サーバーで有効なユーザー名とパスワードを指定します。この構造に関する詳細については、550 ページの『SQLE-REG-NWBINDERY』を参照してください。

pSqlca

出力。 *sqlca* 構造を指すポインターです。 この構造に関する詳細は、520ページの『SQLCA』を参照してください。

REXX API 構文

この API は、SQLDB2 インターフェースを使って、REXX から呼び出すことができます。 13ページの『API の説明の編成』、または *アプリケーション開発の手引き* を参照してください。構文の記述については、*コマンド解説書* を参照してください。

使用上の注意

Registry の値が SQL_NWBINDERY の場合、この API は、 *sqle_reg_nwbindery* 構造で指定された NetWare ユーザー名とパスワードを使用して、データベース・マネージャー構成ファイルで指定された NetWare ファイル・サーバー (FILESERVER) にログオンします。データベース・マネージャー構成ファイルで指定されたオブジェクト名 (OBJECTNAME) は、 NetWare ファイル・サーバー・バインダリーから削除されます。

指定した NetWare ユーザー名およびパスワードには、監視またはそれと同等の権限が必要です。

この API は、 DB2 サーバーからローカルに発行しなければなりません。リモートにはサポートされていません。

IPX/SPX フィールドを再構成する場合、または DB2 サーバーの IPX/SPX インターネットワーク・アドレスを変更する場合は、変更を行う前にまずネットワーク・ファイル・サーバーから DB2 サーバーの登録を解除し、次に変更を行って、最後に登録し直してください。

参照資料

286ページの『sqleregs - 登録』

sqledrpd - データベースの消去

データベースの内容とそのすべてのログ・ファイルを削除し、データベースをアンカタログし、さらにデータベースのサブディレクトリーを削除します。

効力範囲

省略時値では、この API は $HOME/sqllib/db2nodes.cfg にリストされているすべてのノードに影響を与えます。

許可

以下のいずれかです。

- *sysadm*
- *sysctrl*

必須接続

インスタンス。リモート・データベースを呼び出す場合、その前に ATTACH を呼び出す必要はありません。データベースがリモートとしてカタログされている場合、リモート・ノードへのインスタンス接続は呼び出しの期間中に確立されます。

API 組み込みファイル

sqlenv.h

C API 構文

```
/* File: sqlenv.h */
/* API: Drop Database */
/* ... */
SQL_API_RC SQL_API_FN
  sqledrpd (
    _SQLOLDCHAR * pDbAlias,
    struct sqlca * pSqlca);
/* ... */
```

汎用 API 構文

```
/* File: sqlenv.h */
/* API: Drop Database */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgdrpd (
    unsigned short Reserved1,
    unsigned short DbAliasLen,
    struct sqlca * pSqlca,
    _SQLOLDCHAR * pReserved2,
    _SQLOLDCHAR * pDbAlias);
/* ... */
```

API パラメーター

Reserved1
将来の使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 バイトの無符号整数 (バイト単位) です。

pSqlca
出力。 *sqlca* 構造を指すポインターです。 この構造に関する詳細は、520ページの『SQLCA』を参照してください。

pReserved2
ヌル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・ポインターです。将来の使用のために予約されています。

pDbAlias
入力。消去されるデータベースの別名を含むストリングを指定します。これは、システム・データベース・ディレクトリーにある実際のデータベース名を参照するための名前です。

REXX API 構文

```
DROP DATABASE dbalias
```

REXX API パラメーター

dbalias
消去するデータベースの別名。

サンプル・プログラム

C	¥sqllib¥samples¥c¥dbconf.sqc
COBOL	¥sqllib¥samples¥cobol¥dbconf.sqb
REXX	¥sqllib¥samples¥rexx¥dbconf.cmd

使用上の注意

sqledrpd はすべてのユーザー・データとログ・ファイルを削除します。復元操作の後でロールフォーワード回復用のログ・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。データベースを消去する前に、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。

消去するには、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。消去されるデータベースがローカル・データベース・ディレクトリーの最後の項目である場合、ローカル・データベース・ディレクトリーは自動的に削除されます。

この API がリモート・クライアント (または同一マシンの別のインスタンス) から呼び出される場合、指定された別名はクライアントのシステム・データベース・ディレクトリーから除去されます。それに対応するデータベース名は、サーバーのシステム・データベース・ディレクトリーから除去されます。

この API は DATALINK 列を介してリンクされているすべてのファイルをリンク解除します。リンク解除操作は DB2 データ・リンク・マネージャーで非同期に実行されるので、その効果が即座に DB2 データ・リンク・マネージャーで現れなかったり、リンク解除されたファイルが他の操作に即時に使用できるようにならなかったりする場合があります。 API が呼び出されると、そのデータベースへのすべての DB2 データ・リンク・マネージャー構成は使用可能になるはずです。そうでない場合、ドロップ・データベースが失敗します。

参照資料

174ページの『sqlecadb - データベースのカタログ』

186ページの『sqlecrea - データベースの作成』

183ページの『sqlecran - ノードでのデータベースの作成』

216ページの『sqledpan - ノードでのデータベースの消去』

301ページの『sqleuncd - データベースのアンカタログ』

sqledrpn - ノードのドロップの検査

ノード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。ノードを消去できるかどうかを示すメッセージが戻されます。

効力範囲

この API は、それが発行されたノードにのみ影響を与えます。

許可

以下のいずれかです。

- *sysadm*
- *sysctrl*

API 組み込みファイル

sqlenv.h

C API 構文

```
/* File: sqlenv.h */
/* API: Drop Node Verify */
/* ... */
SQL_API_RC SQL_API_FN
  sqledrpn (
    unsigned short Action,
    void * pReserved,
    struct sqlca * pSqlca);
/* ... */
```

汎用 API 構文

```
/* File: sqlenv.h */
/* API: Drop Node Verify */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgdrpn (
    unsigned short Reserved1,
    struct sqlca * pSqlca,
    void * pReserved2,
    unsigned short Action);
/* ... */
```

API パラメーター

Reserved1

pReserved2 の長さのために予約されています。

pSqlca

出力。 *sqlca* 構造を指すポインターです。 この構造に関する詳細は、520ページの『SQLCA』を参照してください。

pReserved2

ヌルに設定されたスペア・ポインター、または 0 を指すスペア・ポインター。将来の使用のために予約されています。

Action

要求されたアクション。有効値は以下のとおりです。

SQL_DROPNODE_VERIFY

REXX API 構文

この API は、SQLDB2 インターフェースを使って、REXX から呼び出すことができます。 13ページの『API の説明の編成』、または *アプリケーション開発の手引き* を参照してください。構文については、*コマンド解説書* を参照してください。

使用上の注意

ノード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` ファイルからそのノードの項目を除去します。そうすると、データベース・システムからノードが除去されます。

ノード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、以下の処置を実行してください。

1. ノードにデータが含まれている場合は、355ページの『sqludrdt - ノード・グループの再配分』を使用して、データをノードから除去するためにデータを再配分してください。 **sqludrdt** API のノードの消去オプション、または ALTER NODEGROUP ステートメントを使用して、データベースの任意のノード・グループからノードを除去します。この除去操作は、ノード・グループにノードが含まれているデータベースごとに行う必要があります。詳細については、*SQL 解説書* を参照してください。
2. ノードで定義されているイベント・モニターを除去します。
3. **sqledrpn** を再実行して、データベースが使用されなくなっていることを確かめます。

参照資料

161ページの『sqleaddn - ノードの追加』

276ページの『sqlepstp - データベース・マネージャーの停止』

sqledtin - 切断

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

ありません

必須接続

ありません。既存のインスタンス接続を除去します。

API 組み込みファイル

sqlenv.h

C API 構文

```
/* File: sqlenv.h */
/* API: Detach */
/* ... */
SQL_API_RC SQL_API_FN
  sqledtin (
    struct sqlca * pSqlca);
/* ... */
```

汎用 API 構文

```
/* File: sqlenv.h */
/* API: Detach */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgdtin (
    struct sqlca * pSqlca);
/* ... */
```

API パラメーター

pSqlca

出力。 *sqlca* 構造を指すポインターです。 この構造に関する詳細は、520ページの『SQLCA』を参照してください。

REXX API 構文

```
DETACH
```

サンプル・プログラム

C	¥sqllib¥samples¥c¥dbinst.c
COBOL	¥sqllib¥samples¥cobol¥dbinst.cbl
REXX	¥sqllib¥samples¥rexx¥dbinst.cmd

参照資料

169ページの『sqleatin - 接続』

sqlefmem - メモリーの解放

DB2 API によって割り振られたメモリーを、呼び出し側に代わって解放します。この API は、 149ページの『sqlbtcq - 表スペース・コンテナーの照会』および 132ページの『sqlbmtsq - 表スペースの照会』とともに使用するよう意図されています。

許可

ありません

必須接続

ありません

API 組み込みファイル

sqlenv.h

C API 構文

```
/* File: sqlenv.h */
/* API: Free Memory */
/* ... */
SQL_API_RC SQL_API_FN
  sqlefmem (
    struct sqlca * pSqlca,
    void * pBuffer);
/* ... */
```

汎用 API 構文

```
/* File: sqlenv.h */
/* API: Free Memory */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgfmem (
    struct sqlca * pSqlca,
    void * pBuffer);
/* ... */
```

API パラメーター

pSqlca

出力。 *sqlca* 構造を指すポインターです。 この構造に関する詳細は、520ページの『SQLCA』を参照してください。

pBuffer

入力。解放されるメモリーを指すポインターです。

サンプル・プログラム

C	¥sqllib¥samples¥c¥tspace.sqc
COBOL	¥sqllib¥samples¥cobol¥tspace.sqb

sqlefrce - アプリケーションの強制終了

システムからローカルまたはリモートのユーザーやアプリケーションを強制終了し、サーバー上での保守を可能にします。

考慮事項: 割り込みできない操作 (たとえば、RESTORE DATABASE) を強制終了する場合、データベースが利用可能になるには、その操作の再実行が正常終了しなければなりません。

効力範囲

この API は、 `$HOME/sqllib/db2nodes.cfg` ファイルにリストされているすべてのノードに影響を与えます。

区分データベース環境では、この API を強制されているアプリケーションの調整プログラム・ノードから発行する必要はありません。この API は、区分データベース環境ではどのノード (データベース区画サーバー) からでも発行できます。

許可

以下のいずれかです。

- *sysadm*
- *sysctrl*

必須接続

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接続する必要があります。接続がない場合、この API はローカルに実行されます。

API 組み込みファイル

sqlenv.h

C API 構文

```
/* File: sqlenv.h */
/* API: Force Application */
/* ... */
SQL_API_RC SQL_API_FN
  sqlefrce (
    long NumAgentIds,
    sqluint32 * pAgentIds,
    unsigned short ForceMode,
    struct sqlca * pSqlca);
/* ... */
```

汎用 API 構文

```
/* File: sqlenv.h */
/* API: Force Application */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgfrce (
    struct sqlca * pSqlca,
    unsigned short ForceMode,
    sqluint32 * pAgentIds,
    long NumAgentIds);
/* ... */
```

API パラメーター

pSqlca

出力。 *sqlca* 構造を指すポインターです。 この構造に関する詳細は、520ページの『SQLCA』を参照してください。

ForceMode

入力。 FORCE APPLICATION 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、 FORCE APPLICATION 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、FORCE APPLICATION 呼び出しが完了してから、指定されたユーザーが終了するまでに若干時間がかかることがあります。

このパラメーターは、 SQL_ASYNCH (sqlenv で定義) に設定しなければなりません。

pAgentIds

入力。無符号の長整数の配列を指すポインターです。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。活動アプリケーションのエージェント ID をリストするには、 31ページの『db2GetSnapshot - スナップショットの入手』を使用してください。

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 の配列の要素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) に設定された場合、すべてのユーザーが強制終了されます。このパラメーターがゼロに設定された場合、エラーが戻されます。

REXX API 構文

```
FORCE APPLICATION {ALL | :agentidarray} [MODE ASYNC]
```

REXX API パラメーター

ALL すべてのアプリケーションがデータベースへの接続から切断されます。

agentidarray

終了されるエージェント ID のリストを含む複合 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。 FORCE APPLICATION 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。

サンプル・プログラム

C	¥sqllib¥samples¥c¥dbstop.sqc
COBOL	¥sqllib¥samples¥cobol¥dbstop.sqb
REXX	¥sqllib¥samples¥rexx¥dbstop.cmd

使用上の注意

db2stop は強制終了の間は実行できません。データベース・マネージャーは活動状態のままなので、その後のデータベース・マネージャー操作は **db2start** を呼び出さなくても処理できます。

データベースの保全性を確保するため、終了できるのは、アイドル中のユーザー、または割り込み可能なデータベース操作を実行中のユーザーだけです。

FORCE を出した後でも、データベースはまだ接続要求を受諾します。すべてのユーザーを完全に強制終了するために、追加の FORCE が必要になる場合があります。

強制終了されるユーザーのエージェント ID を収集するには、データベース・システム・モニター機能を使用します。詳細については、システム・モニター *手引きおよび解説書* を参照してください。

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、指定した要素の合計数を含む配列をポインターが指していることを確認する必要があります。 *NumAgentIds* が SQL_ALL_USERS に設定されている場合、その配列は無視されます。

ユーザーが接続終了したとき、データベースの一貫性を確認するために ROLLBACK が実行されます。

強制切断できるすべてのユーザーを強制切断します。指定されたエージェント ID が 1 つ以上見つからない場合、 *sqlca* 構造の *sqlcode* が 1230 に設定されます。たとえば、エージェント ID の収集と **sqlefrce** の呼び出しの間にユーザーがサインオフすると、エージェント ID が見つからない場合があります。API を呼び出すユーザーは決して強制終了されません。

エージェント ID は繰り返し使用することができます。そして、そのエージェント ID が、データベース・システム・モニターによる収集の後に、アプリケーションを強制終了するために使用されることもあります。したがって、あるユーザーがサインオフした場合、別のユーザーがサインオンして、そのサインオフしたユーザーと同じエージェント ID を、この再利用プロセスによって獲得することができます。しかし、このときに、間違ったユーザーを強制終了してしまうおそれもあります。

参照資料

169ページの『sqleatin - 接続』

229ページの『sqledtin - 切断』

276ページの『sqlepstp - データベース・マネージャーの停止』

31ページの『db2GetSnapshot - スナップショットの入手』

sqlegdad - DCS データベースのカタログ

リモート・データベースに関する情報を、データベース接続サービス (DCS) ディレクトリーに保管します。このようなデータベースには、 DB2 コネクトなどのアプリケーション・リクエスター (AR) を介してアクセスします。システム・データベース・ディレクトリー内のデータベース名と一致する名前が DCS ディレクトリー項目にある場合、指定した AR を呼び出して、データベースが存在するリモート・サーバーに SQL 要求を転送します。 DB2 コネクトおよび DCS ディレクトリーの項目に関する詳細については、 *DB2 コネクト 使用者の手引き* を参照してください。

許可

以下のいずれかです。

- *sysadm*
- *sysctrl*

必須接続

ありません

API 組み込みファイル

sqlenv.h

C API 構文

```
/* File: sqlenv.h */
/* API: Catalog DCS Database */
/* ... */
SQL_API_RC SQL_API_FN
  sqlegdad (
    struct sql_dir_entry * pDCSDirEntry,
    struct sqlca * pSqlca);
/* ... */
```

汎用 API 構文

```
/* File: sqlenv.h */
/* API: Catalog DCS Database */
/* ... */
SQL_API_RC SQL_API_FN
  sqlggdad (
    struct sqlca * pSqlca,
    struct sql_dir_entry * pDCSDirEntry);
/* ... */
```

API パラメーター

pSqlca
出力。 *sqlca* 構造を指すポインターです。 この構造に関する詳細は、520ページの『SQLCA』を参照してください。

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です。この構造に関する詳細については、507ページの『SQL-DIR-ENTRY』を参照してください。

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
宛先データベース名。

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。

サンプル・プログラム

C	¥sqllib¥samples¥c¥dcscat.c
COBOL	¥sqllib¥samples¥cobol¥dcscat.cbl
REXX	¥sqllib¥samples¥rexx¥dcscat.cmd

使用上の注意

DB2 コネクトプログラムは、以下のような DRDA アプリケーション・サーバーへの接続を提供します。

- System/370 および System/390 アーキテクチャーのホスト・コンピューター上の DB2 (OS/390 版) 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 (VM および VSE 版) データベース。
- Application System/400 (AS/400) ホスト・コンピューター上の OS/400 データベース。

データベース・マネージャーは、データベース接続サービス・ディレクトリーがなければ自分で作成します。このディレクトリーは、使用しているデータベース・マネージャー・インスタンスを含むパスに保管されます。また、データベースの外側で保持されます。

データベースは、システム・データベース・ディレクトリーにリモート・データベースとしてもカタログしなければなりません。

253ページの『sqlegdsc - DCS ディレクトリー走査のオープン』、247ページの『sqlegdge - データベース用 DCS ディレクトリー項目の入手』、250ページの『sqlegdgt - DCS ディレクトリー項目の入手』、および 242ページの『sqlegdcl - DCS ディレクトリー走査のクローズ』を使用して、DCS ディレクトリーの内容をリストします。

注: ディレクトリーのキャッシュが使用可能にされている場合 (331ページの『sqlfxsys - データベース・マネージャー構成の入手』の構成パラメーター *dir_cache* を参照)、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。

参照資料

244ページの『sqlegdel - DCS データベースのアンカタログ』

sqlegdcl - DCS ディレクトリー走査のクローズ

253ページの『sqlegdsc - DCS ディレクトリー走査のオープン』によって割り振られたリソースを解放します。

許可

ありません

必須接続

ありません

API 組み込みファイル

sqlenv.h

C API 構文

```
/* File: sqlenv.h */
/* API: Close DCS Directory Scan */
/* ... */
SQL_API_RC SQL_API_FN
  sqlegdcl (
    struct sqlca * pSqlca);
/* ... */
```

汎用 API 構文

```
/* File: sqlenv.h */
/* API: Close DCS Directory Scan */
/* ... */
SQL_API_RC SQL_API_FN
  sqlggdcl (
    struct sqlca * pSqlca);
/* ... */
```

API パラメーター

pSqlca

出力。 *sqlca* 構造を指すポインターです。 この構造に関する詳細は、520ページの『SQLCA』を参照してください。

REXX API 構文

```
CLOSE DCS DIRECTORY
```

サンプル・プログラム

C	¥sqllib¥samples¥c¥dcscat.c
COBOL	¥sqllib¥samples¥cobol¥dcscat.cbl
REXX	¥sqllib¥samples¥rexx¥dcscat.cmd

参照資料

250ページの『sqlegdgt - DCS ディレクトリー項目の入手』

253ページの『sqlegdsc - DCS ディレクトリー走査のオープン』

sqlegdel - DCS データベースのアンカタログ

データベース接続サービス (DCS) ディレクトリーから項目を削除します。

許可

以下のいずれかです。

- *sysadm*
- *sysctrl*

必須接続

ありません

API 組み込みファイル

sqlenv.h

C API 構文

```
/* File: sqlenv.h */
/* API: Uncatalog DCS Database */
/* ... */
SQL_API_RC SQL_API_FN
  sqlegdel (
    struct sql_dir_entry * pDCSDirEntry,
    struct sqlca * pSqlca);
/* ... */
```

汎用 API 構文

```
/* File: sqlenv.h */
/* API: Uncatalog DCS Database */
/* ... */
SQL_API_RC SQL_API_FN
  sqlggdel (
    struct sqlca * pSqlca,
    struct sql_dir_entry * pDCSDirEntry);
/* ... */
```

API パラメーター

pSqlca
出力。 *sqlca* 構造を指すポインターです。 この構造に関する詳細は、520ページの『SQLCA』を参照してください。

pDCSDirEntry
入力 / 出力。データベース接続サービス・ディレクトリー構造を指すポインターです。この構造に関する詳細については、507ページの『SQL-DIR-ENTRY』を参照してください。この構造の *ldb* フィールドには、削除するデータベースのローカル名を入れてください。一致するローカル・データベース名がある DCS ディレクトリー項目は、削除前にこの構造にコピーされます。

REXX API 構文

```
UNCATALOG DCS DATABASE dbname [USING :value]
```

REXX API パラメーター

dbname
削除されるディレクトリー項目のローカル・データベース名。

value ディレクトリー項目情報が戻される複合 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 変数内の要素数 (常に 7)

XXX.1 RELEASE

XXX.2 LDB

XXX.3 TDB

XXX.4 AR

XXX.5 PARMS

XXX.6 COMMENT

XXX.7 RESERVED

サンプル・プログラム

C ¥sqllib¥samples¥c¥dcscat.c

COBOL ¥sqllib¥samples¥cobol¥dcscat.cbl

REXX ¥sqllib¥samples¥rexx¥dcscat.cmd

使用上の注意

DCS データベースは、 301ページの『sqleuncd - データベースのアンカタログ』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、 238ページの『sqlegdad - DCS データベースのカタログ』を使用してください。

ノードにカタログされている DCS データベースをリストするには、 253ページの『sqlegdsc - DCS ディレクトリー走査のオープン』、250ページの『sqlegdgt - DCS ディレクトリー項目の入手』、および 242ページの『sqlegdcl - DCS ディレクトリー走査のクローズ』を使用してください。

ディレクトリーのキャッシュが使用可能にされている場合 (331ページの『sqlfxsys - データベース・マネージャー構成の入手』の構成パラメーター *dir_cache* を参照)、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。

参照資料

238ページの『sqlegdad - DCS データベースのカタログ』

242ページの『sqlegdcl - DCS ディレクトリー走査のクローズ』

247ページの『sqlegdge - データベース用 DCS ディレクトリー項目の入手』

250ページの『sqlegdgt - DCS ディレクトリー項目の入手』

253ページの『sqlegdsc - DCS ディレクトリー走査のオープン』

301ページの『sqleuncd - データベースのアンカタログ』

sqlegdge - データベース用 DCS ディレクトリー項目の入手

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

ありません

必須接続

ありません

API 組み込みファイル

sqlenv.h

C API 構文

```
/* File: sqlenv.h */
/* API: Get DCS Directory Entry for Database */
/* ... */
SQL_API_RC SQL_API_FN
  sqlegdge (
    struct sql_dir_entry * pDCSDirEntry,
    struct sqlca * pSqlca);
/* ... */
```

汎用 API 構文

```
/* File: sqlenv.h */
/* API: Get DCS Directory Entry for Database */
/* ... */
SQL_API_RC SQL_API_FN
  sqlggdge (
    struct sqlca * pSqlca,
    struct sql_dir_entry * pDCSDirEntry);
/* ... */
```

API パラメーター

pSqlca

出力。 *sqlca* 構造を指すポインターです。 この構造に関する詳細は、520ページの『SQLCA』を参照してください。

pDCSDirEntry
入力 / 出力。データベース接続サービス・ディレクトリー構造へのポインターです。この構造に関する詳細については、507ページの『SQL-DIR-ENTRY』を参照してください。この構造の *ldb* フィールドは、検索する DCS ディレクトリー項目のあるデータベースのローカル名で充てんしてください。構造内の残りのフィールドは、この API の戻りに埋め込まれます。

REXX API 構文

```
GET DCS DIRECTORY ENTRY FOR DATABASE dbname [USING :value]
```

REXX API パラメーター

dbname
取得するディレクトリー項目のローカル・データベース名を指定します。

value ディレクトリー項目情報が戻される複合 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	変数内の要素数 (常に 7)
XXX.1	RELEASE
XXX.2	LDB
XXX.3	TDB
XXX.4	AR
XXX.5	PARMS
XXX.6	COMMENT
XXX.7	RESERVED

サンプル・プログラム

C	¥sqllib¥samples¥c¥dcscat.c
COBOL	¥sqllib¥samples¥cobol¥dcscat.cbl
REXX	¥sqllib¥samples¥rexx¥dcscat.cmd

参照資料

238ページの『sqlegdad - DCS データベースのカタログ』

242ページの『sqlegdcl - DCS ディレクトリー走査のクローズ』

244ページの『sqlegdel - DCS データベースのアンカタログ』

250ページの『sqlegdgt - DCS ディレクトリー項目の入手』

253ページの『sqlegdsc - DCS ディレクトリー走査のオープン』

sqlegdgt - DCS ディレクトリー項目の入手

データベース接続サービス (DCS) ディレクトリー項目のコピーを、アプリケーションが提供したバッファーへ転送します。

許可

ありません

必須接続

ありません

API 組み込みファイル

sqlenv.h

C API 構文

```
/* File: sqlenv.h */
/* API: Get DCS Directory Entries */
/* ... */
SQL_API_RC SQL_API_FN
  sqlegdgt (
    short * pNumEntries,
    struct sql_dir_entry * pDCSDirEntries,
    struct sqlca * pSqlca);
/* ... */
```

汎用 API 構文

```
/* File: sqlenv.h */
/* API: Get DCS Directory Entries */
/* ... */
SQL_API_RC SQL_API_FN
  sqlggdgt (
    struct sqlca * pSqlca,
    short * pNumEntries,
    struct sql_dir_entry * pDCSDirEntries);
/* ... */
```

API パラメーター

pSqlca

出力。 *sqlca* 構造を指すポインターです。 この構造に関する詳細は、520ページの『SQLCA』を参照してください。

pNumEntries

入力 / 出力。呼び出し側のバッファーにコピーされる項目数を示す短整数を指すポインターです。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です。この構造に関する詳細については、507ページの『SQL-DIR-ENTRY』を参照してください。バッファーには、 *pNumEntries* パラメーターで指定された数の項目を保留するだけの十分な大きさが必要です。

REXX API 構文

```
GET DCS DIRECTORY ENTRY [USING :value]
```

REXX API パラメーター

value ディレクトリー項目情報が戻される複合 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	変数内の要素数 (常に 7)
XXX.1	RELEASE
XXX.2	LDB
XXX.3	TDB
XXX.4	AR
XXX.5	PARMS
XXX.6	COMMENT
XXX.7	RESERVED

サンプル・プログラム

C	¥sqllib¥samples¥c¥dcscat.c
COBOL	¥sqllib¥samples¥cobol¥dcscat.cbl
REXX	¥sqllib¥samples¥rexx¥dcscat.cmd

使用上の注意

GET DCS DIRECTORY ENTRIES を出す前に、 253ページの『sqlegdsc - DCS ディレクトリー走査のオープン』(項目のカウントを戻す 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・ディレクトリー走査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。

参照資料

242ページの『sqlegdcl - DCS ディレクトリー走査のクローズ』

247ページの『sqlegdge - データベース用 DCS ディレクトリー項目の入手』

253ページの『sqlegdsc - DCS ディレクトリー走査のオープン』

sqlegdsc - DCS ディレクトリー走査のオープン

データベース接続サービス・ディレクトリー項目のコピーをメモリーに保管するとともに、項目の数を戻します。このコピーは、ディレクトリーがオープンする時点のディレクトリーのスナップショットです。

この API への呼び出しの後にディレクトリー自体に変更が加えられることがあっても、このコピーが更新されることはありません。項目を検索するには、250ページの『sqlegdgt - DCS ディレクトリー項目の入手』を使用してください。この API の呼び出しに関連したリソースを解放するには、242ページの『sqlegdcl - DCS ディレクトリー走査のクローズ』を使用してください。

許可

ありません

必須接続

ありません

API 組み込みファイル

sqlenv.h

C API 構文

```
/* File: sqlenv.h */
/* API: Open DCS Directory Scan */
/* ... */
SQL_API_RC SQL_API_FN
  sqlegdsc (
    short * pNumEntries,
    struct sqlca * pSqlca);
/* ... */
```

汎用 API 構文

```
/* File: sqlenv.h */
/* API: Open DCS Directory Scan */
/* ... */
SQL_API_RC SQL_API_FN
  sqlggdsc (
    struct sqlca * pSqlca,
    short * pNumEntries);
/* ... */
```

API パラメーター

pSqlca
出力。 *sqlca* 構造を指すポインターです。 この構造に関する詳細は、520ページの『SQLCA』を参照してください。

pNumEntries
出力。ディレクトリー項目の数が戻される 2 バイト域のアドレスを示します。

REXX API 構文

```
OPEN DCS DIRECTORY
```

サンプル・プログラム

C	¥sqllib¥samples¥c¥dcscat.c
COBOL	¥sqllib¥samples¥cobol¥dcscat.cbl
REXX	¥sqllib¥samples¥rexx¥dcscat.cmd

使用上の注意

走査の呼び出し側では、戻された値 *pNumEntries* を使用して、項目を受け取るのに十分なメモリーを割り振ります。コピーがすでに保留されているのに走査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。

参照資料

242ページの『sqlegdcl - DCS ディレクトリー走査のクローズ』

247ページの『sqlegdge - データベース用 DCS ディレクトリー項目の入手』

250ページの『sqlegdgt - DCS ディレクトリー項目の入手』

sqlegins - インスタンスの入手

DB2INSTANCE 環境変数の値を戻します。

許可

ありません

必須接続

ありません

API 組み込みファイル

sqlenv.h

C API 構文

```
/* File: sqlenv.h */
/* API: Get Instance */
/* ... */
SQL_API_RC SQL_API_FN
  sqlegins (
    _SQLOLDCHAR * pInstance,
    struct sqlca * pSqlca);
/* ... */
```

汎用 API 構文

```
/* File: sqlenv.h */
/* API: Get Instance */
/* ... */
SQL_API_RC SQL_API_FN
  sqlggins (
    struct sqlca * pSqlca,
    _SQLOLDCHAR * pInstance);
/* ... */
```

API パラメーター

pSqlca

出力。 *sqlca* 構造を指すポインターです。 この構造に関する詳細は、520ページの『SQLCA』を参照してください。

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です。このバッファーには、最低 8 バイトの長さが必要です。

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。

サンプル・プログラム

C	¥sqllib¥samples¥c¥dbinst.c
COBOL	¥sqllib¥samples¥cobol¥dbinst.cbl
REXX	¥sqllib¥samples¥rexx¥dbinst.cmd

使用上の注意

DB2INSTANCE 環境変数内の値が、ユーザーの接続するインスタンスである必要はありません。

ユーザーが現在接続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、ヌルの引き数を指定して 169ページの『sqleatin - 接続』を呼び出してください。

sqleintr - 割り込み

要求を停止させます。この API は、アプリケーションの制御の切れ目信号ハンドラーから呼び出されます。この制御の切れ目信号ハンドラーは、省略時値に設定することができます。この制御の切れ目信号ハンドラーは、 260ページの『sqleisig - 信号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

ありません

必須接続

ありません

API 組み込みファイル

sqlenv.h

C API 構文

```
/* File: sqlenv.h */
/* API: Interrupt */
/* ... */
SQL_API_RC SQL_API_FN
  sqleintr (
    void);
/* ... */
```

汎用 API 構文

```
/* File: sqlenv.h */
/* API: Interrupt */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgintr (
    void);
/* ... */
```

API パラメーター

ありません

REXX API 構文

```
INTERRUPT
```

例

```
call SQLDBS 'INTERRUPT'
```

使用上の注意

割り込みハンドラーからは、 **sqleintr** 以外のデータベース・マネージャー API も呼び出さないようにしてください。しかし、システムがそのことを行わずに済むよう保護することはありません。

コミットまたはロールバックの状態にあるデータベース・トランザクションはすべて、割り込みを行うことができません。

割り込まれたデータベース・マネージャー要求は、割り込まれたことを示すコードを戻します。

以下の表は、割り込み操作が他の API で実行されるときのアクションを示しています。

表 7. 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。媒体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	ありません。 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	ありません。これらの 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	ありません。 FORCE APPLICATION は完了します。

表 7. INTERRUPT アクション (続き)

データベース活動	アクション
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	ユーティリティーが取り消されます。表は、実行前の状態のままです。
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・レベルの復元には不適切です。
ROLLBACK	ありません。 ROLLBACK は完了します。
ディレクトリー・サービス	ディレクトリーは、一貫性のある状態を保ちます。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 ステートメントに先行する既存の状態に設定します。
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 ステートメントに先行する既存の状態に設定します。

参照資料

260ページの『sqleisig - 信号ハンドラーのインストール』

sqleisig - 信号ハンドラーのインストール

省略時割り込み (通常、 Ctrl+C または Ctrl+BREAK あるいはその両方) 信号ハンドラーをインストールします。この省略時のハンドラーが割り込み信号を検出すると、信号がリセットされ、 257ページの『sqleintr - 割り込み』が呼び出されます。

許可

ありません

必須接続

ありません

API 組み込みファイル

sqlenv.h

C API 構文

```
/* File: sqlenv.h */
/* API: Install Signal Handler */
/* ... */
SQL_API_RC SQL_API_FN
  sqleisig (
    struct sqlca * pSqlca);
/* ... */
```

汎用 API 構文

```
/* File: sqlenv.h */
/* API: Install Signal Handler */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgisig (
    struct sqlca * pSqlca);
/* ... */
```

API パラメーター

pSqlca

出力。 *sqlca* 構造を指すポインターです。 この構造に関する詳細は、520ページの『SQLCA』を参照してください。

REXX API 構文

```
INSTALL SIGNAL HANDLER
```

サンプル・プログラム

C	¥sqllib¥samples¥c¥dbcmt.c
COBOL	¥sqllib¥samples¥cobol¥ish.cbl
REXX	¥sqllib¥samples¥rexx¥dbcmt.cmd

使用上の注意

アプリケーションが信号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、単純な信号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。

割り込み信号ハンドラーを正しく機能させるために、この API を呼び出してください。

アプリケーションがより精巧な割り込み処理スキーマを必要とする場合、 257ページの『sqleintr - 割り込み』も呼び出せる、信号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信号関数を使用してください。カスタマイズされた信号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、 257ページの『sqleintr - 割り込み』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信号ハンドラーが正しく機能するようにしてください。

参照資料

257ページの『sqleintr - 割り込み』

sqlemgdb - データベースの移行

以前のバージョン (バージョン 2.x 以降) の DB2 データベースを現行の形式に変換します。

許可

sysadm

必須接続

この API によってデータベース接続が確立されます。

API 組み込みファイル

sqlenv.h

C API 構文

```
/* File: sqlenv.h */
/* API: Migrate Database */
/* ... */
SQL_API_RC SQL_API_FN
  sqlemgdb (
    _SQLOLDCHAR * pDbAlias,
    _SQLOLDCHAR * pUserName,
    _SQLOLDCHAR * pPassword,
    struct sqlca * pSqlca);
/* ... */
```

汎用 API 構文

```
/* File: sqlenv.h */
/* API: Migrate Database */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgmgdb (
    unsigned short PasswordLen,
    unsigned short UserNameLen,
    unsigned short DbAliasLen,
    struct sqlca * pSqlca,
    _SQLOLDCHAR * pPassword,
    _SQLOLDCHAR * pUserName,
    _SQLOLDCHAR * pDbAlias);
/* ... */
```

API パラメーター

PasswordLen
入力。パスワードの長さを示す 2 バイトの無符号整数 (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 バイトの無符号整数 (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 バイトの無符号整数 (バイト単位) です。

pSqlca
出力。 *sqlca* 構造を指すポインターです。 この構造に関する詳細は、520ページの『SQLCA』を参照してください。

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を指定します。ヌルにすることもできます。

pUserName
入力。アプリケーションのユーザー名を含むストリングを指定します。ヌルにすることもできます。

pDbAlias
入力。システム・データベース・ディレクトリーにカタログされているデータベースの別名を含むストリングを指定します。

REXX API 構文

```
MIGRATE DATABASE dbalias [USER username USING password]
```

REXX API パラメーター

dbalias
移行するデータベースの別名を指定します。

username
データベースの再始動に使用されるユーザー名を指定します。

password
ユーザー名の認証に使用されるパスワードを指定します。

サンプル・プログラム

C	¥sqllib¥samples¥c¥migrate.c
COBOL	¥sqllib¥samples¥cobol¥migrate.cbl
REXX	¥sqllib¥samples¥rexx¥migrate.cmd

使用上の注意

この API はデータベースを新しいバージョンに移行するだけで、移行したデータベースを以前の古いバージョンに変換することはできません。

移行の前にデータベースをカタログする必要があります。

データベースの移行に関する詳細については、*概説およびインストール* を参照してください。

sqlencls - ノード・ディレクトリー走査のクローズ

270ページの『sqlenops - ノード・ディレクトリー走査のオープン』によって割り振られたリソースを解放します。

許可

ありません

必須接続

ありません

API 組み込みファイル

sqlenv.h

C API 構文

```
/* File: sqlenv.h */
/* API: Close Node Directory Scan */
/* ... */
SQL_API_RC SQL_API_FN
  sqlencls (
    unsigned short Handle,
    struct sqlca * pSqlca);
/* ... */
```

汎用 API 構文

```
/* File: sqlenv.h */
/* API: Close Node Directory Scan */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgncls (
    unsigned short Handle,
    struct sqlca * pSqlca);
/* ... */
```

API パラメーター

Handle

入力。関連する OPEN NODE DIRECTORY SCAN API から戻される識別子です。

pSqlca

出力。 *sqlca* 構造を指すポインターです。 この構造に関する詳細は、520ページの『SQLCA』を参照してください。

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。

サンプル・プログラム

C	¥sqllib¥samples¥c¥nodecat.sqc
COBOL	¥sqllib¥samples¥cobol¥nodecat.sqb
REXX	¥sqllib¥samples¥rexx¥nodecat.cmd

参照資料

267ページの『sqlengne - ノード・ディレクトリー次項目の入手』

270ページの『sqlenops - ノード・ディレクトリー走査のオープン』

sqlengne - ノード・ディレクトリー次項目の入手

270ページの『sqlenops - ノード・ディレクトリー走査のオープン』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

ありません

必須接続

ありません

API 組み込みファイル

sqlenv.h

C API 構文

```
/* File: sqlenv.h */
/* API: Get Next Node Directory Entry */
/* ... */
SQL_API_RC SQL_API_FN
  sqlengne (
    unsigned short Handle,
    struct sqleninfo ** ppNodeDirEntry,
    struct sqlca * pSqlca);
/* ... */
```

汎用 API 構文

```
/* File: sqlenv.h */
/* API: Get Next Node Directory Entry */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgngne (
    unsigned short Handle,
    struct sqleninfo ** ppNodeDirEntry,
    struct sqlca * pSqlca);
/* ... */
```

API パラメーター

Handle

入力。 270ページの『sqlenops - ノード・ディレクトリー走査のオープン』から戻された識別子です。

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 270ページの『sqlenops - ノード・ディレクトリー走査のオープン』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。*sqleninfo* 構造に関する詳細については、 568ページの『SQLENINFO』を参照してください。

pSqlca

出力。 *sqlca* 構造を指すポインターです。 この構造に関する詳細は、520ページの『SQLCA』を参照してください。

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 によって戻された識別子を含む REXX ホスト変数。

value ノード項目情報が戻される複合 REXX ホスト変数。名前が指定されなかった場合、名前 `SQLNINFO` が使用されます。以下の項目において、XXX はホスト変数名を表しています (対応するフィールド名は API によって戻される構造から取られています)。

XXX.0 変数内の要素数 (通常 16)

XXX.1 NODENAME

XXX.2 LOCALLU

XXX.3 PARTNERLU

XXX.4 MODE

XXX.5	COMMENT
XXX.6	RESERVED
XXX.7	PROTOCOL (プロトコル・タイプ)
XXX.8	ADAPTER (NetBIOS アダプター番号)
XXX.9	RESERVED
XXX.10	SYMDESTNAME (記号宛先名)
XXX.11	SECURITY (機密保護タイプ)
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)

サンプル・プログラム

C	¥sqllib¥samples¥c¥nodecat.c
COBOL	¥sqllib¥samples¥cobol¥nodecat.cbl
REXX	¥sqllib¥samples¥rexx¥nodecat.cmd

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、走査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、(*pNumEntries* は 270ページの『sqlenops - ノード・ディレクトリー走査のオープン』によって戻されます) 全ディレクトリーを走査することができます。

参照資料

265ページの『sqlencls - ノード・ディレクトリー走査のクローズ』

270ページの『sqlenops - ノード・ディレクトリー走査のオープン』

sqlenops - ノード・ディレクトリー走査のオープン

ノード・ディレクトリーのコピーをメモリーに保管するとともに、項目の数を戻します。このコピーは、ディレクトリーがオープンする時点のディレクトリーのスナップショットです。後になってディレクトリー自体に変更が加えられることがあっても、このコピーが更新されることはありません。

ノード・ディレクトリーの中で次々にノード項目に関する情報を調べていくには、 267ページの『sqlengne - ノード・ディレクトリー次項目の入手』を使用してください。走査をクローズするには、265ページの『sqlencls - ノード・ディレクトリー走査のクローズ』を使用してください。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

ありません

必須接続

ありません

API 組み込みファイル

sqlenv.h

C API 構文

```
/* File: sqlenv.h */
/* API: Open Node Directory Scan */
/* ... */
SQL_API_RC SQL_API_FN
  sqlenops (
    unsigned short * pHandle,
    unsigned short * pNumEntries,
    struct sqlca * pSqlca);
/* ... */
```

汎用 API 構文

```
/* File: sqlenv.h */
/* API: Open Node Directory Scan */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgnops (
    unsigned short * pHandle,
    unsigned short * pNumEntries,
    struct sqlca * pSqlca);
/* ... */
```

API パラメーター

pHandle
出力。この API から戻された識別子です。この識別子を、267ページの『sqlengne - ノード・ディレクトリー次項目の入手』と 265ページの『sqlencls - ノード・ディレクトリー走査のクローズ』とに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です。 この構造に関する詳細は、520ページの『SQLCA』を参照してください。

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複合 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 変数内の要素数 (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。

サンプル・プログラム

C	¥sqllib¥samples¥c¥nodecat.c
COBOL	¥sqllib¥samples¥cobol¥nodecat.cbl
REXX	¥sqllib¥samples¥rexx¥nodecat.cmd

使用上の注意

この API が割り振った記憶域は、 265ページの『sqlencls - ノード・ディレクトリー走査のクローズ』を呼び出すこと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・ディレクトリー走査を発行することができます。とはいえ、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・ディレクトリー走査をオープンすることができます。

参照資料

265ページの『sqlencls - ノード・ディレクトリー走査のクローズ』

267ページの『sqlengne - ノード・ディレクトリー次項目の入手』

sqlepstart - データベース・マネージャーの始動

複数ノード環境で定義された単一のノードまたはすべてのノードで、現行のデータベース・マネージャー・インスタンスのバックグラウンド処理を開始します。

この API はクライアントでは無効です。

効力範囲

複数ノード環境では、この API は *nodenum* パラメーターが使用されている場合を除いて (551ページの『SQLE-START-OPTIONS』を参照してください)、`$HOME/sqllib/db2nodes.cfg` ファイルでリストされているすべてのノードに影響を与えます。

許可

以下のいずれかです。

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

注: OS/2 では、 *ss_logon* データベース・マネージャー構成パラメーターが `0` に設定されている場合、必要な権限はありません。

必須接続

ありません

API 組み込みファイル

sqlenv.h

C API 構文

```
/* File: sqlenv.h */
/* API: Start Database Manager */
/* ... */
SQL_API_RC SQL_API_FN
  sqlepstart (
    struct sqle_start_options * pStartOptions,
    struct sqlca * pSqlca);
/* ... */
```

汎用 API 構文

```
/* File: sqlenv.h */
/* API: Start Database Manager */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgpstart (
    struct sqle_start_options * pStartOptions,
    struct sqlca * pSqlca);
/* ... */
```

API パラメーター

pStartOptions

sqle_start_options 構造を指すポインターです。この構造は始動オプションを含みます。このポインターはヌルにすることができます。この構造に関する詳細については、551ページの『SQLE-START-OPTIONS』を参照してください。

pSqlca

出力。 *sqlca* 構造を指すポインターです。 この構造に関する詳細は、520ページの『SQLCA』を参照してください。

REXX API 構文

この API は、SQLDB2 インターフェースを使って、REXX から呼び出すことができます。 13ページの『API の説明の編成』、または *アプリケーション開発の手引き* を参照してください。構文については、*コマンド解説書* を参照してください。

サンプル・プログラム

C	¥sqllib¥samples¥c¥dbstart.c
COBOL	¥sqllib¥samples¥cobol¥dbstart.cbl
REXX	¥sqllib¥samples¥rexx¥dbstart.cmd

使用上の注意

クライアント・ノードでこの API を呼び出す必要はありません。そのようにすれば古いクライアントとの互換性を得ることができますが、データベース・マネージャーには何の効果もありません。

一度始動したデータベース・マネージャー・インスタンスは、そのデータベース・マネージャー・インスタンスを使用するアプリケーション・プログラムがすべて終了してしまっても、ユーザーがそのデータベース・マネージャー・インスタンスを停止させるまで、無期限に実行を継続します。

複数ノード・データベース環境でパラメーターが指定されていない場合、データベース・マネージャーはノード構成ファイルに指定されているすべての並列ノードで始動します。

API 呼び出しが処理中の場合、必ず適当なノードを始動した後で、データベースへの要求を出すようにしてください。

db2cshrc ファイルはサポートされていないため、環境の定義には使用できません。

UNIX プラットフォーム上では、 **sqlepstart** は SIGINT および SIGALRM シグナルをサポートしています。 SIGINT シグナルは、「CTRL+C」を押すと出されます。 SIGALRM シグナルは、 *start_stop_time* 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た時間になると出されます。どちらかのシグナルが出されると、進行中のすべての始動処理に割り込みが起こり、割り込みが起こった各ノードから $HOME/sqllib/log/db2start. *timestamp*.log エラー・ログ・ファイルにメッセージが戻されます (SIGINT の場合は SQL1044N、 SIGALRM の場合は SQL6037N)。すでに始動しているノードは影響を受けません。始動中のノードで「CTRL+C」を押す場合は、必ずそのノードで **db2stop** を出してから、再始動を試みるようにしてください。

参照資料

161ページの『sqleaddn - ノードの追加』

276ページの『sqlepstp - データベース・マネージャーの停止』

sqlepstp - データベース・マネージャーの停止

現行のデータベース・マネージャー・インスタンスを停止します。明示的に停止されないかぎり、データベース・マネージャーは活動状態を保持します。データベースに接続しているアプリケーションが 1 つでもあれば、この API はデータベース・マネージャー・インスタンスを停止しません。データベース接続がないものの、インスタンス接続がある場合、この API はインスタンス接続を強制実行して、データベース・マネージャーを停止します。また、この API は未解決のデータベース活動化をすべて非活動化してから、データベース・マネージャーを停止します。

この API を使用して、 db2nodes.cfg ファイルからノードを除去することもできます (MPP システムのみ)。

この API はクライアントでは無効です。

効力範囲

複数ノード環境では、この API は *nodenum* パラメーターが使用されている場合を除いて (563ページの『SQLEDBSTOPOPT』を参照してください)、 $HOME/sqllib/db2nodes.cfg ファイルでリストされているすべてのノードに影響を与えます。

許可

以下のいずれかです。

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

注: OS/2 では、 *ss_logon* データベース・マネージャー構成パラメーターが 0 に設定されている場合、必要な権限はありません。

必須接続

ありません

API 組み込みファイル

sqlenv.h

C API 構文

```
/* File: sqlenv.h */
/* API: Stop Database Manager */
/* ... */
SQL_API_RC SQL_API_FN
  sqlepstp (
    struct sqledbstopopt * pStopOptions,
    struct sqlca * pSqlca);
/* ... */
```

汎用 API 構文

```
/* File: sqlenv.h */
/* API: Stop Database Manager */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgpstp (
    struct sqledbstopopt * pStopOptions,
    struct sqlca * pSqlca);
/* ... */
```

API パラメーター

pStopOptions

sqledbstopopt 構造を指すポインター。この構造は停止オプションを含みます。このポインターはヌルにすることができます。この構造に関する詳細については、 563ページの『SQLEDBSTOPOPT』を参照してください。

pSqlca

出力。 *sqlca* 構造を指すポインターです。 この構造に関する詳細は、520ページの『SQLCA』を参照してください。

REXX API 構文

この API は、SQLDB2 インターフェースを使って、REXX から呼び出すことができます。 13ページの『API の説明の編成』、または *アプリケーション開発の手引き* を参照してください。構文については、*コマンド解説書* を参照してください。

サンプル・プログラム

C	¥sqllib¥samples¥c¥dbstop.c
COBOL	¥sqllib¥samples¥cobol¥dbstop.cbl
REXX	¥sqllib¥samples¥rexx¥dbstop.cmd

使用上の注意

クライアント・ノードでこの API を呼び出す必要はありません。そのようにすれば古いクライアントとの互換性を得ることができますが、データベース・マネージャーには何の効果もありません。

一度始動したデータベース・マネージャー・インスタンスは、そのデータベース・マネージャー・インスタンスを使用するアプリケーション・プログラムがすべて終了してしまっても、ユーザーがそのデータベース・マネージャー・インスタンスを停止させるまで、無期限に実行を継続します。

アプリケーション・プログラムがまだデータベースに接続されているためにデータベース・マネージャーを停止できない場合は、まず 233ページの『sqlefrce - アプリケーションの強制終了』を使用してすべてのユーザーを切断するか、FORCE オプションを使用して **sqlepstp** API を再び呼び出してください。

現在、以下の情報は複数ノード環境にのみ適用されます。

- パラメーターが 1 つも指定されていない場合、データベース・マネージャーはノード構成ファイルでリストされているノードごとに停止します。`db2diag.log` ファイルには、他のノードが遮断中であることを示すメッセージが含まれている場合があります。
- 前回 **sqlepstp** を呼び出した後で MPP システムに追加したノードは、`db2nodes.cfg` ファイルで更新されます。
- UNIX プラットフォーム上で、この API は SIGALRM シグナルをサポートしています。このシグナルは *start_stop_time* データベース・マネージャー構成パラメーターで指定された時間になると出されます。このシグナルが出されると、進行中のすべての停止処理に割り込みが起こり、割り込みが起こった各ノードから `$HOME/sqllib/log/db2stop.` *timestamp*`.log` エラー・ログ・ファイルにメッセージ SQL6037N が戻されます。すでに停止しているノードは影響を受けません。
- `db2cshrc` ファイルはサポートされていないため、 PROFILE パラメーターの値として指定することはできません。

参照資料

158ページの『sqle_deactivate_db - データベースの非活動化』

226ページの『sqledrpn - ノードのドロップの検査』

233ページの『sqlefrce - アプリケーションの強制終了』

273ページの『sqlepstart - データベース・マネージャーの始動』

sqleqryc - クライアントの照会

アプリケーション・プロセスの現行の接続設定を戻します。適切な接続設定およびその値については、535ページの『SQLE-CONN-SETTING』を参照してください。

許可

ありません

必須接続

ありません

API 組み込みファイル

sqlenv.h

C API 構文

```
/* File: sqlenv.h */
/* API: Query Client */
/* ... */
SQL_API_RC SQL_API_FN
  sqleqryc (
    struct sqle_conn_setting * pConnectionSettings,
    unsigned short NumSettings,
    struct sqlca * pSqlca);
/* ... */
```

汎用 API 構文

```
/* File: sqlenv.h */
/* API: Query Client */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgqryc (
    struct sqle_conn_setting * pConnectionSettings,
    unsigned short NumSettings,
    struct sqlca * pSqlca);
/* ... */
```

API パラメーター

pConnectionSettings

入力 / 出力。接続設定のタイプおよび値を指定する *sqle_conn_setting* 構造を指すポインターです。ユーザーは、 *NumSettings* 個の接続設定構造の配列を定義し、この配列内の各要素の *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、各要素の *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。この構造に関する詳細については、535ページの『SQLE-CONN-SETTING』を参照してください。

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 ～ 7) を指定します。

pSqlca

出力。 *sqlca* 構造を指すポインターです。 この構造に関する詳細は、520ページの『SQLCA』を参照してください。

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複合 REXX ホスト変数。以下の項目において、XXX はホスト変数名を表しています。

XXX.1 CONNECTION タイプの現行接続設定。

XXX.2 SQLRULES の現行接続設定。

XXX.3 COMMIT の発行時にどの接続が解放されるのかを示す現行接続設定。

XXX.4 SYNCPOINT オプションの現行接続設定。 2 フェーズ・コミットの意味を適用するためにトランザクション管理プログラムが使用されるべきかどうか、単一のトランザクション内で複数のデータベースがアクセスされる場合に、更新されるデータベースが 1 つだけであることをデータベース・マネージャーが確認するべ

きかどうか、あるいはこのようなオプションがいずれも使用されないかを示します。

XXX.5 NETBIOS アダプターに関連して同時に存在する接続の最大数を示す、現行接続設定。

XXX.6 据え置かれた PREPARE の現行接続設定。

サンプル・プログラム

C ¥sqllib¥samples¥c¥client.c

COBOL ¥sqllib¥samples¥cobol¥client.cbl

REXX ¥sqllib¥samples¥rexx¥client.cmd

使用上の注意

アプリケーション・プロセスの接続設定は、実行中にいつでも照会できます。

QUERY CLIENT が正常に出されると、 *sqle_conn_setting* 構造のフィールドには、アプリケーション・プロセスの現行の接続設定が含まれます。 SET CLIENT がまだ呼び出されていない場合、設定値には、 SQL ステートメントがすでに処理されている場合にのみ、プリコンパイル・オプションの値が使われます。そうでない場合には、プリコンパイル・オプションの省略時値が使われます。

分散作業単位 (DUOW) に関する詳細については、 *管理の手引き* を参照してください。

参照資料

283ページの『sqleqryi - クライアント情報の照会』

294ページの『sqlesetc - クライアントの設定』

sqleqryi - クライアント情報の照会

既存のクライアント情報を戻します。この API はデータベース別名の指定を許可するため、アプリケーションは特定の接続と関連したクライアント情報を照会することができます。 298ページの『sqleseti - クライアント情報の設定』が前に確立された値でない場合、ヌルを戻します。

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

ありません

必須接続

ありません

API 組み込みファイル

sqlenv.h

C API 構文

```
/* File: sqlenv.h */
/* API: Query Client Information */
/* ... */
SQL_API_RC SQL_API_FN
  sqleqryi (
    unsigned short DbAliasLen,
    char * pDbAlias,
    unsigned short NumItems,
    struct sqle_client_info* pClient_Info,
    struct sqlca * pSqlca);
/* ... */
```

汎用 API 構文

```
/* File: sqlenv.h */
/* API: Query Client Information */
/* ... */
SQL_API_RC SQL_API_FN
  sqleqryi (
    unsigned short DbAliasLen,
    char * pDbAlias,
    unsigned short NumItems,
    struct sqle_client_info* pClient_Info,
    struct sqlca * pSqlca);
/* ... */
```

API パラメーター

DbAliasLen
入力。データベース別名の長さを示す 2 バイトの無符号整数 (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です。

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です。その構造のそれぞれには、戻される値を示すタイプ・フィールドと、その戻される値を指すポインターが含まれています。ポインターが指す領域は、要求されている値を十分収容できる大きさでなければなりません。この構造の詳細については、532ページの『SQLE-CLIENT-INFO』を参照してください。

pSqlca
出力。 *sqlca* 構造を指すポインターです。 この構造に関する詳細は、520ページの『SQLCA』を参照してください。

サンプル・プログラム

C ¥sqllib¥samples¥c¥cli_info.c

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。 298ページの『sqleseti - クライアント情報の設定』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、およびヌル文字で終了するストリング (¥0) を戻します。

参照資料

298ページの『sqleseti - クライアント情報の設定』

sqleregs - 登録

ネットワーク・サーバー上で DB2 サーバーを定義します。 DB2 サーバーのネットワーク・アドレスは、ファイル・サーバー上の指定した登録簿に保管されます。そのアドレスは、ファイル・サーバーにおいて、 IPX/SPX 通信プロトコルを使用するクライアント・アプリケーションによって検索することができます。

許可

ありません

必須接続

ありません

API 組み込みファイル

sqlenv.h

C API 構文

```
/* File: sqlenv.h */
/* API: Register */
/* ... */
SQL_API_RC SQL_API_FN
  sqleregs (
    unsigned short Registry,
    void * pRegisterInfo,
    struct sqlca * pSqlca);
/* ... */
```

汎用 API 構文

```
/* File: sqlenv.h */
/* API: Register */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgregs (
    unsigned short Registry,
    void * pRegisterInfo,
    struct sqlca * pSqlca);
/* ... */
```

API パラメーター

Registry

入力。 DB2 サーバーを登録するネットワーク・ファイル・サーバーを指定します。このリリースでは、 SQL_NWBINDERY (sqlenv で定義された NetWare ファイル・サーバー・バインダリー) だけがサポートされます。

pRegisterInfo

入力。 *sqle_reg_nwbindery* 構造を指すポインターです。この構造では、呼び出し側がネットワーク・ファイル・サーバーにおいて有効なユーザー名とパスワードを指定します。この構造に関する詳細については、550ページの『SQLE-REG-NWBINDERY』を参照してください。

pSqlca

出力。 *sqlca* 構造を指すポインターです。 この構造に関する詳細は、520ページの『SQLCA』を参照してください。

REXX API 構文

この API は、SQLDB2 インターフェースを使って、REXX から呼び出すことができます。 13ページの『API の説明の編成』、または *アプリケーション開発の手引き* を参照してください。構文については、*コマンド解説書* を参照してください。

サンプル・プログラム

C	¥sqllib¥samples¥c¥regder.c
COBOL	¥sqllib¥samples¥cobol¥regder.cbl

使用上の注意

この API は、DB2 サーバー・マシン (この API が呼び出されたマシン) の IPX/SPX アドレスを判別した後、データベース・マネージャー構成ファイルに指定された *objectname* の値を使用して NetWare ファイル・サーバー・バインダリーにオブジェクトを作成します。 DB2 サーバーの IPX/SPX アドレスは、そのオブジェクトの特性として保管されます。クライアントが IPX/SPX ファイル・サーバー・アドレス指定を用いて DB2 データベースに接続するためには、ノード・ディレクトリーにある IPX/SPX ノードを (サーバーで指定されたものと同じ FILESERVER および OBJECTNAME を使用して) カタログする必要があります。

指定した NetWare ユーザー名およびパスワードには、管理権限またはそれと同等の権限が必要です。

この API は、 DB2 サーバーからローカルに発行しなければなりません。リモートにはサポートされていません。

DB2 のインストールおよび構成が完了したら、 DB2 サーバーをネットワーク・ファイル・サーバーに一度登録する必要があります (IPX/SPX クライアントが*直接アドレス指定* によってこの DB2 サーバーに接続する場合だけは除きます)。その後、IPX/SPX フィールドを再構成するか、または DB2 サーバーの IPX/SPX インターネットワーク・アドレスが変更される場合には、まずネットワーク・サーバー上で DB2 サーバーの登録を解除し、次に変更を行って、最後に登録し直してください。

参照資料

219ページの『sqledreg - 登録解除』

sqlesact - 会計ストリングの設定

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

ありません

必須接続

ありません

API 組み込みファイル

sqlenv.h

C API 構文

```
/* File: sqlenv.h */
/* API: Set Accounting String */
/* ... */
SQL_API_RC SQL_API_FN
  sqlesact (
    char * pAccountingString,
    struct sqlca * pSqlca);
/* ... */
```

汎用 API 構文

```
/* File: sqlenv.h */
/* API: Set Accounting String */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgsact (
    unsigned short AccountingStringLen,
    char * pAccountingString,
    struct sqlca * pSqlca);
/* ... */
```

API パラメーター

AccountingStringLen
入力。会計ストリングの長さを示す 2 バイトの無符号整数 (バイト単位) を指定します。

pAccountingString
入力。会計データ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です。 この構造に関する詳細は、520ページの『SQLCA』を参照してください。

サンプル・プログラム

C	¥sqllib¥samples¥c¥setact.c
COBOL	¥sqllib¥samples¥cobol¥setact.cbl

使用上の注意

会計データ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。会計ストリングは、最大 SQL_ACCOUNT_STR_SZ (sqlenv で定義) で指定されたバイト数の長さにまですることができます。それよりも長い場合は、切り捨てられます。 DRDA サーバーへの伝送時に、会計ストリングが正しく変換されるようにするため、文字 A ～ Z、0 ～ 9、および下線記号 (_) だけを使用するようにしてください。

参照資料

DB2 コネクト 使用者の手引き には、会計ストリングと、それをサポートする DRDA サーバーについての詳細が記載されています。

298ページの『sqleseti - クライアント情報の設定』

sqlesdeg - 実行時処理度の設定

指定された活動アプリケーションの SQL ステートメントに、区画内での最大の実行時並列処理度を設定します。この API は、CREATE INDEX の並列処理には影響を与えません。

効力範囲

この API は、 $HOME/sqllib/db2nodes.cfg ファイルにリストされているすべてのノードに影響を与えます。

許可

以下のいずれかです。

- *sysadm*
- *sysctrl*

必須接続

インスタンス。リモート・サーバーにおける最大の実行時並列処理度を変更するには、まず、そのサーバーに接続することが必要です。接続が存在しない場合、SET RUNTIME DEGREE ステートメントは失敗します。

API 組み込みファイル

sqlenv.h

C API 構文

```
/* File: sqlenv.h */
/* API: Set Runtime Degree */
/* ... */
SQL_API_RC SQL_API_FN
  sqlesdeg (
    sqlint32 NumAgentIds,
    sqluint32 * pAgentIds,
    sqlint32 Degree,
    struct sqlca * pSqlca);
/* ... */
```

汎用 API 構文

```
/* File: sqlenv.h */
/* API: Set Runtime Degree */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgsdeg (
    struct sqlca * pSqlca,
    sqlint32 Degree,
    sqluint32 * pAgentIds,
    sqlint32 NumAgentIds);
/* ... */
```

API パラメーター

pSqlca

出力。 *sqlca* 構造を指すポインターです。 この構造に関する詳細は、520ページの『SQLCA』を参照してください。

Degree

入力。最大の実行時並列処理度の新規の値を指定します。値の範囲は 1 〜 32767 です。

pAgentIds

入力。無符号の長整数の配列を指すポインターです。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。活動アプリケーションのエージェント ID をリストするには、 31ページの『db2GetSnapshot - スナップショットの入手』を使用してください。

NumAgentIds

入力。新規の並列処理度の値が適用される活動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 の配列の要素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、新規の並列処理度はすべての活動アプリケーションに適用されます。このパラメーターがゼロに設定された場合、エラーが戻されます。

REXX API 構文

この API は、SQLDB2 インターフェースを使って、REXX から呼び出すことができます。 13ページの『API の説明の編成』、または *アプリケーション開発の手引き* を参照してください。構文については、*コマンド解説書* を参照してください。

サンプル・プログラム

C ¥sqllib¥samples¥c¥setrundg.c

使用上の注意

活動アプリケーションのエージェント ID と並列処理度を収集するには、データベース・システム・モニター機能を使用します。詳細については、システム・モニター *手引きおよび解説書* を参照してください。

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、指定した要素の合計数を含む配列をポインターが指していることを確認する必要があります。 *NumAgentIds* が SQL_ALL_USERS に設定されている場合、その配列は無視されます。

指定されたエージェント ID の 1 つかそれ以上が見つからない場合には、認識されないエージェント ID は無視され、機能が続行されます。エラーは戻されません。エージェント ID は、たとえば、エージェント ID が収集されてから API が呼び出されるまでの間にユーザーがサインオフした場合などには、見つからないことがあります。

エージェント ID は再生され、さらに、データベース・システム・モニターによる収集のしばらく後で、アプリケーションの並列処理度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、新規の並列処理度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。

参照資料

31ページの『db2GetSnapshot - スナップショットの入手』

sqlesetc - クライアントの設定

アプリケーション用の接続設定を指定します。適切な接続設定およびその値については、 535ページの『SQLE-CONN-SETTING』を参照してください。

許可

ありません

必須接続

ありません

API 組み込みファイル

sqlenv.h

C API 構文

```
/* File: sqlenv.h */
/* API: Set Client */
/* ... */
SQL_API_RC SQL_API_FN
  sqlesetc (
    struct sqle_conn_setting * pConnectionSettings,
    unsigned short NumSettings,
    struct sqlca * pSqlca);
/* ... */
```

汎用 API 構文

```
/* File: sqlenv.h */
/* API: Set Client */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgsetc (
    struct sqle_conn_setting * pConnectionSettings,
    unsigned short NumSettings,
    struct sqlca * pSqlca);
/* ... */
```

API パラメーター

pConnectionSettings

入力。 *sqle_conn_setting* 構造を指すポインターです。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、この配列の各要素の *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。この構造に関する詳細については、535ページの『SQLE-CONN-SETTING』を参照してください。

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 ～ 7) を指定します。

pSqlca
出力。 *sqlca* 構造を指すポインターです。 この構造に関する詳細は、520ページの『SQLCA』を参照してください。

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複合 REXX ホスト変数。以下の項目において、XXX はホスト変数名を表しています。

XXX.0 確立される接続設定の数。

XXX.1 CONNECTION タイプの設定方法を指定します。有効な値は以下のとおりです。

- **1** Type 1 CONNECT
- **2** Type 2 CONNECT

XXX.2 SQLRULES の設定方法を指定します。有効な値は以下のとおりです。

- **DB2** DB2 規則に従って type 2 CONNECT を処理します。
- **STD** 標準規則に従って type 2 CONNECT を処理します。

XXX.3 データベースへの接続を切断する際の、切断の有効範囲の設定方法を指定します。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 トランザクション管理プログラム (TM) を使用して、 2 フェーズ・コミットを調整します。

ONEPHASE 1 フェーズ・コミットを使用します。

NONE 単一の更新プログラムに複数の読み取りプログラムという形を適用しません。

XXX.5 NETBIOS アダプターを使用している場合に、同時に存在できる接続の最大数を指定します。

XXX.6 PREPARE ステートメントを実行すべきときを指定します。有効な値は以下のとおりです。

NO PREPARE ステートメントは、それが発行された時点で実行されます。

YES PREPARE ステートメントは、対応する OPEN、DESCRIBE、または EXECUTE ステートメントが発行されるまで実行されません。ただし、PREPARE INTO ステートメントは据え置かれません。

ALL PREPARE INTO ステートメントも据え置かれる点を除き、YES と同じです。

サンプル・プログラム

C ¥sqllib¥samples¥c¥client.c

COBOL ¥sqllib¥samples¥cobol¥client.cbl

REXX ¥sqllib¥samples¥rexx¥client.cmd

使用上の注意

この API が成功すると、それに続く作業単位内の接続では、指定された接続設定が使用されます。この API が失敗した場合、接続設定は未変更のままです。

アプリケーションの接続設定は、既存の接続がない場合 (たとえば、接続が確立される前、あるいは RELEASE ALL や COMMIT の後など) にのみ変更できます。

いったん SET CLIENT API が正常に実行されると、接続設定は固定され、再び SET CLIENT API を実行しないかぎり変更できません。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、指定変更されます。

分散作業単位 (DUOW) に関する詳細については、 *管理の手引き* を参照してください。

参照資料

280ページの『sqleqryc - クライアントの照会』

298ページの『sqleseti - クライアント情報の設定』

sqleseti - クライアント情報の設定

接続がすでに存在する場合、アプリケーションが特定の接続と関連したクライアント情報を設定することを許可します。

TP モニターまたは 3 層のクライアント / サーバー環境では、クライアントの代わりに作動しているアプリケーション・サーバーだけでなく、クライアントについての情報も獲得する必要があります。この API を使うことにより、アプリケーション・サーバーはクライアントのユーザー ID、ワークステーション情報、プログラム情報、および他の会計情報を DB2 サーバーに渡すことができます。そうでない場合、アプリケーション・サーバーの情報だけが渡され、たいてい、その情報は同じアプリケーション・サーバーを介して行う多くのクライアント呼び出しと同じです。

アプリケーションは、クライアント情報が既存のすべての接続と、今後行われる接続に合わせて設定される場合に備え、別名を指定しないことを選択することができます。この API は作業単位の外部で変更される情報を、SQL の実行前か、コミットまたはロールバックの後のどちらかに許可するだけです。呼び出しが正常に終了した場合、接続の値は次の機会に送られ、その接続で送信される次の SQL 要求でグループ化されます。正常な呼び出しは、値が受け入れられていること、および後続の接続にそれらの値が伝搬することを意味します。

この API は、データベースへの接続より前に値を確立するために使用するか、接続が確立されてからは値を設定または修正するために使用できます。

許可

ありません

必須接続

ありません

API 組み込みファイル

sqlenv.h

C API 構文

```
/* File: sqlenv.h */
/* API: Set Client Information */
/* ... */
SQL_API_RC SQL_API_FN
  sqleseti (
    unsigned short DbAliasLen,
    char * pDbAlias,
    unsigned short NumItems,
    struct sqle_client_info* pClient_Info,
    struct sqlca * pSqlca);
/* ... */
```

汎用 API 構文

```
/* File: sqlenv.h */
/* API: Set Client Information */
/* ... */
SQL_API_RC SQL_API_FN
  sqleseti (
    unsigned short DbAliasLen,
    char * pDbAlias,
    unsigned short NumItems,
    struct sqle_client_info* pClient_Info,
    struct sqlca * pSqlca);
/* ... */
```

API パラメーター

DbAliasLen
入力。データベース別名の長さを示す 2 バイトの無符号整数 (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です。

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。この構造の詳細については、532ページの『SQLE-CLIENT-INFO』を参照してください。

pSqlca
出力。 *sqlca* 構造を指すポインターです。 この構造に関する詳細は、520ページの『SQLCA』を参照してください。

サンプル・プログラム

C ¥sqllib¥samples¥c¥cli_info.c

使用上の注意

別名が提供された場合、別名への接続がすでに存在していなければならず、その別名へのすべての接続は変更を継承します。情報は、その別名への接続が中断されるまで保存されます。別名が提供されなかった場合、すべての既存の接続の設定は変更され、将来の接続が変更を継承します。情報は、プログラムが終了するまで保存されます。

フィールド名は、提供できる情報のタイプのガイドラインを表します。たとえば、TP モニター・アプリケーションは、SQL_CLIENT_INFO_APPLNAM フィールドに、アプリケーション名と共に TP モニター・トランザクション ID を提供することができます。これにより、 DB2 トランザクション ID を TP モニター・トランザクション ID と関連付けることができるので、 DB2 サーバー上でのモニターと会計の機能が向上します。

現在、この API は DB2 (OS/390 版) バージョン 5 以降にのみ情報を渡します。すべての情報 (会計ストリングを除く) 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。

参照資料

283ページの『sqleqryi - クライアント情報の照会』

289ページの『sqlesact - 会計ストリングの設定』

294ページの『sqlesetc - クライアントの設定』

sqleuncd - データベースのアンカタログ

システム・データベース・ディレクトリーから項目を削除します。

許可

以下のいずれかです。

- *sysadm*
- *sysctrl*

必須接続

ありません

API 組み込みファイル

sqlenv.h

C API 構文

```
/* File: sqlenv.h */
/* API: Uncatalog Database */
/* ... */
SQL_API_RC SQL_API_FN
  sqleuncd (
    _SQLOLDCHAR * pDbAlias,
    struct sqlca * pSqlca);
/* ... */
```

汎用 API 構文

```
/* File: sqlenv.h */
/* API: Uncatalog Database */
/* ... */
SQL_API_RC SQL_API_FN
  sqlguncd (
    unsigned short DbAliasLen,
    struct sqlca * pSqlca,
    _SQLOLDCHAR * pDbAlias);
/* ... */
```

API パラメーター

DbAliasLen
入力。データベース別名の長さを示す 2 バイトの無符号整数 (バイト単位) です。

pSqlca
出力。 *sqlca* 構造を指すポインターです。 この構造に関する詳細は、520ページの『SQLCA』を参照してください。

pDbAlias
入力。アンカタログするデータベースの別名を含むストリングを指定します。

REXX API 構文

```
UNCATALOG DATABASE dbname
```

REXX API パラメーター

dbname
アンカタログするデータベースの別名を指定します。

サンプル・プログラム

C	¥sqllib¥samples¥c¥dbcat.c
COBOL	¥sqllib¥samples¥cobol¥dbcat.cbl
REXX	¥sqllib¥samples¥rexx¥dbcat.cmd

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、222ページの『sqledrpd - データベースの消去』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、174ページの『sqlecadb - データベースのカタログ』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、212ページの『sqledosd - データベース・ディレクトリー走査のオープン』、 208ページの

『sqledgne - データベース・ディレクトリーの次項目の入手』、および 206ページの『sqledcls - データベース・ディレクトリー走査のクローズ』を使用してください。

最初にデータベースをアンカタログし、次に別のタイプを指定してデータベースを再カタログすることにより、下位レベル・サーバーと通信する際に使用される、データベースの認証タイプを変更できます。

ディレクトリーのキャッシュが使用可能にされている場合 (331ページの『sqlfxsys - データベース・マネージャー構成の入手』の構成パラメーター *dir_cache* を参照)、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。

参照資料

174ページの『sqlecadb - データベースのカタログ』

206ページの『sqledcls - データベース・ディレクトリー走査のクローズ』

208ページの『sqledgne - データベース・ディレクトリーの次項目の入手』

212ページの『sqledosd - データベース・ディレクトリー走査のオープン』

sqleuncn - ノードのアンカタログ

ノード・ディレクトリーから項目を削除します。

許可

以下のいずれかです。

- *sysadm*
- *sysctrl*

必須接続

ありません

API 組み込みファイル

sqlenv.h

C API 構文

```
/* File: sqlenv.h */
/* API: Uncatalog Node */
/* ... */
SQL_API_RC SQL_API_FN
  sqleuncn (
    _SQLOLDCHAR * pNodeName,
    struct sqlca * pSqlca);
/* ... */
```

汎用 API 構文

```
/* File: sqlenv.h */
/* API: Uncatalog Node */
/* ... */
SQL_API_RC SQL_API_FN
  sqlguncn (
    unsigned short NodeNameLen,
    struct sqlca * pSqlca,
    _SQLOLDCHAR * pNodeName);
/* ... */
```

API パラメーター

NodeNameLen
入力。ノード名の長さを示す 2 バイトの無符号整数 (バイト単位) です。

pSqlca
出力。 *sqlca* 構造を指すポインターです。 この構造に関する詳細は、520ページの『SQLCA』を参照してください。

pNodeName
入力。アンカタログするノードの名前を含むストリングを指定します。

REXX API 構文

```
UNCATALOG NODE nodename
```

REXX API パラメーター

nodename
アンカタログするノードの名前。

サンプル・プログラム

C	¥sqllib¥samples¥c¥nodecat.c
COBOL	¥sqllib¥samples¥cobol¥nodecat.cbl
REXX	¥sqllib¥samples¥rexx¥nodecat.cmd

使用上の注意

ノードを再カタログする場合には、196ページの『sqlectnd - ノードのカタログ』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、270ページの『sqlenops - ノード・ディレクトリー走査のオープン』、 267ページの『sqlengne - ノード・ディレクトリー次項目の入手』、および 265ページの『sqlencls - ノード・ディレクトリー走査のクローズ』を使用してください。

ディレクトリーのキャッシュが使用可能にされている場合 (331ページの『sqlfxsys - データベース・マネージャー構成の入手』の構成パラメーター *dir_cache* を参照)、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。

参照資料

196ページの『sqlectnd - ノードのカタログ』

265ページの『sqlencls - ノード・ディレクトリー走査のクローズ』

267ページの『sqlengne - ノード・ディレクトリー次項目の入手』

270ページの『sqlenops - ノード・ディレクトリー走査のオープン』

sqlfddb - データベース構成の省略時値の入手

データベース構成ファイルにある個々の項目の省略時値を戻します。

許可

ありません

必須接続

インスタンス。リモート・データベースの構成を設定する場合でも、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には、呼び出しの期間中にリモート・ノードへのインスタンス接続が確立されます。

API 組み込みファイル

sqlutil.h

C API 構文

```
/* File: sqlutil.h */
/* API: Get Database Configuration Defaults */
/* ... */
SQL_API_RC SQL_API_FN
  sqlfddb (
    char * pDbAlias,
    unsigned short NumItems,
    struct sqlfupd * pItemList,
    struct sqlca * pSqlca);
/* ... */
```

汎用 API 構文

```
/* File: sqlutil.h */
/* API: Get Database Configuration Defaults */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgddb (
    unsigned short DbAliasLen,
    unsigned short NumItems,
    struct sqlfupd * pItemList,
    struct sqlca * pSqlca,
    char * pDbAlias);
/* ... */
```

API パラメーター

DbAliasLen
入力。データベース別名の長さを示す 2 バイトの無符号整数 (バイト単位) です。

NumItems
入力。戻される項目の数を示します。最小有効値は 1 です。

pItemList
入力 / 出力。 *NumItems* 個の *sqlfupd* 構造の配列を指すポインターです。構造のそれぞれには、戻される値を示すトークン・フィールド、および構成値を位置づける箇所を示すポインター・フィールドが含まれています。この構造に関する詳細については、572ページの『SQLFUPD』を参照してください。

pSqlca
出力。 *sqlca* 構造を指すポインターです。 この構造に関する詳細は、520ページの『SQLCA』を参照してください。

pDbAlias
入力。データベース別名を含むストリングを指定します。

サンプル・プログラム

C	¥sqllib¥samples¥c¥d_dbconf.c
COBOL	¥sqllib¥samples¥cobol¥d_dbconf.cbl

使用上の注意

アプリケーションは、戻されるデータ要素のそれぞれに十分なメモリーを割り振る必要があります。たとえば、*newlogpath* に戻される値は、最大 242 バイトの長さになります。

DB2 は、更新不能パラメーターの現行値を戻します。

エラーが生じた場合には、戻された情報は無効になります。構成ファイルが無効な場合には、エラー・メッセージが戻されます。その場合には、データベースをバックアップ版から復元しなければなりません。

データベース構成パラメーターを推奨されているデータベース・マネージャーの省略時値に設定するには、 313ページの『sqlfrdb - データベース構成のリセット』を使用してください。

データベース構成パラメーターの要旨については、 *コマンド解説書* を参照してください。これらのパラメーターの調整に関する詳細については、 *管理の手引き* を参照してください。

参照資料

313ページの『sqlfrdb - データベース構成のリセット』

319ページの『sqlfudb - データベース構成の更新』

327ページの『sqlfxdb - データベース構成の入手』

sqlfdsys - データベース・マネージャー構成の省略時値の入手

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省略時値を戻します。

許可

ありません

必須接続

なし、またはインスタンス。現行インスタンス (**DB2INSTANCE** 環境変数の値で定義された) でデータベース・マネージャーの構成操作を実行する場合、インスタンス接続は必要ありません。しかし、現行以外のインスタンスでデータベース・マネージャーの構成操作を実行する場合には、インスタンス接続が必要になります。現行以外のインスタンスのデータベース・マネージャー構成を表示するには、まず最初にそのインスタンスに接続することが必要です。

API 組み込みファイル

sqlutil.h

C API 構文

```
/* File: sqlutil.h */
/* API: Get Database Manager Configuration Defaults */
/* ... */
SQL_API_RC SQL_API_FN
  sqlfdsys (
    unsigned short NumItems,
    struct sqlfupd * pItemList,
    struct sqlca * pSqlca);
/* ... */
```

汎用 API 構文

```
/* File: sqlutil.h */
/* API: Get Database Manager Configuration Defaults */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgdsys (
    unsigned short NumItems,
    struct sqlfupd * pItemList,
    struct sqlca * pSqlca);
/* ... */
```

API パラメーター

NumItems
入力。戻される項目の数を示します。最小有効値は 1 です。

pItemList
入力 / 出力。 *NumItems* 個の *sqlfupd* 構造の配列を指すポインターです。構造のそれぞれには、戻される値を示すトークン・フィールド、および構成値を位置づける箇所を示すポインター・フィールドが含まれています。この構造に関する詳細については、572ページの『SQLFUPD』を参照してください。

pSqlca
出力。 *sqlca* 構造を指すポインターです。 この構造に関する詳細は、520ページの『SQLCA』を参照してください。

サンプル・プログラム

C	¥sqllib¥samples¥c¥d_dbmcon.c
COBOL	¥sqllib¥samples¥cobol¥d_dbmcon.cbl

使用上の注意

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) への接続が存在する場合、それらのインスタンスに接続されたサーバーの省略時のデータベース・マネージャー構成パラメーターが戻されます。そのようなインスタンスが存在しない場合には、ローカルの省略時データベース・マネージャー構成パラメーターが戻されます。

エラーが生じた場合には、戻された情報は無効になります。構成ファイルが無効な場合には、エラー・メッセージが戻されます。そのような場合には、データベース・マネージャーを再インストールして回復を実行する必要があります。

更新不能パラメーターの現行値は、省略時値として戻されます。

データベース・マネージャー構成パラメーターを推奨されているデータベース・マネージャーの省略時値に設定するには、 316ページの『sqlfrsys - データベース・マネージャー構成のリセット』を使用してください。

データベース・マネージャー構成パラメーターの要旨については、 *コマンド解説書* を参照してください。これらのパラメーターの調整に関する詳細については、 *管理の手引き* を参照してください。

参照資料

316ページの『sqlfrsys - データベース・マネージャー構成のリセット』

323ページの『sqlfusys - データベース・マネージャー構成の更新』

331ページの『sqlfxsys - データベース・マネージャー構成の入手』

sqlfrdb - データベース構成のリセット

特定のデータベースの構成ファイルをシステム省略時値にリセットします。

効力範囲

この API は、それが発行されたノードにのみ影響を与えます。

許可

以下のいずれかです。

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

必須接続

インスタンス。明示的な接続は必要ありません。データベースがリモートとしてリストされている場合には、呼び出しの期間中にリモート・ノードへのインスタンス接続が確立されます。

API 組み込みファイル

sqlutil.h

C API 構文

```
/* File: sqlutil.h */
/* API: Reset Database Configuration */
/* ... */
SQL_API_RC SQL_API_FN
  sqlfrdb (
    _SQLOLDCHAR * pDbAlias,
    struct sqlca * pSqlca);
/* ... */
```

汎用 API 構文

```
/* File: sqlutil.h */
/* API: Reset Database Configuration */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgrdb (
    unsigned short DbAliasLen,
    struct sqlca * pSqlca,
    char * pDbAlias);
/* ... */
```

API パラメーター

DbAliasLen
入力。データベース別名の長さを示す 2 バイトの無符号整数 (バイト単位) です。

pSqlca
出力。 *sqlca* 構造を指すポインターです。 この構造に関する詳細は、520ページの『SQLCA』を参照してください。

pDbAlias
入力。データベース別名を含むストリングを指定します。

REXX API 構文

```
RESET DATABASE CONFIGURATION FOR dbname
```

REXX API パラメーター

dbname
構成ファイルに関連したデータベースの別名。

サンプル・プログラム

C	¥sqllib¥samples¥c¥dbconf.c
COBOL	¥sqllib¥samples¥cobol¥dbconf.cbl
REXX	¥sqllib¥samples¥rexx¥dbconf.cmd

使用上の注意

この API は、構成全体 (更新不能パラメーターを除く) をリセットします。

現行のデータベース構成パラメーターのリストを表示または出力するには、327ページの『sqlfxdb - データベース構成の入手』を使用してください。

データベース構成パラメーターの省略時値を表示するには、 307ページの『sqlfddb - データベース構成の省略時値の入手』を使用してください。

構成可能パラメーターの値を変更するには、 319ページの『sqlfudb - データベース構成の更新』を使用してください。

データベース構成ファイルに加えられた変更は、変更がメモリーにロードされてはじめて有効になります。ファイルがメモリーにロードされる前に、すべてのアプリケーションがデータベースから切断されていなければなりません。

エラーが発生した場合、データベース構成ファイルは変更されません。

チェックサムが無効の場合は、データベース構成ファイルをリセットすることができません。これは、正しい API を使用せずにデータベース構成ファイルを変更した場合に起こる可能性があります。その場合、データベースを復元して、データベース構成ファイルをリセットする必要が生じます。

データベース構成パラメーターの要旨については、 *コマンド解説書* を参照してください。これらのパラメーターに関する詳細については、 *管理の手引き* を参照してください。

参照資料

307ページの『sqlfddb - データベース構成の省略時値の入手』

319ページの『sqlfudb - データベース構成の更新』

327ページの『sqlfxdb - データベース構成の入手』

sqlfrsys - データベース・マネージャー構成のリセット

データベース・マネージャー構成ファイルのパラメーターを、システム省略時値にリセットします。

許可

sysadm

必須接続

なし、またはインスタンス。現行インスタンス (**DB2INSTANCE** 環境変数の値で定義された) でデータベース・マネージャーの構成操作を実行する場合、インスタンス接続は必要ありません。しかし、現行以外のインスタンスでデータベース・マネージャーの構成操作を実行する場合には、インスタンス接続が必要になります。現行以外のインスタンスのデータベース・マネージャー構成をリセットするには、まず最初にそのインスタンスに接続することが必要です。

API 組み込みファイル

sqlutil.h

C API 構文

```
/* File: sqlutil.h */
/* API: Reset Database Manager Configuration */
/* ... */
SQL_API_RC SQL_API_FN
  sqlfrsys (
    struct sqlca * pSqlca);
/* ... */
```

汎用 API 構文

```
/* File: sqlutil.h */
/* API: Reset Database Manager Configuration */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgrsys (
    struct sqlca * pSqlca);
/* ... */
```

API パラメーター

pSqlca
出力。 *sqlca* 構造を指すポインターです。 この構造に関する詳細は、520ページの『SQLCA』を参照してください。

REXX API 構文

```
RESET DATABASE MANAGER CONFIGURATION
```

サンプル・プログラム

C	¥sqllib¥samples¥c¥dbmconf.c
COBOL	¥sqllib¥samples¥cobol¥dbmconf.cbl
REXX	¥sqllib¥samples¥rexx¥dbmconf.cmd

使用上の注意

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) への接続が存在する場合、それらのインスタンスに接続されたサーバーのデータベース・マネージャー構成パラメーターがリセットされます。そのようなインスタンスが存在しない場合には、ローカルのデータベース・マネージャー構成パラメーターがリセットされます。

この API は、構成全体 (更新不能パラメーターを除く) をリセットします。

現行のデータベース・マネージャー構成パラメーターのリストを表示または出力するには、 331ページの『sqlfxsys - データベース・マネージャー構成の入手』を使用してください。

データベース・マネージャー構成パラメーターの省略時値を表示するには、310ページの『sqlfdsys - データベース・マネージャー構成の省略時値の入手』を使用してください。

構成可能なパラメーターの値を変更するには、 323ページの『sqlfusys - データベース・マネージャー構成の更新』を使用してください。

データベース・マネージャー構成ファイルに加えられたほとんどの変更は、変更がメモリーにロードされてはじめて有効になります。サーバーの構成パラメ

ーターの場合、このことは **db2start** の実行中に起こります。クライアントの構成パラメーターの場合、このことはアプリケーションが再始動されるときに起こります。

エラーが発生した場合、データベース・マネージャー構成ファイルに変更は加えられません。

チェックサムが無効の場合は、データベース・マネージャー構成ファイルをリセットすることができません。これは、正しい API を使用せずにデータベース・マネージャー構成ファイルを変更した場合に起こる可能性があります。その場合、データベース・マネージャーを再インストールして、データベース・マネージャー構成ファイルをリセットする必要が生じます。

データベース・マネージャー構成パラメーターの要旨については、*コマンド解説書* を参照してください。これらのパラメーターに関する詳細については、*管理の手引き* を参照してください。

参照資料

310ページの『sqlfdsys - データベース・マネージャー構成の省略時値の入手』

323ページの『sqlfusys - データベース・マネージャー構成の更新』

331ページの『sqlfxsys - データベース・マネージャー構成の入手』

sqlfudb - データベース構成の更新

特定のデータベース構成ファイルにある個々の項目を修正します。

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。

効力範囲

この API は、それが発行されたノードにのみ影響を与えます。

許可

以下のいずれかです。

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

必須接続

インスタンス。明示的な接続は必要ありません。データベースがリモートとしてリストされている場合には、呼び出しの期間中にリモート・ノードへのインスタンス接続が確立されます。

API 組み込みファイル

sqlutil.h

C API 構文

```
/* File: sqlutil.h */
/* API: Update Database Configuration */
/* ... */
SQL_API_RC SQL_API_FN
  sqlfudb (
    _SQLOLDCHAR * pDbAlias,
    unsigned short NumItems,
    struct sqlfupd * pItemList,
    struct sqlca * pSqlca);
/* ... */
```

汎用 API 構文

```
/* File: sqlutil.h */
/* API: Update Database Configuration */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgudb (
    unsigned short DbAliasLen,
    unsigned short NumItems,
    unsigned short * pItemListLens,
    struct sqlfupd * pItemList,
    struct sqlca * pSqlca,
    char * pDbAlias);
/* ... */
```

API パラメーター

DbAliasLen

入力。データベース別名の長さを示す 2 バイトの無符号整数 (バイト単位) です。

NumItems

入力。修正される項目の数を指定します。最小有効値は 1 です。

pItemListLens

入力。 *pItemList* に指定された新しい構成フィールド値のそれぞれの長さを示す 2 バイトの無符号整数の配列を示します。 *newlogpath* などのように、ストリングだけを含むフィールドには、長さを指定する必要があります。たとえば、 *newlogpath* が *pItemList* 配列内で 5 番目の要素である場合、その長さは *pItemListLens* 配列でも 5 番目の要素でなければなりません。

pItemList

入力。 *NumItems* 個の *sqlfupd* 構造の配列を指すポインターです。その構造のそれぞれには、更新される値を示すトークン・フィールド、および新規の値を示すポインター・フィールドが含まれています。この構造に関する詳細については、572ページの『SQLFUPD』を参照してください。

pSqlca

出力。 *sqlca* 構造を指すポインターです。 この構造に関する詳細は、520ページの『SQLCA』を参照してください。

pDbAlias

入力。データベース別名を含むストリングを指定します。

REXX API 構文

```
UPDATE DATABASE CONFIGURATION FOR dbname USING :values
```

REXX API パラメーター

dbname
構成ファイルに関連したデータベースの別名。

values
どの構成フィールドを修正するかを示すトークンを含む複合 REXX ホスト変数。トークンおよび新規の値は、アプリケーションによってフィールドごとに提供されます。以下に示すのは、変数の要素です。 XXX はホスト変数名を表しています。

XXX.0 提供されたフィールド数の 2 倍 (残りの変数内のデータ要素の数)。

XXX.1 最初のトークン。

XXX.2 最初のフィールドに提供された値。

XXX.3 2 番目のトークン。

XXX.4 2 番目のフィールドに提供された値。

XXX.5 以降、3 番目、4 番目 ... と続きます。

サンプル・プログラム

C	¥sqllib¥samples¥c¥dbconf.c
COBOL	¥sqllib¥samples¥cobol¥dbconf.cbl
REXX	¥sqllib¥samples¥rexx¥dbconf.cmd

使用上の注意

データベース構成パラメーターのリストを表示または出力するには、 327ページの『sqlfxdb - データベース構成の入手』を使用してください。

データベース構成パラメーターの省略時値を表示するには、 307ページの『sqlfddb - データベース構成の省略時値の入手』を使用してください。

データベース構成パラメーターを省略時値にリセットするには、 313ページの『sqlfrdb - データベース構成のリセット』を使用してください。

これらのパラメーターの省略時値は、構成されたデータベース・ノードのタイプ (サーバー、クライアント、またはリモート・クライアントを持つサーバー) ごとに異なる場合があります。それぞれのノード・タイプで設定できる範囲と省略時値については、 *管理の手引き* を参照してください。各構成項目ごとの有効な *token* 値が、 572ページの表53 にリストされています。

すべてのパラメーターが更新可能なわけではありません。

データベース構成ファイルに加えられたほとんどの変更は、変更がメモリーにロードされてはじめて有効になります。ファイルがメモリーにロードされる前に、すべてのアプリケーションがデータベースから切断されていなければなりません。

エラーが発生した場合、データベース構成ファイルは変更されません。

チェックサムが無効な場合、データベース構成ファイルは更新できません。これは、正しい API を使用せずにデータベース構成ファイルを変更した場合に起こる可能性があります。その場合、データベースを復元して、データベース構成ファイルをリセットする必要が生じます。

データベース構成パラメーターの要旨については、 *コマンド解説書* を参照してください。これらのパラメーターに関する詳細については、 *管理の手引き* を参照してください。

参照資料

307ページの『sqlfddb - データベース構成の省略時値の入手』

313ページの『sqlfrdb - データベース構成のリセット』

327ページの『sqlfxdb - データベース構成の入手』

sqlfusys - データベース・マネージャー構成の更新

データベース・マネージャー構成ファイル内にある個々の項目を修正します。

許可

sysadm

必須接続

なし、またはインスタンス。現行インスタンス (**DB2INSTANCE** 環境変数の値で定義された) でデータベース・マネージャーの構成操作を実行する場合、インスタンス接続は必要ありません。しかし、現行以外のインスタンスでデータベース・マネージャーの構成操作を実行する場合には、インスタンス接続が必要になります。現行以外のインスタンスのデータベース・マネージャー構成を更新するには、まず最初にそのインスタンスに接続することが必要です。

API 組み込みファイル

sqlutil.h

C API 構文

```
/* File: sqlutil.h */
/* API: Update Database Manager Configuration */
/* ... */
SQL_API_RC SQL_API_FN
  sqlfusys (
    unsigned short NumItems,
    struct sqlfupd * pItemList,
    struct sqlca * pSqlca);
/* ... */
```

汎用 API 構文

```
/* File: sqlutil.h */
/* API: Update Database Manager Configuration */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgusys (
    unsigned short NumItems,
    unsigned short * pItemListLens,
    struct sqlfupd * pItemList,
    struct sqlca * pSqlca);
/* ... */
```

API パラメーター

NumItems
入力。修正される項目の数を指定します。最小有効値は 1 です。

pItemListLens
入力。 *pItemList* に指定された新しい構成フィールド値のそれぞれの長さを示す 2 バイトの無符号整数の配列を示します。 *dftdbpath* などのように、ストリングだけを含むフィールドには、長さを指定することが必要です。たとえば、*dftdbpath* が *pItemList* 配列内で 5 番目の要素である場合、その長さは *pItemListLens* 配列内でも 5 番目の要素でなければなりません。

pItemList
入力。 *NumItems* 個の *sqlfupd* 構造の配列を指すポインターです。その構造のそれぞれには、更新される値を示すトークン・フィールド、および新規の値を示すポインター・フィールドが含まれています。この構造に関する詳細については、572ページの『SQLFUPD』を参照してください。

pSqlca
出力。 *sqlca* 構造を指すポインターです。 この構造に関する詳細は、520ページの『SQLCA』を参照してください。

REXX API 構文

```
UPDATE DATABASE MANAGER CONFIGURATION USING :values
```

REXX API パラメーター

values

修正される構成フィールドを示すトークンを含む複合 REXX ホスト変数。トークンおよび新規の値は、アプリケーションによってフィールドごとに提供されます。以下に示すのは、変数の要素です。 XXX はホスト変数名を表しています。

XXX.0 変数内の要素の数。この値は、修正するフィールドの数の 2 倍です。

XXX.1 最初のトークン。

XXX.2 最初のフィールドの新規の値。

XXX.3 2 番目のトークン。

XXX.4 2 番目のフィールドの新規の値。

XXX.5 以降、3 番目、4 番目 ... と続きます。

サンプル・プログラム

C	¥sqllib¥samples¥c¥dbmconf.c
COBOL	¥sqllib¥samples¥cobol¥dbmconf.cbl
REXX	¥sqllib¥samples¥rexx¥dbmconf.cmd

使用上の注意

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) への接続が存在する場合、それらのインスタンスに接続されたサーバーのデータベース・マネージャー構成パラメーターが戻されます。そのようなインスタンスが存在しない場合には、ローカルのデータベース・マネージャー構成パラメーターが更新されます。

データベース・マネージャー構成パラメーターのリストを表示または出力するには、 331ページの『sqlfxsys - データベース・マネージャー構成の入手』を使用してください。

データベース・マネージャー構成パラメーターを推奨されているデータベース・マネージャーの省略時値にリセットするには、 316ページの『sqlfrsys - データベース・マネージャー構成のリセット』を使用してください。

これらのパラメーターの省略時値は、構成されたデータベース・ノードのタイプ (サーバー、クライアント、またはリモート・クライアントを持つサーバー) ごとに異なる場合があります。それぞれのノード・タイプで設定できる範囲と

省略時値については、 *管理の手引き* を参照してください。各構成項目ごとの有効な *token* 値が、 575ページの表55 にリストされています。

すべてのパラメーターが更新可能なわけではありません。

データベース・マネージャー構成ファイルに加えられたほとんどの変更は、変更がメモリーにロードされてはじめて有効になります。サーバーの構成パラメーターの場合、このことは **db2start** の実行中に起こります。クライアントの構成パラメーターの場合、このことはアプリケーションが再始動されるときに起こります。

エラーが発生した場合、データベース・マネージャー構成ファイルに変更は加えられません。

チェックサムが無効の場合は、データベース・マネージャー構成ファイルを更新することができません。これは、正しい API を使用せずにデータベース・マネージャー構成ファイルを変更した場合に起こる可能性があります。その場合、データベース・マネージャーを再インストールして、データベース・マネージャー構成ファイルをリセットする必要が生じます。

データベース・マネージャー構成パラメーターの要旨については、*コマンド解説書* を参照してください。これらのパラメーターに関する詳細については、*管理の手引き* を参照してください。

参照資料

310ページの『sqlfdsys - データベース・マネージャー構成の省略時値の入手』

316ページの『sqlfrsys - データベース・マネージャー構成のリセット』

331ページの『sqlfxsys - データベース・マネージャー構成の入手』

sqlfxdb - データベース構成の入手

データベース構成ファイル内にある個々の項目の値を戻します。

データベース構成パラメーターの要旨については、 *コマンド解説書* を参照してください。これらのパラメーターに関する詳細については、 *管理の手引き* を参照してください。

効力範囲

この API は、それが呼び出されるノードについての情報のみを戻します。

許可

ありません

必須接続

インスタンス。リモート・データベースの構成を設定する場合でも、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 組み込みファイル

sqlutil.h

C API 構文

```
/* File: sqlutil.h */
/* API: Get Database Configuration */
/* ... */
SQL_API_RC SQL_API_FN
  sqlfxdb (
    _SQLOLDCHAR * pDbAlias,
    unsigned short NumItems,
    struct sqlfupd * pItemList,
    struct sqlca * pSqlca);
/* ... */
```

汎用 API 構文

```
/* File: sqlutil.h */
/* API: Get Database Configuration */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgxdb (
    unsigned short DbAliasLen,
    unsigned short NumItems,
    struct sqlfupd * pItemList,
    struct sqlca * pSqlca,
    char * pDbAlias);
/* ... */
```

API パラメーター

DbAliasLen
入力。データベース別名の長さを示す 2 バイトの無符号整数 (バイト単位) です。

NumItems
入力。戻される項目の数を示します。最小有効値は 1 です。

pItemList
入力 / 出力。 *NumItem* 個の *sqlfupd* 構造の配列を指すポインターです。その構造のそれぞれには、戻される値を示すトークン・フィールド、および構成値を位置づける箇所を示すポインター・フィールドが含まれています。この構造に関する詳細については、572ページの『SQLFUPD』を参照してください。

pSqlca
出力。 *sqlca* 構造を指すポインターです。 この構造に関する詳細は、520ページの『SQLCA』を参照してください。

pDbAlias
入力。データベース別名を含むストリングを指定します。

REXX API 構文

```
GET DATABASE CONFIGURATION FOR database_alias USING :values
```

REXX API パラメーター

database_alias
特定のデータベース構成ファイルに関連したデータベースの別名。

values
戻される構成フィールドを示すトークンを含む複合 REXX ホスト変数。トークンはアプリケーションによって提供され、API が値を戻します。以下に示すのは、変数の要素です。 XXX はホスト変数名を表しています。

XXX.0	戻されたフィールド数の 2 倍 (残りの変数内のデータ要素の数)。
XXX.1	最初のトークン。
XXX.2	最初のフィールドに戻された値。
XXX.3	2 番目のトークン。
XXX.4	2 番目のフィールドに戻された値。
XXX.5	以降、3 番目、4 番目 ... と続きます。

サンプル・プログラム

C	¥sqllib¥samples¥c¥dbconf.c
COBOL	¥sqllib¥samples¥cobol¥dbconf.cbl
REXX	¥sqllib¥samples¥rexx¥dbconf.cmd

使用上の注意

pItemList のトークン値のリストに示されていないデータベース構成ファイル内の項目は、アプリケーションにはアクセス不能です。

アプリケーションは、戻されるデータ要素のそれぞれに十分なメモリーを割り振る必要があります。たとえば、*newlogpath* に戻される値は、最大 242 バイトの長さになります。

エラーが生じた場合には、戻された情報は無効になります。構成ファイルが無効な場合には、エラー・メッセージが戻されます。その場合には、データベースをバックアップ版から復元しなければなりません。

データベース構成パラメーターをデータベース・マネージャーの省略時値に設定するには、 313ページの『sqlfrdb - データベース構成のリセット』を使用してください。

これらのパラメーターに関する詳細については、 *管理の手引き* を参照してください。

参照資料

307ページの『sqlfddb - データベース構成の省略時値の入手』

313ページの『sqlfrdb - データベース構成のリセット』

319ページの『sqlfudb - データベース構成の更新』

sqlfxsys - データベース・マネージャー構成の入手

データベース・マネージャー構成ファイル内にある個々の項目の値を戻します。

データベース・マネージャー構成パラメーターの要旨については、*コマンド解説書* を参照してください。これらのパラメーターに関する詳細については、*管理の手引き* を参照してください。

許可

ありません

必須接続

現行インスタンス (**DB2INSTANCE** 環境変数の値で定義された) でデータベース・マネージャーの構成操作を実行する場合、インスタンス接続は必要ありません。しかし、現行以外のインスタンスでデータベース・マネージャーの構成操作を実行する場合には、インスタンス接続が必要になります。現行以外のインスタンスのデータベース・マネージャー構成を表示するには、まず最初にそのインスタンスに接続することが必要です。

API 組み込みファイル

sqlutil.h

C API 構文

```
/* File: sqlutil.h */
/* API: Get Database Manager Configuration */
/* ... */
SQL_API_RC SQL_API_FN
  sqlfxsys (
    unsigned short NumItems,
    struct sqlfupd * pItemList,
    struct sqlca * pSqlca);
/* ... */
```

汎用 API 構文

```
/* File: sqlutil.h */
/* API: Get Database Manager Configuration */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgxsys (
    unsigned short NumItems,
    struct sqlfupd * pItemList,
    struct sqlca * pSqlca);
/* ... */
```

API パラメーター

NumItems
入力。修正される項目の数を指定します。最小有効値は 1 です。

pItemList
入力 / 出力。 *NumItems* 個の *sqlfupd* 構造の配列を指すポインターです。構造のそれぞれには、戻される値を示すトークン・フィールド、および構成値を位置づける箇所を示すポインター・フィールドが含まれています。この構造に関する詳細については、572ページの『SQLFUPD』を参照してください。

pSqlca
出力。 *sqlca* 構造を指すポインターです。 この構造に関する詳細は、520ページの『SQLCA』を参照してください。

REXX API 構文

```
GET DATABASE MANAGER CONFIGURATION USING :values
```

REXX API パラメーター

values
戻される構成フィールドを示すトークンを含む複合ホスト変数。トークンはアプリケーションによって提供され、 API が値を戻します。
XXX はホスト変数名を表しています。

XXX.0 残りの変数内のデータ要素の実数。

XXX.1 最初のトークン。

XXX.2 最初のフィールドに戻された値。

XXX.3 2 番目のトークン。

XXX.4 2 番目のフィールドに戻された値。

XXX.5 以降、3 番目、4 番目 ... と続きます。

サンプル・プログラム

C ¥sqllib¥samples¥c¥dbmconf.c

COBOL ¥sqllib¥samples¥cobol¥dbmconf.cbl

REXX ¥sqllib¥samples¥rexx¥dbmconf.cmd

使用上の注意

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) への接続が存在する場合、それらのインスタンスに接続されたサーバーのデータベース・マネージャー構成パラメーターが戻されます。そのようなインスタンスが存在しない場合には、ローカルのデータベース・マネージャー構成パラメーターが戻されます。

アプリケーションは、戻されるデータ要素のそれぞれに十分なメモリーを割り振る必要があります。たとえば、*dftdbpath* に戻される値は、最大 215 バイトの長さになります。

エラーが生じた場合には、戻された情報は無効になります。構成ファイルが無効な場合には、エラー・メッセージが戻されます。そのような場合には、データベース・マネージャーを再インストールして回復を実行する必要があります。

構成パラメーターをデータベース・マネージャーによって出荷された省略時値に設定するには、 316ページの『sqlfrsys - データベース・マネージャー構成のリセット』を使用してください。

これらのパラメーターに関する詳細については、 *管理の手引き* を参照してください。

参照資料

310ページの『sqlfdsys - データベース・マネージャー構成の省略時値の入手』

316ページの『sqlfrsys - データベース・マネージャー構成のリセット』

323ページの『sqlfusys - データベース・マネージャー構成の更新』

sqlgaddr - アドレスの入手

ある変数のアドレスを別の変数の中に入れます。 FORTRAN や COBOL など、ポインター操作を提供しないホスト言語で使用されます。

許可

ありません

必須接続

ありません

API 組み込みファイル

sqlutil.h

汎用 API 構文

```
/* File: sqlutil.h */
/* API: Get Address */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgaddr (
    char * pVariable,
    char ** ppOutputAddress);
/* ... */
```

API パラメーター

pVariable
入力。アドレスが戻される変数を示します。

ppOutputAddress
出力。変数アドレスが戻される 4 バイトの領域を示します。

使用上の注意

この API を使用するのは COBOL および FORTRAN 言語だけです。

参照資料

335ページの『sqlgdref - アドレスの参照解除』

sqlgdref - アドレスの参照解除

ポインターによって定義されるバッファーからのデータを、アプリケーションが直接アクセスできる変数にコピーします。 FORTRAN や COBOL など、ポインター操作を提供しないホスト言語で使用されます。この API を使用して、必要なデータを指すポインターを戻す API (たとえば、 267ページの『sqlengne - ノード・ディレクトリー次項目の入手』など) からの結果を取得することができます。

許可

ありません

必須接続

ありません

API 組み込みファイル

sqlutil.h

汎用 API 構文

```
/* File: sqlutil.h */
/* API: Dereference Address */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgdref (
    unsigned int NumBytes,
    char * pTargetBuffer,
    char ** ppSourceBuffer);
/* ... */
```

API パラメーター

NumBytes
入力。転送されるバイト数を示す整数です。

pTargetBuffer
出力。データの移動先の領域を示します。

ppSourceBuffer
入力。対象データを含む領域を指すポインターです。

使用上の注意

この API を使用するのは COBOL および FORTRAN 言語だけです。

参照資料

334ページの『sqlgaddr - アドレスの入手』

sqlgmcpy - メモリーのコピー

あるメモリーの領域から別のメモリーの領域へデータをコピーします。
FORTRAN や COBOL など、メモリー・ブロックのコピー機能を提供しないホスト言語で使用されます。

許可

ありません

必須接続

ありません

API 組み込みファイル

sqlutil.h

汎用 API 構文

```
/* File: sqlutil.h */
/* API: Copy Memory */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgmcpy (
    void * pTargetBuffer,
    const void * pSource,
    sqluint32 NumBytes);
/* ... */
```

API パラメーター

pTargetBuffer
出力。データの移動先の領域を示します。

pSource
入力。データの移動元の領域を示します。

NumBytes
入力。転送されるバイト数を示す 4 バイトの無符号整数です。

使用上の注意

この API を使用するのは COBOL および FORTRAN 言語だけです。

参照資料

334ページの『sqlgaddr - アドレスの入手』

sqlogstt - SQLSTATE メッセージの入手

SQLSTATE に関連したメッセージ・テキストを検索します。

許可

ありません

必須接続

ありません

API 組み込みファイル

sql.h

C API 構文

```
/* File: sql.h */
/* API: Get SQLSTATE Message */
/* ... */
SQL_API_RC SQL_API_FN
  sqlogstt (
    char * pBuffer,
    short BufferSize,
    short LineWidth,
    char * pSqlstate);
/* ... */
```

汎用 API 構文

```
/* File: sql.h */
/* API: Get SQLSTATE Message */
/* ... */
SQL_API_RC SQL_API_FN
  sqlggstt (
    short BufferSize,
    short LineWidth,
    char * pSqlstate,
    char * pBuffer);
/* ... */
```

API パラメーター

BufferSize
入力。検索したメッセージ・テキストを保留するストリング・バッファーのサイズ (バイト単位) です。

LineWidth
入力。メッセージ・テキストの各行ごとの最大行幅を示します。各行は、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、このフィールドはヌル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、必ずヌル文字で終了しなければなりません。

pBuffer
出力。メッセージ・テキストが配置されるストリング・バッファーを指すポインターです。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、切り捨てはヌル・ストリング終了文字を見込んで行われます。

REXX API 構文

```
GET MESSAGE FOR SQLSTATE sqlstate INTO :msg [LINEWIDTH width]
```

REXX API パラメーター

sqlstate
メッセージ・テキストが検索される SQLSTATE。

msg メッセージが入れられる REXX 変数。

width メッセージ・テキストの各行の最大行幅。行は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。

サンプル・プログラム

C ¥sqllib¥samples¥c¥utilapi.c

COBOL ¥sqllib¥samples¥cobol¥checkerr.cbl

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 形式化メッセージのバイト数を示す正の整数です。呼び出し側が入力したバッファー・サイズよりもこの数の方が大きい場合、メッセージは切り捨てられます。

-1 メッセージ書式化サービスを機能させるには、利用可能なメモリーが不十分です。要求されたメッセージは、戻されません。

-2 SQLSTATE の形式が間違っています。この形式は英数字でなければならず、長さは 2 桁あるいは 5 桁のどちらかです。

-3 メッセージ・ファイルがアクセス不能または正しくありません。

-4 行幅が 0 未満です。

-5 無効 *sqlca*、不良バッファー・アドレス、または不良バッファー長を示します。

戻りコードが -1 または -3 の場合、メッセージ・バッファーには、問題に関するより詳細な情報が含まれています。

参照資料

103ページの『sqlaintp - エラー・メッセージの入手』

sqluadau - 許可の入手

データベース・マネージャー構成ファイルおよび許可システム・カタログ・ビュー (SYSCAT.DBAUTH) の中の値から、現行ユーザーの権限を報告します。

許可

ありません

必須接続

データベース

API 組み込みファイル

sqlutil.h

C API 構文

```
/* File: sqlutil.h */
/* API: Get Authorizations */
/* ... */
SQL_API_RC SQL_API_FN
  sqluadau (
    struct sql_authorizations * pAuthorizations,
    struct sqlca * pSqlca);
/* ... */
```

汎用 API 構文

```
/* File: sqlutil.h */
/* API: Get Authorizations */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgadau (
    struct sql_authorizations * pAuthorizations,
    struct sqlca * pSqlca);
/* ... */
```

API パラメーター

pAuthorizations

入力 / 出力。 *sql_authorizations* 構造を指すポインターです。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* の最初の要素は、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。*sql_authorizations* 構造に関する詳細については、 504ページの『SQL-AUTHORIZATIONS』を参照してください。

pSqlca
出力。 *sqlca* 構造を指すポインターです。 この構造に関する詳細は、520ページの『SQLCA』を参照してください。

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複合 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	変数内の要素の数 (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

サンプル・プログラム

C	¥sqllib¥samples¥c¥dbauth.sqc
COBOL	¥sqllib¥samples¥cobol¥dbauth.sqb
REXX	¥sqllib¥samples¥rexx¥dbauth.cmd

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* 構造の各要素には `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。

sqlubkp - データベースのバックアップ

データベースまたは表スペースのバックアップ・コピーを作成します。

効力範囲

この API は、それが実行されるノードにのみ影響を与えます。

許可

以下のいずれかです。

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

必須接続

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。

注: 指定したデータベースへの接続がすでに確立されている場合は、その接続がバックアップ操作に使用されます。その接続は、バックアップの完了時に終了します。

API 組み込みファイル

sqlutil.h

C API 構文

```
/* File: sqlutil.h */
/* API: Backup Database */
/* ... */
SQL_API_RC SQL_API_FN
  sqlubkp (
    char * pDbAlias,
    sqluint32     BufferSize,
    sqluint32     BackupMode,
    sqluint32     BackupType,
    sqluint32     CallerAction,
    char * pApplicationId,
    char * pTimestamp,
    sqluint32     NumBuffers,
    struct sqlu_tablespace_bkrst_list * pTablespaceList,
    struct sqlu_media_list * pMediaTargetList,
    char * pUserName,
    char * pPassword,
    void * pReserved2,
    sqluint32 VendorOptionsSize,
```

```
    void * pVendorOptions,
    sqluint32 Parallelism,
    sqluint32 * pBackupSize,
    void * pReserved4,
    void * pReserved3,
    struct sqlca * pSqlca);
/* ... */
```

汎用 API 構文

```
/* File: sqlutil.h */
/* API: Backup Database */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgbkp (
    unsigned short DbAliasLen,
    unsigned short UserNameLen,
    unsigned short PasswordLen,
    unsigned short * pReserved1,
    char * pDbAlias,
    sqluint32 BufferSize,
    sqluint32 BackupMode,
    sqluint32 BackupType,
    sqluint32 CallerAction,
    char * pApplicationId,
    char * pTimestamp,
    sqluint32 NumBuffers,
    struct sqlu_tablespace_bkrst_list * pTablespaceList,
    struct sqlu_media_list * pMediaTargetList,
    char * pUserName,
    char * pPassword,
    void * pReserved2,
    sqluint32 VendorOptionsSize,
    void * pVendorOptions,
    sqluint32 Parallelism,
    sqluint32 * pBackupSize,
    void * pReserved4,
    void * pReserved3,
    struct sqlca * pSqlca);
/* ... */
```

API パラメーター

DbAliasLen

入力。データベース別名の長さを示す 2 バイトの無符号整数 (バイト単位) です。

UserNameLen

入力。ユーザー名の長さを示す 2 バイトの無符号整数 (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 バイトの無符号整数 (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。

pReserved1
将来の使用のために予約されています。

pDbAlias
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。

BufferSize
入力。バッファー・サイズを 4KB 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。省略時値は 1024 単位です。

BackupMode
入力。バックアップ・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。

SQLUB_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。

SQLUB_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: DB2 バックアップ・ユーティリティーでは、 SMS LOB オブジェクトに S ロックを掛け、それ以外のすべてのオブジェクトに IN ロックを掛けるため、sysibm.systables に IX ロックが掛かっていると、オンライン・バックアップ操作がタイムアウトになる場合があります。

BackupType
入力。実行するバックアップのタイプを指定します。有効な値 (sqlutil で定義) は、以下のとおりです。

SQLUB_FULL
データベースの全バックアップ。

SQLUB_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、 *pTablespaceList* パラメーターに表スペースのリストを提供してください。

CallerAction

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

SQLUB_BACKUP

バックアップを開始します。

SQLUB_NOINTERRUPT

バックアップを開始します。バックアップを自動実行するよう指定します。通常ユーザーの介入を要求するシナリオは、呼び出し側への最初の戻りなしに試行されるか、エラーを生成します。この呼び出し側アクションは、たとえば、バックアップに必要な媒体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。

SQLUB_CONTINUE

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

SQLUB_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。

SQLUB_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。特定の媒体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。

SQLUB_PARM_CHECK

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。

SQLUB_PARM_CHECK_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。

pApplicationId

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

pTimestamp

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

NumBuffers

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。

pTablespaceList

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。
598ページの『SQLU-TABLESPACE-BKRST-LIST』を参照してください。

pMediaTargetList

入力。この構造を使用することにより、呼び出し側はバックアップ操作の宛先を指定することができます。提供される情報は、*media_type* フィールドの値によって異なります。 *media_type* についての有効な値 (sqlutil で定義) は、以下のとおりです。

SQLU_LOCAL_MEDIA

ローカル装置 (テープ、ディスク、またはディスケットの組み合わせが可能です)。 *sqlu_media_entry* 構造のリストを提供してください。 OS/2 または Windows オペレーティング・システムでは、項目をテープ装置名ではなく、ディレクトリー・パスだけにすることができます。

SQLU_TSM_MEDIA

TSM。 *sqlu_media_entry* 構造によってバックアップ・イメージのパスを指定していない場合は、 *sqlu_media_list_targets* 構造の *media* ポインターをヌルに初期設定してください。 DB2 に付属の TSM 共用ライブラリーが使用されます。別のバージョ

ンの TSM 共用ライブラリーが必要な場合には、SQLU_OTHER_MEDIA を使用し、共用ライブラリー名を入力してください。

SQLU_OTHER_MEDIA
ベンダー製品。 *sqlu_vendor* 構造に共用ライブラリー名を提供します。

SQLU_USER_EXIT
ユーザー出口。追加の入力は必要ありません (OS/2 でのみ使用可能です)。

詳細については、 592ページの『SQLU-MEDIA-LIST』、および*管理の手引き* を参照してください。

pUserName
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。

pPassword
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。

pReserved2
将来の使用のために予約されています。

VendorOptionsSize
入力。 *pVendorOptions* 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。

pVendorOptions
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、バイト逆転が行われず、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。

Parallelism
入力。並列処理度 (バッファー・マニピュレーターの数) を指定します。

pBackupSize
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。ヌルに設定することができます。

pReserved4
将来の使用のために予約されています。

pReserved3
将来の使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です。 この構造に関する詳細は、520ページの『SQLCA』を参照してください。

REXX API 構文

```
BACKUP DATABASE dbalias USING :value [USER username USING password]
[TABLESPACE :tablespacenames] [ONLINE]
[LOAD vendor-library [OPTIONS vendor-options] [OPEN num-sessions SESSIONS] |
 TO :target-area |
 USE TSM [OPEN num-sessions SESSIONS] |
 USER_EXIT]
[ACTION caller-action] [WITH num-buffers BUFFERS] [BUFFERSIZE buffer-size]
[PARALLELISM parallelism-degree]
```

REXX API パラメーター

dbalias
バックアップされるデータベースの別名。

value データベースのバックアップ情報が戻される複合 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 変数内の要素の数 (常に 2)。

XXX.1 バックアップ・イメージのタイム・スタンプ。

XXX.2 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アプリケーション ID。

username
データベースのバックアップに使用されるユーザー名を識別します。

password
ユーザー名の認証に使用されるパスワード。

tablespacenames
バックアップされる表スペースのリストを含む複合 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 バックアップされる表スペースの数。

XXX.1 最初の表スペース名。

XXX.2 2 番目の表スペース名。

XXX.3 以降、3 番目、4 番目 ... と続きます。

vendor-library
使用されるベンダーの入出力バックアップ / 復元関数を含む共用ライブラリーの名前 (OS/2 または Windows オペレーティング・システムでは DLL)。これには、全パスを入れてもかまいません。全パスが指定されない場合は、省略時値として、ユーザー出口プログラムが存在するパスが使用されます。

vendor-options
ベンダー関数によって必要とされる情報。

num-sessions
TSM またはベンダー製品と共に使用する入出力セッションの数。

target-area
ローカル装置。テープ、ディスク、またはディスケットの組み合わせが可能です。 592ページの『SQLU-MEDIA-LIST』のリストを提供してください。 OS/2 または Windows オペレーティング・システムでは、項目をテープ装置名ではなく、ディレクトリー・パスだけにすることができます。

caller-action
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。

SQLUB_BACKUP
バックアップを開始します。

SQLUB_NOINTERRUPT
バックアップを開始します。バックアップを不在時実行するよう指定します。通常ユーザーの介入を要求するシナリオは、呼び出し側への最初の戻りなしに試行されるか、エラーを生成します。この呼び出し側アクションは、たとえば、バックアップに必要な媒体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。

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

SQLUB_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。

SQLUB_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。特定の媒体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。

SQLUB_PARM_CHECK
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。

num-buffers
使用されるバックアップ・バッファーの数。

buffer-size
バックアップ・バッファー・サイズ (4KB の割り振り単位)。最小値は 8 単位です。

parallelism-degree
バッファー・マニピュレーターの数。

サンプル・プログラム

C	¥sqllib¥samples¥c¥backrest.c
COBOL	¥sqllib¥samples¥cobol¥backrest.cbl

使用上の注意

データベース・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、オンラインおよびオフラインのバックアップ、バックアップ・ファイル名、およびサポートされる装置に関する詳細については、 *コマンド解説書* を参照してください。

バックアップの概説については、 *管理の手引き* の『データベースの回復』を参照してください。

参照資料

262ページの『sqlemgdb - データベースの移行』

465ページの『sqluroll - データベースのロールフォワード』

447ページの『sqlurestore - データベースの復元』

sqludrdt - ノード・グループの再配分

ノード・グループ内のノードにデータを再配分します。現行のデータ配分 (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では、現行のデータ配分に基づいて、移動する区分を選択します。

この API は、カタログ・ノードからのみ呼び出すことができます。どのノードが各データベースごとのカタログ・ノードかを判別するには、 LIST DATABASE DIRECTORY コマンド (*コマンド解説書* を参照) を使用してください。

効力範囲

この API は、ノード・グループ内のすべてのノードに影響を与えます。

許可

以下のいずれかです。

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

API 組み込みファイル

sqlutil.h

C API 構文

```
/* File: sqlutil.h */
/* API: Redistribute Nodegroup */
/* ... */
SQL_API_RC SQL_API_FN
  sqludrdt (
    char * pNodeGroupName,
    char * pTargetPMapFileName,
    char * pDataDistFileName,
    SQL_PDB_NODE_TYPE * pAddList,
    unsigned short AddCount,
    SQL_PDB_NODE_TYPE * pDropList,
    unsigned short DropCount,
    unsigned char DataRedistOption,
    struct sqlca * pSqlca);
/* ... */
```

汎用 API 構文

```
/* File: sqlutil.h */
/* API: Redistribute Nodegroup */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgdrdt (
    unsigned short NodeGroupNameLen,
    unsigned short TargetPMapFileNameLen,
    unsigned short DataDistFileNameLen,
    char * pNodeGroupName,
    char * pTargetPMapFileName,
    char * pDataDistFileName,
    SQL_PDB_NODE_TYPE * pAddList,
    unsigned short AddCount,
    SQL_PDB_NODE_TYPE * pDropList,
    unsigned short DropCount,
    unsigned char DataRedistOption,
    struct sqlca * pSqlca);
/* ... */
```

API パラメーター

NodeGroupNameLen
ノード・グループの名前の長さ。

TargetPMapFileNameLen
ターゲット区分化マップ・ファイルの名前の長さ。

DataDistFileNameLen
データ配分ファイルの名前の長さ。

pNodeGroupName
再配分が行われるノード・グループの名前。

pTargetPMapFileName
ターゲット区分化マップが入っているファイルの名前。ファイル名の一部としてディレクトリー・パスを指定しない場合には、現行ディレクトリーが使用されます。このパラメーターは、 *DataRedistOption* 値が T である場合に使用されます。ファイルは文字形式でなければならず、4 096 項目 (複数ノードのノード・グループの場合) または 1 項目 (単一ノードのノード・グループの場合) を含まなければなりません。ファイル内の項目は、ノード番号を示します。項目は自由形式にすることができます。

pDataDistFileName
入力配分情報が入っているファイルの名前。ファイル名の一部としてデ

ィレクトリー・パスを指定しない場合には、現行ディレクトリーが使用されます。このパラメーターは、 *DataRedistOption* 値が U である場合に使用されます。ファイルは文字形式でなければならず、 4 096 個の正の整数項目を含まなければなりません。ファイル内の各項目は、対応する区分の重みを示します。 4 096 個の値の合計は、 4 294 967 295 以下でなければなりません。

pAddList
データの再配分時にノード・グループに追加するノードのリスト。リスト内の項目は、SQL_PDB_NODE_TYPE の形式でなければなりません。

AddCount
ノード・グループに追加するノードの数。

pDropList
データの再配分時にノード・グループから除去するノードのリスト。リスト内の項目は、SQL_PDB_NODE_TYPE の形式でなければなりません。

DropCount
ノード・グループから除去するノードの数。

DataRedistOption
実行されるデータ再配分のタイプを示す 1 つの文字。可能な値は以下のとおりです。

U 均等に配分するためにノード・グループを再配分することを指定します。 *pDataDistFileName* がヌルである場合には、現行のデータ配分が均等である (つまり、各ハッシュ区分が同じ量のデータを表している) ものとみなされます。
pDataDistFileName がヌルでない場合には、このファイル内の値が現行のデータ配分を表しているものとみなされます。
DataRedistOption が U である場合には、 *pTargetPMapFileName* がヌルでなければなりません。

追加リストに指定されたノードがノード・グループに追加され、除去リストに指定されたデータがノード・グループから除去されます。

T *pTargetPMapFileName* を用いてノード・グループを再配分することを指定します。このオプションを使用する場合には、*pDataDistFileName*、*pAddList*、および *pDropList* がヌルでなければならず、 *AddCount* と *DropCount* の両方がゼロでなければなりません。

C 失敗した再配分操作を続行することを指定します。このオプションを使用する場合には、*pTargetPMapFileName*、*pDataDistFileName*、 *pAddList*、および *pDropList* がヌルでなければならず、 *AddCount* と *DropCount* の両方がゼロでなければなりません。

R 失敗した再配分操作をロールバックすることを指定します。このオプションを使用する場合には、*pTargetPMapFileName*、*pDataDistFileName*、 *pAddList*、および *pDropList* がヌルでなければならず、 *AddCount* と *DropCount* の両方がゼロでなければなりません。

pSqlca

出力。 *sqlca* 構造を指すポインターです。 この構造に関する詳細は、520ページの『SQLCA』を参照してください。

REXX API 構文

この API は、SQLDB2 インターフェースを使って、REXX から呼び出すことができます。 13ページの『API の説明の編成』、または *アプリケーション開発の手引き* を参照してください。構文については、*コマンド解説書* を参照してください。

使用上の注意

再分配操作が実行されると、メッセージ・ファイルが次のディレクトリーに書き込まれます。

- サブディレクトリーとファイル名に、*database-name.nodegroup-name.timestamp* という形式を使った、 UNIX ベースのシステムにある `$HOME/sqllib/redist` ディレクトリー。
- サブディレクトリーとファイル名に、*database-name¥first-eight-characters-of-the-nodegroup-name¥date¥time* という形式を使った、 OS/2 または Windows オペレーティング・システムにある `$HOME¥sqllib¥redist¥` ディレクトリー。

タイム・スタンプ値は、API が呼び出された時刻です。

このユーティリティーは、処理中に断続的な COMMIT を実行します。

ノード・グループにノードを追加するには、 ALTER NODEGROUP ステートメントを使用してください。このステートメントでは、ノード・グループに関連する表スペース用のコンテナーを定義することができます。詳細については、*SQL 解説書* を参照してください。

注: *sysadm* または *sysctrl* 権限があるユーザーには、 DB2 パラレル・エディション (AIX 版) バージョン 1 の構文が、 ADD NODE および DROP NODE オプション付きでサポートされます。 ADD NODE オプションを指定すると、コンテナーは、ノード・グループ内のノード番号が最も低い既存のノード上のコンテナーと同様に作成されます。

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。ノード・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。それにより、無効なパッケージに対する最初の SQL 要求の実行での初期遅延がなくなります。再配分メッセージ・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。

さらに、ノード・グループの再配分操作が完了した後で、 477ページの『sqlustat - 統計の実行』を発行して統計を更新することをお勧めします。

DATA CAPTURE CHANGES で定義された複写した表 (複数の場合もある) を含むノード・グループは再配分できません。

ノードグループに、宣言された既存の一時表を含むユーザー一時表スペースがある場合、再配分を行うことはできません。

参照資料

114ページの『sqlarbnd - 再バインド』

sqluexpr - エクスポート

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

許可

以下のいずれかです。

- *sysadm*
- *dbadm*

または、関係するそれぞれの表または視点に対する CONTROL または SELECT 特権

必須接続

データベース。暗黙的な接続が可能である場合には、省略時データベースへの接続が確立されます。

API 組み込みファイル

sqlutil.h

C API 構文

```
/* File: sqlutil.h */
/* API: Export */
/* ... */
SQL_API_RC SQL_API_FN
  sqluexpr (
    char * pDataFileName,
    sqlu_media_list * pLobPathList,
    sqlu_media_list * pLobFileList,
    struct sqldcol * pDataDescriptor,
    struct sqlchar * pActionString,
    char * pFileType,
    struct sqlchar * pFileTypeMod,
    char * pMsgFileName,
    short CallerAction,
    struct sqluexpt_out*  pOutputInfo,
    void * pReserved,
    struct sqlca * pSqlca);
/* ... */
```

汎用 API 構文

```
/* File: sqlutil.h */
/* API: Export */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgexpr (
    unsigned short DataFileNameLen,
    unsigned short FileTypeLen,
    unsigned short MsgFileNameLen,
    char * pDataFileName,
    sqlu_media_list * pLobPathList,
    sqlu_media_list * pLobFileList,
    struct sqldcol * pDataDescriptor,
    struct sqlchar * pActionString,
    char * pFileType,
    struct sqlchar * pFileTypeMod,
    char * pMsgFileName,
    short CallerAction,
    struct sqluexpt_out*  pOutputInfo,
    void * pReserved,
    struct sqlca * pSqlca);
/* ... */
```

API パラメーター

DataFileNameLen
入力。データ・ファイル名の長さを示す 2 バイトの無符号整数 (バイト単位) です。

FileTypeLen
入力。ファイル・タイプの長さを示す 2 バイトの無符号整数 (バイト単位) です。

MsgFileNameLen
入力。メッセージ・ファイル名の長さを示す 2 バイトの無符号整数 (バイト単位) です。

pDataFileName
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。

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

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

詳細については、 592ページの『SQLU-MEDIA-LIST』を参照してください。

pLobFileList

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

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

詳細については、 592ページの『SQLU-MEDIA-LIST』を参照してください。

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

pDataDescriptor

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エクスポート・ユーティリティーがどのように解釈するかが判別されます。このパラメーターに有効な値 (sqlutil で定義) は、以下のとおりです。

SQL_METH_N

名前。出力ファイルで使用する列名を指定します。

SQL_METH_D

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

詳細については、 526ページの『SQLDCOL』を参照してください。

pActionString

入力。有効な動的 SQL SELECT ステートメントを含む *sqlchar* 構造を指すポインターです。この構造には、2 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。

外部ファイルの列 (*pDataDescriptor* からの) 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要素から取られます。

詳細については、 522ページの『SQLCHAR』を参照してください。

注: タイプ表に使用する構文については、 *コマンド解説書* で説明されています。

pFileType

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

SQL_DEL

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

SQL_WSF

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

SQL_IXF

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

pFileTypeMod

入力。 2 バイトの長さフィールドと、 1 つまたは複数の処理オプションを指定する文字の配列を含む *sqldcol* 構造を指すポインターです。このポインターがヌルであるか、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、このアクションは省略時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。

詳細については、 522ページの『SQLCHAR』 およびコマンド*解説書*を参照してください。

pMsgFileName

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

CallerAction

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

SQLU_INITIAL

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

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

SQLU_CONTINUE

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

SQLU_TERMINATE

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

pOutputInfo

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。この構造に関する詳細については、
600ページの『SQLUEXPT-OUT』を参照してください。

pReserved
将来の使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です。 この構造に関する詳細は、520ページの『SQLCA』を参照してください。

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 (*データ移動ユーティリティー手引きおよび解説書* を参照)。

dcoldata
エクスポート・ファイルで使用する列名を含む複合 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 (残りの変数内の要素の数)

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ヌルの場合、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。

サンプル・プログラム

C	¥sqllib¥samples¥c¥impexp.sqc
COBOL	¥sqllib¥samples¥cobol¥impexp.sqb
REXX	¥sqllib¥samples¥rexx¥impexp.cmd

使用上の注意

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

SELECT ステートメントでは表別名を使用できます。

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

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

外部列名配列 *pDataDescriptor* の列数 (*dcolnum*) が SELECT ステートメントによって生成される列数と同じでない場合には、警告メッセージが出されます。この場合、外部ファイルに書き込まれる列数はそれらのうち小さい方の数になります。出力ファイルを生成するために、余分のデータベース列または外部列名が使用されることはありません。

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

PC/IXF インポートは、データベース間でデータを移動する場合に使用します。行区切り文字を含む文字データが区切り付き 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 (たとえば、OS/2 と AIX システム間を移動される) 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。

PC/IXF ファイル形式指定によって、OS/2 (IBM OS/2 拡張サービス、OS/2 拡張版、および DB2 (OS/2 版)) データベースと、DB2 (AIX 版) データベースとの間でデータの移行を、 (1) エクスポート、(2) OS/2 と AIX 間でのファイルの 2 進形式でのコピー、 (3) インポートという手順で行えるようになります。ソース・データベースと宛先データベースの両方が同一のクライアントからアクセス可能な場合には、ファイルのコピーというステップは必要ありません。

DB2 コネクトは、DB2 (OS/390 版)、DB2 (VM および VSE 版)、および DB2 (OS/400 版) などの DRDA サーバーから表をエクスポートするために使用できます。 PC/IXF エクスポートだけがサポートされています。

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

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

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

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

IXF 以外のファイル形式の場合は、階層の走査の方法とエクスポートする副表とが DB2 に知らされるよう、走査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 順序が省略時の順序になります。 OUTER 関数によって指定される省略時順序を使うこともできます。

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

DB2 データ・リンク・マネージャーについての考慮事項

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

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

   これにより、EXPORT の実行時に更新トランザクションが進行しなくなります。
2. EXPORT コマンドを発行する。
3. それぞれのデータ・リンク・サーバーで、 **dlfm_export** ユーティリティーを実行する。 **dlfm_export** ユーティリティーへの入力は制御ファイル名です。これは、エクスポート・ユーティリティーによって生成されます。このようにすると、制御ファイル内でリストされるファイルの tar (または同等の) アーカイブが生成されます。
4. QUIESCE TABLESPACES FOR TABLE tablename RESET コマンドを発行する。

   この操作により表が更新可能になります。

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

EXPORT が正常に実行されると、以下のファイルが生成されます。

- EXPORT コマンドで指定されたエクスポート・データ・ファイル。このファイルの DATALINK 列の値は、 420ページで説明されている形式になります。 DATALINK 列の値が SQL NULL 値の場合、処理は他のデータ型の場合と同様になります。

- 各データ・リンク・サーバーについて生成される制御ファイル *server_name* (Windows NT オペレーティング・システムでは、単一の制御ファイル ctrlfile.lst がすべてのデータ・リンク・サーバーによって使用されます)。これらの制御ファイルは、 <data-file path>/dlfm/YYYYMMDD /HHMMSS ディレクトリーに置かれます (Windows NT オペレーティング・システムでは、 ctrlfile.lst は <data-file path>¥dlfm ¥YYYYMMDD ¥HHMMSS ディレクトリーに置かれます)。 YYYYMMDD は日付 (年月日) を表し、HHMMSS は時刻 (時分秒) を表します。

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

表 8. 有効なファイル・タイプ修飾子 (エクスポート)

修飾子	説明
すべてのファイル形式	
lobsinfile	*lob-path* は、LOB 値を含むファイルへのパスを指定します。
DEL (区切り付き ASCII) ファイル形式	
chardel*x*	*x* は、単一文字のストリング区切り文字です。省略時値は、二重引用符 (") です。指定した文字は、文字ストリングを囲む二重引用符の代わりに使用されます。 [a] 単一引用符 (') も、文字ストリングの区切り文字として指定できます。 `modified by chardel''`

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

datesiso	日付形式。すべての日付データ値を ISO 形式 (*"YYYY-MM-DD"*) でエクスポートします。 [b]
decplusblank	正符号文字。正の 10 進値の接頭部として、正符号 (+) ではなくブランク・スペースを使用します。省略時アクションでは、10 進値に正符号の接頭部が付けられます。

表 8. 有効なファイル・タイプ修飾子 (エクスポート) (続き)

修飾子	説明
decpt*x*	*x* は、小数点文字としてピリオドの代わりに使用される単一の置換文字です。省略時値は、ピリオド (.) です。指定した文字は、小数点文字としてピリオドの代わりに使用されます。 [a]
dldel*x*	*x* は、単一文字の DATALINK 区切り文字です。省略時値は、セミコロン (;) です。指定した文字は、 DATALINK 値のフィールド間区切り文字としてセミコロンの代わりに使用されます。 DATALINK 値には 2 つ以上の副次値を指定できるので、この区切り文字が必要です。 [a] **注:** *x* は、行、列、または文字ストリングの区切り文字とは異なる文字にしてください。
nodoubledel	二重になっている区切り文字の認識を抑止します。詳細については、371ページの『区切り文字についての制限事項』を参照してください。
WSF ファイル形式	
1	Lotus 1-2-3 リリース 1 または Lotus 1-2-3 リリース 1a との互換がある WSF ファイルを作成します。 [c] この値が省略時値です。
2	Lotus Symphony リリース 1.0 との互換がある WSF ファイルを作成します。 [c]
3	Lotus 1-2-3 バージョン 2、または Lotus Symphony リリース 1.1 との互換がある WSF ファイルを作成します。 [c]
4	DBCS 文字を含む WSF ファイルを作成します。

表 8. 有効なファイル・タイプ修飾子 (エクスポート) (続き)

修飾子	説明
注: 1. サポートされていないファイル・タイプを MODIFIED BY オプションで使用しようとしても、エクスポート・ユーティリティーは警告を出しません。この場合、エクスポート操作が失敗し、エラー・コードが戻されます。 2. [a] 『区切り文字についての制限事項』には、代替区切り文字として使用できる文字に適用される制限事項がリストされています。 3. [b] 通常、エクスポート・ユーティリティーでの記述形式は次のとおりです。 • 日付データ: *YYYYMMDD* の形式 • 文字 (日付) データ: *YYYY-MM-DD* の形式 • 時刻データ: *HH.MM.SS* の形式 • タイム・スタンプ・データ: *YYYY-MM-DD-HH.MM.SS.uuuuuu* の形式 エクスポート操作用の SELECT ステートメントで指定されたすべての日時列に入れられるデータも、上記の形式になります。 4. [c] これらのファイルは、 *filetype-mod* パラメーター・ストリングに、 L (Lotus 1-2-3 の場合) または S (Symphony の場合) と指定することにより、特定の製品に送信することもできます。指定できるのは、1 つの値または製品指定子だけです。	

区切り文字についての制限事項

選択した区切り文字が移動するデータの一部となっていないかどうかを確かめるのは、ユーザーの責任です。データの一部になっている場合、予期しないエラーが発生する可能性があります。データの移動時は、以下の制限事項が、列、ストリング、DATALINK、および小数点区切り文字に適用されます。

- 区切り文字は相互に排他的である。
- 区切り文字として 2 進ゼロ、改行文字、ブランク・スペースを使用することはできない。
- 省略時の小数点 (.) をストリング区切り文字として使用することはできない。
- 以下の文字は、 ASCII ファミリーと EBCDIC ファミリーのコード・ページでは異なった仕方で指定されています。
  - シフトイン (0x0F) とシフトアウト (0x0E) 文字を、 EBCDIC MBCS データ・ファイルの区切り文字として使用することはできない。
  - MBCS、EUC、または DBCS コード・ページの区切り文字は、 0x40 より大きくなければならない (EBCDIC MBCS データの省略時小数点 0x4b は例外)。

- ASCII コード・ページまたは EBCDIC MBCS コード・ページでのデータ・ファイルの省略時区切り文字は、以下のとおりです。

  ```
  " (0x22、二重引用符。ストリング区切り文字)
  , (0x2c、コンマ。列区切り文字)
  ```

- EBCDIC SBCS コード・ページでのデータ・ファイルの省略時区切り文字は、以下のとおりです。

  ```
  " (0x7F、二重引用符。ストリング区切り文字)
  , (0x6B、コンマ。列区切り文字)
  ```

- ASCII データ・ファイルの省略時小数点は 0x2e (ピリオド) です。
- EBCDIC データ・ファイルの省略時小数点は 0x4B (ピリオド) です。
- サーバーのコード・ページがクライアントのコード・ページと異なっている場合は、非省略時区切り文字の 16 進表示を指定するようお勧めします。たとえば、次のようにできます。

  ```
  db2 load from ... modified by chardel0x0C coldelX1e ...
  ```

DEL ファイルでの二重になっている文字区切り文字の認識のサポートに関する以下の情報は、エクスポート、インポート、およびロード・ユーティリティーに適用されます。

- 文字区切り文字は、 DEL ファイルにある文字ベースのフィールド内で使用できる。これは、CHAR、VARCHAR、LONG VARCHAR、または CLOB 型のフィールドに適用されます (lobsinfile 指定時は例外)。文字区切り文字で囲まれた文字区切り文字の対が検出されると、データベースにインポートまたはロードされます。たとえば、次のようにできます。

  ```
  "What a ""nice"" day!"
  ```

  これは、次のようにインポートされます。

  ```
  What a "nice" day!
  ```

  エクスポートの場合、この規則は逆に適用されます。たとえば、次のようにできます。

  ```
  I am 6" tall.
  ```

  これは、DEL ファイルに次のようにエクスポートされます。

  ```
  "I am 6"" tall."
  ```

- DBCS 環境では、パイプ (|) 文字区切り文字はサポートされていません。

参照資料

381ページの『sqluimpr - インポート』

408ページの『sqluload - ロード』

sqlugrpn - 行の区分化番号の入手

区分化キー値に基づいて区分番号およびノード番号を戻します。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ノードを判別することができます。

区分化データ構造 (611ページの『SQLUPI』) は、この API への入力となります。この構造は、379ページの『sqlugtpi - 表の区分化情報の入手』によって戻すことができます。別の入力としては、対応する区分化キー値の文字表示があります。出力は、区分化ストラテジーによって生成された区分番号と、区分化マップからの対応するノード番号です。区分化マップ情報が提供されていない場合には、区分番号のみが戻されます。この API は、データ配分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。

効力範囲

この API は、 `db2nodes.cfg` ファイル内の任意のノードから呼び出すことができます。

許可

ありません

API 組み込みファイル

sqlutil.h

C API 構文

```
/* File: sqlutil.h */
/* API: Get Row Partitioning Number */
/* ... */
SQL_API_RC SQL_API_FN
  sqlugrpn (
    unsigned short num_ptrs,
    unsigned char ** ptr_array,
    unsigned short * ptr_lens,
    unsigned short ctrycode,
    unsigned short codepage,
    struct sqlupi * part_info,
    short * part_num,
    SQL_PDB_NODE_TYPE * node_num,
    unsigned short chklvl,
    struct sqlca * sqlca,
    short dataformat,
    void * pReserved1,
    void * pReserved2);
/* ... */
```

汎用 API 構文

```
/* File: sqlutil.h */
/* API: Get Row Partitioning Number */
/* ... */
SQL_API_RC SQL_API_FN
  sqlggrpn (
    unsigned short num_ptrs,
    unsigned char ** ptr_array,
    unsigned short * ptr_lens,
    unsigned short ctrycode,
    unsigned short codepage,
    struct sqlupi * part_info,
    short * part_num,
    SQL_PDB_NODE_TYPE * node_num,
    unsigned short chklvl,
    struct sqlca * sqlca,
    short dataformat,
    void * pReserved1,
    void * pReserved2);
/* ... */
```

API パラメーター

num_ptrs

ptr_array 内のポインターの数。この値は、 *part_info* に指定されるものと同じでなければなりません。つまり、*part_info->sqld* です。

ptr_array

part_info に指定された区分化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。ヌル値が必要とされる場合には、対応するポインターがヌルに設定されます。

ptr_lens

part_info に指定された区分化キーの各パーツに対応する値の文字表示の長さを含むポインターの配列。

ctrycode

ターゲット・データベースの国別コード。有効な国別コード値のリストについては、*概説およびインストール* を参照してください。

この値は、データベース構成ファイルから入手することができます (*コマンド解説書* の GET DATABASE CONFIGURATION コマンドを参照してください)。

codepage

ターゲット・データベースのコード・ページ。有効なコード・ページ値のリストについては、*概説およびインストール* を参照してください。

この値は、データベース構成ファイルから入手することができます (*コマンド解説書* の GET DATABASE CONFIGURATION コマンドを参照してください)。

part_info

sqlupi 構造を指すポインター。この構造の詳細については、611ページの『SQLUPI』を参照してください。

part_num

区分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。ポインターがヌルの場合、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無符号整

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検査されます。

sqlca 出力。 *sqlca* 構造を指すポインターです。この構造の詳細については、520ページの『SQLCA』を参照してください。

dataformat
区分化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
すべての区分化キー値は文字ストリングによって表示されます。これは省略時値です。

SQL_IMPLIEDDECIMAL_FORMAT
暗黙指定されている小数点の位置が列定義によって決定されます。たとえば、列定義が DECIMAL(8,2) である場合、値 12345 は 123.45 として処理されます。

SQL_PACKEDDECIMAL_FORMAT
すべての 10 進数列区分化キー値はパック 10 進数形式になります。

SQL_BINARYNUMERICS_FORMAT
すべての数値区分化キー値はビッグ・エンディアン 2 進数形式になります。

pReserved1
将来の使用のために予約されています。

pReserved2
将来の使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、区分化キーとして定義できるものと同じです。

CHAR、VARCHAR、GRAPHIC、および VARGRAPHIC は、この API を呼び出す前にターゲット・コード・ページに変換しなければなりません。

数値および日時データ・タイプの場合、文字表示は、 API が呼び出されるそれぞれのシステムのコード・ページで表記しなければなりません。

node_num がヌルでない場合には、区分化マップを提供しなければなりません。つまり、 `part_info->pmaplen` を 2 または 8 192 のどちらかにする必要があります。そうでなければ、SQLCODE -6038 が戻されます。

区分化キーを定義しなければなりません。つまり、part_info->sqld をゼロよりも大きくしてください。そうでなければ、SQLCODE -2032 が戻されます。

ヌル値不可の区分化列にヌル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。

参照資料

327ページの『sqlfxdb - データベース構成の入手』

379ページの『sqlugtpi - 表の区分化情報の入手』

355ページの『sqludrdt - ノード・グループの再配分』

sqlugtpi - 表の区分化情報の入手

アプリケーションは、表の区分化情報を入手できます。区分化情報には、区分化マップと、区分化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 374ページの『sqlugrpn - 行の区分化番号の入手』に渡して、表の任意の行の区分番号とノード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、区分化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。

効力範囲

この API は、 db2nodes.cfg ファイルで定義されている任意のノードから実行できます。

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

必須接続

データベース

API 組み込みファイル

sqlutil.h

C API 構文

```
/* File: sqlutil.h */
/* API: Get Table Partitioning Information */
/* ... */
SQL_API_RC SQL_API_FN
  sqlugtpi (
    unsigned char * tablename,
    struct sqlupi * part_info,
    struct sqlca * sqlca);
/* ... */
```

汎用 API 構文

```
/* File: sqlutil.h */
/* API: Get Table Partitioning Information */
/* ... */
SQL_API_RC SQL_API_FN
  sqlggtpi (
    unsigned short tn_length,
    unsigned char * tablename,
    struct sqlupi * part_info,
    struct sqlca * sqlca);
/* ... */
```

API パラメーター

tn_length
表名の長さを示す 2 バイトの無符号整数。

tablename
表の完全修飾名。

part_info
sqlupi 構造を指すポインター。この構造の詳細については、611ページの『SQLUPI』を参照してください。

pSqlca
出力。 *sqlca* 構造を指すポインターです。 この構造に関する詳細は、520ページの『SQLCA』を参照してください。

参照資料

374ページの『sqlugrpn - 行の区分化番号の入手』

355ページの『sqludrdt - ノード・グループの再配分』

sqluimpr - インポート

サポートされているファイル形式を用いて、外部ファイルから表、階層、または視点にデータを挿入します。 408ページの『sqluload - ロード』はより高速にこれを行うことができますが、このロード・ユーティリティーは、階層レベルではロードするデータをサポートしません。

許可

- INSERT オプションを使用して IMPORT する場合、以下のいずれかが必要です。
  - *sysadm*
  - *dbadm*
  - 関係するそれぞれの表または視点に対する CONTROL 特権
  - 関係するそれぞれの表または視点に対する INSERT および SELECT 特権
- INSERT_UPDATE、REPLACE、または REPLACE_CREATE オプションを使用して、既存の表に IMPORT する場合、以下のいずれかが必要です。
  - *sysadm*
  - *dbadm*
  - 表または視点に対する CONTROL 特権
- CREATE、または REPLACE_CREATE オプションを使用して、新規の表または階層に IMPORT する場合、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)
    - 階層全体で REPLACE_CREATE オプションが使用されている場合、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

必須接続

データベース。暗黙的な接続が可能である場合には、省略時データベースへの接続が確立されます。

API 組み込みファイル

sqlutil.h

C API 構文

```
/* File: sqlutil.h */
/* API: Import */
/* ... */
SQL_API_RC SQL_API_FN
  sqluimpr (
    char * pDataFileName,
    sqlu_media_list * pLobPathList,
    struct sqldcol * pDataDescriptor,
    struct sqlchar * pActionString,
    char * pFileType,
    struct sqlchar * pFileTypeMod,
    char * pMsgFileName,
    short CallerAction,
    struct sqluimpt_in*  pImportInfoIn,
    struct sqluimpt_out*  pImportInfoOut,
    sqlint32 * pNullIndicators,
    void * pReserved,
    struct sqlca * pSqlca);
/* ... */
```

汎用 API 構文

```
/* File: sqlutil.h */
/* API: Import */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgimpr (
    unsigned short DataFileNameLen,
    unsigned short FileTypeLen,
    unsigned short MsgFileNameLen,
    char * pDataFileName,
    sqlu_media_list * pLobPathList,
    struct sqldcol * pDataDescriptor,
    struct sqlchar * pActionString,
    char * pFileType,
    struct sqlchar * pFileTypeMod,
    char * pMsgFileName,
    short CallerAction,
    struct sqluimpt_in*  pImportInfoIn,
    struct sqluimpt_out*  pImportInfoOut,
    sqlint32 * NullIndicators,
    void * pReserved,
    struct sqlca * pSqlca);
/* ... */
```

API パラメーター

DataFileNameLen
入力。入力ファイル名の長さを示す 2 バイトの無符号整数 (バイト単位) です。

FileTypeLen
入力。入力ファイル・タイプの長さを示す 2 バイトの無符号整数 (バイト単位) です。

MsgFileNameLen
入力。メッセージ・ファイル名の長さを示す 2 バイトの無符号整数 (バイト単位) です。

pDataFileName
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。

pLobPathList
入力。 *media_type* SQLU_LOCAL_MEDIA を使用する *sqlu_media_list*、および LOB ファイルがあるクライアント上のパスをリストする *sqlu_media_entry* 構造を示します。

詳細については、 592ページの『SQLU-MEDIA-LIST』を参照してください。

pDataDescriptor

入力。外部ファイルからインポートするよう選択された列に関する情報を含む *sqldcol* 構造を指すポインターです。 *dcolmeth* フィールドの値によって、このパラメーターに提供される残りの情報をインポート・ユーティリティーがどのように解釈するかが判別されます。このパラメーターに有効な値 (`sqlutil` で定義) は、以下のとおりです。

SQL_METH_N

名前。外部入力ファイルの列は、列名によって選択されます。

SQL_METH_P

位置。外部入力ファイルの列は、列の位置によって選択されます。

SQL_METH_L

ロケーション。外部入力ファイルの列は、列のロケーションによって選択されます。以下のいずれかの条件のために無効であるロケーションの対を指定したインポート呼び出しは、データベース・マネージャーによって拒否されます。

- 開始または終了ロケーションが有効な範囲 (1 ～ 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、ヌル可能列がヌルで埋め込まれることを示します。

SQL_METH_D

省略時値。 *pDataDescriptor* がヌルの場合、または `SQL_METH_D` に設定されている場合、外部入力ファイルの列の省略時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。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。

詳細については、 526ページの『SQLDCOL』を参照してください。

pActionString

入力。 2 バイトの長さフィールドと、データをインポートする列を識別する文字の配列を含む *sqlchar* 構造を指すポインターです。

文字配列の形式は、以下のようになります。

```
{INSERT | INSERT_UPDATE | REPLACE | CREATE | REPLACE_CREATE}
INTO {tname[(tcolumn-list)] |
[{ALL TABLES | (tname[(tcolumn-list)][, tname[(tcolumn-list)]])}]
[IN] HIERARCHY {STARTING tname | (tname[, tname])}
[UNDER sub-table-name | AS ROOT TABLE]}
[DATALINK SPECIFICATION datalink-spec]
```

INSERT

既存の表データを変更することなく、インポートされたデータを表に追加します。

INSERT_UPDATE

1 次キー値が表にない場合はインポートした行を追加し、 1 次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 1 次キーがあり、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、 1 次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、視点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。

考慮事項: 既存のデータを削除した後にエラーが発生した場合、そのデータは失われてしまいます。

CREATE

指定された表が定義されていない場合に、指定された PC/IXF ファイルにある情報を使用して表定義と列の内容が作成されます。データベース・マネージャーによりファイルが事前にエクスポートされている場合、索引も作成されます。指定された表がすでに存在している場合、エラーが戻されます。このオプションは、PC/IXF ファイル形式にのみ有効です。

REPLACE_CREATE

指定された表が定義されている場合に、 PC/IXF ファイルにある PC/IXF 行情報を使用して表の内容が置き換えられます。表がまだ定義されていない場合は、指定された PC/IXF ファイルにある情報を使用して表定義と行の内容が作成されます。 DB2

により PC/IXF ファイルが事前にエクスポートされている場合は、索引も作成されます。このオプションは、PC/IXF ファイル形式にのみ有効です。

考慮事項: 既存のデータを削除した後にエラーが発生した場合、そのデータは失われてしまいます。

tname データが挿入される表、タイプ表、視点、またはオブジェクト視点の名前。下位のサーバーの場合を除き、 REPLACE、INSERT_UPDATE、または INSERT には、修飾または非修飾の名前を使わなければならないようなときでも、別名を指定することができます。視点の場合、読み取り専用視点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視点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。タイプ表に指定されている列のリストがない場合、それぞれの副表のすべての列にデータが挿入されます。

sub-table-name
CREATE オプションで 1 つまたは複数の副表を作成する際に、親表を指定します。

ALL TABLES
階層専用の暗黙キーワード。階層をインポートする際、省略時には*走査順序リスト* 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、省略時順序を使用するよう指定します。

UNDER
階層および CREATE 専用のキーワード。新しい階層、副階層、または副表を、指定された副表の下に作成するよう指定します。

AS ROOT TABLE
階層および CREATE 専用のキーワード。新しい階層、副階層、または副表を、独立型の階層として作成するよう指定します。

DATALINK SPECIFICATION *datalink-spec*
DB2 データ・リンクに関連するパラメーターを指定します。これらのパラメーターは、 IMPORT コマンドと同じ構文を使用して指定できます (*コマンド解説書* を参照)。

tname および *tcolumn-list* は、 SQL INSERT ステートメントの *tablename* および *colname* リストに対応し、同一の制限の下にあります。

tcolumn-list 内の列と、外部列 (指定または暗黙指定された) とは、リストまたは構造における位置に従って対応付けられます (*sqldcol* 構造で指定された最初の列からのデータは、 *tcolumn-list* の最初の要素に対応する表または視点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 (一部のヌル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。

詳細については、 522ページの『SQLCHAR』を参照してください。

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

SQL_ASC
区切りなし ASCII。

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

SQL_IXF
IXF (統合交換フォーマットの PC バージョン)。表からデータ

をエクスポートする場合の推奨方式で、同じ表または別のデータベース・マネージャー表にそれを再インポートすることが可能です。

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

ファイル形式の詳細については、 *データ移動ユーティリティー 手引きおよび解説書* の付録『エクスポート / インポート ロード・ユーティリティーのファイル形式』を参照してください。

pFileTypeMod
入力。 2 バイトの長さフィールドと、 1 つまたは複数の処理オプションを指定する文字の配列を含む構造を指すポインターです。このポインターがヌルであるか、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、このアクションは省略時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。

詳細については、 522ページの『SQLCHAR』、および*コマンド解説書*を参照してください。

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

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

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

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

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

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

pImportInfoIn
入力。追加の入力パラメーターを含む *sqluimpt_in* 構造を指すオプション・ポインターです。この構造の詳細については、
601ページの『SQLUIMPT-IN』を参照してください。

pImportInfoOut
出力。追加の出力パラメーターを含む *sqluimpt_out* 構造を指すオプション・ポインターです。この構造の詳細については、
602ページの『SQLUIMPT-OUT』を参照してください。

NullIndicators
入力。 ASC ファイルの場合にのみ使用します。列データがヌル可能であるかどうかを示す整数の配列です。この配列内の要素数は、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。この配列の要素とデータ・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、要素の数は、 *pDataDescriptor* パラメーターの *dcolnum* フィールドと同じでなければなりません。配列の各要素には、ヌル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、または表列がヌル可能ではないことを示すゼロが含まれます。要素がゼロでない場合は、データ・ファイル内の識別された列に、 `Y` と `N` のどちらかが含まれているはずです。 `Y` は表列のデータがヌルであることを示し、 `N` は表列のデータがヌルではないことを示します。

pReserved
将来の使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です。 この構造に関する詳細は、520ページの『SQLCA』を参照してください。

REXX API 構文

```
IMPORT FROM datafile OF filetype
[MODIFIED BY :filetmod]
[METHOD {L|N|P} USING :dcoldata]
[COMMITCOUNT :commitcnt] [RESTARTCOUNT :restartcnt]
MESSAGES msgfile
{INSERT|REPLACE|CREATE|INSERT_UPDATE|REPLACE_CREATE}
INTO tname [(:columns)]
[OUTPUT INTO :output]

CONTINUE IMPORT

STOP IMPORT
```

REXX API パラメーター

datafile
データのインポート元となるファイルの名前。

filetype
外部インポート・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 区切り付き ASCII

ASC 区切りなし 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 (*コマンド解説書* を参照。)

L|N|P 外部入力ファイル内の列の選択に使用される方式を指定する文字。有効な値は以下のとおりです。

L ロケーション

N 名前

P 位置

dcoldata
外部入力ファイルからインポートするよう選択された列に関する情報を含む、複合 REXX ホスト変数。構造の内容は、指定した方式 によって異なります。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。

- ロケーションの方式

  XXX.0 残りの変数内の要素数。

  XXX.1 この列の入力ファイルにおける開始ロケーションを示す数値。この列が、データベース表の最初の列になります。

  XXX.2 2 番目の列の終了ロケーションを示す数値。

  XXX.3 この列の入力ファイルにおける開始ロケーションを示す数値。この列が、データベース表の 2 番目の列になります。

  XXX.4 2 番目の列の終了ロケーションを示す数値。

  XXX.5 以降、3 番目、4 番目 ... と続きます。

- 名前の方式

  XXX.0 ホスト変数内に含まれている列名の数。

  XXX.1 最初の名前。

  XXX.2 2 番目の名前。

  XXX.3 以降、3 番目、4 番目 ... と続きます。

- 位置の方式

  XXX.0 ホスト変数内に含まれている列名の数。

  XXX.1 外部入力ファイルにおける列の位置。

  XXX.2 外部入力ファイルにおける列の位置。

  XXX.3 以降、3 番目、4 番目 ... と続きます。

tname ターゲット表または視点の名前。読み取り専用視点にデータをインポートすることはできません。

columns
データがインポートされる表や視点内の列の名前を含む、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。

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。

commitcnt

commitcnt 個のレコードがインポートされるたびに COMMIT を実行します。

restartcnt

インポート操作はレコード *restartcnt* + 1 から開始されます。最初の *restartcnt* 個のレコードはスキップされます。

output

インポートに関する情報が渡される複合 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.1 インポート操作中に外部入力ファイルから読み取られたレコードの数。

XXX.2 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。

XXX.3 ターゲット表に挿入された行の数。

XXX.4 インポートされたレコードからの情報で更新されたターゲット表の行数。

XXX.5 インポートできなかったレコードの数。

XXX.6 正常にインポートされ、データベースにコミットされたレコードの数 (挿入、更新、スキップ、および拒否された行を含む)。

サンプル・プログラム

C	¥sqllib¥samples¥c¥impexp.sqc
COBOL	¥sqllib¥samples¥cobol¥impexp.sqb
REXX	¥sqllib¥samples¥rexx¥impexp.cmd

使用上の注意

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

インポート・ユーティリティーは、 SQL INSERT ステートメントを使用してターゲット表に行を追加します。このユーティリティーは、入力ファイル中の各行のデータにつき 1 つずつ INSERT ステートメントを発行します。
INSERT ステートメントが失敗した場合、以下の 2 通りの結果のいずれかになります。

- 後続の INSERT ステートメントが成功すると予測される場合には、警告メッセージがメッセージ・ファイルに書き込まれ、処理が継続されます。
- 後続の INSERT ステートメントが失敗すると予測され、データベースが損傷する可能性がある場合には、エラー・メッセージが書き込まれ、処理が停止されます。

このユーティリティーは、 REPLACE または REPLACE_CREATE 操作時に以前の行が削除された後、自動 COMMIT を実行します。したがって、表オブジェクトが切り捨てられた後、システムに障害が起こったり、アプリケーションがデータベース・マネージャーに割り込んだりすると、元のデータのすべてが失われてしまいます。これらのオプションを使用する前に、元のデータがもはや必要ないことを確認してください。

CREATE、REPLACE、または REPLACE_CREATE 操作時にログが満杯になると、このユーティリティーは挿入されたレコードに対して自動 COMMIT を実行します。自動 COMMIT の後に、システムに障害が起こるか、またはアプリケーションがデータベース・マネージャーに割り込むと、部分的にデータの挿入された表はデータベース内に残ります。 REPLACE または REPLACE_CREATE オプションを使用してインポート操作全体を再実行するか、または *restartcnt* パラメーターを正常にインポートされた行の数に設定して INSERT を使用してください。

省略時解釈には、 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、 *commitcnt* パラメーターがゼロでない場合は実行されます。ログが満杯だと、ROLLBACK が実行されます。

インポート・ユーティリティーが COMMIT を実行するたびに、 2 つのメッセージがメッセージ・ファイルに書き込まれます。一方は、コミットされるレコードの数を示し、もう一方は、 COMMIT の成功後に書き込まれます。障害の後にインポート操作を再開するときには、スキップするレコードの数 (最後の正常なコミットから判別される) を指定してください。

インポート・ユーティリティーでは、多少の非互換性問題がある入力データは受け入れられます (たとえば、文字データは埋め込みまたは切り捨てを用いて

インポートできます。数値データは異なる数値データ・タイプを用いてインポートできます)。しかし、大きな非互換性問題のあるデータは受け入れられません。

オブジェクト表に何らかの従属 (それ自体への従属は除く) がある場合は、そのオブジェクト表を REPLACE または REPLACE_CREATE することはできません。また、オブジェクト視点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、そのオブジェクト視点を 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視点を置換するには、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消去します。
2. インポート・ユーティリティーを実行します。
3. 表を更新して、外部キーを再作成します。

外部キーの作成中にエラーが発生した場合には、データを修正して、参照保全を保持するようにしてください。

PC/IXF ファイルから表を作成するときは、参照制約と外部キー定義は保存されません。 (以前に SELECT * を使用してエクスポートされたデータの場合、1 次キー定義は保持されます。)

リモート・データベースへのインポートでは、サーバーに、入力データ・ファイルのコピー、出力メッセージ・ファイル、およびデータベースのサイズ拡大を見込んだ十分なディスク・スペースが必要とされます。

インポート操作がリモート・データベースに対して実行され、出力メッセージ・ファイルが非常に長くなってしまった場合 (60KB を超過)、クライアントのユーザーに戻されるメッセージ・ファイルは、インポートの途中でメッセージを失っているおそれがあります。メッセージ情報の最初の 30KB と最後の 30KB は、常に保存されます。

PC/IXF ファイルのリモート・データベースへのインポートは、 PC/IXF ファイルがディスケットにあるときよりも、ハード・ディスクにあるときの方がより速く行うことができます。 *pDataDescriptor* で非省略時値を使用したり、*pActionString* 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。タイプ表の場合、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 (たとえば、 OS/2 と AIX システム間を移動される) 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。 PC/IXF ファイル形式指定によって、OS/2 (IBM OS/2 拡張サービス、 OS/2 拡張版、および DB2 (OS/2 版)) データベースと、 DB2 (AIX 版) データベースとの間でデータの移行を、(1) エクスポート、 (2) OS/2 と AIX 間でのファイルの 2 進形式でのコピー、 (3) インポートという手順で行えるようになります。ソース・データベースと宛先データベースの両方が同一のクライアントからアクセス可能な場合には、ファイルのコピーというステップは必要ありません。

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みなします。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 (AIX 版) 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 コネクトは、DB2 (OS/390 版)、DB2 (VM および VSE 版)、および DB2 (OS/400 版) などの DRDA サーバーにデータをインポートするために使用できます。サポートされているのは、PC/IXF インポート (INSERT オプション) だけです。 *restartcnt* パラメーターもサポートされていますが、 *commitcnt* パラメーターはサポートされていません。

タイプ表で 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。タイプ表で CREATE 以外のオプションを使用するときは、走査順序リストによって走査順序を指定できます。このため、走査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走査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回復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、または要約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、視点を作成することはできません。

DB2 では、個々のパーツが OS/2 システムから AIX システムにコピーされる複数パーツ PC/IXF ファイルがサポートされています。

Windows NT オペレーティング・システムの場合は、以下のとおりです。

- 論理分割された PC/IXF ファイルのインポートはサポートされていません。
- 不正な形式の PC/IXF または WSF ファイルのインポートは、サポートされていません。

DB2 データ・リンク・マネージャーについての考慮事項

DB2 インポート・ユーティリティーを実行する前に、以下のようにしてください。

1. 参照されているファイルを適切なデータ・リンク・サーバーにコピーする。 **dlfm_import** ユーティリティーを使用すれば、 **dlfm_export** ユーティリティーで生成されたアーカイブからファイルを抽出することができます。
2. 必要な接頭名を DB2 データ・リンク・マネージャーに登録する。必要な場合は、データベースの登録など他の管理タスクを行うこともできます。

3. 必要な場合は、(DATALINK 列の) URL のデータ・リンク・サーバー情報を、 SQL 表のエクスポート済みデータから更新する。 (元の構成のデータ・リンク・サーバーのターゲット位置が同じである場合、データ・リンク・サーバー名を更新する必要はありません。)
4. データ・リンク・サーバーを、 DB2 データ・リンク・マネージャー構成ファイルのターゲット構成で定義する。

インポート・ユーティリティーがターゲット・システムで実行されると、DATALINK 列に関連するデータは、SQL INSERT により、基礎となっている DB2 表にロードされます (他の列の場合と同様)。

挿入操作の際、DATALINK 列処理では、ターゲット・データベースでの列指定に従って、適切なデータ・リンク・サーバーのファイルがリンクされます。

入力ファイルでの DATALINK 情報の表現

入力ファイルでの DATALINK 情報の表現方法の説明については、 420 ページを参照してください。

表 9. 有効なファイル・タイプ修飾子 (インポート)

修飾子	説明
すべてのファイル形式	
compound=*x*	*x* は、1 ～ 100 の数値です。非アトミック複合 SQL を使用してデータを挿入します。毎回 *x* 個のステートメントが試行されます。 この修飾子を指定した場合、トランザクション・ログに十分な大きさがないと、インポート操作は失敗します。トランザクション・ログには、 COMMITCOUNT で指定された行数、またはデータ・ファイル内の行数 (COMMITCOUNT が指定されていない場合) が入るだけの大きさが必要です。それで、トランザクション・ログがあふれるのを避けるため、 COMMITCOUNT を指定するようお勧めします。 この修飾子には、INSERT_UPDATE モード、階層形式の表、および usedefaults、identitymissing、 identityignore、generatedmissing、generatedignore の各修飾子との互換性はありません。
generatedignore	この修飾子はインポート・ユーティリティーに対して、すべての生成列のデータがデータ・ファイル内にあるものの、それらのデータは無視するべきものであることを通知します。その結果、生成列のすべての値はユーティリティーが生成します。この修飾子は、generatedmissing 修飾子とともに使用することはできません。
generatedmissing	この修飾子が指定されている場合、ユーティリティーは、生成列のデータが入力データ・ファイルに入っていない (ヌルも入っていない) ものと見なし、行ごとに値を生成します。この修飾子は、generatedignore 修飾子とともに使用することはできません。

表 9. 有効なファイル・タイプ修飾子 (インポート) (続き)

修飾子	説明
identityignore	この修飾子はインポート・ユーティリティーに対して、識別列のデータがデータ・ファイル内にあるものの、それらのデータは無視するべきものであることを通知します。その結果、すべての識別値はユーティリティーが生成します。 GENERATED ALWAYS 識別列と GENERATED BY DEFAULT 識別列のどちらの場合も動作は同じになります。つまり、GENERATED ALWAYS 列の場合には、拒否される行はありません。この修飾子は、identitymissing 修飾子とともに使用することはできません。
identitymissing	この修飾子が指定されている場合、ユーティリティーは、識別列のデータが入力データ・ファイルに入っていない (ヌルも入っていない) ものと見なし、行ごとに値を生成します。 GENERATED ALWAYS 識別列と GENERATED BY DEFAULT 識別列のどちらの場合も動作は同じになります。この修飾子は、identityignore 修飾子とともに使用することはできません。
lobsinfile	*lob-path* は、LOB 値を含むファイルへのパスを指定します。
no_type_id	単一の副表にインポートするときにのみ有効です。通常の使用法としては、正規の表からデータをエクスポートしてから、インポート操作を呼び出し (この修飾子を使用して)、データを単一の副表に変換します。
nodefaults	ターゲット表の列のソース列が明示的に指定されておらず、表の列がヌル不可の場合に、省略時値がロードされません。このオプションが指定されていない場合、ターゲット表の 1 つの列のソース列が明示的に指定されていないと、以下のいずれかの処理が行われます。 • 列の省略時値を指定できる場合は、省略時値がロードされます。 • 列がヌル可能で、その列に省略時値が指定できない場合は、ヌルがロードされます。 • 列がヌル可能ではなく、その列に省略時値も指定できない場合は、エラーが戻され、ユーティリティーは処理を停止します。
usedefaults	ターゲット表の列のソース列が指定されているが、 1 つまたは複数の行インスタンスのデータが含まれていない場合は、省略時値がロードされます。欠落データの例は、以下のとおりです。 • DEL ファイルの場合: 列に ",," が指定されています。 • ASC ファイルの場合: 列のヌル標識が「yes」に設定されています。 • DEL/ASC/WSF ファイルの場合: 十分な数の列がない行、または元の指定に対して長さが十分でない行。 このオプションが指定されていない場合、ソース列に行インスタンスのデータがないと、以下のいずれかの処理が行われます。 • その列がヌル可能な場合は、ヌルがロードされます。 • その列がヌル可能でない場合、ユーティリティーはその行を拒否します。

表 9. 有効なファイル・タイプ修飾子 (インポート) (続き)

修飾子	説明
ASCII ファイル形式 (ASC/DEL)	
dateformat="*x*"	*x* はソース・ファイル内の日付の形式です。[a] 有効な日付要素は以下のとおりです。 YYYY - 年 (0000 ～ 9999 の 4 桁) M - 月 (1 ～ 12 の 1 桁または 2 桁) MM - 月 (1 ～ 12 の 2 桁; M と相互に排他) D - 日 (1 ～ 31 の 1 桁または 2 桁) DD - 日 (1 ～ 31 の 2 桁; D と相互に排他) DDD - 元日から数えた日数 (001 ～ 366 の 3 桁; 他の日または月要素と相互に排他) 指定されていない要素には、省略時値の 1 が割り当てられます。日付形式の例を以下に示します。 "D-M-YYYY" "MM.DD.YYYY" "YYYYDDD"
implieddecimal	暗黙指定されている小数点の位置が列定義によって決定され、値の終わりにあるとはみなされなくなります。たとえば値 12345 は、12345.00 ではなく、 123.45 として DECIMAL(8,2) 列にロードされます。
noeofchar	任意指定のファイル終わり文字 x'1A' が、ファイルの終わりとして認識されません。通常の文字の場合のように処理が続行されます。

表 9. 有効なファイル・タイプ修飾子 (インポート) (続き)

修飾子	説明
timeformat="*x*"	*x* はソース・ファイル内の時刻の形式です。 [a] 有効な時刻要素は以下のとおりです。 H - 時 (12 時間系では 0 ～ 12 の 1 桁または 2 桁。 24 時間系では 0 ～ 24 の 1 桁または 2 桁) HH - 時 (12 時間系では 0 ～ 12 の 2 桁。 24 時間系では 0 ～ 24 の 2 桁; H と相互に排他) M - 分 (0 ～ 59 の 1 桁または 2 桁) MM - 分 (0 ～ 59 の 2 桁; M と相互に排他) S - 秒 (0 ～ 59 の 1 桁または 2 桁) SS - 秒 (0 ～ 59 の 2 桁; S と相互に排他) SSSSS - 夜の 12 時から数えた秒数 (00000 ～ 86399 の 5 桁; 他の時刻要素と相互に排他) TT - 正午の標識 (AM または PM) 指定されていない要素には、省略時値の 0 が割り当てられます。時刻形式の例を以下に示します。 "HH:MM:SS" "HH.MM TT" "SSSSS"

表 9. 有効なファイル・タイプ修飾子 (インポート) (続き)

修飾子	説明

timestampformat="x"	x はソース・ファイル内のタイム・スタンプの形式です。 [a] 有効なタイム・スタンプ要素は以下のとおりです。
	
	YYYY - 年 (0000 ～ 9999 の 4 桁)
	M - 月 (1 ～ 12 の 1 桁または 2 桁)
	MM - 月 (1 ～ 12 の 2 桁; M と相互に排他)
	D - 日 (1 ～ 31 の 1 桁または 2 桁)
	DD - 日 (1 ～ 31 の 2 桁; D と相互に排他)
	DDD - 元日から数えた日数 (001 ～ 366 の 3 桁; 他の月日要素と相互に排他)
	H - 時 (12 時間系では 0 ～ 12 の 1 桁または 2 桁。24 時間系では 0 ～ 24 の 1 桁または 2 桁)
	HH - 時 (12 時間系では 0 ～ 12 の 2 桁。24 時間系では 0 ～ 24 の 2 桁; H と相互に排他)
	M - 分 (0 ～ 59 の 1 桁または 2 桁)
	MM - 分 (0 ～ 59 の 2 桁; M と相互に排他)
	S - 秒 (0 ～ 59 の 1 桁または 2 桁)
	SS - 秒 (0 ～ 59 の 2 桁; S と相互に排他)
	SSSSS - 夜の 12 時から数えた秒数 (00000 ～ 86399 の 5 桁; 他の時刻要素と相互に排他)
	UUUUUU - マイクロ秒 (000000 ～ 999999 の 6 桁)
	TT - 正午の標識 (AM または PM)
	
	YYYY、M、MM、D、DD、または DDD 要素に値が指定されていない場合、省略時値の 1 が割り当てられます。それ以外の要素に値が指定されていない場合、省略時値の 0 が割り当てられます。タイム・スタンプ形式の例を以下に示します。
	
	`"YYYY/MM/DD HH:MM:SS.UUUUUU"`
	
	以下の例では、ユーザー定義の日付および時刻の形式を含んでいるデータを、 schedule という表にインポートする方法を示しています。
	
	`db2 import from delfile2 of del`
	`modified by timestampformat="yyyy.mm.dd hh:mm tt"`
	`insert into schedule`

ASC (区切りなし ASCII) ファイル形式	

表 9. 有効なファイル・タイプ修飾子 (インポート) (続き)

修飾子	説明
nochecklengths	nochecklengths が指定されていると、ソース・データの列定義がターゲット表の列のサイズを超えるものであっても、各行のインポートが試行されます。このような行が正常にインポートされるのは、コード・ページ変換でソース・データが縮小する場合です。たとえば、ソースにある 4 バイトの EUC データがターゲットで 2 バイトの DBCS データに縮小すれば、必要スペースは半分になります。このオプションが特に役立つのは、列の定義は不一致であるがソース・データが常に適合することが分かっている場合です。
nullindchar=*x*	*x* は単一文字です。この修飾子は、ヌル値を表す文字を *x* に変更します。 *x* の省略時値は Y です。 [b] 文字が英字である場合、この修飾子は、 EBCDIC データ・ファイルで大文字小文字を区別します。たとえば、ヌル標識文字が文字 N として指定されている場合は、 n もヌル標識として認識されます。
reclen=*x*	*x* は、最大値 32 767 の整数です。各行ごとに *x* 個の文字が読み取られ、行の終わりを示すのに改行文字は使用されません。

striptblanks	データを可変長フィールドにロードする際に、後書きブランク・スペースを切り捨てます。このオプションを指定しないと、ブランク・スペースは保持されます。
	
	以下の例では、striptblanks により、インポート・ユーティリティーが後書きブランク・スペースを切り捨てるよう指定しています。
	
	`db2 import from myfile.asc of asc`
	`  modified by striptblanks`
	`  method l (1 10, 12 15) messages msgs.txt`
	`  insert into staff`
	
	このオプションは、 striptnulls と一緒に指定することはできません。これらのオプションは、相互に排他的です。
	注: このオプションは以前の t オプションを置き換えるもので、後方互換性のためにのみサポートされています。

striptnulls	データを可変長フィールドにロードする際に、後書きヌル (0x00 文字) を切り捨てます。このオプションを指定しないと、ヌルは保持されます。 このオプションは、 striptblanks と一緒に指定することはできません。これらのオプションは、相互に排他的です。 **注:** このオプションは以前の padwithzero オプションを置き換えるもので、後方互換性のためにのみサポートされています。
	DEL (区切り付き ASCII) ファイル形式

表 9. 有効なファイル・タイプ修飾子 (インポート) (続き)

修飾子	説明

chardel*x*	*x* は、単一文字のストリング区切り文字です。省略時値は、二重引用符 (") です。指定した文字は、二重引用符の代わりに、文字ストリングを囲む文字として使用されます。 [bc]
	
	単一引用符 (') も、文字ストリングの区切り文字として指定できます。以下の例では、`chardel''` が指定されており、インポート・ユーティリティーは検出するすべての単一引用符 (') を文字ストリングの区切り文字として解釈します。
	
	`db2 "import from myfile.del of del`
	`modified by chardel''`
	`method p (1, 4) insert into staff (id, years)"`

coldel*x*	*x* は、単一文字の列区切り文字です。省略時値は、コンマ (,) です。指定した文字は、コンマの代わりに、列の終わりを示す文字として使用されます。 [bc]
	
	以下の例では、`coldel;` が指定されており、インポート・ユーティリティーは検出するすべてのセミコロン (;) を列の区切り文字として解釈します。
	
	`db2 import from myfile.del of del`
	`modified by coldel;`
	`messages msgs.txt insert into staff`

datesiso	日付形式。すべての日付データ値を ISO 形式でインポートします。
decplusblank	正符号文字。正の 10 進値の接頭部として、正符号 (+) ではなくブランク・スペースを使用します。省略時アクションでは、10 進値に正符号の接頭部が付けられます。

decpt*x*	*x* は、小数点文字としてピリオドの代わりに使用される単一の置換文字です。省略時値は、ピリオド (.) です。指定した文字は、小数点文字としてピリオドの代わりに使用されます。 [bc]
	
	以下の例では、`decpt;` が指定されており、インポート・ユーティリティーは検出するすべてのセミコロン (;) を小数点として解釈します。
	
	`db2 "import from myfile.del of del`
	`modified by chardel'`
	`decpt; messages msgs.txt insert into staff"`

表 9. 有効なファイル・タイプ修飾子 (インポート) (続き)

修飾子	説明

delprioritychar	区切り文字の現在の省略時優先順位は、レコード区切り文字、文字区切り文字、列区切り文字の順になっています。この修飾子は、区切り文字の優先順位を文字区切り文字、レコード区切り文字、列区切り文字と逆順にすることにより、以前の優先順位に依存する既存のアプリケーションを保護します。構文は次のとおりです。
	`db2 import ... modified by delprioritychar ...`
	たとえば、以下のような DEL データ・ファイルがあるとします。
	`"Smith, Joshua",4000,34.98<row delimiter>`
	`"Vincent,<row delimiter>, is a manager", ...`
	`... 4005,44.37<row delimiter>`
	`delprioritychar` 修飾子が指定されている場合、このデータ・ファイルは 2 行しかありません。 2 番目の <row delimiter> は 2 番目の行の最初のデータ列の一部と解釈されますが、最初と 3 番目の <row delimiter> は実レコードの区切り文字と解釈されます。この修飾子が指定されていないと、このデータ・ファイルは 3 行になり、各行は <row delimiter> によって区切られます。

dldel*x*	*x* は、単一文字の DATALINK 区切り文字です。省略時値は、セミコロン (;) です。指定した文字は、 DATALINK 値のフィールド間区切り文字としてセミコロンの代わりに使用されます。 DATALINK 値には 2 つ以上の副次値を指定できるので、この区切り文字が必要です。[bc] **注:** *x* は、行、列、または文字ストリングの区切り文字とは異なる文字にしてください。
keepblanks	タイプ CHAR、VARCHAR、LONG VARCHAR、または CLOB の各フィールドの前後のブランクを保持します。このオプションが指定されていない場合、文字区切り文字の外側にある前後のブランクはすべて除去され、ブランクになっているすべてのフィールドについては表にヌルが挿入されます。
nodoubledel	二重になっている区切り文字の認識を抑止します。詳細については、371ページの『区切り文字についての制限事項』を参照してください。
IXF ファイル形式	
forcein	コード・ページが不一致でもデータを受け入れ、コード・ページ間の変換を抑止するようユーティリティーに指示します。 固定長ターゲット・フィールドは、そのデータが入るだけの十分な大きさがあるかどうかチェックされます。`nochecklengths` が指定されていると、チェックは実行されず、各行のインポートが試行されます。

表 9. 有効なファイル・タイプ修飾子 (インポート) (続き)

修飾子	説明
indexixf	既存の表に現在定義されているすべての索引を消去し、 PC/IXF ファイルの索引定義から新しい索引を作成するよう、ユーティリティーに指示します。このオプションを使用できるのは、表の内容が置き換えられる場合だけです。視点では使用できません。また、*insert-column* が指定されている場合にも使用できません。
indexschema=*schema*	指定したスキーマ を索引作成時の索引名に使用します。スキーマ が指定されていない (しかし、 `indexschema` が*指定*されている) 場合は、接続ユーザー ID を使用します。このキーワードが指定されていない場合は、IXF ファイルのスキーマを使用します。
nochecklengths	`nochecklengths` が指定されていると、ソース・データの列定義がターゲット表の列のサイズを超えるものであっても、各行のインポートが試行されます。このような行が正常にインポートされるのは、コード・ページ変換でソース・データが縮小する場合です。たとえば、ソースにある 4 バイトの EUC データがターゲットで 2 バイトの DBCS データに縮小すれば、必要スペースは半分になります。このオプションが特に役立つのは、列の定義は不一致であるがソース・データが常に適合することが分かっている場合です。

表 9. 有効なファイル・タイプ修飾子 (インポート) (続き)

修飾子	説明

注:
1. サポートされていないファイル・タイプを MODIFIED BY オプションで使用しようとしても、インポート・ユーティリティーは警告を出しません。この場合、インポート操作が失敗し、エラー・コードが戻されます。
2. [a] 日付形式ストリングは必ず二重引用符で囲まなければなりません。フィールド区切り文字には、 a ～ z、A ～ Z、および 0 ～ 9 を含めることはできません。フィールド区切り文字は、 DEL ファイル形式の文字区切り文字またはフィールド区切り文字と同じにすることはできません。要素の開始および終了位置が明らかな場合、フィールド区切り文字は任意指定です。開始および終了位置が明らかでないのは、項目の長さが一定でない D、H、M、または S などの要素が使用されている場合です (修飾の仕方によって異なります)。
タイム・スタンプ形式の場合、月の記述子と分の記述子のどちらも文字 M を使用するため、区別が明白であるように注意しなければなりません。月のフィールドは他の日付フィールドに隣接していなければなりません。分のフィールドは他の時刻フィールドに隣接していなければなりません。以下のタイム・スタンプ形式は、あいまいな形式の例です。
`"M" (月と分のどちらかがはっきりしない)`
`"M:M" (どちらが何を表しているのか分からない)`
`"M:YYYY:M" (どちらも月として解釈される)`
`"S:M:YYYY" (時刻値と日付値の両方に隣接している)`
あいまいな形式になっている場合、ユーティリティーはエラー・メッセージを報告し、操作は失敗します。
以下のタイム・スタンプ形式では、何を指しているのかがはっきりしています。
`"M:YYYY" (月)`
`"S:M" (分)`
`"M:YYYY:S:M" (月....分)`
`"M:H:YYYY:M:D" (分....月)`
注: 二重引用符や円記号などの文字の前には、拡張文字 (たとえば、¥) を付けなければなりません。
3. [b] この文字は、ソース・データのコード・ページで指定してください。
文字コード・ポイント (文字記号ではない) を、 xJJ または 0xJJ の構文 (JJ はコード・ポイントの 16 進表記) を使用して指定できます。たとえば、# 文字を列区切り文字として指定するには、以下のいずれかを使用します。
`... modified by coldel# ...`
`... modified by coldel0x23 ...`
`... modified by coldelX23 ...`
4. [c] 371ページの『区切り文字についての制限事項』には、代替区切り文字として使用できる文字に適用される制限事項がリストされています。

参照資料

360ページの『sqluexpr - エクスポート』

408ページの『sqluload - ロード』

sqluload - 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、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、完全修飾ファイルまたは名前付きパイプの形式とすることができます。テープは OS/2 ではサポートされていません。ロード・ユーティリティーは、階層レベルでのデータのロードをサポートしていません。

効力範囲

このコマンドは、直接接続の接続先となる区画だけに影響し、ロード・ユーティリティーは、1 つのデータベース区画だけを操作します。

リモートで接続しているクライアント上にあるデータのロードは、以下の条件ではサポートされていません。

- クライアントが接続しているデータベースが DB2 エンタープライズ拡張エディション環境である。
- クライアントが接続しているデータベースが、すでにカタログ化されているデータベースに対してカタログ化されている。

許可

以下のいずれかです。

- *sysadm*
- *dbadm*
- データベースのロード権限と以下のもの
  - 表の INSERT 特権 (ロード・ユーティリティーが INSERT モード、TERMINATE モード、または RESTART モードで呼び出される場合)。TERMINATE モードは直前のロード挿入操作を終了するためのもので、RESTART モードは直前のロード挿入操作を再開するためのものです。
  - 表の INSERT および DELETE 特権 (ロード・ユーティリティーが REPLACE モード、 TERMINATE モード、または RESTART モードで呼び出される場合)。 TERMINATE モードは直前のロード置換操作を終了するためのもので、 RESTART モードは直前のロード置換操作を再開するためのものです。
  - 例外表の INSERT 特権 (例外表をロード操作の一部として使用する場合)。

注: すべてのロード処理 (および一般的にはすべての DB2 サーバー処理) がインスタンス所有者に所有されており、これらの処理すべてがインスタンス

所有者の ID を使用して必要なファイルにアクセスするため、インスタンス所有者はデータ・ファイルを入力するために読み取りアクセスを持っている必要があります。これらの入力データ・ファイルは、コマンドを呼び出すのが誰かに関係なくインスタンス所有者によって読み取り可能でなければなりません。

必須接続

データベース。暗黙的な接続が可能である場合には、省略時データベースへの接続が確立されます。

インスタンス。明示的な接続は必要ありません。データベースへの接続が確立されている場合には、ローカル・インスタンスへの暗黙的な接続が試みられます。

API 組み込みファイル

sqlutil.h

C API 構文

```
/* File: sqlutil.h */
/* API: Load */
/* ... */
SQL_API_RC SQL_API_FN
  sqluload (
    sqlu_media_list * pDataFileList,
    sqlu_media_list * pLobPathList,
    struct sqldcol * pDataDescriptor,
    struct sqlchar * pActionString,
    char * pFileType,
    struct sqlchar * pFileTypeMod,
    char * pLocalMsgFileName,
    char * pRemoteMsgFileName,
    short CallerAction,
    struct sqluload_in * pLoadInfoIn,
    struct sqluload_out * pLoadInfoOut,
    sqlu_media_list * pWorkDirectoryList,
    sqlu_media_list * pCopyTargetList,
    sqlint32 * pNullIndicators,
    void * pReserved,
    struct sqlca * pSqlca);
/* ... */
```

汎用 API 構文

```
/* File: sqlutil.h */
/* API: Load */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgload (
    unsigned short FileTypeLen,
    unsigned short LocalMsgFileNameLen,
    unsigned short RemoteMsgFileNameLen,
    sqlu_media_list * pDataFileList,
    sqlu_media_list * pLobPathList,
    struct sqldcol * pDataDescriptor,
    struct sqlchar * pActionString,
    char * pFileType,
    struct sqlchar * pFileTypeMod,
    char * pLocalMsgFileName,
    char * pRemoteMsgFileName,
    short CallerAction,
    struct sqluload_in * pLoadInfoIn,
    struct sqluload_out * pLoadInfoOut,
    sqlu_media_list * pWorkDirectoryList,
    sqlu_media_list * pCopyTargetList,
    sqlint32 * pNullIndicators,
    void * pReserved,
    struct sqlca * pSqlca);
/* ... */
```

API パラメーター

FileTypeLen

入力。ファイル・タイプの長さを示す 2 バイトの無符号整数 (バイト単位)。

LocalMsgFileNameLen

入力。ローカル・メッセージ・ファイル名の長さを示す 2 バイトの無符号整数 (バイト単位)。

RemoteMsgFileNameLen

入力。一時ファイル・パス名の長さを示す 2 バイトの無符号整数 (バイト単位)。

pDataFileList

入力。ソース・ファイル、装置、ベンダー、またはパイプ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。テープは OS/2 ではサポートされていません。

この構造に提供される情報は、*media_type* フィールドの値によって異なります。有効な値 (sqlutil で定義) は、以下のとおりです。

SQLU_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。この値は、ファイル、装置、および名前付きパイプの場合に使用されます。

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。この値は、完全修飾ファイル、および名前付きパイプの場合に使用されます。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な識別子が入ります。 *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、異なる順序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* には共用ライブラリー名、 *filename* にはロードされるデータに固有な識別子が入ります。 *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、異なる順序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。

ファイル名を入力する場合には、必ず完全修飾名にしてください。詳細については、 592ページの『SQLU-MEDIA-LIST』を参照してください。

pLobPathList

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。テープは OS/2 ではサポートされていません。

この構造に提供される情報は、*media_type* フィールドの値によって異なります。有効な値 (`sqlutil` で定義) は、以下のとおりです。

SQLU_LOCAL_MEDIA

この値に設定されている場合、呼び出し側から *sqlu_media_entry* 構造によって情報が提供されます。 *sessions* フィールドは、提供される *sqlu_media_entry* 構造の数を示します。

SQLU_TSM_MEDIA

この値に設定されている場合、*sqlu_vendor* 構造が使用されます。 *filename* には、ロードされるデータに固有な識別子が入ります。 *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、異なる順序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* には共用ライブラリー名、*filename* にはロードされるデータに固有な識別子が入ります。 *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、異なる順序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。

詳細については、 592ページの『SQLU-MEDIA-LIST』を参照してください。

pDataDescriptor

入力。外部ファイルからロードするよう選択された列に関する情報を含む *sqldcol* 構造を指すポインター。

pFileType パラメーターが SQL_ASC に設定されている場合、この構造の *dcolmeth* フィールドは、必ず SQL_METH_L に設定してください。ユーザーは、ロードされる列ごとに開始ロケーションと終了ロケーションを指定します。

ファイル・タイプ SQL_DEL の場合、 *dcolmeth* は SQL_METH_P か SQL_METH_D のどちらかにすることができます。 SQL_METH_P の場合、ユーザーは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。

詳細については、 526ページの『SQLDCOL』を参照してください。

pActionString

入力。 2 バイトの長さフィールドと、表に影響するアクションを指定する文字の配列を含む *sqlchar* 構造を指すポインター。

文字配列の形式は、以下のようになります。

```
"INSERT|REPLACE|RESTART|TERMINATE
INTO tbname [(column_list)]
[DATALINK SPECIFICATION datalink-spec]
[FOR EXCEPTION e_tbname]"
```

INSERT

既存の表データを変更することなく、ロードされたデータを表に追加します。

REPLACE

表から既存データをすべて削除し、ロードされたデータを挿入します。表定義および索引定義は変更されません。

RESTART

以前に割り込んだロード操作を再開します。ロード操作は、最後にロード、作成、または削除を行った一貫性ポイントから自動的に続行します。

TERMINATE
以前に割り込んだロード操作を終了し、一貫性ポイントが設定されていたとしても、操作をその開始時点までロールバックします。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保留状態でなければ、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保留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、または未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。未修飾の表名を指定すると、その表は CURRENT SCHEMA で修飾されます。

(*column_list***)**
データの挿入先の表の列名のリスト。列名は、コンマで区切らなければなりません。名前にスペースまたは小文字が含まれている場合には、それを引用符で囲まなければなりません。

DATALINK SPECIFICATION *datalink-spec*
DB2 データ・リンクに関連するパラメーターを指定します。これらのパラメーターは、 LOAD コマンドと同じ構文を使って指定できます (*コマンド解説書* を参照)。

FOR EXCEPTION *e_tbname*
エラー状態の行がコピーされる例外表を指定します。固有索引または 1 次キー索引に違反した行がすべてコピーされます。例外表には、DATALINK 例外も取り込まれます。

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

SQL_ASC
区切りなし ASCII。

SQL_DEL
区切り付き 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。

ファイル形式の詳細については、 *データ移動ユーティリティー 手引きおよび解説書* の付録『エクスポート / インポート / ロード・ユーティリティーのファイル形式』を参照してください。

pFileTypeMod

入力。 2 バイトの長さフィールドと、 1 つまたは複数の処理オプションを指定する文字の配列を含む構造を指すポインター。このポインターがヌルであるか、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、このアクションは省略時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。

詳細については、 522ページの『SQLCHAR』およびコマン*ド解説書* を参照してください。

pLocalMsgFileName

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。

pRemoteMsgFileName

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、メッセージや一貫性ポイントを格納したり、フェーズ情報を削除したりするために作成されます。一時ファイルの詳細については、 *データ移動ユーティリティー 手引きおよび解説書* を参照してください。

CallerAction

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

SQLU_INITIAL

最初の呼び出し。この値 (または SQLU_NOINTERRUPT) は、API への最初の呼び出しの際には必ず使用してください。

SQLU_NOINTERRUPT
最初の呼び出し。処理を中断しません。この値 (または SQLU_INITIAL) は、API への最初の呼び出しの際には必ず使用してください。

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

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

SQLU_TERMINATE
処理の終了。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。

pLoadInfoIn
入力。追加の入力パラメーターを含む *sqluload_in* 構造を指すオプション・ポインター。
この構造の詳細については、604ページの『SQLULOAD-IN』を参照してください。

pLoadInfoOut

出力。追加の出力パラメーターを含む *sqluload_out* 構造を指すオプション・ポインター。
この構造の詳細については、609ページの『SQLULOAD-OUT』を参照してください。

pWorkDirectoryList

予約済み。

pCopyTargetList

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、または共用ライブラリーのリストを提供するときに使用します。

この構造に入力する値は、*media_type* フィールドの値によって異なります。このフィールドに有効な値 (`sqlutil` で定義) は、以下のとおりです。

SQLU_LOCAL_MEDIA

コピーをローカル媒体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、異なる順序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。

詳細については、 592ページの『SQLU-MEDIA-LIST』を参照してください。

pNullIndicators

入力。 ASC ファイルの場合にのみ使用します。列データがヌル可能で

あるかどうかを示す整数の配列です。この配列の要素と、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、要素の数は、*pDataDescriptor* パラメーターの *dcolnum* フィールドと同じでなければなりません。配列の各要素には、ヌル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、または表列がヌル可能ではないことを示すゼロが含まれます。要素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 は表列のデータがヌルであることを示し、 N は表列のデータがヌルではないことを示します。

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。この構造に関する詳細については、 520ページの『SQLCA』を参照してください。

REXX API 構文

この API は、SQLDB2 インターフェースを使って、REXX から呼び出すことができます。 13ページの『API の説明の編成』、または *アプリケーション開発の手引き* を参照してください。構文の記述については、*コマンド解説書* を参照してください。

サンプル・プログラム

C	¥sqllib¥samples¥c¥tload.sqc
COBOL	¥sqllib¥samples¥cobol¥tload.sqb

使用上の注意

データは、入力ファイルに入って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。固有キーの重複を処理するのに、例外表が使用されます。ユーティリティーは参照保全を強制したり、制約検査を実行したり、またはロードする表に従属する要約表を更新したりすることはありません。参照制約または検査制約を含む表は、検査保留状態に置かれます。 REFRESH IMMEDIATE で定義されており、ロード中の表に従属する要約表も、検査保留状態に置かれます。表の検査保留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、複写された要約表では実行できません。

クラスター化が必要な場合、データはロードの前にクラスター化索引でソートする必要があります。

DB2 データ・リンク・マネージャーについての考慮事項

それぞれの DATALINK 列ごとに、括弧内に 1 つの列を指定できます。それぞれの列の指定は、1 つ以上の DL_LINKTYPE、*接頭部* 、および DL_URL_SUFFIX の指定で構成されます。*接頭部* の情報は、DL_URL_REPLACE_PREFIX、または DL_URL_DEFAULT_PREFIX の指定のいずれかになります。

指定できる DATALINK 列の数は、表で定義されている DATALINK 列の数と同じです。指定の順序は、 (INSERT INTO (insert-column, ...) によって指定される場合) insert-column リスト内にある DATALINK 列の順序、あるいは (insert-column が指定されていない場合) 表定義内にある DATALINK 列の順序に従います。

たとえば、表に列 C1、C2、C3、C4、および C5 があり、そのうち列 C2 と C5 だけが DATALINK 型であり、 insert-column リストが (C1、C5、C3、C2) である場合、 2 つの DATALINK 列を指定する必要があります。最初の列の指定は C5 用であり、2 番目の列の指定は C2 用です。 insert-column リストを指定しない場合、最初の列の指定は C2 用になり、 2 番目の列の指定は C5 用になります。

複数の DATALINK 列があり、一部の列では特別な指定が必要ではない場合、列の指定には、指定の順序をはっきりと示すために、少なくとも括弧を含める必要があります。どの列にも指定が行われない場合は、空の括弧のリスト全体を除くことができます。したがって、省略時値で十分な場合には、DATALINK を指定する必要はありません。

FILE LINK CONTROL で定義された DATALINK 列を含む表にデータがロードされる場合は、ロード・ユーティリティーを呼び出す前に、以下のステップを実行してください。 (すべての DATALINK 列が NO LINK CONTROL で定義されている場合、これらのステップは必要ありません)。

1. DATALINK 列の値によって参照されるデータ・リンク・サーバーに、 DB2 データ・リンク・マネージャーがインストールされていることを確認する。
2. データベースが DB2 データ・リンク・マネージャーに登録されていることを確認する。
3. DATALINK 値として挿入されるすべてのファイルを、適切なデータ・リンク・サーバーにコピーする。

4. データ・リンク・サーバー上の DB2 データ・リンク・マネージャーの接頭部名 (複数可) を定義する。
5. DB2 データ・リンク・マネージャー構成ファイルの DATALINK データ (ロードされる) によって参照されるデータ・リンク・サーバーを登録する。

ロード・ユーティリティーの実行中に、DB2 とデータ・リンク・サーバーとの間の接続が失敗し、ロード操作も失敗してしまう場合があります。この状況が発生する場合には、以下を実行します。

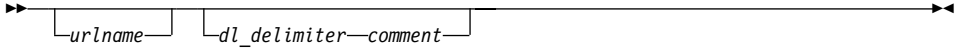
1. データ・リンク・サーバーおよび DB2 データ・リンク・マネージャーを始動する。
2. ロードの再始動操作を呼び出す。

ロード操作時に失敗したリンクは、データ保全性違反とみなされ、固有索引違反と同じ方法で処理されます。したがって、1 つ以上の DATALINK 列のある表のロードでは、特別な例外が定義されています。詳細については、*SQL 解説書* で例外についての説明を参照してください。

入力ファイルでの DATALINK 情報の表現

LINKTYPE (現在は URL だけがサポートされている) は、DATALINK 情報の一部として指定されていません。 LINKTYPE は LOAD または IMPORT コマンドで指定され、 PC/IXF 型の入力ファイルの場合は、適切な列記述子レコードで指定されます。これについては、 *データ移動ユーティリティー 手引きおよび解説書* の『エクスポート / インポート / ロード・ユーティリティーのファイル形式』で説明されています。

URL LINKTYPE の DATALINK 情報の構文は、次のとおりです。



urlname と *comment* は両方とも任意指定です。どちらも指定しないと、ヌル値が割り当てられます。

urlname

この URL 名は有効な URL 構文に適合していなければなりません。

注:

1. スキーム名としては、″http″ と ″file″ だけが許可されています。

2. URL 名の接頭部 (スキーム、ホスト、およびポート) は任意指定です。接頭部がない場合は、ロード・ユーティリティーまたはインポート・ユーティリティーの DL_URL_DEFAULT_PREFIX または DL_URL_REPLACE_PREFIX 指定の接頭部が使われます。これらも指定されていない場合、接頭部は省略時値の ″file://localhost″ になります。したがって、ローカル・ファイルでは、LOAD または IMPORT コマンド内に DATALINK 列を指定せずに、全パス名のファイル名を URL 名として入力することができます。
3. 接頭部が URL 名に付けられている場合も、ロードまたはインポート操作時には、 DL_URL_REPLACE_PREFIX で指定した異なる接頭部名によってオーバーライドされます。
4. (DL_URL_SUFFIX を指定した場合、それを追加した後の) ″path″ は、リモート・サーバーにあるリモート・ファイルの全パス名です。相対パス名は許可されていません。 HTTP サーバーの省略時のパス接頭部は使用されません。

dl_delimiter

区切り付き ASCII (DEL) ファイル形式では、dldel 修飾子で指定した文字、あるいは LOAD または IMPORT コマンドでの省略時の文字。区切りなし ASCII (ASC) ファイル形式の場合、これは文字シーケンス **¥;** (円記号およびその後のセミコロン) に対応している必要があります。空白文字 (ブランク、タブなど) は、このパラメーターに指定した値の前後に置くことができます。

comment

DATALINK 値のコメント部分。区切り付き ASCII (DEL) ファイル形式で指定する場合、 *comment* テキストは、文字ストリング区切り文字で囲む必要があります。これは省略時には二重引用符 (″) になります。この文字ストリング区切り文字は、 LOAD または IMPORT コマンドで MODIFIED BY *filetype-mod* を指定することによりオーバーライドすることができます。

コメントを指定しない場合、このコメントは省略時値である長さがゼロのストリングになります。

以下に示すのは、区切り付き ASCII (DEL) ファイル形式での DATALINK データの例です。

- `http://www.almaden.ibm.com:80/mrep/intro.mpeg; "Intro Movie"`

  これは、以下の部分とともに格納されます。

  – scheme = http

  - server = www.almaden.ibm.com
  - path = /mrep/intro.mpeg
  - comment = ″Intro Movie″
- `file://narang/u/narang; "InderPal's Home Page"`
  これは、以下の部分とともに格納されます。
  - scheme = file
  - server = narang
  - path = /u/narang
  - comment = ″InderPal’s Home Page″
- `file:/home/ff.gg; "hi there"`
  これは、以下の部分とともに格納されます。
  - scheme = file
  - server = localhost
  - path = /home/ff.gg
  - comment = ″hi there″

以下に示すのは、区切りなし ASCII (ASC) ファイル形式での DATALINK データの例です。

- `http://www.almaden.ibm.com:80/mrep/intro.mpeg¥;Intro Movie`
  これは、以下の部分とともに格納されます。
  - scheme = http
  - server = www.almaden.ibm.com
  - path = /mrep/intro.mpeg
  - comment = ″Intro Movie″
- `file://narang/u/narang¥; InderPal's Home Page`
  これは、以下の部分とともに格納されます。
  - scheme = file
  - server = narang
  - path = /u/narang
  - comment = ″InderPal’s Home Page″
- `file:/home/ff.gg¥; hi there`
  これは、以下の部分とともに格納されます。
  - scheme = file
  - server = localhost

- path = /home/ff.gg
- comment = ″hi there″

以下に示すのは、列へのロードまたはインポートの指定が DL_URL_DEFAULT_PREFIX (″http://qso″) であるとみなされるときの、DATALINK データの例です。

- `file://narang/pics/xxx.jpeg?search_pat`
  これは、以下の部分とともに格納されます。
  - scheme = file
  - server = narang
  - path = /pics/xxx.jpeg
  - comment = NULL string
- `/u/me/myfile.ps`
  これは、以下の部分とともに格納されます。
  - scheme = http
  - server = qso
  - path = /u/me/myfile.ps
  - comment = NULL string

表 10. 有効なファイル・タイプ修飾子 (LOAD)

修飾子	説明
すべてのファイル形式	
anyorder	この修飾子は、*cpu_parallelism* パラメーターとともに使用され、ソース・データの順序を保持する必要がないことを指定します。そのため、SMP システムでは、パフォーマンスがかなり向上します。*cpu_parallelism* の値が 1 である場合、このオプションは無視されます。一貫性ポイント後の破損回復ではデータを順番にロードする必要があるため、 SAVECOUNT > 0 の場合には、このオプションはサポートされません。

表 10. 有効なファイル・タイプ修飾子 (LOAD) (続き)

修飾子	説明
fastparse	ユーザー提供の列値に対して、簡略化された構文チェックを実行します。これにより、パフォーマンスが向上します。このオプションの下でロードした表は、体系的に正確であることが保証され、ユーティリティーは、セグメント化違反またはトラップを防ぐための十分なデータ・チェックを実行することが保証されます。正しい形式のデータが正しくロードされます。 たとえば、ASC ファイル内の整数列のフィールド項目として 123qwr4 の値が検出された場合、この値は有効な数値を表すものではないので、ロード・ユーティリティーは通常、構文エラーのフラグを付けます。 fastparse が指定されている場合、構文エラーは検出されず、整数フィールドには任意の数値がロードされます。この修飾子を使用する場合は、正しい形式のデータだけを使用するように注意してください。 ASCII データでこのオプションを使用するとパフォーマンスはかなり向上しますが、 PC/IXF データでこのオプションを使用しても fastparse のパフォーマンスはそれほど向上しません。これは、IXF が 2 進形式であり、 fastparse が ASCII から内部形式への構文解析および変換に影響するためです。
generatedignore	この修飾子はロード・ユーティリティーに対して、すべての生成列のデータがデータ・ファイル内にあるものの、それらのデータは無視するべきものであることを通知します。ヌル可能な生成列の場合、列にはヌルがロードされます。ヌル不可の生成列の場合、生成列のデータ・タイプの省略時値がロードされます。ロード操作の最後に、 SET INTEGRITY ステートメントを呼び出して、ロードされた値を、生成列の定義に従って計算された値に強制置換することができます。この修飾子は、 generatedmissing または generatedoverride 修飾子とともに使用することはできません。
generatedmissing	この修飾子が指定されている場合、ユーティリティーは、生成列のデータが入力データ・ファイルに入っていない (ヌルも入っていない) ものと見なし、ヌルを列にロードします。ロード操作の最後に、 SET INTEGRITY ステートメントを使用して、ヌルを、生成列の定義に従って計算された値に置換することができます。この修飾子は、generatedoverride または generatedoverride 修飾子とともに使用することはできません。

表 10. 有効なファイル・タイプ修飾子 (LOAD) (続き)

修飾子	説明
generatedoverride	この修飾子はロード・ユーティリティーに対し、表内のすべての生成列に関して、明示的な非ヌル・データを受け入れる (これらのタイプの列に関する通常の規則に反する) ように指示します。これが役立つのは、別のデータベース・システムからデータを移行する場合や、 ROLLFORWARD DATABASE コマンドで DROPPED TABLE RECOVERY オプションを使用して回復したデータから表をロードする場合です。この修飾子を使用した場合、ヌル不可の生成列でデータまたはヌル・データの入っていない行は拒否されます (SQL3116W)。 **注:** このオプションが使用されていると、ロード・ユーティリティーは生成列の値を妥当性検査しません。 この修飾子は、 `generatedmissing` または `generatedignore` 修飾子とともに使用することはできません。
identityignore	この修飾子はロード・ユーティリティーに対して、識別列のデータがデータ・ファイル内にあるものの、それらのデータは無視するべきものであることを通知します。その結果、すべての識別値はユーティリティーが生成します。 GENERATED ALWAYS 識別列と GENERATED BY DEFAULT 識別列のどちらの場合も動作は同じになります。つまり、GENERATED ALWAYS 列の場合には、拒否される行はありません。この修飾子は、 `identitymissing` または `identityoverride` 修飾子とともに使用することはできません。
identitymissing	この修飾子が指定されている場合、ユーティリティーは、識別列のデータが入力データ・ファイルに入っていない (ヌルも入っていない) ものと見なし、行ごとに値を生成します。 GENERATED ALWAYS 識別列と GENERATED BY DEFAULT 識別列のどちらの場合も動作は同じになります。この修飾子は、 `identityignore` または `identityoverride` 修飾子とともに使用することはできません。
identityoverride	この修飾子を使用するのは、 GENERATED ALWAYS として定義されている識別列が、ロードされる表にある場合だけです。この修飾子はユーティリティーに対し、そのような列に関して、明示的な非ヌル・データを受け入れる (これらのタイプの識別列に関する通常の規則に反する) ように指示します。これが役立つのは、別のデータベース・システムからデータを移行するときに GENERATED ALWAYS として表を定義しなければならない場合や、 ROLLFORWARD DATABASE コマンドで DROPPED TABLE RECOVERY オプションを使用して回復したデータから表をロードする場合です。この修飾子を使用した場合、識別列でデータまたはヌル・データの入っていない行は拒否されます (SQL3116W)。この修飾子は、 `identitymissing` または `identityignore` 修飾子とともに使用することはできません。 **注:** このオプションが使用されていると、ロード・ユーティリティーは、表の識別列内の値の固有性の保守または検査を行いません。

表 10. 有効なファイル・タイプ修飾子 (LOAD) (続き)

修飾子	説明
indexfreespace=*x*	*x* は 0 ～ 99 の整数です。この値は各索引ページのうち、索引のロード時に空きスペースとして残される部分のパーセンテージとして解釈されます。ページの最初の項目は制限なしで追加されます。残りの項目は、空きスペースのしきい値のパーセントを維持できる限り追加されます。省略時値は、CREATE INDEX の実行時に使用した値です。 この値は、CREATE INDEX ステートメントで指定した PCTFREE 値に優先し、索引の葉ページだけに影響します。
lobsinfile	*lob-path* は、 LOB 値を含むファイルへのパスを指定します。ASC、DEL、または IXF ロード入力ファイルには、 LOB 列に LOB データが入っているファイルの名前が含まれています。
noheader	ヘッダー検査コードをスキップします (単一ノードのノードグループ内にある表へのロード操作にのみ適用されます)。 オートローダー・ユーティリティーは、 マルチノード・ノードグループの表にデータを提供している各ファイルにヘッダーを書き込みます。このヘッダーには、ノード番号、区分化マップ、および区分化キーの指定が含まれています。ロード・ユーティリティーには、データが正しいノードでロードされたことを検査するために、この情報が必要です。単一ノード・ノードグループに存在する表にファイルをロードする場合、ヘッダーは存在しないため、このオプションを指定すると、ロード・ユーティリティーはヘッダー検査コードをスキップします。
norowwarnings	拒否された行についてのすべての警告を抑制します。
pagefreespace=*x*	*x* は 0 ～ 100 の整数です。この値は各データ・ページのうち、空きスペースとして残される部分のパーセンテージとして解釈されます。 最小行サイズのために、指定した値が無効である場合 (たとえば、最も小さい行の大きさが 3 000 バイトで、*x* の値が 50 である場合)、その行は新しいページに置かれます。 100 の値を指定した場合、各行が新しいページに置かれます。 **注:** 表の PCTFREE 値は、ページごとに指定された空きスペースの量を決定します。ロード操作の `pagefreespace` 値、または表の PCTFREE 値が設定されていない場合、ユーティリティーは、それぞれのページにできるだけ大きなスペースを割り当てます。`pagefreespace` で設定した値は、その表について指定された PCTFREE 値をオーバーライドします。
totalfreespace=*x*	*x* は 0 ～ 100 の整数です。この値は表内の合計ページのうち、表の終わりに空きスペースとして追加される部分のパーセンテージとして解釈されます。たとえば、*x* が 20 で、表に 100 個のデータ・ページが存在する場合、20 個の空のページが追加されます。この表のデータ・ページの合計数は、120 になります。

表 10. 有効なファイル・タイプ修飾子 (LOAD) (続き)

修飾子	説明
usedefaults	ターゲット表の列のソース列が指定されているが、 1 つまたは複数の行インスタンスのデータが含まれていない場合は、省略時値がロードされます。欠落データの例は、以下のとおりです。 • DEL ファイルの場合: 列に ",," が指定されています。 • DEL/ASC/WSF ファイルの場合: 十分な数の列がない行、または元の指定に対して長さが十分でない行。 このオプションが指定されていない場合、ソース列に行インスタンスのデータがないと、以下のいずれかの処理が行われます。 • その列がヌル可能な場合は、ヌルがロードされます。 • その列がヌル可能でない場合、ユーティリティーはその行を拒否します。
ASCII ファイル形式 (ASC/DEL)	
codepage=*x*	*x* は ASCII 文字ストリングです。この値は、入力データ・セット内のデータのコード・ページとして解釈されます。ロード操作時には、文字データ (および文字で指定された数値データ) は、このコード・ページからデータベースのコード・ページへ変換されます。 以下の規則が適用されます。 • 純粋な DBCS (グラフィック)、混合 DBCS、および EUC の場合、区切り文字は x00 ～ x3F の範囲に制限されます。 • EBCDIC コード・ページで指定された DEL データの場合、区切り文字をシフトインおよびシフトアウト DBCS 文字と一致させることはできません。 • `nullindchar` には、標準の ASCII セットに含まれる (コード・ポイント x20 ～ x7F の範囲の) 記号を指定する必要があります。これは、ASCII 記号およびコード・ポイントを示します。EBCDIC データでは、コード・ポイントが異なるとしても、対応する記号を使用することができます。

表 10. 有効なファイル・タイプ修飾子 (LOAD) (続き)

修飾子	説明
dateformat="*x*"	*x* はソース・ファイル内の日付の形式です。 [a] 有効な日付要素は以下のとおりです。 YYYY - 年 (0000 ～ 9999 の 4 桁) M - 月 (1 ～ 12 の 1 桁または 2 桁) MM - 月 (1 ～ 12 の 2 桁; M と相互に排他) D - 日 (1 ～ 31 の 1 桁または 2 桁) DD - 日 (1 ～ 31 の 2 桁; D と相互に排他) DDD - 元日から数えた日数 (001 ～ 366 の 3 桁; 他の日または月要素と相互に排他) 指定されていない要素には、省略時値の 1 が割り当てられます。日付形式の例を以下に示します。 "D-M-YYYY" "MM.DD.YYYY" "YYYYDDD"

表 10. 有効なファイル・タイプ修飾子 (LOAD) (続き)

修飾子	説明

dumpfile = *x*	*x* は、拒否された行が書き込まれる先の例外ファイルの (サーバー・ノードによる) 完全修飾名です。 1 レコード当たり、最大で 32KB のデータが書き込まれます。次の例は、ダンプ・ファイルを指定する方法を示すものです。
	`db2 load from data of del`
	`modified by dumpfile = /u/user/filename`
	`insert into table_name`
	注:
	1. 区分データベース環境では、並行して実行されるロード操作が同じファイルへ書き込まないように、パスはロード元のノードに対してローカルでなければなりません。
	2. ファイルの内容は、非同期バッファー・モードでディスクに書き込まれます。ロード操作が失敗したり割り込まれる場合には、ディスクにコミットされるレコード数が正確に知らされておらず、LOAD RESTART 後の一貫性が保証されていない可能性があります。 1 度の受け渡しで始動と完了を行うロード操作で、このファイルが完了したと見なされているだけの場合もあります。
	3. この修飾子は、複数のファイル拡張子の付くファイル名をサポートしていません。たとえば、次のような例を考えます。
	`dumpfile = /home/svtdbm6/DUMP.FILE`
	ロード・ユーティリティーでは、上のファイル名は受け入れられますが、
	`dumpfile = /home/svtdbm6/DUMP.LOAD.FILE`
	このファイル名は受け入れられません。

implieddecimal	暗黙指定されている小数点の位置が列定義によって決定され、値の終わりにあるとはみなされなくなります。たとえば値 `12345` は、`12345.00` ではなく、 `123.45` として DECIMAL(8,2) 列にロードされます。

表 10. 有効なファイル・タイプ修飾子 (LOAD) (続き)

修飾子	説明
timeformat="*x*"	*x* はソース・ファイル内の時刻の形式です。 [a] 有効な時刻要素は以下のとおりです。 H - 時 (12 時間系では 0 ～ 12 の 1 桁または 2 桁。 24 時間系では 0 ～ 24 の 1 桁または 2 桁) HH - 時 (12 時間系では 0 ～ 12 の 2 桁。 24 時間系では 0 ～ 24 の 2 桁; H と相互に排他) M - 分 (0 ～ 59 の 1 桁または 2 桁) MM - 分 (0 ～ 59 の 2 桁; M と相互に排他) S - 秒 (0 ～ 59 の 1 桁または 2 桁) SS - 秒 (0 ～ 59 の 2 桁; S と相互に排他) SSSSS - 夜の 12 時から数えた秒数 (00000 ～ 86399 の 5 桁; 他の時刻要素と相互に排他) TT - 正午の標識 (AM または PM) 指定されていない要素には、省略時値の 0 が割り当てられます。時刻形式の例を以下に示します。 "HH:MM:SS" "HH.MM TT" "SSSSS"

表 10. 有効なファイル・タイプ修飾子 (LOAD) (続き)

修飾子	説明

timestampformat="*x*"	*x* はソース・ファイル内のタイム・スタンプの形式です。 [a] 有効なタイム・スタンプ要素は以下のとおりです。
	
	YYYY - 年 (0000 ～ 9999 の 4 桁)
	M - 月 (1 ～ 12 の 1 桁または 2 桁)
	MM - 月 (1 ～ 12 の 2 桁;
	M と相互に排他)
	D - 日 (1 ～ 31 の 1 桁または 2 桁)
	DD - 日 (1 ～ 31 の 2 桁;
	D と相互に排他)
	DDD - 元日から数えた日数 (001 ～ 366 の 3 桁;
	他の月日要素と相互に排他)
	H - 時 (12 時間系では 0 ～ 12 の 1 桁または
	2 桁。
	24 時間系では 0 ～ 24 の 1 桁または 2 桁)
	HH - 時 (12 時間系では 0 ～ 12 の 2 桁。
	24 時間系では 0 ～ 24 の 2 桁;
	H と相互に排他)
	M - 分 (0 ～ 59 の 1 桁または 2 桁)
	MM - 分 (0 ～ 59 の 2 桁;
	M と相互に排他)
	S - 秒 (0 ～ 59 の 1 桁または 2 桁)
	SS - 秒 (0 ～ 59 の 2 桁;
	S と相互に排他)
	SSSSS - 夜の 12 時から数えた秒数
	(00000 ～ 86399 の 5 桁;
	他の時刻要素と相互に排他)
	UUUUUU - マイクロ秒
	(000000 ～ 999999 の 6 桁)
	TT - 正午の標識 (AM または PM)
	
	YYYY、M、MM、D、DD、または DDD 要素に値が指定されていない場合、省略時値の 1 が割り当てられます。それ以外の要素に値が指定されていない場合、省略時値の 0 が割り当てられます。タイム・スタンプ形式の例を以下に示します。
	
	`"YYYY/MM/DD HH:MM:SS.UUUUUU"`
	
	以下の例では、ユーザー定義の日付および時刻の形式を含んでいるデータを、 schedule という表にインポートする方法を示しています。
	
	`db2 import from delfile2 of del`
	`modified by timestampformat="yyyy.mm.dd hh:mm tt"`
	`insert into schedule`

noeofchar	任意指定のファイル終わり文字 x'1A' が、ファイルの終わりとして認識されません。通常の文字の場合のように処理が続行されます。
ASC (区切りなし ASCII) ファイル形式	

表 10. 有効なファイル・タイプ修飾子 (LOAD) (続き)

修飾子	説明
binarynumerics	数値 (ただし DECIMAL ではないもの) データは、文字表記ではなく、 2 進形式でなければなりません。これにより、コストのかかる変換を避けられます。 このオプションは、reclen オプションによって指定した固定長レコードを使用した、定位置 ASC でのみサポートされています。 noeofchar オプションが前提となります。 以下の規則が適用されます。 • BIGINT、INTEGER、および SMALLINT を除き、データ型間の変換は実行されません。 • データ長は、それぞれのターゲット列定義と一致していなければなりません。 • FLOAT は、IEEE 浮動小数点形式でなければなりません。 • ロード・ソース・ファイルの 2 進データは、ロード操作を実行するプラットフォームに関係なく、ビッグ・エンディアンであるとみなされます。 **注:** この修飾子の影響を受ける列のデータに、ヌルを指定することはできません。この修飾子を使用すると、ブランク (通常はヌルと解釈される) は、 2 進数の値であると解釈されます。
nochecklengths	nochecklengths が指定されていると、ソース・データの列定義がターゲット表の列のサイズを超えるものであっても、各行のロードが試行されます。このような行が正常にロードされるのは、コード・ページ変換でソース・データが縮小する場合です。たとえば、ソースにある 4 バイトの EUC データがターゲットで 2 バイトの DBCS データに縮小すれば、必要スペースは半分になります。このオプションが特に役立つのは、列の定義は不一致であるがソース・データが常に適合することが分かっている場合です。
nullindchar=*x*	*x* は単一文字です。この修飾子は、ヌル値を表す文字を *x* に変更します。 *x* の省略時値は Y です。[b] 文字が英字である場合、この修飾子は、EBCDIC データ・ファイルで大文字小文字を区別します。たとえば、ヌル標識文字が文字 N として指定されている場合は、 n もヌル標識として認識されます。

表 10. 有効なファイル・タイプ修飾子 (LOAD) (続き)

修飾子	説明

packeddecimal	binarynumerics 修飾子は DECIMAL フィールド・タイプを含まないため、パック 10 進数データを直接ロードします。
	
	このオプションは、reclen オプションによって指定した固定長レコードを使用した、定位置 ASC でのみサポートされています。 noeofchar オプションが前提となります。
	
	この符号ニブルでサポートされている値は、以下のとおりです。
	`+ = 0xC 0xA 0xE 0xF`
	`- = 0xD 0xB`
	
	注: この修飾子の影響を受ける列のデータに、ヌルを指定することはできません。この修飾子を使用すると、ブランク (通常はヌルと解釈される) は、 2 進数の値であると解釈されます。
	
	サーバー・プラットフォームに関係なく、ロード・ソース・ファイルの 2 進データのバイト配列はビッグ・エンディアンであるとみなされます。つまり、OS/2 または Windows オペレーティング・システムでこの修飾子を使用する場合、バイト配列を逆転させてはなりません。

reclen=*x*	*x* は、最大値 32 767 の整数です。各行ごとに *x* 個の文字が読み取られ、行の終わりを示すのに改行文字は使用されません。
striptblanks	データを可変長フィールドにロードする際に、後書きブランク・スペースを切り捨てます。このオプションを指定しないと、ブランク・スペースは保持されます。 このオプションは、 striptnulls と一緒に指定することはできません。これらのオプションは、相互に排他的です。 **注:** このオプションは以前の t オプションを置き換えるもので、後方互換性のためにのみサポートされています。
striptnulls	データを可変長フィールドにロードする際に、後書きヌル (0x00 文字) を切り捨てます。このオプションを指定しないと、ヌルは保持されます。 このオプションは、 striptblanks と一緒に指定することはできません。これらのオプションは、相互に排他的です。 **注:** このオプションは以前の padwithzero オプションを置き換えるもので、後方互換性のためにのみサポートされています。

表 10. 有効なファイル・タイプ修飾子 (LOAD) (続き)

修飾子	説明

zoneddecimal	BINARYNUMERICS 修飾子は DECIMAL フィールド・タイプを含まないため、ゾーン 10 進数データをロードします。このオプションは、RECLEN オプションによって指定した固定長レコードを使用した、定位置 ASC でのみサポートされています。 NOEOFCHAR オプションが前提となります。
	
	ハーフバイト符号値は、以下のいずれかです。
	`+ = 0xC 0xA 0xE 0xF`
	`- = 0xD 0xB`
	
	数字としてサポートされている値は `0x0` ～ `0x9` です。
	
	ゾーンとしてサポートされている値は `0x3` および `0xF` です。

	DEL (区切り付き ASCII) ファイル形式
chardel*x*	*x* は、単一文字のストリング区切り文字です。省略時値は、二重引用符 (") です。指定した文字は、二重引用符の代わりに、文字ストリングを囲む文字として使用されます。 [bc] 単一引用符 (') も、文字ストリングの区切り文字として指定できます。 `modified by chardel''`
coldel*x*	*x* は、単一文字の列区切り文字です。省略時値は、コンマ (,) です。指定した文字は、コンマの代わりに、列の終わりを示す文字として使用されます。 [bc]
datesiso	日付形式。すべての日付データ値を ISO 形式でロードします。
decplusblank	正符号文字。正の 10 進値の接頭部として、正符号 (+) ではなくブランク・スペースを使用します。省略時アクションでは、10 進値に正符号の接頭部が付けられます。
decpt*x*	*x* は、小数点文字としてピリオドの代わりに使用される単一の置換文字です。省略時値は、ピリオド (.) です。指定した文字は、小数点文字としてピリオドの代わりに使用されます。 [bc]

表 10. 有効なファイル・タイプ修飾子 (LOAD) (続き)

修飾子	説明

delprioritychar	区切り文字の現在の省略時優先順位は、レコード区切り文字、文字区切り文字、列区切り文字の順になっています。この修飾子は、区切り文字の優先順位を文字区切り文字、レコード区切り文字、列区切り文字と逆順にすることにより、以前の優先順位に依存する既存のアプリケーションを保護します。構文は次のとおりです。
	`db2 load ... modified by delprioritychar ...`
	たとえば、以下のような DEL データ・ファイルがあるとします。
	`"Smith, Joshua",4000,34.98<row delimiter>`
	`"Vincent,<row delimiter>, is a manager", ...`
	`... 4005,44.37<row delimiter>`
	`delprioritychar` 修飾子が指定されている場合、このデータ・ファイルは 2 行しかありません。 2 番目の <row delimiter> は 2 番目の行の最初のデータ列の一部と解釈されますが、最初と 3 番目の <row delimiter> は実レコードの区切り文字と解釈されます。この修飾子が指定されていないと、このデータ・ファイルは 3 行になり、各行は <row delimiter> によって区切られます。

dldel*x*	*x* は、単一文字の DATALINK 区切り文字です。省略時値は、セミコロン (;) です。指定した文字は、 DATALINK 値のフィールド間区切り文字としてセミコロンの代わりに使用されます。 DATALINK 値には 2 つ以上の副次値を指定できるので、この区切り文字が必要です。 bcd **注:** *x* は、行、列、または文字ストリングの区切り文字とは異なる文字にしてください。

keepblanks	タイプ CHAR、VARCHAR、LONG VARCHAR、または CLOB の各フィールドの前後のブランクを保持します。このオプションが指定されていない場合、文字区切り文字の外側にある前後のブランクはすべて除去され、ブランクになっているすべてのフィールドについては表にヌルが挿入されます。
	以下の例では、データ・ファイル内の前後のスペースをすべて保持しながら、 TABLE1 という表にデータをロードする方法を示しています。
	`db2 load from delfile3 of del`
	`modified by keepblanks`
	`insert into table1`

nodoubledel	二重になっている区切り文字の認識を抑止します。詳細については、371ページの『区切り文字についての制限事項』を参照してください。
IXF ファイル形式	

表 10. 有効なファイル・タイプ修飾子 (LOAD) (続き)

修飾子	説明
forcein	コード・ページが不一致でもデータを受け入れ、コード・ページ間の変換を抑止するようユーティリティーに指示します。 固定長ターゲット・フィールドは、そのデータが入るだけの十分な大きさがあるかどうかチェックされます。 nochecklengths が指定されていると、チェックは実行されず、各行のロードが試行されます。
nochecklengths	nochecklengths が指定されていると、ソース・データの列定義がターゲット表の列のサイズを超えるものであっても、各行のロードが試行されます。このような行が正常にロードされるのは、コード・ページ変換でソース・データが縮小する場合です。たとえば、ソースにある 4 バイトの EUC データがターゲットで 2 バイトの DBCS データに縮小すれば、必要スペースは半分になります。このオプションが特に役立つのは、列の定義は不一致であるがソース・データが常に適合することが分かっている場合です。

表 10. 有効なファイル・タイプ修飾子 (LOAD) (続き)

修飾子	説明

注:
1. サポートされていないファイル・タイプを MODIFIED BY オプションで使用しようとしても、ロード・ユーティリティーは警告を出しません。この場合、ロード操作が失敗し、エラー・コードが戻されます。
2. [a] 日付形式ストリングは必ず二重引用符で囲まなければなりません。フィールド区切り文字には、 a ～ z、A ～ Z、および 0 ～ 9 を含めることはできません。フィールド区切り文字は、 DEL ファイル形式の文字区切り文字またはフィールド区切り文字と同じにすることはできません。要素の開始および終了位置が明らかな場合、フィールド区切り文字は任意指定です。開始および終了位置が明らかでないのは、項目が長さが一定でない D、H、M、または S などの要素が使用されている場合です (修飾の仕方によって異なります)。
タイム・スタンプ形式の場合、月の記述子と分の記述子のどちらも文字 M を使用するため、区別が明白であるように注意しなければなりません。月のフィールドは他の日付フィールドに隣接していなければなりません。分のフィールドは他の時刻フィールドに隣接していなければなりません。以下のタイム・スタンプ形式は、あいまいな形式の例です。
`"M" (月と分のどちらかがはっきりしない)`
`"M:M" (どちらが何を表しているのか分からない)`
`"M:YYYY:M" (どちらも月として解釈される)`
`"S:M:YYYY" (時刻値と日付値の両方に隣接している)`
あいまいな形式になっている場合、ユーティリティーはエラー・メッセージを報告し、操作は失敗します。
以下のタイム・スタンプ形式では、何を指しているのかがはっきりしています。
`"M:YYYY" (月)`
`"S:M" (分)`
`"M:YYYY:S:M" (月....分)`
`"M:H:YYYY:M:D" (分....月)`
注: 二重引用符や円記号などの文字の前には、拡張文字 (たとえば、¥) を付けなければなりません。
3. [b] この文字は、ソース・データのコード・ページで指定してください。
文字コード・ポイント (文字記号ではない) を、 xJJ または 0xJJ の構文 (JJ はコード・ポイントの 16 進表記) を使用して指定できます。たとえば、# 文字を列区切り文字として指定するには、以下のいずれかを使用します。
`... modified by coldel# ...`
`... modified by coldel0x23 ...`
`... modified by coldelX23 ...`
4. [c] 371ページの『区切り文字についての制限事項』には、代替区切り文字として使用できる文字に適用される制限事項がリストされています。
5. [d] DATALINK 区切り文字が URL 構文内で有効な文字である場合でも、ロード操作の効力範囲内では、その特別な意味はなくなります。	

参照資料

75ページの『db2LoadQuery - ロードの照会』

483ページの『sqluvqdp - 表の表スペースの静止』

sqlurcon - 調整

表の DATALINK データ用のファイルへの参照を妥当性検査します。ファイルへの参照を設定できない行は、例外表 (指定してある場合) へコピーされ、入力表で修正されます。

許可

以下のいずれかです。

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

必須接続

データベース

API 組み込みファイル

sqlutil.h

C API 構文

```
/* File: sqlutil.h */
/* API: Reconcile */
/* ... */
SQL_API_RC SQL_API_FN
  sqlurcon (
    char * pTableName,
    char * pExTableName,
    char * pReportFileName,
    void * pReserved,
    struct sqlca * pSqlca);
/* ... */
```

汎用 API 構文

```
/* File: sqlutil.h */
/* API: Reconcile */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgrcon (
    unsigned short TableNameLen,
    char * pTableName,
    unsigned short ExTableNameLen,
    char * pExTableName,
    unsigned short ReportFleNameLen,
    char * pReportFileName,
    void * pReserved,
    struct sqlca * pSqlca);
/* ... */
```

API パラメーター

TableNameLen
入力。表名の長さを示す 2 バイトの無符号整数 (バイト単位)。

pTableName
入力。調整を実行する表を指定します。別名、完全修飾、または未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。未修飾の表名を指定すると、その表は現行許可 ID で修飾されます。

ExTableNameLen
入力。例外表の名前の長さを示す 2 バイトの無符号整数 (バイト単位)。

pExTableName
入力。 DATALINK 値へのリンク障害が発生している行がコピーされる例外表を指定します。

ReportFileNameLen
入力。レポート・ファイル名の長さを示す 2 バイトの無符号整数 (バイト単位)。

pReportFileName
入力。調整時にリンク解除されるファイルについての情報を含むファイルを指定します。この名前は、完全修飾名である必要があります (たとえば、 /u/johnh/report)。調整ユーティリティーは、指定されたファイル名に .ulk という拡張子を追加します (たとえば、report.ulk)。

pReserved
将来の利用のために予約されています。

pSqlca
出力。 *sqlca* 構造を指すポインターです。 この構造に関する詳細は、520ページの『SQLCA』を参照してください。

使用上の注意

調整時には、表データに従って存在しているファイルで、データ・リンク・ファイル・マネージャーのメタデータに従って存在しているわけではないものをリンクしようとします。ただし、このことはこれ以外に競合がない場合です。

調整は、表内のすべての DATALINK データに関して実行されます。ファイル参照を再設定できない場合は、違反の行が例外表 (指定してある場合) に挿入されます。これらの行は、入力表からは削除されません。ファイル参照の保全性を確実にするために、問題の DATALINK 値はヌル化されます。列がヌル可能ではないものとして定義されている場合、 DATALINK 値はゼロの長さの URL に置き換えられます。

例外表を指定していない場合、ファイル参照を再設定できない DATALINK 列値は、列 ID やコメントとともに例外レポート・ファイル (<*pReportFileName*>.exp) にコピーされます。

調整処理の終了時に、この表はデータ・リンク調整保留 (DRP) 状態から解放されます。

sqlureot - 表の再編成

断片化されたデータを消去するために行を再構成したり、情報を圧縮したりすることによって、表を再編成します。

効力範囲

この API は、ノード・グループ内のすべてのノードに影響を与えます。

許可

以下のいずれかです。

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

必須接続

データベース

API 組み込みファイル

sqlutil.h

C API 構文

```
/* File: sqlutil.h */
/* API: Reorganize Table */
/* ... */
SQL_API_RC SQL_API_FN
  sqlureot (
    _SQLOLDCHAR * pTableName,
    _SQLOLDCHAR * pIndexName,
    _SQLOLDCHAR * pTablespace,
    struct sqlca * pSqlca);
/* ... */
```

汎用 API 構文

```
/* File: sqlutil.h */
/* API: Reorganize Table */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgreot (
    unsigned short TablespaceLen,
    unsigned short IndexNameLen,
    unsigned short TableNameLen,
    struct sqlca * pSqlca,
    _SQLOLDCHAR * pTablespace,
    _SQLOLDCHAR * pIndexName,
    _SQLOLDCHAR * pTableName);
/* ... */
```

API パラメーター

TablespaceLen
入力。表スペース・ストリングの長さを示す 2 バイトの無符号整数 (バイト単位)。表スペースが指定されていない場合は、ゼロに設定してください。

IndexNameLen
入力。索引名の長さを示す 2 バイトの無符号整数 (バイト単位)。索引が指定されていない場合は、ゼロに設定してください。

TableNameLen
入力。表名の長さを示す 2 バイトの無符号整数 (バイト単位)。

pSqlca
出力。 *sqlca* 構造を指すポインターです。 この構造に関する詳細は、520ページの『SQLCA』を参照してください。

pTablespace
入力。呼び出し側が表の再編成時に 2 次的な作業域を必要とする場合の、システム一時表スペースの名前を含むストリング。ヌルにすることもできます。

pIndexName
入力。ユーザー表の再編成時に使用される索引の完全修飾名。再編成される表にあるレコードは、この索引に基づいて物理的に再配列されます。このパラメーターをヌルに設定すると、不特定な順序でデータが再編成されます。

pTableName
入力。再編成する表の名前。別名を指定することもできます。ただし、下位のサーバーの場合には、必ず、表の完全修飾名を使用しなければなりません。

REXX API 構文

```
REORG TABLE tablename [INDEX iname] [USE tablespace_id]
```

REXX API パラメーター

tablename
表の完全修飾名。

iname 表の再編成に使用される索引の完全修飾名。索引名が指定されていない場合は、不特定な順序でデータが再編成されます。

tablespace_id
システム一時表スペースの名前。

サンプル・プログラム

C	¥sqllib¥samples¥c¥dbstat.sqc
COBOL	¥sqllib¥samples¥cobol¥dbstat.sqb
REXX	¥sqllib¥samples¥rexx¥dbstat.cmd

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。

何度も修正されたために使用しないデータが残り、アクセスのパフォーマンスが著しく遅くなっている表は、再編成の対象候補になります。表に再編成が必要であるかどうかを判別するには、 *コマンド解説書* の『REORGCHK』を使用してください。 REORGANIZE TABLE を呼び出す前に、すべてのデータベース操作を完了させ、すべてのロックを解除するようにしてください。これは、WITH HOLD でオープンしているすべてのカーソルをクローズした後で COMMIT を発行するか、または ROLLBACK を発行することによって行われます。表の再編成が済んだら、 477ページの『sqlustat - 統計の実行』を使用して表統計を更新してください。さらに、114ページの『sqlarbnd - 再バインド』を使用して、その表を使用しているパッケージを再バインドしてください。

表がいくつかのノードに区分化されており、表の再編成がいずれかのノードで失敗した場合には、失敗したノードだけが表の再編成をロールバックします。

注: 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回復に使用します。

索引の名前を指定した場合、データベース・マネージャーは、索引内の配列を基にし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*、クラスター化索引が存在する場合、データはクラスター化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空き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。

REORGANIZE TABLE を視点に関して使用することはできません。

DMS 表が存在す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は、 REORGANIZE TABLE を DMS 表に関して使用することはできません。

表の再編成に続いて表スペースのロールフォワード回復を完了させるには、データと LONG 表スペースの両方で、ロールフォワードが使用可能でなければなりません。

表に COMPACT オプションを使用しない LOB 列がある場合、 LOB DATA 記憶オブジェクトは、表の再編成後、かなり大きくなる可能性があります。これは、行が再編成された順序と、使用された (SMS/DMS) 表スペースのタイプが原因で起こる可能性があります。

DB2 バージョン 2 サーバーは、表の再編成を求める下位クライアントの要求をサポートしません。バージョン 2 以前のサーバーは表スペースをサポートしないため、バージョン 2 クライアントが下位サーバーと一緒に使用されるときには、 *pTablespace* パラメーターはバージョン 1 の *path* パラメーターとして扱われます。

バージョン 2 クライアントがバージョン 2 サーバー上の表を再編成することを要求し、その再編成の対象として、 *pTablespace* パラメーターに一時表スペースではなくパスが含まれる場合 (たとえば、バージョン 2 クライアント上で実行されている古いアプリケーションが一時ファイル・パスを指定している場

合)、 REORG は作業ファイルを入れるシステム一時表スペースをユーザーの代わりに選択します。パス区切り文字 (/ または ¥) を含む有効なシステム一時表スペース名を指定してはなりません。これは、その名前が一時パス (バージョン 2 以前の要求) として解釈され、 REORG がユーザーの代わりにシステム一時表スペースを選択するためです。

REORGANIZE TABLE では、索引拡張に基づく索引は使用できません。

参照資料

114ページの『sqlarbnd - 再バインド』

477ページの『sqlustat - 統計の実行』

sqlurestore - データベースの復元

345ページの『sqlubkp - データベースのバックアップ』を使用してバックアップが取られていたデータベースが損傷または破壊された場合に、そのデータベースを再作成します。復元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復元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復元することが可能です。

このユーティリティーは、以前の 2 つのリリースで作成した DB2 データベースを復元するためにも使用できます。

このユーティリティーでは、表スペース・レベルのバックアップから復元することもできます。

注: この API は **sqlurst** (DB2 バージョン 5.0) を取り替えるもので、 DB2 データ・リンク・マネージャーとともに使う必要があります。 DB2 データ・リンク・マネージャー機能が必要ない場合は、**sqlurst** を使用できます。

効力範囲

この API は、それが呼び出されたノードにのみ影響を与えます。

許可

既存のデータベースに復元するには、次のいずれかが必要です。

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

新規のデータベースに復元するには、次のいずれかが必要です。

- *sysadm*
- *sysctrl*

必須接続

既存のデータベースに復元する場合、データベース。この API を呼び出せば、指定したデータベースへの接続が自動的に確立されます。

新規のデータベースに復元する場合、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。

現行のインスタンス (**DB2INSTANCE** 環境変数の値で定義) とは異なるインスタンスで新規のデータベースへの復元を行うには、まず、新規のデータベースを存在させるインスタンスに接続することが必要です。

API 組み込みファイル

sqlutil.h

C API 構文

```
/* File: sqlutil.h */
/* API: Restore Database */
/* ... */
SQL_API_RC SQL_API_FN
  sqlurestore (
    char * pSourceDbAlias,
    char * pTargetDbAlias,
    sqluint32 BufferSize,
    sqluint32 RollforwardMode,
    sqluint32 DatalinkMode,
    sqluint32 RestoreType,
    sqluint32 RestoreMode,
    sqluint32 CallerAction,
    char * pApplicationId,
    char * pTimestamp,
    char * pTargetPath,
    sqluint32 NumBuffers,
    char * pReportFile,
    struct sqlu_tablespace_bkrst_list * pTablespaceList,
    struct sqlu_media_list * pMediaSourceList,
    char * pUserName,
    char * pPassword,
    void * pReserved2,
    sqluint32 VendorOptionsSize,
    void * pVendorOptions,
    sqluint32 Parallelism,
    void * pRestoreInfo,
    void * pContainerPageList,
    void * pReserved3,
    struct sqlca * pSqlca);
/* ... */
```

汎用 API 構文

```
/* File: sqlutil.h */
/* API: Restore Database */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgrestore (
   unsigned short SourceDbAliasLen,
   unsigned short TargetDbAliasLen,
   unsigned short TimestampLen,
   unsigned short TargetPathLen,
   unsigned short UserNameLen,
   unsigned short PasswordLen,
   unsigned short ReportFileLen,
   unsigned short Reserved2Len,
   char * pSourceDbAlias,
   char * pTargetDbAlias,
   sqluint32 BufferSize,
   sqluint32 RollforwardMode,
   sqluint32 DatalinkMode,
   sqluint32 RestoreType,
   sqluint32 RestoreMode,
   sqluint32 CallerAction,
   char * pApplicationId,
   char * pTimestamp,
   char * pTargetPath,
   sqluint32 NumBuffers,
   char * pReportFile,
   struct sqlu_tablespace_bkrst_list * pTablespaceList,
   struct sqlu_media_list * pMediaSourceList,
   char * pUserName,
   char * pPassword,
   void * pReserved2,
   sqluint32 VendorOptionsSize,
   void * pVendorOptions,
   sqluint32 Parallelism,
   unsigned short RestoreInfoSize,
   void * pRestoreInfo,
   unsigned short ContainerPageListSize,
   void * pContainerPageList,
   void * pReserved3,
   struct sqlca * pSqlca);
/* ... */
```

API パラメーター

SourceDbAliasLen

入力。ソース・データベースの別名の長さを示す 2 バイトの無符号整数 (バイト単位)。

TargetDbAliasLen
入力。宛先データベースの別名の長さを示す 2 バイトの無符号整数 (バイト単位)。宛先データベースの別名が指定されていない場合は、ゼロに設定してください。

TimestampLen
入力。タイム・スタンプの長さを示す 2 バイトの無符号整数 (バイト単位)。タイム・スタンプが提供されていない場合は、ゼロに設定してください。

TargetPathLen
入力。ターゲット・ディレクトリーの長さを示す 2 バイトの無符号整数 (バイト単位)。ターゲット・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。

UserNameLen
入力。ユーザー名の長さを示す 2 バイトの無符号整数 (バイト単位)。ユーザー名が提供されていない場合は、ゼロに設定してください。

PasswordLen
入力。パスワードの長さを示す 2 バイトの無符号整数 (バイト単位)。パスワードが提供されていない場合は、ゼロに設定してください。

ReportFileLen
入力。レポート・ファイル名の長さを示す 2 バイトの無符号整数 (バイト単位)。レポート・ファイル名が提供されていない場合は、ゼロに設定してください。

Reserved2Len
入力。予約済み領域の長さを示す 2 バイトの無符号整数 (バイト単位)。ゼロに設定してください。

pSourceDbAlias
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。

pTargetDbAlias
入力。宛先データベースの別名を含むストリング。このパラメーターがヌルの場合、*pSourceDbAlias* の別名が使用されます。

BufferSize
入力。バッファー・サイズを 4KB 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。省略時値は 1024 単位です。

注: 復元用に入力するバッファー・サイズは、バックアップ・イメージを作成するのに使用するバッファー・サイズと同じか、または整数倍でなければなりません。

RollforwardMode

入力。復元の終了時に、データベースをロールフォワード保留状態に置くかどうかを示します。有効な値 (sqlutil で定義) は、以下のとおりです。

SQLUD_ROLLFWD

データベースの復元が成功した後、データベースをロールフォワード保留状態にします。

SQLUD_NOROLLFWD

データベースの復元が成功した後、データベースをロールフォワード保留状態にしません。

復元が成功した後で、データベースがロールフォワード保留状態にある場合は、 465ページの『sqluroll - データベースのロールフォワード』を実行してからでなければデータベースを使用できません。

DatalinkMode

入力。 DATALINK 列を持つ表が DataLink_Reconcile_Pending (DRP) 状態に入れられるかどうか、およびリンクされたファイルの調整が実行されるかどうかを指定します。有効な値 (sqlutil で定義) は、以下のとおりです。

SQLUD_DATALINK

調整操作を実行します。定義された DATALINK 列を含む表では、RECOVERY YES オプションを指定する必要があります。

SQLUD_NODATALINK

調整操作を実行しません。 DATALINK 列を持つ表は、DataLink_Reconcile_Pending (DRP) 状態に入れられます。定義された DATALINK 列を含む表では、RECOVERY YES オプションを指定する必要があります。

RestoreType

入力。復元のタイプを指定します。有効な値 (sqlutil で定義) は、以下のとおりです。

SQLUD_FULL

バックアップ・イメージからすべてのものを復元します。このタイプを指定した場合、復元はオフラインで実行されます。

SQLUD_ONLINE_TABLESPACE
表スペース・レベルのバックアップだけを復元します。このタイプを指定した場合、復元はオンラインで実行されます。

SQLUD_HISTORY
回復活動記録ファイルだけを復元します。

RestoreMode
入力。復元がオフラインとオンラインのどちらで実行されるかを指定します。有効な値 (sqlutil で定義) は、以下のとおりです。

SQLUD_OFFLINE
オフライン復元操作を実行します。

SQLUD_ONLINE
オンライン復元操作を実行します。

CallerAction
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 (sqlutil で定義) は、以下のとおりです。

SQLUD_RESTORE
復元を開始します。

SQLUD_NOINTERRUPT
復元を開始します。復元を自動実行するよう指定します。通常ユーザーの介入を要求するシナリオは、呼び出し側への最初の戻りなしに試行されるか、エラーを生成します。この呼び出し側アクションは、たとえば、復元に必要な媒体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。

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

SQLUD_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、復元を終了します。

SQLUD_DEVICE_TERMINATE
復元ユーティリティー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、復元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復元を再び呼び出すと、警告が生成される原因となった装置が使用装置のリストから除外されます。

SQLUD_PARM_CHECK
復元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。

SQLUD_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。

復元 API の最初の呼び出しでは、 *CallerAction* を必ず SQLUD_RESTORE、 SQLUD_NOINTERRUPT、 SQLUD_RESTORE_STORDEF、または SQLUD_PARM_CHECK に設定してください。

pApplicationId
出力。長さ SQLU_APPLID_LEN+1 (sqlutil で定義) のバッファーを提供します。復元 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。この識別子を指定してデータベース・システム・モニター API を呼び出せば、アプリケーションのいくつかの局面をモニターできます。

pTimestamp
入力。バックアップ・イメージのタイム・スタンプを示すストリング。指定したソース内にバックアップ・イメージが 1 つしかない場合、このフィールドはオプションです。

pTargetPath
入力。宛先データベースのディレクトリーの相対または完全修飾名を含むストリング。復元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。

NumBuffers
入力。復元に使用するバッファーの数。

pReportFile
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。復元時にリンク解除されるファイル (高速調整の結果) が報告されます。

pTablespaceList
復元される 1 つまたは複数の表スペースを指定します。表スペース・バックアップ・イメージから、バックアップ・イメージのサブセットまたは表スペースを復元する場合に使用されます。

以下の制限が適用されます。

- データベースは回復可能でなければなりません。つまり、*ログ保存* またはユーザー出口が使用可能になっていなければなりません。
- 復元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、表スペース復元機能を使用して、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。
- この関数は、OS/2 上のユーザー出口からの復元時には利用不能です。
- ロールフォワード・ユーティリティーは、MPP 環境で復元される表スペースが、同じ表スペースを含む他のノードと必ず同期化されるようにします。

注: バックアップ後に名前が変更された表スペースを復元する場合、復元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。

pMediaSourceList

入力。バックアップ・イメージのソース媒体。 592ページの『SQLU-MEDIA-LIST』を参照してください。呼び出し側がこの構造に提供する必要のある情報は、 *media_type* フィールドの値によって異なります。このフィールドに有効な値 (`sqlutil` で定義) は、以下のとおりです。

SQLU_LOCAL_MEDIA

ローカル装置 (テープ、ディスクまたはディスケットの組み合わせ)。 *sqlu_media_entry* 構造のリストを提供してください。OS/2 または Windows オペレーティング・システムでは、項目をテープ装置名ではなく、ディレクトリー・パスだけにすることができます。

SQLU_TSM_MEDIA

TSM。追加の入力は必要ありません。 DB2 に付属の TSM 共用ライブラリーが使用されます。別のバージョンの TSM が必要な場合には、 `SQLU_OTHER_MEDIA` を使用し、共用ライブラリー名を入力してください。

SQLU_OTHER_MEDIA

ベンダー製品。 *sqlu_vendor* 構造に共用ライブラリー名を提供します。

SQLU_USER_EXIT
ユーザー出口。追加の入力は必要ありません (OS/2 でのみ使用可能です)。

詳細については、*管理の手引き* を参照してください。

pUserName
入力。接続に使用されるユーザー名を含むストリング。

pPassword
入力。接続用の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。

pReserved2
将来の利用のために予約されています。

VendorOptionsSize
入力。ベンダー・オプション・フィールドの長さ。 65535 バイト以下でなければなりません。

pVendorOptions
入力。情報をアプリケーションからベンダー関数に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、バイト逆転が行われず、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。

Parallelism
入力。区画内並列処理度 (バッファー・マニピュレーターの数)。

RestoreInfoSize
将来の利用のために予約されています。

pRestoreInfo
将来の利用のために予約されています。

ContainerPageListSize
将来の利用のために予約されています。

pContainerPageList
将来の利用のために予約されています。

pReserved3
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です。 この構造に関する詳細は、520ページの『SQLCA』を参照してください。

REXX API 構文

```
RESTORE DATABASE source-database-alias [USING :value] [USER username USING password]
[TABLESPACE :tablespacenames] [ONLINE | HISTORY FILE ]
[LOAD shared-library [OPTIONS vendor-options] [OPEN num-sessions SESSIONS] |
 FROM :source-area | USE TSM [OPEN num-sessions SESSIONS] | USER_EXIT]
[TAKEN AT timestamp] [TO target-directory] [INTO target-database-alias]
[ACTION caller-action] [WITH num-buffers BUFFERS] [BUFFERSIZE buffer-size]
[WITHOUT ROLLING FORWARD] [PARALLELISM parallelism-degree]
```

REXX API パラメーター

source-database-alias
データベースのバックアップ・イメージが取られたソース・データベースの別名。

value データベースの復元情報が戻される複合 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 変数内の要素数 (通常 1)

XXX.1 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アプリケーション ID

username
接続に使用されるユーザー名を識別します。

password
ユーザー名の認証に使用されるパスワード。

tablespacenames
復元される表スペースのリストを含む複合 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 復元される表スペースの数

XXX.1 最初の表スペース名

XXX.2 2 番目の表スペース名

XXX.3 以降、3 番目、4 番目 ... と続きます。

HISTORY FILE
バックアップから活動記録ファイルを復元することを指定します。

shared-library
使用されるベンダーの入出力復元関数を含む共用ライブラリーの名前 (OS/2 または Windows オペレーティング・システムでは DLL)。これには、全パスを入れてもかまいません。全パスが指定されない場合は、省略時値として、ユーザー出口プログラムが存在するパスが使用されます。

vendor-options
ベンダー関数によって必要とされる情報。

num-sessions
TSM またはベンダー製品と共に使用する入出力セッションの数。

source-area
バックアップ・イメージが存在するディレクトリーまたは装置を示す複合 REXX ホスト変数。省略時値は現行ディレクトリーです。 OS/2 または Windows オペレーティング・システムでは、項目をテープ装置名ではなく、ディレクトリー・パスだけにすることができます。

timestamp
データベース・バックアップのタイム・スタンプ。

target-directory
宛先データベースのディレクトリー。

target-database-alias
宛先データベースの別名。宛先データベースが存在しない場合には、作成されます。

caller-action
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。

SQLUD_RESTORE
復元を開始します。

SQLUD_NOINTERRUPT
復元を開始します。復元を自動実行するよう指定します。通常ユーザーの介入を要求するシナリオは、呼び出し側への最初の戻りなしに試行されるか、エラーを生成します。この呼び出し側アクションは、たとえば、復元に必要な媒体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。

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

SQLUD_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、復元を終了します。

SQLUD_DEVICE_TERMINATE
復元ユーティリティー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、復元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復元を再び呼び出すと、警告が生成される原因となった装置が使用装置のリストから除外されます。

SQLUD_PARM_CHECK
復元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。

SQLUD_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。

num-buffers
使用されるバックアップ・バッファーの数。

buffer-size
バックアップ・バッファー・サイズ (4KB の割り振り単位)。最小値は 16 単位です。

parallelism-degree
バッファー・マニピュレーターの数。

サンプル・プログラム

C	¥sqllib¥samples¥c¥backrest.c
COBOL	¥sqllib¥samples¥cobol¥backrest.cbl

使用上の注意

オフラインの復元の場合、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。復元されるデータベースにアプリケーション (呼び出し側のアプリケーションを含む) がすでに接続している場合、このユーティリティー

は失敗します。さらに、オペレーティング・システムの復元ユーティリティーが、復元の実行に使用されており、アプリケーション (呼び出し側のアプリケーションを含む) が同じワークステーション上の任意のデータベースにすでに接続されている場合、要求は失敗します。接続が成功すると、 API は復元が完了するまで他のアプリケーションを締め出します。

現行のデータベース構成ファイルは、それが使用不能でない限り、バックアップ・コピーによって置換されません。ファイルが置換される場合には、警告メッセージが戻されます。

データベースまたは表スペースは、 345ページの『sqlubkp - データベースのバックアップ』を使用してバックアップされていなければなりません。

呼び出し側アクションが SQLUD_NOINTERRUPT の場合、復元はアプリケーションにプロンプトを出すことなく継続されます。呼び出し側のアクションが SQLUD_RESTORE で、ユーティリティーが既存のデータベースに復元しようとしている場合、ユーティリティーは、何らかのユーザー介入を要求するメッセージとともに、アプリケーションに制御を戻します。ユーザーとの対話の処理が終了した後、後続の呼び出しにおいて処理が継続される (SQLUD_CONTINUE) か、終了 (SQLUD_TERMINATE) されるかを示す呼び出し側アクションを設定して、RESTORE DATABASE を再び呼び出します。このユーティリティーは処理を終了させ、 *sqlca* で SQLCODE を戻します。

終了時に装置をクローズするには、呼び出し側アクションを SQLUD_DEVICE_TERMINATE に設定してください。たとえば、2 つのテープ装置を使用して 3 つのテープ・ボリュームから復元を行おうとする場合、テープの 1 つが復元されると、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、媒体装置の終了を通知する呼び出し側アクション SQLUD_DEVICE_TERMINATE を指定して API に戻ります。これで、デバイス・ドライバーは終了しますが、復元に関連する残りの装置は、復元セット内の全セグメントが復元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、復元セット内のセグメントの数が最後の媒体装置に置かれます)。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検査を実行したい場合には、呼び出し側アクションを SQLUD_PARM_CHECK に設定してください。

宛先変更した復元を実行する場合には、呼び出し側アクションを SQLUD_RESTORE_STORDEF に設定し、 145ページの『sqlbstsc - 表スペース・コンテナーの設定』との組み合わせで使用してください。詳細については、*管理の手引き* を参照してください。

エラーが発生すると、このユーティリティーは終了し、 *sqlca* 構造でエラーを戻します。

データベース復元の重要な段階でシステム障害が発生した場合、正常な復元が実行されるまで、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回復に使用できるよう構成されておらず、さらにログ保存パラメーターとユーザー出口パラメーターのいずれかが使用可能になっている、使用可能な現行の構成ファイルがある場合、ユーザーは、復元に続いて、データベースの新しいバックアップをとるか、またはこれらのパラメーターを使用不能にしてから、データベースに接続することが必要です。

復元が失敗すると、復元されたデータベースは除去されませんが (既存のデータベース以外への復元の場合を除き)、使用不能になります。

バックアップ上の回復活動記録ファイルを復元するように復元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回復活動記録ファイル上に復元されます。事実上、復元されるバックアップの後に活動記録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、活動記録ファイルを新規またはテスト・データベースに復元させることにより、実行された更新を破棄することなく、活動記録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回復が使用可能だった場合には、 **sqlurestore** の実行が成功した後に **sqluroll** 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。データベースが回復可能な場合、復元の完了後に、省略時解釈で保留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、呼び出し側が復元後のデータベースのロールフォワードを必要としない場合、*RollforwardMode* パラメーターを SQLUD_NOROLLFWD に設定することができます。これにより、復元後にデータベースがすぐに使用可能になります。バック

アップ・イメージがオンラインで作成されている場合は、呼び出し元は復元が完了した時点で、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。

参照資料

145ページの『sqlbstsc - 表スペース・コンテナーの設定』

262ページの『sqlemgdb - データベースの移行』

327ページの『sqlfxdb - データベース構成の入手』

345ページの『sqlubkp - データベースのバックアップ』

465ページの『sqluroll - データベースのロールフォワード』

sqlurlog - ログの非同期読み取り

呼び出し側が DB2 共通サーバーのデータベース・ログから特定のログ・レコードを抽出できるようにします。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。この API は、回復可能データベース・ログがある (構成パラメーター LOGRETAIN または USEREXIT がオンにされている) データベース上でのみ使用できます。

許可

以下のいずれかです。

- *sysadm*
- *dbadm*

必須接続

データベース

API 組み込みファイル

sqlutil.h

C API 構文

```
/* File: sqlutil.h */
/* API: Asynchronous Read Log */
/* ... */
SQL_API_RC SQL_API_FN
  sqlurlog (
    sqluint32 CallerAction,
    SQLU_LSN * pStartLsn,
    SQLU_LSN * pEndLsn,
    char * pLogBuffer,
    sqluint32 LogBufferSize,
    SQLU_RLOG_INFO * pReadLogInfo,
    struct sqlca * pSqlca);
/* ... */
```

API パラメーター

CallerAction
入力。実行するアクションを指定します。

SQLU_RLOG_READ
開始ログ順序番号から終了ログ順序番号までデータベース・ログを読み取り、この範囲内にある伝搬可能なログ・レコードをすべて戻します。

SQLU_RLOG_READ_SINGLE
開始ログ順序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。

SQLU_RLOG_QUERY
データベース・ログを照会します。照会した結果は、SQLU_RLOG_INFO 構造 (597ページの『SQLU-RLOG-INFO』を参照) を介して戻されます。

pStartLsn
入力。開始ログ順序番号は、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。

pEndLsn
入力。終了ログ順序番号は、ログの読み取りを終了する相対バイト・アドレスを指定します。この値は、*startLsn* の値より大きくなければなりません。実際のログ・レコードの終わりである必要はありません。

pLogBuffer
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 バイトのログ順序番号 (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。

LogBufferSize
出力。バイト単位でログ・バッファーのサイズを指定します。

pReadLogInfo
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。この構造の詳細については、597ページの『SQLU-RLOG-INFO』を参照してください。

pSqlca
出力。 *sqlca* 構造を指すポインター。この構造の詳細については、520ページの『SQLCA』を参照してください。

サンプル・プログラム

C ¥sqllib¥samples¥c¥asynrlog.sqc

使用上の注意

要求されるアクションがログの読み取りであれば、呼び出し側はログ順序番号の範囲と、ログ・レコードを保持するバッファーを提供します。
ASYNCHRONOUS READ LOG API は、要求された LSN の範囲にあるログを順番に読み取ります。さらに、DATA CAPTURE オプションが CHANGES である表に関連したログ・レコードと、現在活動状態にあるログの情報が入った SQLU_RLOG_INFO 構造を戻します。要求されたアクションが照会であれば、API は、現在活動状態にあるログの情報が入った SQLU_RLOG_INFO 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 (SQLU-RLOG-INFO) 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。現在活動状態にあるログの終わりは、ログ情報の読み取り構造の curActiveLSN メンバーに入ります。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。

- curActiveLSN の値
- 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

ログ情報の読み取り構造の詳細については、 597ページの『SQLU-RLOG-INFO』を参照してください。

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。 **sqlurlog** によって戻される各種の DB2 共通サーバー・ログ・レコードについては、 685ページの『付録F. DB2 共通サーバー・ログ・レコード』に記載されています。

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、SQLU-RLOG-INFO で戻された、最後に読み取られたレコードの LSN に 1 を追加します。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実行依頼します。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、現在活動状態にあるログが最後まで読み取られたことを示します。

sqluroll - データベースのロールフォワード

データベース・ログ・ファイルに記録されたトランザクションを適用することによって、データベースを回復します。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復元された後、あるいは媒体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。ロールフォワード回復を用いてデータベースを回復するには、前もってデータベースが回復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 *logretain*、 *userexit*、あるいはこの両方がオンに設定されていなければなりません)。

効力範囲

複数ノード環境では、この API はカタログ・ノードからのみ呼び出すことができます。データベースまたは表スペースの、特定時を指定したロールフォワード呼び出しは、 `db2nodes.cfg` ファイルにリストされているすべてのノードに影響を与えます。ログの終わりを指定したデータベースまたは表スペース・ロールフォワード呼び出しは、指定されたノードに影響を与えます。ノードが 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ノードに影響を与えます。特定のノードでロールフォワードが必要ない場合、そのノードは無視されます。

許可

以下のいずれかです。

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

必須接続

なし。この API は、データベース接続を確立します。

API 組み込みファイル

sqlutil.h

C API 構文

```
/* File: sqlutil.h */
/* API: Rollforward Database */
/* ... */
SQL_API_RC SQL_API_FN
  sqluroll (
    struct rfwd_input * pRfwdInput,
    struct rfwd_output * pRfwdOuput,
    struct sqlca * pSqlca);
/* ... */
```

汎用 API 構文

```
/* File: sqlutil.h */
/* API: Rollforward Database */
/* ... */
SQL_API_RC SQL_API_RN
  sqlgroll (
    struct grfwd_input * grfwdin,
    struct rfwd_output * rfwdout,
    struct sqlca * sqlca);

SQL_STRUCTURE grfwd_input
{
   unsigned short DbAliasLen,
   unsigned short StopTimeLen,
   unsigned short UserNameLen,
   unsigned short PasswordLen,
   unsigned short OverflowLogPathLen,
   unsigned short ReportFileLen,      /* NOTE: This parameter is no longer used */
                                      /*       for the DB2 Data Links Manager. */
   sqluint32 Version,
   char * pDbAlias,
   unsigned short CallerAction,
   char * pStopTime,
   char * pUserName,
   char * pPassword,
   char * pOverflowLogPath,
   unsigned short NumChngLgOvrflw,
   struct sqlurf_newlogpath * pChngLogOvrflw,
   unsigned short ConnectMode,
   struct sqlu_tablespace_bkrst_list * pTablespaceList,
   short AllNodeFlag,
   short NumNodes,
   SQL_PDB_NODE_TYPE * pNodeList,
   short NumNodeInfo,
   unsigned short DlMode,             /* NOTE: This parameter is no longer used */
                                      /*       for the Data Links Manager. */
   char * pReportFile,                /* NOTE: This parameter is no longer used */
                                      /*       for the Data Links Manager. */
   char * pDroppedTblID,
   char * pExportDir
}
/* ... */
```

API パラメーター

pRfwdInput
入力。 *rfwd_input* 構造を指すポインター。この構造の詳細については、 497ページの『RFWD-INPUT』を参照してください。

pRfwdOutput
出力。 *rfwd_output* 構造を指すポインター。この構造の詳細については、 500ページの『RFWD-OUTPUT』を参照してください。

DbAliasLen
入力。データベース別名の長さを示す 2 バイトの無符号整数 (バイト単位)。

StopTimeLen
入力。停止時刻パラメーターの長さを示す 2 バイトの無符号整数 (バイト単位)。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。

UserNameLen
入力。ユーザー名の長さを示す 2 バイトの無符号整数 (バイト単位)。ユーザー名が提供されていない場合は、ゼロに設定してください。

PasswordLen
入力。パスワードの長さを示す 2 バイトの無符号整数 (バイト単位)。パスワードが提供されていない場合は、ゼロに設定してください。

OverflowLogPathLen
入力。オーバーフロー・ログ・パスの長さを示す 2 バイトの無符号整数 (バイト単位)。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。

ReportFileLen
入力。このパラメーターは現在は使用されていないので、ゼロに設定しておく必要があります。

Version
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。

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ている別名です。

CallerAction

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

SQLUM_ROLLFWD

pPointInTime で指定された時点までロールフォワードします。データベースのロールフォワードの場合、データベースは*ロールフォワード保留* 状態のままになります。表スペースのロールフォワードの場合、表スペースは*ロールフォワード処理* 状態のままになります。

SQLUM_STOP

ロールフォワード回復を終了します。新しいログ・レコードは処理されず、コミットされていないトランザクションはバックアウトされます。データベースまたは表スペースの*ロールフォワード保留* 状態はオフになります。同義語は SQLUM_COMPLETE です。

SQLUM_ROLLFWD_STOP

pPointInTime で指定された時点までロールフォワードし、ロールフォワード回復を終了します。データベースまたは表スペースの*ロールフォワード保留* 状態はオフになります。同義語は SQLUM_ROLLFWD_COMPLETE です。

SQLUM_QUERY

pNextArcFileName、 *pFirstDelArcFileName*、*pLastDelArcFileName*、および *pLastCommitTime* の値を照会します。データベース状況とノード番号を戻します。

SQLUM_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。

SQLUM_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、回復保留状態に置かれます。

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。

データベースをロールフォワードするとき、テープ装置を使用するロード回復が必要な場合もあります。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。

SQLUM_LOADREC_CONTINUE
警告メッセージを生成した装置を使用し続けます (たとえば、新しいテープが装てんされた場合)。

SQLUM_LOADREC_DEVICE_TERMINATE
警告メッセージを生成した装置の使用を停止します (たとえば、それ以上テープがない場合)。

SQLUM_LOADREC_TERMINATE
ロード回復に使用されているすべての装置を終了させます。

pStopTime
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、データベース回復は停止します。可能なかぎ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。SQLUM_QUERY、SQLUM_PARM_CHECK、およびいずれかのロード回復 (SQLUM_LOADREC_*xxx*) 呼び出し側アクションの場合は、ヌルにすることができます。

pUserName
入力。アプリケーションのユーザー名を含むストリング。ヌル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。ヌルにすることもできます。

pOverflowLogPath
入力。使用される代替ログ・パスを指定します。ユーザーは、アクティブ・ログ・ファイルのほかに、アーカイブ・ログ・ファイルを *logpath* (327ページの『sqlfxdb - データベース構成の入手』を参照) に移動することが必要です。それからでなければ、このユーティリティーでそれらのファイルを使用することはできません。このことは、*logpath* に十分なスペースがない場合に問題になる可能性があります。その問題を解決するために、オーバーフロー・ログ・パスが備えられています。ロールフォワード回復中に、必要なログ・ファイルは、まず *logpath* で探索され、次にオーバーフロー・ログ・パスで探索されます。表スペース

のロールフォワード回復に必要なログ・ファイルは、 *logpath* または オーバーフロー・ログ・パスのいずれかに置かれる可能性があります。呼び出し側がオーバーフロー・ログ・パスを指定しない場合、省略時値は *logpath* です。複数ノード環境では、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。省略時パスは、各ノードの省略時オーバーフロー・ログ・パスです。単一ノード環境では、サーバーがローカルであれば、オーバーフロー・ログ・パスが相対パスであっても構いません。

NumChngLgOvrflw
MPP 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、指定されたノードの省略時オーバーフロー・ログ・パスだけをオーバーライドします。

pChngLogOvrflw
MPP 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、指定されたノードの省略時オーバーフロー・ログ・パスだけをオーバーライドします。

ConnectMode
入力。有効な値 (`sqlutil` で定義) は、以下のとおりです。

SQLUM_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回復の場合には、必ずこの値を指定してください。

SQLUM_ONLINE
オンライン・ロールフォワード。

pTablespaceList
入力。ログの終わりまで、または指定された時点までロールフォワードされる表スペースの名前が入っている構造を指すポインター。指定されない場合には、ロールフォワードを必要とする表スペースが選択されます。

AllNodeFlag
MPP のみ。入力。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。

SQLURF_NODE_LIST
pNodeList で渡されたノード・リスト内のノードに適用されます。

SQLURF_ALL_NODES
すべてのノードに適用されます。 *pNodeList* はヌルでなければなりません。これは省略時値です。

SQLURF_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。

SQLURF_CAT_NODE_ONLY
カタログ・ノードにのみ適用されます。 *pNodeList* はヌルでなければなりません。

NumNodes
入力。 *pNodeList* 配列内のノードの数を指定します。

pNodeList
入力。ロールフォワード操作を実行する対象のノード番号の配列を指すポインター。

NumNodeInfo
入力。出力パラメーター *pNodeInfo* 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ノード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。単一ノード環境では、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 が呼び出されるノードの数と同じにする必要があります。

DlMode
入力。このパラメーターは現在は使用されていないので、ゼロに設定しておく必要があります。

pReportFile
入力。このパラメーターは現在は使用されていないので、ヌルに設定しておく必要があります。

pDroppedTblID
入力。回復が実行されている消去済み表の ID を含むストリング。

pExportDir
入力。削除した表データのエクスポート先のディレクトリー。

pSqlca
出力。 *sqlca* 構造を指すポインターです。 この構造に関する詳細は、520ページの『SQLCA』を参照してください。

REXX API 構文

```
ROLLFORWARD DATABASE database-alias [USING :value] [USER username USING password]
[rollforward_action_clause | load_recovery_action_clause]
where rollforward_action_clause stands for:
    { TO point-in-time [AND STOP] |
      {
        [TO END OF LOGS [AND STOP] | STOP | CANCEL | QUERY STATUS | PARM CHECK }
        [ON {:nodelist | ALL NODES [EXCEPT :nodelist]}]
      }
    }
    [TABLESPACE {ONLINE |:tablespacenames [ONLINE]} ]
    [OVERFLOW LOG PATH default-log-path [:logpaths]]
and load_recovery_action_clause stands for:
    LOAD RECOVERY { CONTINUE | DEVICE_TERMINATE | TERMINATE }
```

REXX API パラメーター

database-alias
ロールフォワードされるデータベースの別名。

value 出力値を含む複合 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 変数内の要素の数

XXX.1 アプリケーション ID

XXX.2 ノードから受信された応答の数

XXX.2.1.1 最初のノード番号

XXX.2.1.2 最初のノードの状態情報

XXX.2.1.3 最初のノードの必要とされる次のアーカイブ・ファイル

XXX.2.1.4 最初のノードの削除される最初のアーカイブ・ファイル

XXX.2.1.5 最初のノードの削除される最後のアーカイブ・ファイル

XXX.2.1.6 最初のノードの最後のコミット時刻

XXX.2.2.1 2 番目のノードのノード番号

XXX.2.2.2 2 番目のノードの状態情報

XXX.2.2.3 2 番目のノードの必要とされる次のアーカイブ・ファイル

XXX.2.2.4 2 番目のノードの削除される最初のアーカイブ・ファイル

XXX.2.2.5 2 番目のノードの削除される最後のアーカイブ・ファイル

XXX.2.2.6 2 番目のノードの最後のコミット時刻

XXX.2.3.x 以降、3 番目、4 番目 ... と続きます。

username

データベースのロールフォワードに使用されるユーザー名を識別します。

password

ユーザー名の認証に使用されるパスワード。

point-in-time

協定世界時 (UTC) で表現された、 ISO 形式 *yyyy-mm-dd-hh.mm.ss.nnnnnn* (年、月、日、時、分、秒、ミリ秒) のタイム・スタンプ。

tablespacenames

ロールフォワードされる表スペースのリストを含む複合 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 ロールフォワードされる表スペースの数

XXX.1 最初の表スペース名

XXX.2 2 番目の表スペース名

XXX.x 以降、3 番目、4 番目 ... と続きます。

default-log-path

回復中にアーカイブ・ログを探索するための省略時オーバーフロー・ログ・パス。

logpaths

回復中にアーカイブ・ログを探索するための代替ログ・パスのリストを含む複合 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 変更されるオーバーフロー・ログ・パスの数

XXX.1.1 最初のノード

XXX.1.2 最初のノードのオーバーフロー・ログ・パス

XXX.2.1 2 番目のノード

XXX.2.2 2 番目のノードのオーバーフロー・ログ・パス

XXX.3.1 3 番目のノード

XXX.3.2 3 番目のノードのオーバーフロー・ログ・パス

XXX.x.1 以降、3 番目、4 番目 ... と続きます。

nodelist

ノードのリストを含む 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 ノードの数

XXX.1 最初のノード

XXX.2 2 番目のノード

XXX.x 以降、3 番目、4 番目 ... と続きます。

サンプル・プログラム

C ¥sqllib¥samples¥c¥backrest.c

COBOL ¥sqllib¥samples¥cobol¥backrest.cbl

使用上の注意

データベース・マネージャーは、アーカイブおよびログ・ファイルに格納された情報を使用して、最後のバックアップのときにデータベースで実行されたトランザクションを再構築します。

この API の呼び出し時に実行されるアクションは、呼び出し前のデータベースの *rollforward_pending* フラグによって異なります。これは、 327ページの『sqlfxdb - データベース構成の入手』を使用して照会できます。データベースがロールフォワード保留状態にある場合、 *rollforward_pending* フラグは DATABASE に設定されています。 1 つまたは複数の表スペースが SQLB_ROLLFORWARD_PENDING または SQLB_ROLLFORWARD_IN_PROGRESS 状態にある場合、フラグは TABLESPACE に設定されています。データベースも表スペースもロールフォワードする必要がない場合は、 *rollforward_pending* フラグが NO に設定されています。

この API の呼び出し時にデータベースがロールフォワード保留状態にある場合は、データベースがロールフォワードされます。表スペースは、異常状態によって 1 つまたは複数の表スペースがオフラインにならない限り、データベー

スのロールフォワードが正常に終了すると正常の状態に戻ります。
rollforward_pending フラグが TABLESPACE に設定されている場合には、ロールフォワード保留状態にある表スペース、あるいは名前によって要求された表スペースだけがロールフォワードされます。

注: 表スペースのロールフォワードが異常終了してしまった場合、ロールフォワード中だった表スペースは、 SQLB_ROLLFORWARD_IN_PROGRESS 状態に置かれます。次に ROLLFORWARD DATABASE を呼び出したときには、SQLB_ROLLFORWARD_IN_PROGRESS 状態にあるそれらの表スペースだけが処理されます。選択された表スペース名のセットに SQLB_ROLLFORWARD_IN_PROGRESS 状態のすべての表スペースが含まれているのではない場合には、要求されていない表スペースが SQLB_RESTORE_PENDING 状態に置かれます。

データベースがロールフォワード保留状態になく、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、ロールフォワード保留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、 SQLUM_QUERY 呼び出し側アクションを指定してこの API を呼び出すと、このログ・ファイルの名前を判別することができます。

ログ・ファイル内のトランザクションは、データベースに再適用されます。ログは、情報が使用可能である限り、あるいは停止時刻パラメーターで指定された時刻まで、順方向に処理されます。

以下の事象が生じると、回復が停止します。

- 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、回復されない可能性があります。
pLastCommitTime 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回復が必要となった場合、エラーが発生する前の時点で回復を停止することを指示するために、

pStopTime 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回復と、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、前回失敗した回復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回復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、ロールフォワード回復が終了するまで、データベースは使用できません。 SQLUM_STOP または SQLUM_ROLLFORWARD_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保留状態をオフにすれば、回復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回復を実行するための API が呼び出されるまで使用不能になります。表スペースをある時点までロールフォワードすると、表スペースは、正常なロールフォワード後にバックアップ保留状態に置かれます。

データベースをロールフォワードするときには、この解説書では扱われていない前提条件や制限が関係してくる場合があります。詳細については、*管理の手引き* を参照してください。

参照資料

408ページの『sqluload - ロード』

447ページの『sqlurestore - データベースの復元』

sqlustat - 統計の実行

表およびそれに関連する索引の特性に関する統計を更新します。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。最適化プログラムは、データへのアクセス・パスを判別するとき、これらの統計を使用します。

表に数多くの更新が加えられたときや、表を再編成した後、あるいは新規の索引を作成した後などに、このユーティリティーを呼び出してください。

統計は、API が実行されるノードに存在する表区分に基づいて収集されます。グローバル表統計は、あるノードで取得された値に、表が完全に保管されているノード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 が呼び出されるノードは、表の区分を含んでいる必要はありません。

- 表の区分を含むノードから API が呼び出されると、ユーティリティーはこのノードで実行されます。
- 表の区分を含まないノードから API が呼び出されると、要求は、表の区分を保持しているノード・グループ内の最初のノードに送られます。その後、このノードでユーティリティーが実行されます。

効力範囲

この API は、`db2nodes.cfg` ファイル内の任意のノードから呼び出すことができます。この API は、カタログ・ノード上のカタログを更新するのに使用できます。

許可

以下のいずれかです。

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

必須接続

データベース

API 組み込みファイル

sqlutil.h

C API 構文

```
/* File: sqlutil.h */
/* API: Run Statistics */
/* ... */
SQL_API_RC SQL_API_FN
  sqlustat (
    _SQLOLDCHAR * pTableName,
    unsigned short NumIndexes,
    _SQLOLDCHAR ** ppIndexList,
    unsigned char StatsOption,
    unsigned char ShareLevel,
    struct sqlca * pSqlca);
/* ... */
```

汎用 API 構文

```
/* File: sqlutil.h */
/* API: Run Statistics */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgstat (
    unsigned short TableNameLen,
    unsigned short NumIndexes,
    unsigned char StatsOption,
    unsigned char ShareLevel,
    unsigned short * pIndexLens,
    struct sqlca * pSqlca,
    _SQLOLDCHAR ** ppIndexList,
    _SQLOLDCHAR * pTableName);
/* ... */
```

API パラメーター

TableNameLen
入力。表名の長さを示す 2 バイトの無符号整数 (バイト単位)。

NumIndexes
入力。この呼び出しで指定された索引の数。この値は、*StatsOption* パラメーターで使用されます。有効な値は以下のとおりです。

0 索引のすべてが計算されます。

n 索引リストにある索引の数。計算される索引の名前は、*ppIndexList* で指定されます。

StatsOption
入力。どの計算が実行されるかを示す統計オプション。有効な値 (sqlutil で定義) は、以下のとおりです。

SQL_STATS_TABLE
表のみ。

SQL_STATS_EXTTABLE_ONLY
拡張 (分配) 統計を含む表統計。

SQL_STATS_BOTH
表と索引の両方。

SQL_STATS_EXTTTABLE_INDEX
表統計 (分配統計を含む) と索引の基本統計の両方。

SQL_STATS_INDEX
索引のみ。

SQL_STATS_EXTINDEX_ONLY
索引の拡張統計のみ。

SQL_STATS_EXTINDEX_TABLE
索引の拡張統計と表の基本統計。

SQL_STATS_ALL
索引の拡張統計と表統計 (分配統計を含む)。

ShareLevel
入力。他のユーザーと関連した、統計の収集方法を指定します。有効な値 (sqlutil で定義) は、以下のとおりです。

SQL_STATS_REF
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。

SQL_STATS_CHG
統計の収集中に、他のユーザーが読み取りおよび書き込みアクセスを行えるようにします。

pIndexLens
入力。索引リストにある索引名のそれぞれの長さを示す 2 バイトの無符号整数の配列 (バイト単位)。

pSqlca

出力。 *sqlca* 構造を指すポインターです。 この構造に関する詳細は、520ページの『SQLCA』を参照してください。

ppIndexList

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。

pTableName

入力。統計を更新する表。別名を指定することもできます。ただし、下位のサーバーの場合には、必ず、表の完全修飾名を使用しなければなりません。

行タイプの場合、*pTableName* は階層のルート表の名前でなければなりません。

REXX API 構文

```
RUNSTATS ON TABLE tname
[WITH :statsopt INDEXES {ALL | USING :value}]
[SHRLEVEL {REFERENCE|CHANGE}]
```

REXX API パラメーター

tname 統計が収集される表の完全修飾名。

statsopt

どの計算が実行されるかを示す統計オプションを含むホスト変数。有効な値は以下のとおりです。

T 指定した表についてのみ基本統計を更新することを示します。これが省略時の値です。

D 指定した表について拡張 (分配) 統計を更新することを示します。

B 指定した表と索引の両方について基本統計を更新することを示します。

E 指定した表について拡張統計を更新することと、索引について基本統計を更新することを示します。

I 指定した索引についてのみ基本統計を更新することを示します。

X 指定した索引についてのみ拡張統計を更新することを示します。

Y 指定した表について基本統計を更新することと、索引について拡張統計を更新することを示します。

A 指定した表と索引の両方について拡張統計を更新することを示します。

value 統計が収集される索引の名前を含む複合 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 この呼び出しで指定された索引の数

XXX.1 最初の完全修飾索引名

XXX.2 2 番目の完全修飾索引名

XXX.3 以降、3 番目、4 番目 ... と続きます。

REFERENCE
更新中に、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。

CHANGE
更新中に、他のユーザーが読み取りおよび書き込みアクセスを行えるようにします。これが省略時の値です。

サンプル・プログラム

C ¥sqllib¥samples¥c¥dbstat.sqc

COBOL ¥sqllib¥samples¥cobol¥dbstat.sqb

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。

RUNSTATS は、以下のような場合に統計を更新するために使用してください。

- 表が何回も修正されている場合 (たとえば、数多くの更新が行われている場合や、大量のデータが挿入または削除されている場合など)
- 表が再編成されている場合
- 新しい索引が作成されている場合

統計が更新された後、97ページの『sqlabndx - バインド』を用い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。

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。

統計は、API が実行されるデータベース区画に存在する表データに基づいて収集されます。区分表全体のグローバル表統計は、あるデータベース区画で取得された値に、その表が区分されているノードグループ内のデータベース区画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区画は、表の区分を含んでいる必要はありません。

- 表の区分を含むデータベース区画から 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区画で実行されます。
- 表の区分を含まないデータベース区画から API が呼び出されると、要求は、表の区分を保持しているノードグループ内の最初のデータベース区画に送られます。その後、このデータベース区画でユーティリティーが実行されます。

この API の一部を実行するときに不整合が検出される場合 (この API を最後に呼び出した後の表でのアクティビティーによる)、警告メッセージが戻されます。たとえば、最初の呼び出しで表分配統計が収集され、 2 番目の呼び出しでは索引統計だけが収集される場合、その表でのアクティビティーによって不整合が検出されると、表分配統計が消去されます。このときには、API をもう一度呼び出して、表分配統計を最新表示することをお勧めします。

参照資料

コマンド解説書 の『REORGCHK』

327ページの『sqlfxdb - データベース構成の入手』

442ページの『sqlureot - 表の再編成』

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、更新可能、および排他の 3 つです。静止関数の結果として生じる表スペースの状態は、 QUIESCED SHARE、QUIESCED UPDATE、および QUIESCED EXCLUSIVE の 3 つです。

効力範囲

単一ノード環境では、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。 MPP 環境では、この API は、1 つのノードにローカルに影響します。ロードが実行されているノードに属する表スペースのその部分だけを静止します。

許可

以下のいずれかです。

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

必須接続

データベース

API 組み込みファイル

sqlutil.h

C API 構文

```
/* File: sqlutil.h */
/* API: Quiesce Tablespaces for Table */
/* ... */
SQL_API_RC SQL_API_FN
  sqluvqdp (
    char * pTableName,
    sqlint32 QuiesceMode,
    void * pReserved,
    struct sqlca * pSqlca);
/* ... */
```

汎用 API 構文

```
/* File: sqlutil.h */
/* API: Quiesce Tablespaces for Table */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgvqdp (
    unsigned short TableNameLen,
    char * pTableName,
    sqlint32 QuiesceMode,
    void * pReserved,
    struct sqlca * pSqlca);
/* ... */
```

API パラメーター

TableNameLen
入力。表名の長さを示す 2 バイトの無符号整数 (バイト単位)。

pTableName
入力。システム・カタログで使用される表名を含むストリング。これは、スキーマ と、ピリオド (.) で区切られた表名の 2 部からなる名前にすることができます。スキーマ が提供されない場合は、 CURRENT SCHEMA が使用されます。システム・カタログ表を指定することはできません。このフィールドは必須です。

QuiesceMode
入力。静止モードを指定します。有効な値 (sqlutil で定義) は、以下のとおりです。

SQLU_QUIESCEMODE_SHARE
共用モードの場合

SQLU_QUIESCEMODE_INTENT_UPDATE
更新可能モードの場合

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です。 この構造に関する詳細は、520ページの『SQLCA』を参照してください。

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。

サンプル・プログラム

C	¥sqllib¥samples¥c¥tload.sqc
COBOL	¥sqllib¥samples¥cobol¥tload.sqb
REXX	¥sqllib¥samples¥rexx¥quitab.cmd

使用上の注意

この API は宣言された一時表ではサポートされません。

共用静止要求を受け取ると、トランザクションは、表スペースに対する共用可能ロックと表に対する共用ロックを要求します。トランザクションがロックを獲得すると、表スペースの状態が QUIESCED SHARE に変更されます。この状態は、他のユーザーがこれと矛盾する状態を保持していない場合にかぎり、

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。

更新可能静止要求が行われると、表スペースは排他可能 (IX) モードでロックされ、表は更新 (U) モードでロックされます。表スペースの状態は、静止者とともに、表スペース表に記録されます。

1 つの表スペースに対する静止者の限度は、常に 5 人です。 QUIESCED EXCLUSIVE は、他の状態と非互換であり、 QUIESCED UPDATE は、別の QUIESCED UPDATE と非互換であるため、 5 という静止者の限度に達する場合は、少なくとも 4 つの QUIESCED SHARE と多くて 1 つの QUIESCED UPDATE がなければなりません。

静止者は表スペースの状態を、あまり制限的でない状態から、より制限的な状態へ (たとえば、S から U へ、または U から X へ) アップグレードさせることができます。しかし、ユーザーがすでに保持している状態より低い状態を要求しても、元の状態が戻されます。つまり、状態がダウングレードされることはありません。

表スペースの静止状態は、 SQLU_QUIESCEMODE_RESET を使用することによって明示的にリセットしなければなりません。

参照資料

408ページの『sqluload - ロード』

第2章 追加の REXX API

この章では、REXX プログラミング言語でのみサポートされている DB2 アプリケーション・プログラミング・インターフェースについて説明します。

© Copyright IBM Corp. 1993, 2000

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。

許可

ありません

必須接続

ありません

REXX API 構文

```
CHANGE SQLISL TO {RR|CS|UR|RS|NC}
```

REXX API パラメーター

RR 反復可能読み取り。

CS カーソル固定。これが省略時の値です。

UR 非コミット読み取り。

RS 読み取り固定。

NC コミットなし。

サンプル・プログラム

REXX ¥sqllib¥samples¥rexx¥chgisl.cmd

第3章 データ構造

この章では、データベース・マネージャーへのアクセスに使用されるデータ構造について説明します。以下に列挙するデータ構造が提供されています。

© Copyright IBM Corp. 1993, 2000

523ページの『SQLDA』
アプリケーションとデータベース・マネージャーの間でデータの集合を転送します。

526ページの『SQLDCOL』
IMPORT および EXPORT API に列の情報を渡します。

530ページの『SQLE-ADDN-OPTIONS』
情報を 161ページの『sqleaddn - ノードの追加』に渡します。

532ページの『SQLE-CLIENT-INFO』
情報をクライアント情報 API に渡します (298ページの『sqleseti - クライアント情報の設定』および 283ページの『sqleqryi - クライアント情報の照会』を参照)。

535ページの『SQLE-CONN-SETTING』
接続設定のタイプおよび値を指定します。

540ページの『SQLE-NODE-APPC』
APPC ノードをカタログするための情報を渡します。

541ページの『SQLE-NODE-APPN』
APPN ノードをカタログするための情報を渡します。

542ページの『SQLE-NODE-CPIC』
CPIC ノードをカタログするための情報を渡します。

543ページの『SQLE-NODE-IPXSPX』
IPX/SPX ノードをカタログするための情報を渡します。

544ページの『SQLE-NODE-LOCAL』
LOCAL ノードをカタログするための情報を渡します。

545ページの『SQLE-NODE-NETB』
NetBIOS ノードをカタログするための情報を渡します。

546ページの『SQLE-NODE-NPIPE』
名前付きパイプ・ノードをカタログするための情報を渡します。

547ページの『SQLE-NODE-STRUCT』
ノードをカタログするための情報を渡します。

549ページの『SQLE-NODE-TCPIP』
TCP/IP ノードをカタログするための情報を渡します。

550ページの『SQLE-REG-NWBINDERY』
NetWare ファイル・サーバー上のバインダリーに DB2 サーバーを登録するか、あるいはバインダリーから DB2 サーバーの登録を解除するための情報を渡します。

551ページの『SQLE-START-OPTIONS』
データベース・マネージャー始動オプションを保持します。

556ページの『SQLEDBCOUNTRYINFO』
アプリケーションとデータベース・マネージャーの間で国別情報を転送します。

557ページの『SQLEDBDESC』
CREATE DATABASE API に作成パラメーターを渡します。

563ページの『SQLEDBSTOPOPT』
データベース・マネージャー停止オプションを保持します。

565ページの『SQLEDINFO』
システムまたはローカル・データベース・ディレクトリーから単一のディレクトリー項目のコピーを戻します。

568ページの『SQLENINFO』
ノード・ディレクトリーから単一のディレクトリー項目のコピーを戻します。

572ページの『SQLFUPD』
構成ファイル情報を渡します。

580ページの『SQLM-COLLECTED』
アプリケーションとデータベース・マネージャーの間でデータベース・システム・モニターの収集カウント情報を転送します。

583ページの『SQLM-RECORDING-GROUP』
アプリケーションとデータベース・マネージャーの間でデータベース・システム・モニターのモニター・グループ情報を転送します。

585ページの『SQLMA』
データベース・モニター要求をアプリケーションからデータベース・マネージャーへ送信します。

589ページの『SQLOPT』
バインド・パラメーターを BIND API へ転送し、プリコンパイル・オプションを PRECOMPILE PROGRAM API に転送します。

591ページの『SQLU-LSN』
ASYNCHRONOUS READ LOG API で使用されるログ順序番号の定義を含みます。

592ページの『SQLU-MEDIA-LIST』
バックアップ・イメージ用のターゲット媒体 (BACKUP) またはソース媒体 (RESTORE) のリストを保持します。インポート、エクスポート、およびロード API によっても使用されます。

597ページの『SQLU-RLOG-INFO』
ASYNCHRONOUS READ LOG API への呼び出しに関する情報を含みます。

598ページの『SQLU-TABLESPACE-BKRST-LIST』
表スペース名のリストを提供します。

600ページの『SQLUEXPT-OUT』
アプリケーションとデータベース・マネージャーの間でエクスポート情報を転送します。

601ページの『SQLUIMPT-IN』
アプリケーションとデータベース・マネージャーの間でインポート情報を転送します。

602ページの『SQLUIMPT-OUT』
アプリケーションとデータベース・マネージャーの間でインポート情報を転送します。

604ページの『SQLULOAD-IN』
アプリケーションとデータベース・マネージャーの間でロード情報を転送します。

609ページの『SQLULOAD-OUT』
アプリケーションとデータベース・マネージャーの間でロード情報を転送します。

611ページの『SQLUPI』
表の区分化マップや区分化キーなど、区分化情報を含みます。

613ページの『SQLXA-RECOVER』
トランザクション API によって使用され、未確定トランザクションのリストを戻します。

616ページの『SQLXA-XID』
トランザクション API によって使用され、トランザクションを識別します。

db2HistData

この構造は、42ページの『db2HistoryGetEntry - 回復活動記録ファイルの次項目の入手』への呼び出しの後に情報を戻すのに使用されます。

表 11. db2HistData 構造のフィールド

フィールド名	データ・タイプ	説明
ioHistDataID	char(8)	記憶域ダンプ用の 8 バイトの構造識別子および ″目印″。有効な値は "SQLUHINF" だけです。このストリングには、記号の定義はありません。
oObjectPart	db2Char	最初の 14 文字は、*yyyymmddhhnnss* の形式のタイム・スタンプで、操作を開始した時を示します。次の 3 文字は順序番号です。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。順序番号によって複数の位置を指定することができます。復元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順序番号 ’001’ に該当) だけが入ります。順序番号と結合されるタイム・スタンプは、固有のものであることが必要です。
oEndTime	db2Char	操作が完了した時を示す *yyyymmddhhnnss* の形式のタイム・スタンプ。
oFirstLog	db2Char	最も古いログ・ファイル ID (範囲は S0000000 から S9999999)。 • オンライン・バックアップについてロールフォワード回復を適用するために必要 • オフライン・バックアップについてロールフォワード回復に適用するために必要 • ロードの開始時に現行だった全データベースまたは表スペース・レベル・バックアップの復元後に適用

表 11. db2HistData 構造のフィールド (続き)

フィールド名	データ・タイプ	説明
oLastLog	db2Char	最新のログ・ファイル ID (範囲は S0000000 から S9999999)。 • オンライン・バックアップについてロールフォワード回復を適用するために必要 • オフライン・バックアップについて現時点へのロールフォワード回復を適用するために必要 • ロード操作の終了時に現行だった全データベースまたは表スペース・レベル・バックアップの復元後に適用 (ロールフォワード回復が適用されない場合は、*oFirstLog* と同じ)
oID	db2Char	固有のバックアップまたは表識別子。
oTableQualifier	db2Char	表修飾子。
oTableName	db2Char	表名。
oLocation	db2Char	バックアップおよびロード・コピーの場合、このフィールドはデータが保管された場所を示します。ファイルに複数の項目が入る操作の場合、 *oObjectPart* によって定義される順序番号は、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。復元およびロード操作の場合、ロケーションは、常に、復元またはロードされたデータの最初の部分 (複数パーツ・バックアップの順序 '001' に該当) が保管された場所を識別します。 *oLocation* のデータは、*oDeviceType* によって解釈方法が異なります。 • ディスクまたはディスケット (`D` または `K`) の場合、完全修飾ファイル名 • テープ (`T`) の場合、ボリューム・レベル • TSM (`A`) の場合、サーバー名 • ユーザー出口またはその他 (`U` または `O`) の場合、自由形式のテキスト
oComment	db2Char	自由形式のテキスト注釈。
oCommandText	db2Char	コマンド・テキストまたは DDL。
oLastLSN	SQLU_LSN	最新のログ順序番号。
oEID	構造体	固有の項目識別子。
poEventSQLCA	構造体	記録された事象の結果 *sqlca*。 *sqlca* 構造についての詳細は、520ページの『SQLCA』を参照してください。
poTablespace	db2Char	表スペース名のリスト。

表 11. db2HistData 構造のフィールド (続き)

フィールド名	データ・タイプ	説明
ioNumTablespaces	db2Uint32	*poTablespace* リストの項目数。各表スペース・バックアップには 1 つ以上の表スペースが含まれます。各表スペース復元操作は 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復元を示している) 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復元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	事象のタイプ。B (バックアップ)、C (コピー)、 D (消去された表)、F (ロールフォワード)、 G (表の再編成)、L (ロード)、Q (静止)、R (復元)、S (統計の実行)、T (表スペースの変更)、および U (将来の使用)。
oObject	char	操作の細分性。D (全データベース)、P (表スペース)、および T (表)。
oOptype	char	操作のタイプ。C (表スペースの変更 (コンテナーの追加))、 E (ログの終了)、F (オフライン)、 I (挿入 (ロード))、N (オンライン)、 P (時点)、R (表スペースの変更 (再バランス))、S (静止の共有)、U (静止の更新)、 X (排他的な静止)、および Z (静止のリセット)。
oStatus	char	項目の状況。D (削除 (将来に使用))、 E (満了)、I (非アクティブ)、 N (コミットしていない)、および Y (コミット済みまたはアクティブ)。
oDeviceType	char	装置タイプ。このフィールドは、*oLocation* フィールドの解釈を判別します。 A (TSM)、C (クライアント)、D (ディスク)、 K (ディスケット)、L (ローカル)、 O (その他 (他のベンダー装置のサポート))、 P (パイプ)、S (サーバー)、T (テープ)、および U (ユーザー出口)。

表 12. db2Char 構造のフィールド

フィールド名	データ・タイプ	説明
pioData	char	文字データ・バッファーを指すポインター。ヌルの場合、データは戻されません。
iLength	db2Uint32	入力。 pioData バッファーのサイズ。
oLength	db2Uint32	出力。 pioData バッファー内にあるデータの有効文字の数。

表 13. db2HistoryEID 構造のフィールド

フィールド名	データ・タイプ	説明
ioNode	SQL_PDB_NODE_TYPE	ノード番号。
ioHID	db2Uint32	ローカル活動記録ファイルの項目 ID。

言語構文

C 構造

```
/* File: db2ApiDf.h */
/* ... */
typedef SQL_STRUCTURE db2HistoryData
{
  char ioHistDataID[8];
  db2Char oObjectPart;
  db2Char oEndTime;
  db2Char oFirstLog;
  db2Char oLastLog;
  db2Char oID;
  db2Char oTableQualifier;
  db2Char oTableName;
  db2Char oLocation;
  db2Char oComment;
  db2Char oCommandText;
  SQLU_LSN oLastLSN;
  db2HistoryEID oEID;
  struct sqlca * poEventSQLCA;
  db2Char * poTablespace;
  db2Uint32 ioNumTablespaces;
  char oOperation;
  char oObject;
  char oOptype;
  char oStatus;
  char oDeviceType
} db2HistoryData;
typedef SQL_STRUCTURE db2Char
{
  char * pioData;
  db2Uint32 ioLength
} db2Char;
typedef SQL_STRUCTURE db2HistoryEID
{
  SQL_PDB_NODE_TYPE ioNode;
  db2Uint32 ioHID
} db2HistoryEID;
/* ... */
```

RFWD-INPUT

この構造は、465ページの『sqluroll - データベースのロールフォワード』に情報を渡すのに使用されます。

表 14. RFWD-INPUT 構造のフィールド

フィールド名	データ・タイプ	説明
VERSION	sqluint32	ロールフォワード・バージョン。
PDBALIAS	ポインター	データベース別名。
CALLERACTION	UNSIGNED SHORT	アクション。
PSTOPTIME	ポインター	停止時刻。
PUSERNAME	ポインター	ユーザー名。
PPASSWORD	ポインター	パスワード。
POVERFLOWLOGPATH	ポインター	オーバーフロー・ログ・パス。
NUMCHNGLGOVRFLW	UNSIGNED SHORT	変更されたオーバーフロー・ログ・パスの数 (MPP のみ)。
PCHNGLOGOVRFLW	構造体	変更されたオーバーフロー・ログ・パス (MPP のみ)。
CONNECTMODE	UNSIGNED SHORT	接続モード。
PTABLESPACELIST	構造体	表スペース名のリストを指すポインター。この構造の詳細については、 598ページの『SQLU-TABLESPACE-BKRST-LIST』を参照してください。
ALLNODEFLAG	SHORT	すべてのノード・フラグ。
NUMNODES	SHORT	ノード・リストのサイズ。
PNODELIST	ポインター	ノード番号のリスト。
NUMNODEINFO	SHORT	500ページの『RFWD-OUTPUT』の *pNodeInfo* のサイズ。
DLMODE	UNSIGNED SHORT	このパラメーターは現在は使用されていません。
PREPORTFILE	ポインター	このパラメーターは現在は使用されていません。
PDROPPEDTBLID	ポインター	回復が実行されている消去済み表の ID を含むストリング。
PEXPORTDIR	ポインター	削除した表データのエクスポート先のディレクトリー。
NODENUM	SQL_PDB_NODE_TYPE	ノード番号。
PATHLEN	UNSIGNED SHORT	新規のログ・パスの長さ。
LOGPATH	CHAR(255)	新規のオーバーフロー・ログ・パス。

言語構文

C 構造

```
/* File: sqlutil.h */
/* Structure: RFWD-INPUT */
/* ... */
SQL_STRUCTURE rfwd_input
{
  sqluint32              version;
  char                   *pDbAlias;
  unsigned short         CallerAction;
  char                   *pStopTime;
  char                   *pUserName;
  char                   *pPassword;
  char                   *pOverflowLogPath;
  unsigned short         NumChngLgOvrflw;
  struct sqlurf_newlogpath *pChngLogOvrflw;
  unsigned short         ConnectMode;
  struct sqlu_tablespace_bkrst_list *pTablespaceList;
  short                  AllNodeFlag;
  short                  NumNodes;
  SQL_PDB_NODE_TYPE      *pNodeList;
  short                  NumNodeInfo;
  unsigned short         DlMode;         /* This parameter is not currently used. */
  char                   *pReportFile;   /* This parameter is not currently used. */
  char                   *pDroppedTblID;
  char                   *pExportDir;
};
/* ... */


/* File: sqlutil.h */
/* Structure: SQLURF-NEWLOGPATH */
/* ... */
SQL_STRUCTURE sqlurf_newlogpath
{
  SQL_PDB_NODE_TYPE      nodenum;
  unsigned short         pathlen;
  char                   logpath[SQL_LOGPATH_SZ+SQL_LOGFILE_NAME_SZ+1];
};
/* ... */
```

COBOL 構造

```
* File: sqlutil.cbl
01 SQL-RFWD-INPUT.
    05 SQL-VERSION            PIC 9(9) COMP-5.
    05 SQL-DBALIAS            USAGE IS POINTER.
    05 SQL-CALLERACTION       PIC 9(4) COMP-5.
    05 FILLER                 PIC X(2).
    05 SQL-STOPTIME           USAGE IS POINTER.
    05 SQL-USERNAME           USAGE IS POINTER.
    05 SQL-PASSWORD           USAGE IS POINTER.
    05 SQL-OVERFLOWLOGPATH    USAGE IS POINTER.
    05 SQL-NUMCHANGE          PIC 9(4) COMP-5.
    05 FILLER                 PIC X(2).
    05 SQL-P-CHNG-LOG-OVRFLW  USAGE IS POINTER.
    05 SQL-CONNECTMODE        PIC 9(4) COMP-5.
    05 FILLER                 PIC X(2).
    05 SQL-P-TABLESPACE-LIST  USAGE IS POINTER.
    05 SQL-ALLNODEFLAG        PIC S9(4) COMP-5.
    05 SQL-NUMNODES           PIC S9(4) COMP-5.
    05 SQL-NODELIST           USAGE IS POINTER.
    05 SQL-NUMNODEINFO        PIC S9(4) COMP-5.
    05 SQL-DLMODE             PIC 9(4) COMP-5.  * This parameter is not
                                                *   currently used.
    05 SQL-REPORTFILE         USAGE IS POINTER. * This parameter is not
                                                *   currently used.
    05 SQL-DROPPEDTBLID       USAGE IS POINTER.
    05 SQL-EXPORTDIR       USAGE IS POINTER.
*


* File: sqlutil.cbl
01 SQLURF-NEWLOGPATH.
    05 SQL-NODENUM            PIC S9(4) COMP-5.
    05 SQL-PATHLEN            PIC 9(4) COMP-5.
    05 SQL-LOGPATH            PIC X(254).
    05 FILLER                 PIC X.
    05 FILLER                 PIC X(1).
*
```

RFWD-OUTPUT

この構造は、465ページの『sqluroll - データベースのロールフォワード』から情報を渡すのに使用されます。

表 15. RFWD-OUTPUT 構造のフィールド

フィールド名	データ・タイプ	説明
PAPPLICATIONID	ポインター	API から戻されたアプリケーション識別子を保持する、長さ SQLU_APPLID_LEN+1 (sqlutil で定義される) のバッファーのアドレス。この識別子を指定してデータベース・システム・モニター API を呼び出せば、アプリケーションのいくつかの局面をモニターできます。この情報が必要ない場合は、ヌル・ポインターを提供してください。複数ノード環境では、カタログ・ノードのアプリケーション識別子だけを戻します。
PNUMREPLIES	ポインター	受信されたノード応答の数。応答するそれぞれのノードが、 *pNodeInfo* の *sqlurf_info* 構造に入ります。単一ノード環境では、このパラメーターの値は 1 でなければなりません。
PNODEINFO	構造体	ノード応答情報。 *NumNodeInfo* 個の *sqlurf_info* 構造のユーザー定義の配列。

表 16. SQLURF-INFO 構造のフィールド

フィールド名	データ・タイプ	説明
NODENUM	SQL_PDB_NODE_TYPE	ノード番号。
STATE	LONG	状態情報。
NEXTARCLOG	UNSIGNED CHAR(13)	必要とされる次のアーカイブ・ログ・ファイルの戻された名前を保持する 12 バイトのバッファー。 SQLUM_QUERY 以外の呼び出し側アクションを指定した場合には、このフィールドに戻される値によって、ファイルへのアクセス時にエラーが生じたことが示されます。考えられる原因は、以下のとおりです。 • データベース・ログ・ディレクトリー内、またはオーバーフロー・ログ・パス・パラメーターで指定されたパス上にファイルが見つからない。 • ユーザー出口プログラムがアーカイブ・ファイルを戻すのに失敗した。

表 16. SQLURF-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
FIRSTARCDEL	UNSIGNED CHAR(13)	回復に必要ではなくなった最初のアーカイブ・ログ・ファイルの戻された名前を保持する 12 バイトのバッファー。このファイルと、*lastarcdel* までのすべてのファイルを移動させて、ディスク上のスペースを空けることができます。 たとえば、 *firstarcdel* と *lastarcdel* で戻された値が `S0000001.LOG` と `S0000005.LOG` である場合には、以下のログ・ファイルを移動できます。 • `S0000001.LOG` • `S0000002.LOG` • `S0000003.LOG` • `S0000004.LOG` • `S0000005.LOG`
LASTARCDEL	UNSIGNED CHAR(13)	データベース・ログ・ディレクトリーから除去できる最後のアーカイブ・ログ・ファイルの戻された名前を保持する 12 バイトのバッファー。
LASTCOMMIT	UNSIGNED CHAR(27)	ISO 形式のタイム・スタンプを含む 26 文字のストリング。この値は、ロールフォワード操作の終了後、最後にコミットされたトランザクションのタイム・スタンプを示します。

STATE に有効な値 (`sqlutil` で定義) は、以下のとおりです。

SQLURFQ_NOT_AVAILABLE
ノードに接続できなかった。

SQLURFQ_NOT_RFW_PENDING
データベースがロールフォワード保留状態ではない。

SQLURFQ_DB_RFW_PENDING
データベースがロールフォワード保留状態である。

SQLURFQ_TBL_RFW_PENDING
表スペースがロールフォワード保留状態である。

SQLURFQ_DB_RFW_IN_PROGRESS
データベースがロールフォワード進行中である。

SQLURFQ_TBL_RFW_IN_PROGRESS
表スペースがロールフォワード進行中である。

SQLURFQ_DB_RFW_STOPPING
STOP 要求の処理中にデータベースのロールフォワードが中断された。

SQLURFQ_TBL_RFW_STOPPING
STOP 要求の処理中に表スペースのロールフォワードが中断された。

言語構文

C 構造

```
/* File: sqlutil.h */
/* Structure: RFWD-OUTPUT */
/* ... */
SQL_STRUCTURE rfwd_output
{
  char             *pApplicationId;
  long             *pNumReplies;
  struct sqlurf_info *pNodeInfo;
};
/* ... */


/* File: sqlutil.h */
/* Structure: SQLURF-INFO */
/* ... */
SQL_STRUCTURE sqlurf_info
{
  SQL_PDB_NODE_TYPE nodenum;
  long             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];
};
/* ... */
```

COBOL 構造

```
* File: sqlutil.cbl
01 SQL-RFWD-OUTPUT.
    05 SQL-APPLID             USAGE IS POINTER.
    05 SQL-NUMREPLIES         USAGE IS POINTER.
    05 SQL-P-NODE-INFO        USAGE IS POINTER.
*
```

```
* File: sqlutil.cbl
01 SQLURF-INFO.
    05 SQL-NODENUM           PIC S9(4) COMP-5.
    05 FILLER                PIC X(2).
    05 SQL-STATE             PIC S9(9) COMP-5.
    05 SQL-NEXTARCLOG        PIC X(12).
    05 FILLER                PIC X.
    05 SQL-FIRSTARCDEL       PIC X(12).
    05 FILLER                PIC X.
    05 SQL-LASTARCDEL        PIC X(12).
    05 FILLER                PIC X.
    05 SQL-LASTCOMMIT        PIC X(26).
    05 FILLER                PIC X.
    05 FILLER                PIC X(2).
*
```

SQL-AUTHORIZATIONS

この構造は、342ページの『sqluadau - 許可の入手』への呼び出しの後、情報を戻すのに使用されます。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 17. SQL-AUTHORIZATIONS 構造のフィールド

フィールド名	説明
SQL_AUTHORIZATIONS_LEN	構造のサイズ。
SQL_SYSADM_AUTH	SYSADM 権限。
SQL_SYSCTRL_AUTH	SYSCTRL 権限。
SQL_SYSMAINT_AUTH	SYSMAINT 権限。
SQL_DBADM_AUTH	DBADM 権限。
SQL_CREATETAB_AUTH	CREATETAB 権限。
SQL_CREATET_NOT_FENC_AUTH	CREATE_NOT_FENCED 権限。
SQL_BINDADD_AUTH	BINDADD 権限。
SQL_CONNECT_AUTH	CONNECT 権限。
SQL_IMPLICIT_SCHEMA_AUTH	IMPLICIT_SCHEMA 権限。
SQL_LOAD_AUTH	LOAD 権限。
SQL_SYSADM_GRP_AUTH	SYSADM 権限を保持するグループに所属するユーザー。
SQL_SYSCTRL_GRP_AUTH	SYSCTRL 権限を保持するグループに所属するユーザー。
SQL_SYSMAINT_GRP_AUTH	SYSMAINT 権限を保持するグループに所属するユーザー。
SQL_DBADM_GRP_AUTH	DBADM 権限を保持するグループに所属するユーザー。
SQL_CREATETAB_GRP_AUTH	CREATETAB 権限を保持するグループに所属するユーザー。
SQL_CREATE_NON_FENC_GRP_AUTH	CREATE_NOT_FENCED 権限を保持するグループに所属するユーザー。
SQL_BINDADD_GRP_AUTH	BINDADD 権限を保持するグループに所属するユーザー。
SQL_CONNECT_GRP_AUTH	CONNECT 権限を保持するグループに所属するユーザー。
SQL_IMPLICIT_SCHEMA_GRP_AUTH	IMPLICIT_SCHEMA 権限を保持するグループに所属するユーザー。
SQL_LOAD_GRP_AUTH	LOAD 権限を保持するグループに所属するユーザー。
注: SYSADM、SYSMAINT、および SYSCTRL は間接権限だけであり、ユーザーに直接付与されることはありません。これらの権限は、ユーザーが所属するグループを通じてのみ使用可能です。	

言語構文

C 構造

```
/* File: sqlutil.h */
/* Structure: SQL-AUTHORIZATIONS */
/* ... */
SQL_STRUCTURE sql_authorizations
{
  short           sql_authorizations_len;
  short           sql_sysadm_auth;
  short           sql_dbadm_auth;
  short           sql_createtab_auth;
  short           sql_bindadd_auth;
  short           sql_connect_auth;
  short           sql_sysadm_grp_auth;
  short           sql_dbadm_grp_auth;
  short           sql_createtab_grp_auth;
  short           sql_bindadd_grp_auth;
  short           sql_connect_grp_auth;
  short           sql_sysctrl_auth;
  short           sql_sysctrl_grp_auth;
  short           sql_sysmaint_auth;
  short           sql_sysmaint_grp_auth;
  short           sql_create_not_fenc_auth;
  short           sql_create_not_fenc_grp_auth;
  short           sql_implicit_schema_auth;
  short           sql_implicit_schema_grp_auth;
  short           sql_load_auth;
  short           sql_load_grp_auth;
};
/* ... */
```

SQL-AUTHORIZATIONS

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 18. SQL-DIR-ENTRY 構造のフィールド

フィールド名	データ・タイプ	説明
STRUCT_ID	SMALLINT	構造識別子。 SQL_DCS_STR_ID (sqlenv で定義) に設定してください。
RELEASE	SMALLINT	(API が割り当てた) リリース・バージョン。
CODEPAGE	SMALLINT	コメント用コード・ページ。
COMMENT	CHAR(30)	データベースのオプション・コメント。
LDB	CHAR(8)	データベースのローカル名。システム・データベース・ディレクトリー内のデータベース別名と一致していなければなりません。
TDB	CHAR(18)	データベースの実名。
AR	CHAR(32)	アプリケーション・クライアントの名前。
PARM	CHAR(512)	トランザクション・プログラムの接頭部、トランザクション名、SQLCODE マッピング・ファイル名、および切断 / 機密保護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、ヌル終了するか、フィールドの長さいっぱいにまでブランクを埋め込む必要があります。		

言語構文

C 構造

```
/* File: sqlenv.h */
/* Structure: SQL-DIR-ENTRY */
/* ... */
SQL_STRUCTURE sql_dir_entry
{
  unsigned short         struct_id;
  unsigned short         release;
  unsigned short         codepage;
  _SQLOLDCHAR            comment[SQL_CMT_SZ + 1];
  _SQLOLDCHAR            ldb[SQL_DBNAME_SZ + 1];
  _SQLOLDCHAR            tdb[SQL_LONG_NAME_SZ + 1];
  _SQLOLDCHAR            ar[SQL_AR_SZ + 1];
  _SQLOLDCHAR            parm[SQL_PARAMETER_SZ + 1];
};
/* ... */
```

COBOL 構造

```
* File: sqlenv.cbl
01 SQL-DIR-ENTRY.
    05 STRUCT-ID             PIC 9(4) COMP-5.
    05 RELEASE-LVL           PIC 9(4) COMP-5.
    05 CODEPAGE              PIC 9(4) COMP-5.
    05 COMMENT               PIC X(30).
    05 FILLER                PIC X.
    05 LDB                   PIC X(8).
    05 FILLER                PIC X.
    05 TDB                   PIC X(18).
    05 FILLER                PIC X.
    05 AR                    PIC X(32).
    05 FILLER                PIC X.
    05 PARM                  PIC X(512).
    05 FILLER                PIC X.
    05 FILLER                PIC X(1).
*
```

SQLA-FLAGINFO

この構造は、標識機能情報を保持するのに使用されます。

表 19. SQLA-FLAGINFO 構造のフィールド

フィールド名	データ・タイプ	説明
VERSION	SMALLINT	SQLA_FLAG_VERSION (sqlaprep で定義) に設定する必要のある入力フィールド。
MSGS	構造体	組み込み *sqla_flagmsgs* 構造。

表 20. SQLA-FLAGMSGS 構造のフィールド

フィールド名	データ・タイプ	説明
COUNT	SMALLINT	標識機能によって戻されるメッセージの数に設定される出力フィールド。
SQLCA	配列	標識機能からの情報を戻す SQLCA 構造の配列。

言語構文

C 構造

```
/* File: sqlaprep.h */
/* Structure: SQLA-FLAGINFO */
/* ... */
SQL_STRUCTURE sqla_flaginfo
{
  short             version;
  short             padding;
  struct            sqla_flagmsgs msgs;
};
/* ... */


/* File: sqlaprep.h */
/* Structure: SQLA-FLAGMSGS */
/* ... */
SQL_STRUCTURE sqla_flagmsgs
{
  short             count;
  short             padding;
  SQL_STRUCTURE sqlca sqlca[SQLA_FLAG_MAXMSGS];
};
/* ... */
```

SQLA-FLAGINFO

COBOL 構造

```
* File: sqlaprep.cbl
01 SQLA-FLAGINFO.
    05 SQLFLAG-VERSION            PIC 9(4) COMP-5.
    05 FILLER                     PIC X(2).
    05 SQLFLAG-MSGS.
        10 SQLFLAG-MSGS-COUNT      PIC 9(4) COMP-5.
        10 FILLER                  PIC X(2).
        10 SQLFLAG-MSGS-SQLCA OCCURS 10 TIMES.
*
```

SQLB-TBS-STATS

この構造は、追加の表スペース統計をアプリケーション・プログラムへ戻すのに使用されます。

表 21. 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。詳細については、*管理の手引き* の『表スペースの設計と選択』を参照してください。 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、最高ウォーターマークは、表スペース・アドレス空間の現在の ″終わり″ です。言い換えれば、表スペースの最後に割り振られたエクステントに続く最初の空きエクステントのページ番号です。 これは、実際には ″最高ウォーターマーク″ ではなく、 ″現行ウォーターマーク″ である (値が減少する可能性があるため) ことに注意してください。 SMS には適用されません。

表スペースの再バランス中、使用可能なページの数には、新しく追加されたコンテナーのページも含まれますが、これらの新しいページは、再バランスが完了するまでは空きページの数に反映されません。表スペースの再バランスが行われていない ときは、使用中のページ数と空きページの数の合計が使用可能なページの数になります。

言語構文

C 構造

```
/* File: sqlutil.h */
/* Structure: SQLB-TBS-STATS */
/* ... */
SQL_STRUCTURE SQLB_TBS_STATS
{
  sqluint32   totalPages;
  sqluint32   useablePages;
  sqluint32   usedPages;
  sqluint32   freePages;
  sqluint32   highWaterMark;
};
/* ... */
```

COBOL 構造

```
* File: sqlutil.cbl
01 SQLB-TBS-STATS.
    05 SQL-TOTAL-PAGES        PIC 9(9) COMP-5.
    05 SQL-USEABLE-PAGES      PIC 9(9) COMP-5.
    05 SQL-USED-PAGES         PIC 9(9) COMP-5.
    05 SQL-FREE-PAGES         PIC 9(9) COMP-5.
    05 SQL-HIGH-WATER-MARK    PIC 9(9) COMP-5.
*
```

SQLB-TBSCONTQRY-DATA

この構造は、コンテナー・データをアプリケーション・プログラムへ戻すのに使用されます。

表 22. 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	コンテナー識別子。
NTBS	INTEGER	通常 1。
TBSID	INTEGER	表スペース識別子。
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
新しい装置を指定します (DMS のみ)。

SQLB_CONT_FILE
ファイルを指定します (DMS のみ)。

SQLB-TBSCONTQRY-DATA

言語構文

C 構造

```
/* File: sqlutil.h */
/* Structure: SQLB-TBSCONTQRY-DATA */
/* ... */
SQL_STRUCTURE SQLB_TBSCONTQRY_DATA
{
  sqluint32   id;
  sqluint32   nTbs;
  sqluint32   tbsID;
  sqluint32   nameLen;
  char        name[SQLB_MAX_CONTAIN_NAME_SZ];
  sqluint32   underDBDir;
  sqluint32   contType;
  sqluint32   totalPages;
  sqluint32   useablePages;
  sqluint32   ok;
};
/* ... */
```

COBOL 構造

```
* File: sqlutbcq.cbl
01 SQLB-TBSCONTQRY-DATA.
    05 SQL-ID                 PIC 9(9) COMP-5.
    05 SQL-N-TBS              PIC 9(9) COMP-5.
    05 SQL-TBS-ID             PIC 9(9) COMP-5.
    05 SQL-NAME-LEN           PIC 9(9) COMP-5.
    05 SQL-NAME               PIC X(256).
    05 SQL-UNDER-DBDIR        PIC 9(9) COMP-5.
    05 SQL-CONT-TYPE          PIC 9(9) COMP-5.
    05 SQL-TOTAL-PAGES        PIC 9(9) COMP-5.
    05 SQL-USEABLE-PAGES      PIC 9(9) COMP-5.
    05 SQL-OK                 PIC 9(9) COMP-5.
*
```

SQLB-TBSPQRY-DATA

この構造は、表スペース・データをアプリケーション・プログラムへ戻すのに使用されます。

表 23. 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)	構造のバージョン識別子。
ID	INTEGER	表スペースの内部識別子。
NAMELEN	INTEGER	表スペース名の長さ。
NAME	CHAR(128)	表スペースのヌル文字で終了する名前。
TOTALPAGES	INTEGER	CREATE TABLESPACE で指定されたページ数 (DMS のみ)。
USEABLEPAGES	INTEGER	TOTALPAGES からオーバーヘッドを引いた数 (DMS のみ)。この値は、次の 4KB の倍数に切り捨てられます。
FLAGS	INTEGER	表スペースのビット属性。
PAGESIZE	INTEGER	表スペースのページ・サイズ (バイト単位)。現在、4KB に固定されています。
EXTSIZE	INTEGER	表スペースのエクステント・サイズ (ページ単位)。
PREFETCHSIZE	INTEGER	事前取り出しサイズ。
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 23. 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
QUIESCEID	INTEGER	表スペースが静止する原因となったオブジェクト QUIESCEOBJ の表スペース ID。
QUIESCEOBJ	INTEGER	表スペースが静止する原因となった表スペース QUIESCEID のオブジェクト ID。
RESERVED	CHAR(32)	将来の使用のために予約されています。

FLAGS に有効な値 (sqlutil で定義) は、以下のとおりです。

SQLB_TBS_SMS
システム管理スペース

SQLB_TBS_DMS
データベース管理スペース

SQLB_TBS_ANY
正規の内容

SQLB_TBS_LONG
長フィールド・データ

SQLB_TBS_SYSTMP
システム一時データ。

SQLB_TBS_USRTMP
ユーザー一時データ。

TBSSTATE に有効な値 (sqlutil で定義) は、以下のとおりです。

SQLB_NORMAL
通常

SQLB_QUIESCED_SHARE
静止: SHARE

SQLB_QUIESCED_UPDATE
静止: UPDATE

SQLB_QUIESCED_EXCLUSIVE
静止: EXCLUSIVE

SQLB_LOAD_PENDING
ロード保留

SQLB_DELETE_PENDING
削除保留

SQLB_BACKUP_PENDING
バックアップ保留

SQLB_ROLLFORWARD_IN_PROGRESS
ロールフォワード進行中

SQLB_ROLLFORWARD_PENDING
ロールフォワード保留

SQLB_RESTORE_PENDING
復元保留

SQLB_DISABLE_PENDING
使用不能化保留

SQLB_REORG_IN_PROGRESS
再編成進行中

SQLB_BACKUP_IN_PROGRESS
バックアップ進行中

SQLB_STORDEF_PENDING
記憶域の定義が必要

SQLB_RESTORE_IN_PROGRESS
復元進行中

SQLB_STORDEF_ALLOWED
記憶域の定義が可能

SQLB_STORDEF_FINAL_VERSION
記憶域の定義が ’最終’ 状態

SQLB_STORDEF_CHANGED
ロールフォワード前に記憶域定義が変更された

SQLB_REBAL_IN_PROGRESS
DMS リバランサーが活動状態

SQLB_PSTAT_DELETION
表スペース削除の進行中

SQLB_PSTAT_CREATION
表スペース作成の進行中

FLAGS2 に有効な値 (`sqlutil` で定義) は、以下のとおりです。

SQLB_STATE_SET
サービスの目的でのみ使用されます。

言語構文

C 構造

```
/* File: sqlutil.h */
/* ... */
SQL_STRUCTURE SQLB_TBSPQRY_DATA
{
   char                         tbspqver[SQLB_SVERSION_SIZE];
   sqluint32                    id;
   sqluint32                    nameLen;
   char                         name[SQLB_MAX_TBS_NAME_SZ];
   sqluint32                    totalPages;
   sqluint32                    useablePages;
   sqluint32                    flags;
   sqluint32                    pageSize;
   sqluint32                    extSize;
   sqluint32                    prefetchSize;
   sqluint32                    nContainers;
   sqluint32                    tbsState;
   char                         lifeLSN[6];
   char                         pad[2];
   sqluint32                    flags2;
   char                         minimumRecTime[SQL_STAMP_STRLEN+1];
   char                         pad1[1];
   sqluint32                    StateChngObj;
   sqluint32                    StateChngID;
   sqluint32                    nQuiescers;
   struct SQLB_QUIESCER_DATA    quiescer[SQLB_MAX_QUIESCERS];
   char                         reserved[32];
};
/* ... */
```

```
/* File: sqlutil.h */
/* ... */
SQL_STRUCTURE SQLB_QUIESCER_DATA
{
   sqluint32   quiesceId;
   sqluint32   quiesceObject;
};
/* ... */
```

COBOL 構造

```
* File: sqlutbsp.cbl
01 SQLB-TBSPQRY-DATA.
    05 SQL-TBSPQVER           PIC X(8).
    05 SQL-ID                 PIC 9(9) COMP-5.
    05 SQL-NAME-LEN           PIC 9(9) COMP-5.
    05 SQL-NAME               PIC X(128).
    05 SQL-TOTAL-PAGES        PIC 9(9) COMP-5.
    05 SQL-USEABLE-PAGES      PIC 9(9) COMP-5.
    05 SQL-FLAGS              PIC 9(9) COMP-5.
    05 SQL-PAGE-SIZE          PIC 9(9) COMP-5.
    05 SQL-EXT-SIZE           PIC 9(9) COMP-5.
    05 SQL-PREFETCH-SIZE      PIC 9(9) COMP-5.
    05 SQL-N-CONTAINERS       PIC 9(9) COMP-5.
    05 SQL-TBS-STATE          PIC 9(9) COMP-5.
    05 SQL-LIFE-LSN           PIC X(6).
    05 SQL-PAD                PIC X(2).
    05 SQL-FLAGS2             PIC 9(9) COMP-5.
    05 SQL-MINIMUM-REC-TIME   PIC X(26).
    05 FILLER                 PIC X.
    05 SQL-PAD1               PIC X(1).
    05 SQL-STATE-CHNG-OBJ     PIC 9(9) COMP-5.
    05 SQL-STATE-CHNG-ID      PIC 9(9) COMP-5.
    05 SQL-N-QUIESCERS        PIC 9(9) COMP-5.
    05 SQL-QUIESCER OCCURS 5 TIMES.
        10 SQL-QUIESCE-ID     PIC 9(9) COMP-5.
        10 SQL-QUIESCE-OBJECT PIC 9(9) COMP-5.
    05 SQL-RESERVED           PIC X(32).
*
```

SQLCA

SQL 連絡域 (SQLCA) 構造は、データベース・マネージャーがエラー情報をアプリケーション・プログラムへ戻すのに使用するものです。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。

フィールドの説明を含め、SQLCA 構造の詳細については、 *SQL 解説書* を参照してください。

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 24. SQLCHAR 構造のフィールド

フィールド名	データ・タイプ	説明
LENGTH	SMALLINT	*DATA* が指す文字ストリングの長さ。
DATA	CHAR(n)	長さ *LENGTH* の文字の配列。

言語構文

C 構造

```
/* File: sql.h */
/* Structure: SQLCHAR */
/* ... */
SQL_STRUCTURE sqlchar
{
  short                  length;
  _SQLOLDCHAR            data[1];
};
/* ... */
```

COBOL 構造

この構造は、ヘッダー・ファイルでは定義されません。以下に示すのは、この構造をどのように定義できるかを示した一例です。

```
* maxlen を該当する値で置き換えてください。
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の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。 REXX では、SQLDA は他の言語と若干異なります。 REXX での SQLDA の使い方については、*アプリケーション開発の手引き* を参照してください。

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。

フィールドの説明を含め、SQLDA 構造の詳細については、 *SQL 解説書* を参照してください。

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 360ページの『sqluexpr - エクスポート』、 381ページの『sqluimpr - インポート』、および 408ページの『sqluload - ロード』へ渡すのに使用されます。

表 25. SQLDCOL 構造のフィールド

フィールド名	データ・タイプ	説明
DCOLMETH	SMALLINT	データ・ファイル内の列の選択に使用する方式を示す文字。
DCOLNUM	SMALLINT	*DCOLNAME* 配列で指定された列の数。
DCOLNAME	配列	*DCOLNUM* 個の *sqldcoln* 構造の配列。

表 26. 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要素を指すポインター。
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 27. 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 28. SQLLOCPAIR 構造のフィールド

フィールド名	データ・タイプ	説明
BEGIN_LOC	SMALLINT	外部ファイル内の列データの開始位置。
END_LOC	SMALLINT	外部ファイル内の列データの終了位置。

DCOLMETH に有効な値 (`sqlutil` で定義) は、以下のとおりです。

SQL_METH_N
名前。インポートまたはロード時には、この構造により提供された列名が、外部ファイルからインポートまたはロードされるデータの識別に使用されます。これらの列名の大文字小文字の区別は、システム・カタロ

グ内の対応する名前の大文字小文字の区別と一致しなければなりません。エクスポート時は、この構造により提供された列名が出力ファイル内の列名として使用されます。

dcolname 配列の各要素の *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* 配列内の要素数を示します。

外部ファイルが列名を含んでいない場合 (たとえば、DEL や ASC 形式ファイルの場合)、この方式は無効です。

SQL_METH_P

位置。インポートまたはロード時には、この構造により提供された開始列位置が、外部ファイルからインポートまたはロードされるデータの識別に使用されます。データのエクスポート時には、この方式は無効です。

dcolnlen フィールドに外部ファイルの列位置が入っていると、*dcolname* 配列の各要素の *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* 配列内の要素数を示します。

もっとも低い有効な列位置の値は、1 (最初の列を示す) です。もっとも高い有効な値は、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 配列の最初の要素の *dcolnptr* フィールドは、 *sqlloctab* 構造 (*sqllocpair* 構造の配列を構成) を指します。この構造内の要素数は、*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。配列内の各要素は、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要素は、列が始まるファイル中のバイトです。 2 番目の要素は、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 とみなされます。列は、オーバーラップさせることができます。

この方式は、ASC ファイルのインポートまたはロードについてのみ有効な方式です。

SQL_METH_D

省略時値。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、省略時名が外部ファイル内の列に使用されます。

sqldcol 構造の *dcolnum* と *dcolname* フィールドはどちらも無視され、外部ファイルの列はそのままの順序で取り出されます。

外部ファイルからの列は、配列の中で複数回使用することができます。外部ファイルからの列をすべて使用する必要はありません。

言語構文

C 構造

```
/* File: sqlutil.h */
/* Structure: SQLDCOL */
/* ... */
SQL_STRUCTURE sqldcol
{
  short           dcolmeth;
  short           dcolnum;
  struct sqldcoln dcolname[1];
};
/* ... */


/* File: sqlutil.h */
/* Structure: SQLDCOLN */
/* ... */
SQL_STRUCTURE sqldcoln
{
  short           dcolnlen;
  char            *dcolnptr;
};
/* ... */


/* File: sqlutil.h */
/* Structure: SQLLOCTAB */
/* ... */
SQL_STRUCTURE sqlloctab
{
  struct sqllocpair locpair[1];
};
/* ... */
```

```
/* File: sqlutil.h */
/* Structure: SQLLOCPAIR */
/* ... */
SQL_STRUCTURE sqllocpair
{
  short           begin_loc;
  short           end_loc;
};
/* ... */
```

COBOL 構造

```
* File: sqlutil.cbl
01 SQL-DCOLDATA.
    05 SQL-DCOLMETH           PIC S9(4) COMP-5.
    05 SQL-DCOLNUM            PIC S9(4) COMP-5.
    05 SQLDCOLN OCCURS 0 TO 255 TIMES DEPENDING ON SQL-DCOLNUM.
        10 SQL-DCOLNLEN       PIC S9(4) COMP-5.
        10 FILLER             PIC X(2).
        10 SQL-DCOLN-PTR      USAGE IS POINTER.
*


* File: sqlutil.cbl
01 SQL-LOCTAB.
    05 SQL-LOC-PAIR OCCURS 1 TIMES.
        10 SQL-BEGIN-LOC      PIC S9(4) COMP-5.
        10 SQL-END-LOC        PIC S9(4) COMP-5.
*
```

SQLE-ADDN-OPTIONS

この構造は、161ページの『sqleaddn - ノードの追加』に情報を渡すのに使用されます。

表 29. SQLE-NODE-APPN 構造のフィールド

フィールド名	データ・タイプ	説明
SQLADDID	CHAR	SQLE_ADDOPTID_V51 に設定しなければならない "目印" 値。
TBLSPACE_TYPE	sqluint32	追加されるノードのために使用されるシステム一時表スペース定義のタイプを指定します。値については、以下を参照してください。
TBLSPACE_NODE	SQL_PDB_NODE_TYPE	システム一時表スペース定義を取得するノード番号を指定します。このノード番号は、db2nodes.cfg ファイルに存在しなければならず、 *tblspace_type* フィールドが SQLE_TABLESPACES_LIKE_NODE に設定される場合にのみ使用されます。

TBLSPACE_TYPE に有効な値 (sqlenv で定義) は、以下のとおりです。

SQLE_TABLESPACES_NONE
システム一時表スペースを作成しません。

SQLE_TABLESPACES_LIKE_NODE
システム一時表スペース用のコンテナーは、指定されたノード用のものと同じでなければなりません。

SQLE_TABLESPACES_LIKE_CATALOG
システム一時表スペース用のコンテナーは、各データベースのカタログ・ノード用のものと同じでなければなりません。

言語構文

C 構造

```
/* File: sqlenv.h */
/* Structure: SQLE-ADDN-OPTIONS */
/* ... */
SQL_STRUCTURE sqle_addn_options
{
  char                   sqladdid[8];
  sqluint32              tblspace_type;
  SQL_PDB_NODE_TYPE      tblspace_node;
};
/* ... */
```

COBOL 構造

```
* File: sqlenv.cbl
01 SQLE-ADDN-OPTIONS.
    05 SQLADDID               PIC X(8).
    05 SQL-TBLSPACE-TYPE      PIC 9(9) COMP-5.
    05 SQL-TBLSPACE-NODE      PIC S9(4) COMP-5.
    05 FILLER                 PIC X(2).
*
```

SQLE-CLIENT-INFO

この構造は、298ページの『sqleseti - クライアント情報の設定』および 283ページの『sqleqryi - クライアント情報の照会』に情報を渡すのに使用されます。

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会計ストリング。最大で 200 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: 情報の設定には、289ページの『sqlesact - 会計ストリングの設定』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 30. 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。

表 30. SQLE-CLIENT-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
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、ヌル値になります。 **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、ヌル値になります。
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 要素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 31. 接続設定

種類	データ・タイプ	説明
SQLE_CLIENT_INFO_USERID	CHAR(255)	クライアントのユーザー ID。サーバーによってはこの値を切り捨てるものもあります。たとえば、DB2 (OS/390 版) サーバーがサポートしている長さは最大 16 文字です。このユーザー ID は識別目的でのみ使用され、許可のために使われることはありません。
SQLE_CLIENT_INFO_ WRKSTNNAME	CHAR(255)	クライアントのワークステーション名。サーバーによってはこの値を切り捨てるものもあります。たとえば、DB2 (OS/390 版) サーバーがサポートしている長さは最大 18 文字です。

表 31. 接続設定 (続き)

種類	データ・タイプ	説明
SQLE_CLIENT_INFO_APPLNAME	CHAR(255)	クライアントのアプリケーション名。サーバーによってはこの値を切り捨てるものもあります。たとえば、DB2 (OS/390 版) 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_ACCTSTR	CHAR(200)	クライアントの会計ストリング。サーバーによってはこの値を切り捨てるものもあります。たとえば、DB2 (OS/390 版) サーバーがサポートしている長さは最大 200 文字です。
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、同じ意味の類似した名前があります。		

言語構文

C 構造

```
/* File: sqlenv.h */
/* Structure: SQLE-CLIENT-INFO */
/* ... */
SQL_STRUCTURE sqle_client_info
{
  unsigned short         type;
  unsigned short         length;
  char                   *pValue;
};
/* ... */
```

COBOL 構造

```
* File: sqlenv.cbl
01 SQLE-CLIENT-INFO.
    05 SQLE-CLIENT-INFO-ITEM OCCURS 4 TIMES.
        10 SQLE-CLIENT-INFO-TYPE   PIC S9(4) COMP-5.
        10 SQLE-CLIENT-INFO-LENGTH PIC S9(4) COMP-5.
        10 SQLE-CLIENT-INFO-VALUE  USAGE IS POINTER.
*
```

SQLE-CONN-SETTING

この構造は、接続設定のタイプおよび値を指定するのに使用します (280ページの『sqleqryc - クライアントの照会』、および 294ページの『sqlesetc - クライアントの設定』を参照してください)。

表 32. SQLE-CONN-SETTING 構造のフィールド

フィールド名	データ・タイプ	説明
TYPE	SMALLINT	設定のタイプ。
VALUE	SMALLINT	設置値。

SQLE-CONN-SETTING TYPE 要素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 33. 接続設定

種類	値	説明
SQL_CONNECT_TYPE	SQL_CONNECT_1	Type 1 CONNECT は、旧リリースの作業単位の意味 (リモート作業単位 (RUOW) に関する規則とも呼ばれる) ごとに 1 つのデータベースを適用します。
	SQL_CONNECT_2	Type 2 CONNECT は、DUOW の作業単位の意味ごとに複数のデータベースをサポートします。
SQL_RULES	SQL_RULES_DB2	現行接続を、確立済みの (休止) 接続に切り替える SQL CONNECT ステートメントを使用可能にします。
	SQL_RULES_STD	SQL CONNECT ステートメントによる新規接続の確立だけを許可します。現行の接続から休止接続へ切り替えるには、SQL SET CONNECTION ステートメントを使用する必要があります。
SQL_DISCONNECT	SQL_DISCONNECT_EXPL	コミット時に SQL RELEASE ステートメントにより解放されるように明示的にマークされた接続を除去します。

表 33. 接続設定 (続き)

種類	値	説明
	SQL_DISCONNECT_COND	コミット時にオープン状態の WITH HOLD カーソルを持たない接続、および SQL RELEASE ステートメントにより解放されるようにマークされた接続を切断します。
	SQL_DISCONNECT_AUTO	コミット時にすべての接続を切断します。
SQL_SYNCPOINT	SQL_SYNC_TWOPHASE	トランザクション管理プログラム (TM) に、このプロトコルをサポートするデータベース間で 2 フェーズ・コミットを調整するよう要求します。
	SQL_SYNC_ONEPHASE	1 フェーズ・コミットを使用して、複数のデータベース・トランザクション内のそれぞれのデータベースにより行われた作業をコミットします。単一の更新プログラムに、複数の読み取り動作という形を適用します。
	SQL_SYNC_NONE	1 フェーズ・コミットを使用して、行われた処理をコミットしますが、単一の更新プログラムに、複数の読み取り動作という形を適用しません。
SQL_MAX_NETBIOS_ CONNECTIONS	1 〜 254	特定のアプリケーションで、NETBIOS アダプターを使用して確立できる同時接続の最大数を指定します。
SQL_DEFERRED_PREPARE	SQL_DEFERRED_PREPARE_NO	PREPARE ステートメントは、それが発行されたときに実行されます。

表 33. 接続設定 (続き)

種類	値	説明
	SQL_DEFERRED_PREPARE_YES	PREPARE ステートメントの実行は、対応する OPEN、DESCRIBE、または EXECUTE ステートメントが発行されるまで据え置かれます。 PREPARE ステートメントは、INTO 文節 (SQLDA がすぐに戻されることを必要とする) を使用する場合には据え置かれません。ただし、パラメーター・マーカーを使用しないカーソルについて PREPARE INTO ステートメントが発行された場合には、PREPARE の実行時にカーソルを事前オープンすることによって、処理が最適化されます。
	SQL_DEFERRED_PREPARE_ALL	パラメーター・マーカーを含む PREPARE INTO ステートメントが*据え置かれる* 点を除き、YES と同じです。 PREPARE INTO ステートメントにパラメーター・マーカーが含まれていない場合には、カーソルの事前オープンが依然として実行されます。 PREPARE ステートメントが SQLDA を戻すために INTO 文節を使用する場合、アプリケーションでは、 OPEN、DESCRIBE、または EXECUTE ステートメントが発行され、戻されるまで、この SQLDA の内容を参照してはなりません。

表 33. 接続設定 (続き)

種類	値	説明
SQL_CONNECT_NODE	0 ～ 999 の範囲、あるいはキーワード SQL_CONN_CATALOG_NODE。	接続が確立される先のノードを指定します。環境変数 **DB2NODE** の値をオーバーライドします。 たとえば、ノード 1、2、および 3 が定義されている場合、クライアントはこれらのノードの 1 つにアクセスできれば十分です。データベースを含むノード 1 だけがカタログされており、このパラメーターが 3 に設定されると、次の接続の試みは、ノード 1 での初期接続の後、ノード 3 での接続になります。
SQL_ATTACH_NODE	0 ～ 999 の範囲。	接続が確立される先のノードを指定します。環境変数 **DB2NODE** の値をオーバーライドします。 たとえば、ノード 1、2、および 3 が定義されている場合、クライアントはこれらのノードの 1 つにアクセスできれば十分です。データベースを含むノード 1 だけがカタログされており、このパラメーターが 3 に設定されると、次の接続の試みは、ノード 1 での初期接続の後、ノード 3 での接続になります。
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、同じ意味の類似した名前があります。		

言語構文

C 構造

```
/* File: sqlenv.h */
/* Structure: SQLE-CONN-SETTING */
/* ... */
SQL_STRUCTURE sqle_conn_setting
{
  unsigned short         type;
  unsigned short         value;
};
/* ... */
```

COBOL 構造

```
* File: sqlenv.cbl
01 SQLE-CONN-SETTING.
    05 SQLE-CONN-SETTING-ITEM OCCURS 7 TIMES.
        10 SQLE-CONN-TYPE  PIC S9(4) COMP-5.
        10 SQLE-CONN-VALUE PIC S9(4) COMP-5.
*
```

SQLE-NODE-APPC

この構造は、APPC ノードをカタログするのに使用されます (196ページの『sqlectnd - ノードのカタログ』を参照)。

表 34. SQLE-NODE-APPC 構造のフィールド

フィールド名	データ・タイプ	説明
LOCAL_LU	CHAR(8)	ローカル LU 名。
PARTNER_LU	CHAR(8)	パートナー LU 名の別名。
MODE	CHAR(8)	モード。
注: この構造に渡された文字フィールドは、ヌル終了するか、フィールドの長さいっぱいにまでブランクを埋め込む必要があります。		

言語構文

C 構造

```
/* File: sqlenv.h */
/* Structure: SQLE-NODE-APPC */
/* ... */
SQL_STRUCTURE sqle_node_appc
{
  _SQLOLDCHAR    local_lu[SQL_LOCLU_SZ + 1];
  _SQLOLDCHAR    partner_lu[SQL_RMTLU_SZ + 1];
  _SQLOLDCHAR    mode[SQL_MODE_SZ + 1];
};
/* ... */
```

COBOL 構造

```
* File: sqlenv.cbl
01 SQL-NODE-APPC.
    05 LOCAL-LU               PIC X(8).
    05 FILLER                 PIC X.
    05 PARTNER-LU             PIC X(8).
    05 FILLER                 PIC X.
    05 TRANS-MODE             PIC X(8).
    05 FILLER                 PIC X.
*
```

SQLE-NODE-APPN

この構造は、APPN ノードをカタログするのに使用されます (196ページの『sqlectnd - ノードのカタログ』を参照)。

表 35. SQLE-NODE-APPN 構造のフィールド

フィールド名	データ・タイプ	説明
NETWORKID	CHAR(8)	ネットワーク ID。
REMOTE_LU	CHAR(8)	リモート LU 名の別名。
LOCAL_LU	CHAR(8)	ローカル LU 名の別名。
MODE	CHAR(8)	モード。
注: この構造に渡された文字フィールドは、ヌル終了するか、フィールドの長さいっぱいにまでブランクを埋め込む必要があります。		

言語構文

C 構造

```
/* File: sqlenv.h */
/* Structure: SQLE-NODE-APPN */
/* ... */
SQL_STRUCTURE sqle_node_appn
{
  _SQLOLDCHAR    networkid[SQL_NETID_SZ + 1];
  _SQLOLDCHAR    remote_lu[SQL_RMTLU_SZ + 1];
  _SQLOLDCHAR    local_lu[SQL_LOCLU_SZ + 1];
  _SQLOLDCHAR    mode[SQL_MODE_SZ + 1];
};
/* ... */
```

COBOL 構造

```
* File: sqlenv.cbl
01 SQL-NODE-APPN.
    05 NETWORKID             PIC X(8).
    05 FILLER                PIC X.
    05 REMOTE-LU             PIC X(8).
    05 FILLER                PIC X.
    05 LOCAL-LU              PIC X(8).
    05 FILLER                PIC X.
    05 TRANS-MODE            PIC X(8).
    05 FILLER                PIC X.
*
```

SQLE-NODE-CPIC

この構造は、CPIC ノードをカタログするのに使用されます (196ページの『sqlectnd - ノードのカタログ』を参照)。

表 36. SQLE-NODE-CPIC 構造のフィールド

フィールド名	データ・タイプ	説明
SYM_DEST_NAME	CHAR(8)	リモート・パートナーの記号宛先名。
SECURITY_TYPE	SMALLINT	機密保護タイプ。
注: この構造に渡された文字フィールドは、ヌル終了するか、フィールドの長さいっぱいにまでブランクを埋め込む必要があります。		

SECURITY_TYPE に有効な値 (sqlenv で定義) は、以下のとおりです。

SQL_CPIC_SECURITY_NONE

SQL_CPIC_SECURITY_SAME

SQL_CPIC_SECURITY_PROGRAM

言語構文

C 構造

```
/* File: sqlenv.h */
/* Structure: SQLE-NODE-CPIC */
/* ... */
SQL_STRUCTURE sqle_node_cpic
{
  _SQLOLDCHAR    sym_dest_name[SQL_SYM_DEST_NAME_SZ+1];
  unsigned short security_type;
};
/* ... */
```

COBOL 構造

```
* File: sqlenv.cbl
01 SQL-NODE-CPIC.
    05 SYM-DEST-NAME          PIC X(8).
    05 FILLER                 PIC X.
    05 FILLER                 PIC X(1).
    05 SECURITY-TYPE          PIC 9(4) COMP-5.
*
```

SQLE-NODE-IPXSPX

この構造は、IPX/SPX ノードをカタログするのに使用されます (196ページの『sqlectnd - ノードのカタログ』 を参照)。

表 37. SQLE-NODE-IPXSPX 構造のフィールド

フィールド名	データ・タイプ	説明
FILESERVER	CHAR(48)	DB2 サーバー・インスタンスが登録されている NetWare ファイル・サーバーの名前。
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・サーバー上でオブジェクト *objectname* として表現されます。サーバーの IPX/SPX インターネットワーク・アドレスは、このオブジェクトから保管および検索されます。
注: この構造に渡された文字フィールドは、ヌル終了するか、フィールドの長さいっぱいにまでブランクを埋め込む必要があります。		

言語構文

C 構造

```
/* File: sqlenv.h */
/* Structure: SQLE-NODE-IPXSPX */
/* ... */
SQL_STRUCTURE sqle_node_ipxspx
{
  char           fileserver[SQL_FILESERVER_SZ+1];
  char           objectname[SQL_OBJECTNAME_SZ+1];
};
/* ... */
```

COBOL 構造

```
* File: sqlenv.cbl
01 SQL-NODE-IPXSPX.
    05 SQL-FILESERVER         PIC X(48).
    05 FILLER                 PIC X.
    05 SQL-OBJECTNAME         PIC X(48).
    05 FILLER                 PIC X.
*
```

SQLE-NODE-LOCAL

この構造は、ローカル・ノードをカタログするのに使用されます (196ページの『sqlectnd - ノードのカタログ』を参照)。

表 38. 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、ヌル終了するか、フィールドの長さいっぱいにまでブランクを埋め込む必要があります。		

言語構文

C 構造

```
/* File: sqlenv.h */
/* Structure: SQLE-NODE-LOCAL */
/* ... */
SQL_STRUCTURE sqle_node_local
{
  char           instance_name[SQL_INSTNAME_SZ+1];
};
/* ... */
```

COBOL 構造

```
* File: sqlenv.cbl
01 SQL-NODE-LOCAL.
    05 SQL-INSTANCE-NAME      PIC X(8).
    05 FILLER                 PIC X.
*
```

SQLE-NODE-NETB

この構造は、NetBIOS をカタログするのに使用されます (196ページの『sqlectnd - ノードのカタログ』を参照)。

表 39. SQLE-NODE-NETB 構造のフィールド

フィールド名	データ・タイプ	説明
ADAPTER	SMALLINT	ローカル LAN アダプター。
REMOTE_NNAME	CHAR(8)	サーバー・インスタンス上のデータベース・マネージャー構成ファイルに保管されている、リモート・ワークステーションの *Nname*。
注: この構造に渡された文字フィールドは、ヌル終了するか、フィールドの長さいっぱいにまでブランクを埋め込む必要があります。		

言語構文

C 構造

```
/* File: sqlenv.h */
/* Structure: SQLE-NODE-NETB */
/* ... */
SQL_STRUCTURE sqle_node_netb
{
  unsigned short adapter;
  _SQLOLDCHAR    remote_nname[SQL_RMTLU_SZ + 1];
};
/* ... */
```

COBOL 構造

```
* File: sqlenv.cbl
01 SQL-NODE-NETB.
    05 ADAPTER                PIC 9(4) COMP-5.
    05 REMOTE-NNAME           PIC X(8).
    05 FILLER                 PIC X.
    05 FILLER                 PIC X(1).
*
```

SQLE-NODE-NPIPE

この構造は、名前付きパイプ・ノードをカタログするのに使用されます (196ページの『sqlectnd - ノードのカタログ』を参照)。

表 40. 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、ヌル終了するか、フィールドの長さいっぱいにまでブランクを埋め込む必要があります。		

言語構文

C 構造

```
/* File: sqlenv.h */
/* Structure: SQLE-NODE-NPIPE */
/* ... */
SQL_STRUCTURE sqle_node_npipe
{
  char           computername[SQL_COMPUTERNAME_SZ+1];
  char           instance_name[SQL_INSTNAME_SZ+1];
};
/* ... */
```

COBOL 構造

```
* File: sqlenv.cbl
01 SQL-NODE-NPIPE.
    05 COMPUTERNAME           PIC X(15).
    05 FILLER                 PIC X.
    05 INSTANCE-NAME          PIC X(8).
    05 FILLER                 PIC X.
*
```

SQLE-NODE-STRUCT

この構造は、ノードをカタログするのに使用されます (196ページの『sqlectnd - ノードのカタログ』を参照)。

表 41. 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	構造識別子。
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、ヌル終了するか、フィールドの長さいっぱいにまでブランクを埋め込む必要があります。		

PROTOCOL に有効な値 (sqlenv で定義) は、以下のとおりです。

SQL_PROTOCOL_APPC

SQL_PROTOCOL_APPN

SQL_PROTOCOL_CPIC

SQL_PROTOCOL_IPXSPX

SQL_PROTOCOL_LOCAL

SQL_PROTOCOL_NETB

SQL_PROTOCOL_NPIPE

SQL_PROTOCOL_SOCKS

SQL_PROTOCOL_TCPIP

言語構文

C 構造

```
/* File: sqlenv.h */
/* Structure: SQLE-NODE-STRUCT */
/* ... */
SQL_STRUCTURE sqle_node_struct
{
  unsigned short struct_id;
  unsigned short codepage;
  _SQLOLDCHAR    comment[SQL_CMT_SZ + 1];
  _SQLOLDCHAR    nodename[SQL_NNAME_SZ + 1];
  unsigned char  protocol;
};
/* ... */
```

COBOL 構造

```
* File: sqlenv.cbl
01 SQL-NODE-STRUCT.
    05 STRUCT-ID             PIC 9(4) COMP-5.
    05 CODEPAGE              PIC 9(4) COMP-5.
    05 COMMENT               PIC X(30).
    05 FILLER                PIC X.
    05 NODENAME              PIC X(8).
    05 FILLER                PIC X.
    05 PROTOCOL              PIC X.
    05 FILLER                PIC X(1).
*
```

SQLE-NODE-TCPIP

この構造は、TCP/IP ノードをカタログするのに使用されます (196ページの『sqlectnd - ノードのカタログ』を参照)。

注: TCP/IP SOCKS ノードをカタログするには、**sqlectnd** API を呼び出す前に、ノード・ディレクトリー構造の PROTOCOL タイプを SQL_PROTOCOL_SOCKS に設定してください (547ページの『SQLE-NODE-STRUCT』を参照)。

表 42. SQLE-NODE-TCPIP 構造のフィールド

フィールド名	データ・タイプ	説明
HOSTNAME	CHAR(255)	DB2 サーバー・インスタンスが存在する TCP/IP ホストの名前。
SERVICE_NAME	CHAR(14)	DB2 サーバー・インスタンスの TCP/IP サービス名または関連するポート番号。
注: この構造に渡された文字フィールドは、ヌル終了するか、フィールドの長さいっぱいにまでブランクを埋め込む必要があります。		

言語構文

C 構造

```
/* File: sqlenv.h */
/* Structure: SQLE-NODE-TCPIP */
/* ... */
SQL_STRUCTURE sqle_node_tcpip
{
  _SQLOLDCHAR    hostname[SQL_HOSTNAME_SZ+1];
  _SQLOLDCHAR    service_name[SQL_SERVICE_NAME_SZ+1];
};
/* ... */
```

COBOL 構造

```
* File: sqlenv.cbl
01 SQL-NODE-TCPIP.
    05 HOSTNAME               PIC X(255).
    05 FILLER                 PIC X.
    05 SERVICE-NAME           PIC X(14).
    05 FILLER                 PIC X.
*
```

SQLE-REG-NWBINDERY

この構造は、NetWare ファイル・サーバー上のバインダリーに DB2 サーバーを登録するか、あるいはバインダリーから DB2 サーバーの登録を解除するのに使用されます (286ページの『sqleregs - 登録』および 219ページの『sqledreg - 登録解除』を参照してください)。

表 43. SQLE-REG-NWBINDERY 構造のフィールド

フィールド名	データ・タイプ	説明
UID	CHAR(48)	NetWare ファイル・サーバーへのログインに使用されるユーザー ID。
PSWD	CHAR(128)	ユーザー ID の妥当性検査に使用されるパスワード。

言語構文

C 構造

```
/* File: sqlenv.h */
/* Structure: SQLE-REG-NWBINDERY */
/* ... */
SQL_STRUCTURE sqle_reg_nwbindery
{
  char                  uid[SQL_NW_UID_SZ+1];
  unsigned short        reserved_len_1;
  char                  pswd[SQL_NW_PSWD_SZ+1];
  unsigned short        reserved_len_2;
};
/* ... */
```

COBOL 構造

```
* File: sqlenv.cbl
01 SQLE-REG-NWBINDERY.
    05 SQL-UID               PIC X(48).
    05 FILLER                PIC X.
    05 FILLER                PIC X(1).
    05 SQL-UID-LEN           PIC 9(4) COMP-5.
    05 SQL-PSWD              PIC X(128).
    05 FILLER                PIC X.
    05 FILLER                PIC X(1).
    05 SQL-PSWD-LEN          PIC 9(4) COMP-5.
*
```

SQLE-START-OPTIONS

この構造は、データベース・マネージャー始動オプションを提供するのに使用されます。

表 44. SQLE-START-OPTIONS 構造のフィールド

フィールド名	データ・タイプ	説明
SQLOPTID	CHAR	SQLE_STARTOPTID_V51 に設定しなければならない "目印" 値。
ISPROFILE	sqluint32	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。
PROFILE	CHAR(236)	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。省略時値は db2profile です。
ISNODENUM	sqluint32	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。
NODENUM	SQL_PDB_NODE_TYPE	ノード番号。
OPTION	sqluint32	アクションを指定します。値については、以下を参照してください。
ISHOSTNAME	sqluint32	ホスト名が指定されるかどうかを示します。
HOSTNAME[a]	CHAR(256)	システム名。
ISPORT	sqluint32	ポート番号が指定されるかどうかを示します。
PORT[a]	SQL_PDB_PORT_TYPE	ポート番号。
ISNETNAME	sqluint32	ネット名が指定されるかどうかを示します。
NETNAME[a]	CHAR(256)	ネット名
TBLSPACE_TYPE	sqluint32	追加されるノードのために使用されるシステム一時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。

表 44. SQLE-START-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_NODE	SQL_PDB_NODE_TYPE	システム一時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。
ISCOMPUTER	sqluint32	コンピューター名が指定されるかどうかを示します。 OS/2 または Windows オペレーティング・システムでのみ有効です。
COMPUTER	CHAR(16)	コンピューター名。 OS/2 または Windows オペレーティング・システムでのみ有効です。
PUSERNAME	CHAR	ログオン・アカウント・ユーザー名。 OS/2 または Windows オペレーティング・システムでのみ有効です。
PPASSWORD	CHAR	ログオン・アカウント・パスワード。 OS/2 または Windows オペレーティング・システムでのみ有効です。
[a] このフィールドは、 *OPTION* フィールドの値が SQLE_ADDNODE または SQLE_RESTART である場合にのみ有効です。		

OPTION に有効な値 (sqlenv で定義) は、以下のとおりです。

SQLE_NONE
通常の db2start 操作を発行します。

SQLE_ADDNODE
ADD NODE コマンドを発行します。

SQLE_RESTART
RESTART DATABASE コマンドを発行します。

SQLE_STANDALONE
STANDALONE モードでノードを開始します。

これらのオプションの詳細については、*コマンド解説書* を参照してください。

TBLSPACE_TYPE に有効な値 (sqlenv で定義) は、以下のとおりです。

SQLE_TABLESPACES_NONE
システム一時表スペースを作成しません。

SQLE_TABLESPACES_LIKE_NODE
システム一時表スペース用のコンテナーは、指定されたノード用のものと同じでなければなりません。

SQLE_TABLESPACES_LIKE_CATALOG
システム一時表スペース用のコンテナーは、各データベースのカタログ・ノード用のものと同じでなければなりません。

言語構文

C 構造

```
/* File: sqlenv.h */
/* Structure: SQLE-START-OPTIONS */
/* ... */
SQL_STRUCTURE sqle_start_options
{
  char                  sqloptid[8];
  sqluint32             isprofile;
  char                  profile[SQL_PROFILE_SZ+1];
  sqluint32             isnodenum;
  SQL_PDB_NODE_TYPE     nodenum;
  sqluint32             option;
  sqluint32             ishostname;
  char                  hostname[SQL_HOSTNAME_SZ+1];
  sqluint32             isport;
  SQL_PDB_PORT_TYPE     port;
  sqluint32             isnetname;
  char                  netname[SQL_HOSTNAME_SZ+1];
  sqluint32             tblspace_type;
  SQL_PDB_NODE_TYPE     tblspace_node;
  sqluint32             iscomputer;
  char                  computer[SQL_COMPUTERNAME_SZ+1];
  char                  *pUserName;
  char                  *pPassword;
};
/* ... */
```

COBOL 構造

```
* File: sqlenv.cbl
01 SQLE-START-OPTIONS.
    05 SQLOPTID              PIC X(8).
    05 SQL-ISPROFILE         PIC 9(9) COMP-5.
    05 SQL-PROFILE           PIC X(235).
    05 FILLER                PIC X.
    05 SQL-ISNODENUM         PIC 9(9) COMP-5.
    05 SQL-NODENUM           PIC S9(4) COMP-5.
    05 FILLER                PIC X(2).
    05 SQL-OPTION            PIC 9(9) COMP-5.
    05 SQL-ISHOSTNAME        PIC 9(9) COMP-5.
    05 SQL-HOSTNAME          PIC X(255).
    05 FILLER                PIC X.
    05 SQL-ISPORT            PIC 9(9) COMP-5.
    05 SQL-PORT              PIC S9(9) COMP-5.
    05 SQL-ISNETNAME         PIC 9(9) COMP-5.
    05 SQL-NETNAME           PIC X(255).
    05 FILLER                PIC X.
    05 SQL-TBLSPACE-TYPE     PIC 9(9) COMP-5.
    05 SQL-TBLSPACE-NODE     PIC S9(4) COMP-5.
    05 FILLER                PIC X(2).
    05 SQL-ISCOMPUTER        PIC 9(9) COMP-5.
    05 SQL-COMPUTER          PIC X(15).
    05 FILLER                PIC X.
    05 SQL-P-USER-NAME       USAGE IS POINTER.
    05 SQL-P-PASSWORD        USAGE IS POINTER.
*
```

SQLEDBCOUNTRYINFO

この構造は、コード・セットおよび地域オプションを 186ページの『sqlecrea - データベースの作成』へ渡すのに使用されます。

表 45. SQLEDBCOUNTRYINFO 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)	データベース地域。

言語構文

C 構造

```
/* File: sqlenv.h */
/* Structure: SQLEDBCOUNTRYINFO */
/* ... */
SQL_STRUCTURE sqledbcountryinfo
{
  char                  sqldbcodeset[SQL_CODESET_LEN + 1];
  char                  sqldblocale[SQL_LOCALE_LEN + 1];
};
/* ... */
```

COBOL 構造

```
* File: sqlenv.cbl
01 SQLEDBCOUNTRYINFO.
    05 SQLDBCODESET           PIC X(9).
    05 FILLER                 PIC X.
    05 SQLDBLOCALE            PIC X(5).
    05 FILLER                 PIC X.
*
```

SQLEDBDESC

データベース記述ブロック (SQLEDBDESC) 構造は、 186ページの『sqlecrea - データベースの作成』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のに使用できます。これらの属性には、データベース・コメント、照合順序、および表スペース定義が含まれます。

表 46. 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識別子および "目印"。値 SQLE_DBDESC_2 (sqlenv で定義) で初期設定されなければならない 8 バイトのストリングです。このフィールドの内容に対して、バージョン制御を目的とした妥当性の検査がなされます。
SQLDBCCP	INTEGER	データベース・コメントのコード・ページ。この値は、データベース・マネージャーによって使用されなくなりました。
SQLDBCSS	INTEGER	データベース照合順序のソースを示す値。値については、以下を参照してください。 **注:** データベースの作成時に IDENTITY 照合順序を指定するには、 SQL_CS_NONE (2 進照合順序を実装している) を指定してください。
SQLDBUDC	CHAR(256)	このフィールドの *n* 番目のバイトには、基礎となる 10 進表記がデータベースのコード・ページ中で *n* であるコード・ポイントの分類の重みが含まれます。 SQLDBCSS が SQL_CS_USER と等しくない場合、このフィールドは無視されます。
SQLDBCMT	CHAR(30)	データベースのコメント。
SQLDBSGP	INTEGER	予約フィールド。使用されなくなりました。
SQLDBNSG	SHORT	データベース内で作成されるファイル・セグメントの数を示す値。このフィールドの最小値は 1 であり、最大値は 256 です。 -1 の値が提供されると、このフィールドは 1 に省略時解釈されます。 **注:** ゼロに設定された SQLDBNSG は、バージョン 1 との互換性のために、省略時値をもたらします。
SQLTSEXT	INTEGER	データベース中にある各表スペースの省略時のエクステント・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 32 に省略時解釈されます。
SQLCATTS	ポインター	カタログ表スペースを定義する表スペース記述制御ブロック SQLETSDESC を指すポインター。ヌルである場合、 SQLTSEXT および SQLDBNSG の値に基づく省略時のカタログ表スペースが作成されます。

表 46. SQLEDBDESC 構造のフィールド (続き)

フィールド名	データ・タイプ	説明
SQLUSRTS	ポインター	ユーザー表スペースを定義する表スペース記述制御ブロック SQLETSDESC を指すポインター。ヌルである場合、 SQLTSEXT および SQLDBNSG の値に基づく省略時のユーザー表スペースが作成されます。
SQLTMPTS	ポインター	システム一時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。ヌルである場合、 SQLTSEXT および SQLDBNSG の値に基づく省略時のシステム一時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 47. 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識別子および "目印"。値 SQLE_DBTSDESC_2 (sqlenv で定義) で初期設定されなければならない 8 バイトのストリングです。このフィールドの内容に対して、バージョン制御を目的とした妥当性の検査がなされます。
SQLEXTNT	INTEGER	表スペースのエクステント・サイズ (4KB ページ単位)。 -1 の値が提供されると、このフィールドは *dft_extent_sz* 構成パラメーターの現行値に省略時解釈されます。
SQLPRFTC	INTEGER	表スペースの事前取り出しサイズ (4KB ページ単位)。 -1 の値が提供されると、このフィールドは *dft_prefetch_sz* 構成パラメーターの現行値に省略時解釈されます。
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、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) に省略時解釈されます。
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、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) に省略時解釈されます。
SQLTSTYP	CHAR(1)	表スペースがシステムによって管理されるか、データベースによって管理されるかを示します。値については、以下を参照してください。
SQLCCNT	SMALLINT	表スペースに割り当てられるコンテナーの数。後続の SQLCTYPE/SQLCSIZE/SQLCLEN/SQLCONTR 値の数を示します。
CONTAINR	配列	*sqlccnt* 個の *SQLETSCDESC* 構造の配列。

表 48. 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 に有効な値 (sqlenv で定義) は、以下のとおりです。

SQL_CS_SYSTEM
システムからの照合順序。

SQL_CS_USER
ユーザーからの照合順序。

SQL_CS_NONE
なし。

SQLE_CS_COMPATABILITY
バージョン 5 以前の照合順序を使用。

SQLTSTYP に有効な値 (sqlenv で定義) は、以下のとおりです。

SQL_TBS_TYP_SMS
システム管理。

SQL_TBS_TYP_DMS
データベース管理。

SQLCTYPE に有効な値 (sqlenv で定義) は、以下のとおりです。

SQL_TBSC_TYP_DEV
装置。 *SQLTSTYP* = SQL_TBS_TYP_DMS の場合のみ有効。

SQL_TBSC_TYP_FILE
ファイル。 *SQLTSTYP* = SQL_TBS_TYP_DMS の場合のみ有効。

SQL_TBSC_TYP_PATH
パス (ディレクトリー)。 *SQLTSTYP* = SQL_TBS_TYP_SMS の場合のみ有効。

言語構文

C 構造

```
/* File: sqlenv.h */
/* Structure: SQLEDBDESC */
/* ... */
SQL_STRUCTURE sqledbdesc
{
  _SQLOLDCHAR    sqldbdid[8];
  sqlint32       sqldbccp;
  sqlint32       sqldbcss;
  unsigned char  sqldbudc[SQL_CS_SZ];
  _SQLOLDCHAR    sqldbcmt[SQL_CMT_SZ+1];
  _SQLOLDCHAR    pad[1];
  sqluint32      sqldbsgp;
  short          sqldbnsg;
  char           pad2[2];
  sqlint32       sqltsext;
  struct SQLETSDESC *sqlcatts;
  struct SQLETSDESC *sqlusrts;
  struct SQLETSDESC *sqltmpts;
};
/* ... */


/* File: sqlenv.h */
/* Structure: SQLETSDESC */
/* ... */
SQL_STRUCTURE SQLETSDESC
{
  char           sqltsdid[8];
  sqlint32       sqlextnt;
  sqlint32       sqlprftc;
  double         sqlpovhd;
  double         sqltrfrt;
  char           sqltstyp;
  char           pad1;
  short          sqlccnt;
  struct SQLETSCDESC containr[1];
};
/* ... */
```

```
/* File: sqlenv.h */
/* Structure: SQLETSCDESC */
/* ... */
SQL_STRUCTURE SQLETSCDESC
{
  char           sqlctype;
  char           pad1[3];
  sqlint32       sqlcsize;
  short          sqlclen;
  char           sqlcontr[SQLB_MAX_CONTAIN_NAME_SZ];
  char           pad2[2];
};
/* ... */
```

COBOL 構造

```
* File: sqlenv.cbl
01 SQLEDBDESC.
    05 SQLDBDID               PIC X(8).
    05 SQLDBCCP               PIC S9(9) COMP-5.
    05 SQLDBCSS               PIC S9(9) COMP-5.
    05 SQLDBUDC               PIC X(256).
    05 SQLDBCMT               PIC X(30).
    05 FILLER                 PIC X.
    05 SQL-PAD                PIC X(1).
    05 SQLDBSGP               PIC 9(9) COMP-5.
    05 SQLDBNSG               PIC S9(4) COMP-5.
    05 SQL-PAD2               PIC X(2).
    05 SQLTSEXT               PIC S9(9) COMP-5.
    05 SQLCATTS               USAGE IS POINTER.
    05 SQLUSRTS               USAGE IS POINTER.
    05 SQLTMPTS               USAGE IS POINTER.
*


* File: sqletsd.cbl
01 SQLETSDESC.
    05 SQLTSDID               PIC X(8).
    05 SQLEXTNT               PIC S9(9) COMP-5.
    05 SQLPRFTC               PIC S9(9) COMP-5.
    05 SQLPOVHD               USAGE COMP-2.
    05 SQLTRFRT               USAGE COMP-2.
    05 SQLTSTYP               PIC X.
    05 SQL-PAD1               PIC X.
    05 SQLCCNT                PIC S9(4) COMP-5.
    05 SQL-CONTAINR OCCURS 001 TIMES.
        10 SQLCTYPE           PIC X.
        10 SQL-PAD1           PIC X(3).
        10 SQLCSIZE           PIC S9(9) COMP-5.
        10 SQLCLEN            PIC S9(4) COMP-5.
        10 SQLCONTR           PIC X(256).
        10 SQL-PAD2           PIC X(2).
*
```

```
* File: sqlenv.cbl
01 SQLETSCDESC.
    05 SQLCTYPE              PIC X.
    05 SQL-PAD1              PIC X(3).
    05 SQLCSIZE              PIC S9(9) COMP-5.
    05 SQLCLEN               PIC S9(4) COMP-5.
    05 SQLCONTR              PIC X(256).
    05 SQL-PAD2              PIC X(2).
*
```

SQLEDBSTOPOPT

この構造は、データベース・マネージャー停止オプションを提供するのに使用されます。

表 49. SQLEDBSTOPOPT 構造のフィールド

フィールド名	データ・タイプ	説明
ISPROFILE	sqluint32	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。
PROFILE	CHAR(236)	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)。 273ページの『sqlepstart - データベース・マネージャーの始動』で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。
ISNODENUM	sqluint32	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。
NODENUM	SQL_PDB_NODE_TYPE	ノード番号。
OPTION	sqluint32	オプション。
CALLERAC	sqluint32	呼び出し側のアクション。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。

OPTION に有効な値 (sqlenv で定義) 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除去します。

これらのオプションの詳細については、*コマンド解説書* を参照してください。

CALLERAC に有効な値 (sqlenv で定義) は、以下のとおりです。

SQLE_DROP
最初の呼び出し。これは省略時値です。

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。

言語構文

C 構造

```
/* File: sqlenv.h */
/* Structure: SQLEDBSTOPOPT */
/* ... */
SQL_STRUCTURE sqledbstopopt
{
  sqluint32              isprofile;
  char                   profile[SQL_PROFILE_SZ+1];
  sqluint32              isnodenum;
  SQL_PDB_NODE_TYPE      nodenum;
  sqluint32              option;
  sqluint32              callerac;
};
/* ... */
```

COBOL 構造

```
* File: sqlenv.cbl
01 SQLEDBSTOPOPT.
    05 SQL-ISPROFILE          PIC 9(9) COMP-5.
    05 SQL-PROFILE            PIC X(235).
    05 FILLER                 PIC X.
    05 SQL-ISNODENUM          PIC 9(9) COMP-5.
    05 SQL-NODENUM            PIC S9(4) COMP-5.
    05 FILLER                 PIC X(2).
    05 SQL-OPTION             PIC 9(9) COMP-5.
    05 SQL-CALLERAC           PIC 9(9) COMP-5.
*
```

SQLEDINFO

この構造は、208ページの『sqledgne - データベース・ディレクトリーの次項目の入手』への呼び出し後に情報を戻すのに使用されます。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・ディレクトリーの両方によって共用されます。

表 50. SQLEDINFO 構造のフィールド

フィールド名	データ・タイプ	説明
ALIAS	CHAR(8)	代替データベース名。
DBNAME	CHAR(8)	データベースの名前。
DRIVE	CHAR(215)	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・ディレクトリーが走査用にオープンしている場合のみ戻されます。 **注:** このフィールドは、OS/2 では CHAR(2) であり、Windows NT では CHAR(12) です。
INTNAME	CHAR(8)	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・ディレクトリーが走査用にオープンしている場合のみ戻されます。
NODENAME	CHAR(8)	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	CHAR(20)	データベース・マネージャーのリリース情報。
COMMENT	CHAR(30)	データベースに関するコメント。
COM_CODEPAGE	SMALLINT	コメントのコード・ページ。使用されません。
TYPE	CHAR(1)	項目タイプ。値については、以下を参照してください。
AUTHENTICATION	SMALLINT	認証タイプ。値については、以下を参照してください。
GLBDBNAME	CHAR(255)	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宛先データベースのグローバル名。
DCEPRINCIPAL	CHAR(1024)	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	SHORT	カタログ・ノード番号。
NODENUM	SHORT	ノード番号。
注: システムとローカル・データベース・ディレクトリーは、両方とも同じ構造を使用しますが、有効なフィールドはそれぞれ特定のものだけです。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。		

TYPE に有効な値 (sqlenv で定義) は、以下のとおりです。

SQL_INDIRECT
現行のインスタンス (**DB2INSTANCE** 環境変数の値によって定義された) 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 に有効な値 (sqlenv で定義) は、以下のとおりです。

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

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

SQL_AUTHENTICATION_DCS
DB2 コネクト用に使用します。

SQL_AUTHENTICATION_DCE
認証は、DCE 機密保護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・ディレクトリーで保持しなくてもよいことになりました。下位レベル (DB2 V2 またはそれ以下) のサーバー以外に接続するときは、この値を指定してください。

言語構文

C 構造

```
/* File: sqlenv.h */
/* Structure: SQLEDINFO */
/* ... */
SQL_STRUCTURE sqledinfo
{
  _SQLOLDCHAR    alias[SQL_ALIAS_SZ];
  _SQLOLDCHAR    dbname[SQL_DBNAME_SZ];
  _SQLOLDCHAR    drive[SQL_DRIVE_SZ];
  _SQLOLDCHAR    intname[SQL_INAME_SZ];
  _SQLOLDCHAR    nodename[SQL_NNAME_SZ];
  _SQLOLDCHAR    dbtype[SQL_DBTYP_SZ];
  _SQLOLDCHAR    comment[SQL_CMT_SZ];
  short          com_codepage;
  _SQLOLDCHAR    type;
  unsigned short authentication;
  char           glbdbname[SQL_DIR_NAME_SZ];
  _SQLOLDCHAR    dceprincipal[SQL_DCEPRIN_SZ];
  short          cat_nodenum;
  short          nodenum;
};
/* ... */
```

COBOL 構造

```
* File: sqlenv.cbl
01 SQLEDINFO.
    05 SQL-ALIAS              PIC X(8).
    05 SQL-DBNAME             PIC X(8).
    05 SQL-DRIVE              PIC X(215).
    05 SQL-INTNAME            PIC X(8).
    05 SQL-NODENAME           PIC X(8).
    05 SQL-DBTYPE             PIC X(20).
    05 SQL-COMMENT            PIC X(30).
    05 FILLER                 PIC X(1).
    05 SQL-COM-CODEPAGE       PIC S9(4) COMP-5.
    05 SQL-TYPE               PIC X.
    05 FILLER                 PIC X(1).
    05 SQL-AUTHENTICATION     PIC 9(4) COMP-5.
    05 SQL-GLBDBNAME          PIC X(255).
    05 SQL-DCEPRINCIPAL       PIC X(1024).
    05 FILLER                 PIC X(1).
    05 SQL-CAT-NODENUM        PIC S9(4) COMP-5.
    05 SQL-NODENUM            PIC S9(4) COMP-5.
*
```

SQLENINFO

この構造は、267ページの『sqlengne - ノード・ディレクトリー次項目の入手』への呼び出しの後に情報を戻します。

表 51. 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。ローカル論理装置。
PARTNER_LU	CHAR(8)	APPN プロトコルに使用されます。パートナー論理装置。
MODE	CHAR(8)	APPN プロトコルに使用されます。伝送サービス・モード。
COMMENT	CHAR(30)	ノードに関するコメント。
COM_CODEPAGE	SMALLINT	コメントのコード・ページ。このフィールドは、データベース・マネージャーによって使用されなくなりました。
ADAPTER	SMALLINT	NetBIOS プロトコルに使用されます。ローカル・ネットワーク・アダプター。
NETWORKID	CHAR(8)	APPN プロトコルに使用されます。ネットワーク ID。
PROTOCOL	CHAR(1)	通信プロトコル。
SYM_DEST_NAME	CHAR(8)	APPC プロトコルに使用されます。記号宛先名。
SECURITY_TYPE	SMALLINT	APPC プロトコルに使用されます。機密保護タイプ。値については、以下を参照してください。
HOSTNAME	CHAR(255)	TCP/IP プロトコル用に使用されます。 DB2 サーバー・インスタンスが存在する TCP/IP ホストの名前。
SERVICE_NAME	CHAR(14)	TCP/IP プロトコルに使用されます。 DB2 サーバー・インスタンスの TCP/IP サービス名または関連するポート番号。
FILESERVER	CHAR(48)	IPX/SPX プロトコル用に使用されます。 DB2 サーバー・インスタンスが登録されている NetWare ファイル・サーバーの名前。

表 51. 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
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・サーバー上でオブジェクト *objectname* として表現されます。サーバーの IPX/SPX インターネットワーク・アドレスは、このオブジェクトから保管および検索されます。
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

言語構文

C 構造

```
/* File: sqlenv.h */
/* Structure: SQLENINFO */
/* ... */
SQL_STRUCTURE sqleninfo
{
  _SQLOLDCHAR    nodename[SQL_NNAME_SZ];
  _SQLOLDCHAR    local_lu[SQL_LOCLU_SZ];
  _SQLOLDCHAR    partner_lu[SQL_RMTLU_SZ];
  _SQLOLDCHAR    mode[SQL_MODE_SZ];
  _SQLOLDCHAR    comment[SQL_CMT_SZ];
  unsigned short com_codepage;
  unsigned short adapter;
  _SQLOLDCHAR    networkid[SQL_NETID_SZ];
  _SQLOLDCHAR    protocol;
  _SQLOLDCHAR    sym_dest_name[SQL_SYM_DEST_NAME_SZ];
  unsigned short security_type;
  _SQLOLDCHAR    hostname[SQL_HOSTNAME_SZ];
  _SQLOLDCHAR    service_name[SQL_SERVICE_NAME_SZ];
  char           fileserver[SQL_FILESERVER_SZ];
  char           objectname[SQL_OBJECTNAME_SZ];
  char           instance_name[SQL_INSTNAME_SZ];
  char           computername[SQL_COMPUTERNAME_SZ];
  char           system_name[SQL_SYSTEM_NAME_SZ];
  char           remote_instname[SQL_REMOTE_INSTNAME_SZ];
  _SQLOLDCHAR    catalog_node_type;
  unsigned short os_type;
};
/* ... */
```

COBOL 構造

```
* File: sqlenv.cbl
01 SQLENINFO.
    05 SQL-NODE-NAME          PIC X(8).
    05 SQL-LOCAL-LU           PIC X(8).
    05 SQL-PARTNER-LU         PIC X(8).
    05 SQL-MODE               PIC X(8).
    05 SQL-COMMENT            PIC X(30).
    05 SQL-COM-CODEPAGE       PIC 9(4) COMP-5.
    05 SQL-ADAPTER            PIC 9(4) COMP-5.
    05 SQL-NETWORKID          PIC X(8).
    05 SQL-PROTOCOL           PIC X.
    05 SQL-SYM-DEST-NAME      PIC X(8).
    05 FILLER                 PIC X(1).
    05 SQL-SECURITY-TYPE      PIC 9(4) COMP-5.
    05 SQL-HOSTNAME           PIC X(255).
    05 SQL-SERVICE-NAME       PIC X(14).
    05 SQL-FILESERVER         PIC X(48).
    05 SQL-OBJECTNAME         PIC X(48).
    05 SQL-INSTANCE-NAME      PIC X(8).
    05 SQL-COMPUTERNAME       PIC X(15).
    05 SQL-SYSTEM-NAME        PIC X(21).
    05 SQL-REMOTE-INSTNAME    PIC X(8).
    05 SQL-CATALOG-NODE-TYPE  PIC X.
    05 SQL-OS-TYPE            PIC 9(4) COMP-5.
*
```

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。これは、データベース構成およびデータベース・マネージャー構成 API と一緒に使用されます。

表 52. SQLFUPD 構造のフィールド

フィールド名	データ・タイプ	説明
TOKEN	UINT16	戻すかまたは更新するための構成値を指定します。
PTRVALUE	ポインター	*TOKEN* によって指定されたデータを保持する、アプリケーション割り振りバッファーへのポインター。

トークン 要素についての有効なデータ・タイプは、以下のとおりです。

Uint16 符号なし、2 バイトの整数

Sint16 符号付き、2 バイトの整数

Uint32 符号なし、4 バイトの整数

Sint32 符号付き、4 バイトの整数

float 4 バイトの浮動小数点数

char(*n*) 長さ *n* のストリング (ヌル終了文字は含みません)。

データベース構成パラメーターの完全な説明については、 *管理の手引き* を参照してください。

以下に、トークン 要素 SQLFUPD についての有効な項目をリストします。

表 53. 更新可能なデータベース構成パラメーター

パラメーター名	トークン	トークン値	データ・タイプ
app_ctl_heap_sz	SQLF_DBTN_APP_CTL_HEAP_SZ	500	Uint16
applheapsz	SQLF_DBTN_APPLHEAPSZ	51	Uint16
audit_buf_sz	SQLF_KTN_AUDIT_BUF_SZ	312	Sint32
autorestart	SQLF_DBTN_AUTO_RESTART	25	Uint16
avg_appls	SQLF_DBTN_AVG_APPLS	47	Uint16
buffpage	SQLF_DBTN_BUFF_PAGE	90	Uint32
catalogcache_sz	SQLF_DBTN_CATALOGCACHE_SZ	56	Sint32
chngpgs_thresh	SQLF_DBTN_CHNGPGS_THRESH	38	Uint16
copyprotect	SQLF_DBTN_COPY_PROTECT	22	Uint16
dbheap	SQLF_DBTN_DB_HEAP	701	Uint64

表 53. 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
dft_degree	SQLF_DBTN_DFT_DEGREE	301	Sint32
dft_extent_sz	SQLF_DBTN_DFT_EXTENT_SZ	54	Uint32
dft_loadrec_ses	SQLF_DBTN_DFT_LOADREC_SES	42	Sint16
dft_prefetch_sz	SQLF_DBTN_DFT_PREFETCH_SZ	40	Sint16
dft_queryopt	SQLF_DBTN_DFT_QUERYOPT	57	Sint32
dft_refresh_age	SQLF_DBTN_DFT_REFRESH_AGE	702	char(22)
dft_sqlmathwarn	SQLF_DBTN_DFT_SQLMATHWARN	309	Sint16
dir_obj_name	SQLF_DBTN_DIR_OBJ_NAME	46	char(255)
discover	SQLF_DBTN_DISCOVER	308	Uint16
dl_expint	SQLF_DBTN_DL_EXPINT	350	Sint32
dl_num_copies	SQLF_DBTN_DL_NUM_COPIES	351	Uint16
dl_time_drop	SQLF_DBTN_DL_TIME_DROP	353	Uint16
dl_token	SQLF_DBTN_DL_TOKEN	602	char(10)
dl_upper	SQLF_DBTN_DL_UPPER	603	Sint16
dlchktime	SQLF_DBTN_ DLCHKTIME	9	Uint32
dyn_query_mgmt	SQLF_DBTN_DYN_QUERY_MGMT	604	Uint16
estore_seg_sz	SQLF_DBTN_ESTORE_SEG_SZ	303	Sint32
indexrec[a]	SQLF_DBTN_INDEXREC	30	Uint16
indexsort	SQLF_DBTN_INDEXSORT	35	Uint16
locklist	SQLF_DBTN_ LOCKLIST	1	Uint16
locktimeout	SQLF_DBTN_LOCKTIMEOUT	34	Sint16
logbufsz	SQLF_DBTN_LOGBUFSZ	33	Uint16
logfilsiz	SQLF_DBTN_LOGFIL_SIZ	92	Uint32
logprimary	SQLF_DBTN_LOGPRIMARY	16	Uint16
logretain[b]	SQLF_DBTN_LOG_RETAIN	23	Uint16
logsecond	SQLF_DBTN_LOGSECOND	17	Uint16
maxappls	SQLF_DBTN_MAXAPPLS	6	Uint16
maxfilop	SQLF_DBTN_MAXFILOP	3	Uint16
maxlocks	SQLF_DBTN_MAXLOCKS	15	Uint16
mincommit	SQLF_DBTN_MINCOMMIT	32	Uint16
newlogpath	SQLF_DBTN_NEWLOGPATH	20	char(242)
num_db_backups	SQLF_DBTN_NUM_DB_BACKUPS	352	Uint16
num_estore_segs	SQLF_DBTN_NUM_ESTORE_SEGS	304	Sint32
num_freqvalues	SQLF_DBTN_NUM_FREQVALUES	36	Uint16
num_iocleaners	SQLF_DBTN_NUM_IOCLEANERS	37	Uint16
num_ioservers	SQLF_DBTN_NUM_IOSERVERS	39	Uint16

表 53. 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
num_quantiles	SQLF_DBTN_NUM_QUANTILES	48	Uint16
pckcachesz	SQLF_DBTN_PCKCACHE_SZ	505	Uint32
rec_his_retentn	SQLF_DBTN_REC_HIS_RETENTN	43	Sint16
seqdetect	SQLF_DBTN_SEQDETECT	41	Uint16
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
tsm_mgmtclass	SQLF_DBTN_TSM_MGMTCLASS	307	char(30)
tsm_nodename	SQLF_DBTN_TSM_NODENAME	306	char(64)
tsm_owner	SQLF_DBTN_TSM_OWNER	305	char(64)
tsm_password	SQLF_DBTN_TSM_PASSWORD	501	char(64)
userexit	SQLF_DBTN_USER_EXIT	24	Uint16
util_heap_sz	SQLF_DBTN_UTIL_HEAP_SZ	55	Uint32

[a] 有効な値 (sqlutil.h で定義) は、以下のとおりです。

```
SQLF_INX_REC_SYSTEM (0)
SQLF_INX_REC_REFERENCE (1)
SQLF_INX_REC_RESTART (2)
```

[b] 有効な値 (sqlutil.h で定義) は、以下のとおりです。

```
SQLF_LOGRETAIN_NO (0)
SQLF_LOGRETAIN_RECOVERY (1)
SQLF_LOGRETAIN_CAPTURE (2)
```

表 54. 更新不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
backup_pending	SQLF_DBTN_BACKUP_PENDING	112	Uint16
codepage	SQLF_DBTN_CODEPAGE	101	Uint16
codeset	SQLF_DBTN_CODESET	120	char(9)[a]
collate_info	SQLF_DBTN_COLLATE_INFO	44	char(260)
country	SQLF_DBTN_COUNTRY	100	Uint16
database_consistent	SQLF_DBTN_CONSISTENT	111	Uint16
database_level	SQLF_DBTN_DATABASE_LEVEL	124	Uint16
log_retain_status	SQLF_DBTN_LOG_RETAIN_STATUS	114	Uint16
loghead	SQLF_DBTN_LOGHEAD	105	char(12)
logpath	SQLF_DBTN_LOGPATH	103	char(242)
multipage_alloc	SQLF_DBTN_MULTIPAGE_ALLOC	506	Uint16

表 54. 更新不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
numsegs	SQLF_DBTN_NUMSEGS	122	Uint16
release	SQLF_DBTN_RELEASE	102	Uint16
restore_pending	SQLF_DBTN_RESTORE_PENDING	503	Uint16
rollfwd_pending	SQLF_DBTN_ROLLFWD_PENDING	113	Uint16
territory	SQLF_DBTN_TERRITORY	121	char(5)[b]
user_exit_status	SQLF_DBTN_USER_EXIT_STATUS	115	Uint16
[a] HP-UX および Solaris では、char(17)。 [b] HP-UX および Solaris では、char(33)。			

データベース・マネージャー構成パラメーターの完全な説明については、 *管理の手引き* を参照してください。

以下に、トークン 要素 SQLFUPD についての有効な項目をリストします。

表 55. 更新可能なデータベース・マネージャー構成パラメーター

パラメーター名	トークン	トークン値	データ・タイプ
agent_stack_sz	SQLF_KTN_AGENT_STACK_SZ	61	Uint16
agentpri	SQLF_KTN_AGENTPRI	26	Sint16
aslheapsz	SQLF_KTN_ASLHEAPSZ	15	Uint32
audit_buf_sz	SQLF_KTN_AUDIT_BUF_SZ	312	Sint32
authentication[a]	SQLF_KTN_AUTHENTICATION	78	Uint16
backbufsz	SQLF_KTN_BACKBUFSZ	18	Uint32
catalog_noauth	SQLF_KTN_CATALOG_NOAUTH	314	Uint16
comm_bandwidth	SQLF_KTN_COMM_BANDWIDTH	307	float
conn_elapse	SQLF_KTN_CONN_ELAPSE	508	Uint16
cpuspeed	SQLF_KTN_CPUSPEED	42	float
datalinks	SQLF_KTN_DATALINKS	603	Sint16
dft_account_str	SQLF_KTN_DFT_ACCOUNT_STR	28	char(25)
dft_client_adpt	SQLF_KTN_DFT_CLIENT_ADPT	82	Uint16
dft_client_comm	SQLF_KTN_DFT_CLIENT_COMM	77	char(31)
dft_monswitches	SQLF_KTN_DFT_MONSWITCHES[b]	29	Uint16
dft_mon_bufpool	SQLF_KTN_DFT_MON_BUFPOOL	33	Uint16
dft_mon_lock	SQLF_KTN_DFT_MON_LOCK	34	Uint16
dft_mon_sort	SQLF_KTN_DFT_MON_SORT	35	Uint16
dft_mon_stmt	SQLF_KTN_DFT_MON_STMT	31	Uint16
dft_mon_table	SQLF_KTN_DFT_MON_TABLE	32	Uint16

表 55. 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_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
dir_obj_name	SQLF_KTN_DIR_OBJ_NAME	75	char(255)
dir_path_name	SQLF_KTN_DIR_PATH_NAME	74	char(255)
dir_type[c]	SQLF_KTN_DIR_TYPE	73	Uint16
discover[d]	SQLF_KTN_DISCOVER	304	Uint16
discover_comm	SQLF_KTN_DISCOVER_COMM	305	char(35)
discover_inst	SQLF_KTN_DISCOVER_INST	308	Uint16
dos_rqrioblk	SQLF_KTN_DOS_RQRIOBLK	72	Uint16
drda_heap_sz	SQLF_KTN_DRDA_HEAP_SZ	41	Uint16
fcm_num_anchors	SQLF_KTN_FCM_NUM_ANCHORS	506	Sint32
fcm_num_buffers	SQLF_KTN_FCM_NUM_BUFFERS	503	Uint32
fcm_num_connect	SQLF_KTN_FCM_NUM_CONNECT	505	Sint32
fcm_num_rqb	SQLF_KTN_FCM_NUM_RQB	504	Uint32
federated	SQLF_KTN_FEDERATED	604	Sint16
fileserver	SQLF_KTN_FILESERVER	47	char(48)
indexrec[e]	SQLF_KTN_INDEXREC	20	Uint16
initdari_jvm	SQLF_KTN_INITDARI_JVM	602	Sint16
intra_parallel	SQLF_KTN_INTRA_PARALLEL	306	Sint16
ipx_socket	SQLF_KTN_IPX_SOCKET	71	char(4)
java_heap_sz	SQLF_KTN_JAVA_HEAP_SZ	310	Sint32
jdk11_path	SQLF_KTN_JDK11_PATH	311	char(255)
keepdari	SQLF_KTN_KEEPDARI	81	Uint16
max_connretries	SQLF_KTN_MAX_CONNRETRIES	509	Uint16
max_coordagents	SQLF_KTN_MAX_COORDAGENTS	501	Sint32
max_logicagents	SQLF_KTN_MAX_LOGICAGENTS	70	Sint32
max_querydegree	SQLF_KTN_MAX_QUERYDEGREE	303	Sint32
max_time_diff	SQLF_KTN_MAX_TIME_DIFF	510	Uint16
maxagents	SQLF_KTN_MAXAGENTS	12	Uint32
maxcagents	SQLF_KTN_MAXCAGENTS	13	Sint32
maxdari	SQLF_KTN_MAXDARI	80	Sint32
maxtotfilop	SQLF_KTN_MAXTOTFILOP	45	Uint16
min_priv_mem	SQLF_KTN_MIN_PRIV_MEM	43	Uint32

表 55. 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
mon_heap_sz	SQLF_KTN_MON_HEAP_SZ	79	Uint16
nname	SQLF_KTN_NNAME	7	char(8)
notifylevel	SQLF_KTN_NOTIFYLEVEL	605	Sint16
num_initagents	SQLF_KTN_NUM_INITAGENTS	500	Uint32
num_initdaris	SQLF_KTN_NUM_INITDARIS	601	Sint32
num_poolagents	SQLF_KTN_NUM_POOLAGENTS	502	Sint32
numdb	SQLF_KTN_NUMDB	6	Uint16
objectname	SQLF_KTN_OBJECTNAME	48	char(48)
priv_mem_thresh	SQLF_KTN_PRIV_MEM_THRESH	44	Sint32
query_heap_sz	SQLF_KTN_QUERY_HEAP_SZ	49	Sint32
restbufsz	SQLF_KTN_RESTBUFSZ	19	Uint32
resync_interval	SQLF_KTN_RESYNC_INTERVAL	68	Uint16
route_obj_name	SQLF_KTN_ROUTE_OBJ_NAME	76	char(255)
rqrioblk	SQLF_KTN_RQRIOBLK	1	Uint16
sheapthres	SQLF_KTN_SHEAPTHRES	21	Uint32
spm_log_file_sz	SQLF_KTN_SPM_LOG_FILE_SZ	90	Sint32
spm_max_resync	SQLF_KTN_SPM_MAX_RESYNC	91	Sint32
spm_name	SQLF_KTN_SPM_NAME	92	char(8)
spm_path_name	SQLF_KTN_SPM_PATH_NAME	313	char(226)
ss_logon	SQLF_KTN_SS_LOGON	309	Uint16
start_stop_time	SQLF_KTN_START_STOP_TIME	511	Uint16
svcename	SQLF_KTN_SVCENAME	24	char(14)
sysadm_group	SQLF_KTN_SYSADM_GROUP	39	char(16)
sysctrl_group	SQLF_KTN_SYSCTRL_GROUP	63	char(16)
sysmaint_group	SQLF_KTN_SYSMAINT_GROUP	62	char(16)
tm_database	SQLF_KTN_TM_DATABASE	67	char(8)
tp_mon_name	SQLF_KTN_TP_MON_NAME	66	char(19)
tpname	SQLF_KTN_TPNAME	25	char(64)
trust_allclnts[f]	SQLF_KTN_TRUST_ALLCLNTS	301	Uint16
trust_clntauth	SQLF_KTN_TRUST_CLNTAUTH	302	Uint16
udf_mem_sz	SQLF_KTN_UDF_MEM_SZ	69	Uint16

表 55. 更新可能なデータベース・マネージャー構成パラメーター (続き)

パラメーター名	トークン	トークン値	データ・タイプ

[a] 有効な値 (sqlenv.h で定義) は、以下のとおりです。

```
SQL_AUTHENTICATION_SERVER (0)
SQL_AUTHENTICATION_CLIENT (1)
SQL_AUTHENTICATION_DCS (2)
SQL_AUTHENTICATION_DCE (3)
SQL_AUTHENTICATION_SVR_ENCRYPT (4)
SQL_AUTHENTICATION_DCS_ENCRYPT (5)
SQL_AUTHENTICATION_DCE_SVR_ENC (6)
SQL_AUTHENTICATION_KERBEROS (7)
SQL_AUTHENTICATION_KRB_SVR_ENC (8)
SQL_AUTHENTICATION_NOT_SPEC (255)
```

[b] SQLF_KTN_DFT_MONSWITCHES は、 Uint16 のパラメーターであり、そのビットは省略時のモニター・スイッチ設定を示します。これによって、いくつかのパラメーターを一度に指定できます。この複合パラメーターを構成する個々のビットは、次のとおりです。

```
Bit 1 (xxxx xxx1): dft_mon_uow
Bit 2 (xxxx xx1x): dft_mon_stmt
Bit 3 (xxxx x1xx): dft_mon_table
Bit 4 (xxxx 1xxx): dft_mon_buffpool
Bit 5 (xxx1 xxxx): dft_mon_lock
Bit 6 (xx1x xxxx): dft_mon_sort
```

[c] 有効な値 (sqlutil.h で定義) は、以下のとおりです。

```
SQLF_DIRTYPE_NONE (0)
SQLF_DIRTYPE_DCE (1)
```

[d] 有効な値 (sqlutil.h で定義) は、以下のとおりです。

```
SQLF_DSCVR_KNOWN (1)
SQLF_DSCVR_SEARCH (2)
```

[e] 有効な値 (sqlutil.h で定義) は、以下のとおりです。

```
SQLF_INX_REC_SYSTEM (0)
SQLF_INX_REC_REFERENCE (1)
```

[f] 有効な値 (sqlutil.h で定義) は、以下のとおりです。

```
SQLF_TRUST_ALLCLNTS_NO (0)
SQLF_TRUST_ALLCLNTS_YES (1)
SQLF_TRUST_ALLCLNTS_DRDAONLY (2)
```

表 56. 更新不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
nodetype[a]	SQLF_KTN_NODETYPE	100	Uint16
release	SQLF_KTN_RELEASE	101	Uint16

表 56. 更新不可能なデータベース・マネージャー構成パラメーター (続き)

パラメーター名	トークン	トークン値	データ・タイプ
[a] 有効な値 (sqlutil.h で定義) は、以下のとおりです。 SQLF_NT_STANDALONE (0) SQLF_NT_SERVER (1) SQLF_NT_REQUESTOR (2) SQLF_NT_STAND_REQ (3) SQLF_NT_MPP (4) SQLF_NT_SATELLITE (5)			

言語構文

C 構造

```
/* File: sqlutil.h */
/* Structure: SQLFUPD */
/* ... */
SQL_STRUCTURE sqlfupd
{
  unsigned short  token;
  char            *ptrvalue;
};
/* ... */
```

COBOL 構造

```
* File: sqlutil.cbl
01 SQL-FUPD.
    05 SQL-TOKEN              PIC 9(4) COMP-5.
    05 FILLER                 PIC X(2).
    05 SQL-VALUE-PTR          USAGE IS POINTER.
*
```

SQLM-COLLECTED

この構造は、データベース・システム・モニター API への呼び出しの後に情報を戻すのに使用されます。これは、SQLM_DBMON_VERSION5_2 レベル以下で実行されるスナップショット要求のためだけに指定されます。

表 57. SQLM-COLLECTED 構造のフィールド

フィールド名	データ・タイプ	説明
SIZE	sqluint32	構造のサイズ。
DB2	sqluint32	廃止。
DATABASES	sqluint32	廃止。
TABLE_DATABASES	sqluint32	廃止。
LOCK_DATABASES	sqluint32	廃止。
APPLICATIONS	sqluint32	廃止。
APPLINFOS	sqluint32	廃止。
DCS_APPLINFOS	sqluint32	廃止。
SERVER_DB2_TYPE	sqluint32	データベース・マネージャー・サーバー・タイプ (sqlutil.h で定義)。
TIME_STAMP	TIMESTAMP	スナップショットがとられた時刻。
GROUP_STATES	OBJECT SQLM_ RECORDING_ GROUP	モニター・スイッチの現在の状態。
SERVER_PRDID	CHAR(20)	サーバー上のデータベース・マネージャーの製品名およびバージョン番号。
SERVER_NNAME	CHAR(20)	サーバーの構成ノード名。
SERVER_ INSTANCE_NAME	CHAR(20)	データベース・マネージャーのインスタンス名。
RESERVED	CHAR(22)	将来の使用のために予約されています。
NODE_NUMBER	UNSIGNED SHORT	データを送信しているノードの番号。
TIME_ZONE_DISP	sqlint32	GMT とローカル時刻の差 (秒単位)。
NUM_TOP_LEVEL_ STRUCTS	sqluint32	スナップショット出力バッファーに戻された高水準構造の合計数。高水準構造は、いくつかの低レベル・データ構造から構成される可能性があります。このカウンターは、現在は廃止された、各高水準構造についての個々のカウンター (*table_databases* など) に代わるものです。
TABLESPACE_ DATABASES	sqluint32	廃止。
SERVER_VERSION	sqluint32	データを戻しているサーバーのバージョン。

データベース・モニターのプログラミングについては、 システム・モニター *手引きおよび解説書* を参照してください。

言語構文

C 構造

```
/* File: sqlmon.h */
/* Structure: SQLM-COLLECTED */
/* ... */
typedef struct sqlm_collected
{
  sqluint32      size;
  sqluint32      db2;
  sqluint32      databases;
  sqluint32      table_databases;
  sqluint32      lock_databases;
  sqluint32      applications;
  sqluint32      applinfos;
  sqluint32      dcs_applinfos;
  sqluint32      server_db2_type;
  sqlm_timestamp time_stamp;
  sqlm_recording_group group_states[SQLM_NUM_GROUPS];
  _SQLOLDCHAR    server_prdid[SQLM_IDENT_SZ];
  _SQLOLDCHAR    server_nname[SQLM_IDENT_SZ];
  _SQLOLDCHAR    server_instance_name[SQLM_IDENT_SZ];
  _SQLOLDCHAR    reserved[22];
  unsigned short node_number;
  long           time_zone_disp;
  sqluint32      num_top_level_structs;
  sqluint32      tablespace_databases;
  sqluint32      server_version;
}sqlm_collected;
/* ... */
```

COBOL 構造

```
* File: sqlmonct.cbl
01 SQLM-COLLECTED.
    05 SQLM-SIZE               PIC 9(9) COMP-5.
    05 DB2                     PIC 9(9) COMP-5.
    05 DATABASES               PIC 9(9) COMP-5.
    05 TABLE-DATABASES         PIC 9(9) COMP-5.
    05 LOCK-DATABASES          PIC 9(9) COMP-5.
    05 APPLICATIONS            PIC 9(9) COMP-5.
    05 APPLINFOS               PIC 9(9) COMP-5.
    05 DCS-APPLINFOS           PIC 9(9) COMP-5.
    05 SERVER-DB2-TYPE         PIC 9(9) COMP-5.
    05 TIME-STAMP.
        10 SECONDS              PIC 9(9) COMP-5.
        10 MICROSEC             PIC 9(9) COMP-5.
    05 GROUP-STATES OCCURS 6.
        10 INPUT-STATE          PIC 9(9) COMP-5.
        10 OUTPUT-STATE         PIC 9(9) COMP-5.
        10 START-TIME.
    05 SERVER-PRDID            PIC X(20).
    05 SERVER-NNAME            PIC X(20).
    05 SERVER-INSTANCE-NAME    PIC X(20).
    05 RESERVED                PIC X(32).
    05 TABLESPACE-DATABASES    PIC 9(9) COMP-5.
    05 SERVER-VERSION          PIC 9(9) COMP-5.
*
```

SQLM-RECORDING-GROUP

この構造は、データベース・システム・モニター API への呼び出しの後に情報を戻すのに使用されます。

表 58. SQLM-RECORDING-GROUP 構造のフィールド

フィールド名	データ・タイプ	説明
INPUT_STATE	INTEGER	特定のモニター・グループに必要な状態。
OUTPUT_STATE	INTEGER	特定のモニター・スイッチの状態に関する戻り情報。
START_TIME	構造体	モニター・グループ・スイッチがオンになったときのタイム・スタンプ。

表 59. SQLM-TIMESTAMP 構造のフィールド

フィールド名	データ・タイプ	説明
SECONDS	INTEGER	1970 年 1 月 1 日 (GMT) から経過した秒数として表現される日時。
MICROSEC	INTEGER	現在の秒で、経過したマイクロ秒数。

input_state と *output_state* の両方について、特定のモニター・スイッチは、80 ページの『db2MonitorSwitches - モニター・スイッチの入手 / 更新』に渡される配列中の索引によって識別されます。索引をスイッチにマップする定数は、SQLM_*XXXX*_SW と呼ばれます。ここで、*XXXX* はモニター・グループの名前です。これらの定数は、sqlmon.h で定義されます。

データベース・モニターのプログラミングについては、 システム・モニター *手引きおよび解説書* を参照してください。

言語構文

C 構造

```
/* File: sqlmon.h */
/* Structure: SQLM-RECORDING-GROUP */
/* ... */
typedef struct sqlm_recording_group
{
  sqluint32      input_state;
  sqluint32      output_state;
  sqlm_timestamp start_time;
}sqlm_recording_group;
/* ... */
```

```
/* File: sqlmon.h */
/* Structure: SQLM-TIMESTAMP */
/* ... */
typedef struct sqlm_timestamp
{
  sqluint32  seconds;
  sqluint32  microsec;
}sqlm_timestamp;
/* ... */
```

COBOL 構造

```
* File: sqlmonct.cbl
01 SQLM-RECORDING-GROUP OCCURS 6 TIMES.
    05 INPUT-STATE             PIC 9(9) COMP-5.
    05 OUTPUT-STATE            PIC 9(9) COMP-5.
    05 START-TIME.
        10 SECONDS              PIC 9(9) COMP-5.
        10 MICROSEC             PIC 9(9) COMP-5.
*
```

```
* File: sqlmonct.cbl
01 SQLM-TIMESTAMP.
    05 SECONDS                 PIC 9(9) COMP-5.
    05 MICROSEC                PIC 9(9) COMP-5.
*
```

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のに使用されます。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 60. 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 61. 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。
OBJ_TYPE	INTEGER	モニターされるオブジェクトのタイプ。
OBJECT	CHAR(36)	モニターされるオブジェクトの名前。*OBJ_TYPE* に名前 (*appl_id* など) またはデータベース別名が必要な場合にのみ指定します。

OBJ_TYPE に有効な値 (sqlmon で定義) は、以下のとおりです。

SQLMA_DB2
DB2 関連情報

SQLMA_DBASE
データベース関連情報

SQLMA_APPL
アプリケーション ID により編成されるアプリケーション情報

SQLMA_AGENT_ID
エージェント ID により編成されるアプリケーション情報

SQLMA_DBASE_TABLES
データベースの表情報

SQLMA_DBASE_APPLS
データベースのアプリケーション情報

SQLMA_DBASE_APPLINFO
データベースの要約アプリケーション情報

SQLMA_DBASE_LOCKS
データベースのロック情報

SQLMA_DBASE_ALL
データベース・マネージャー内の全活動データベースに関するデータベース情報

SQLMA_APPL_ALL
データベース・マネージャー内の全活動アプリケーションに関するアプリケーション情報

SQLMA_APPLINFO_ALL
データベース・マネージャー内の全活動アプリケーションに関する要約アプリケーション情報

SQLMA_DCS_APPLINFO_ALL
データベース・マネージャー内の全活動アプリケーションに関するデータベース接続サービス・アプリケーション情報の要約

SQLMA_DYNAMIC_SQL
動的 SQL のスナップショットの入手

SQLMA_DCS_DBASE
データベース接続サービスのデータベース・レベル情報

SQLMA_DCS_DBASE_ALL
すべてのアクティブ・データベースに関する、データベース接続サービスのデータベース情報

SQLMA_DCS_APPL_ALL
すべての接続に関する、データベース接続サービスのアプリケーション情報

SQLMA_DCS_APPL
アプリケーション ID によって識別される、データベース接続サービスのアプリケーション情報

SQLMA_DCS_APPL_HANDLE
アプリケーション・ハンドルによって識別される、データベース接続サービスのアプリケーション情報

SQLMA_DCS_DBASE_APPLS
データベースへのアクティブな接続すべてに関する、データベース接続サービスのアプリケーション情報

SQLMA_DBASE_TABLESPACES
データベースの表スペース情報

SQLMA_DBASE_REMOTE
DataJoiner データベースの情報。

SQLMA_DBASE_REMOTE_ALL
すべての DataJoiner データベースの情報。

SQLMA_DBASE_APPLS_REMOTE
特定の DataJoiner データベースのアプリケーション情報。

SQLMA_APPLS_REMOTE_ALL
すべての DataJoiner データベースのアプリケーション情報。

データベース・モニターのプログラミングについては、 システム・モニター *手引きおよび解説書* を参照してください。

言語構文

C 構造

```
/* File: sqlmon.h */
/* Structure: SQLMA */
/* ... */
typedef struct sqlma
{
  sqluint32 obj_num;
  sqlm_obj_struct obj_var[1];
}sqlma;
/* ... */


/* File: sqlmon.h */
/* Structure: SQLM-OBJ-STRUCT */
/* ... */
typedef struct sqlm_obj_struct
{
  sqluint32     agent_id;
  sqluint32     obj_type;
  _SQLOLDCHAR   object[SQLM_OBJECT_SZ];
}sqlm_obj_struct;
/* ... */
```

COBOL 構造

```
* File: sqlmonct.cbl
01 SQLMA.
    05 OBJ-NUM                 PIC 9(9) COMP-5.
    05 OBJ-VAR OCCURS 0 TO 100 TIMES DEPENDING ON OBJ-NUM.
        10 AGENT-ID             PIC 9(9) COMP-5.
        10 OBJ-TYPE             PIC 9(9) COMP-5.
        10 OBJECT               PIC X(36).
*
```

SQLOPT

この構造は、97ページの『sqlabndx - バインド』にバインド・オプションを渡し、 107ページの『sqlaprep - プログラムのプリコンパイル』にプリコンパイル・オプションを渡し、 114ページの『sqlarbnd - 再バインド』に再バインド・オプションを渡すのに使用されます。

表 62. 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。この配列中の要素の数は、ヘッダー の *allocated* フィールドの値によって判別されます。

表 63. 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* 配列にある要素の数。
USED	INTEGER	*sqlopt* 構造の *option* 配列内の実際に使用される要素の数。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 64. SQLOPTIONS 構造のフィールド

フィールド名	データ・タイプ	説明
TYPE	INTEGER	バインド / プリコンパイル / 再バインド・オプション・タイプ。
VAL	INTEGER	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 / 再バインド・オプションについて繰り返されます。		

TYPE および *VAL* に有効な値については、 97ページの『sqlabndx - バインド』、107ページの『sqlaprep - プログラムのプリコンパイル』、および 114ページの『sqlarbnd - 再バインド』を参照してください。

言語構文

C 構造

```
/* File: sql.h */
/* Structure: SQLOPT */
/* ... */
SQL_STRUCTURE sqlopt
{
  SQL_STRUCTURE sqloptheader header;
  SQL_STRUCTURE sqloptions   option[1];
};
/* ... */


/* File: sql.h */
/* Structure: SQLOPTHEADER */
/* ... */
SQL_STRUCTURE sqloptheader
{
  sqluint32 allocated;
  sqluint32 used;
};
/* ... */


/* File: sql.h */
/* Structure: SQLOPTIONS */
/* ... */
SQL_STRUCTURE sqloptions
{
  sqluint32 type;
  sqluint32 val;
};
/* ... */
```

COBOL 構造

```
* File: sql.cbl
01 SQLOPT.
    05 SQLOPTHEADER.
        10 ALLOCATED   PIC 9(9) COMP-5.
        10 USED        PIC 9(9) COMP-5.
    05 SQLOPTIONS OCCURS 1 TO 50 DEPENDING ON ALLOCATED.
        10 SQLOPT-TYPE        PIC 9(9) COMP-5.
        10 SQLOPT-VAL         PIC 9(9) COMP-5.
        10 SQLOPT-VAL-PTR     REDEFINES SQLOPT-VAL
*
```

SQLU-LSN

462ページの『sqlurlog - ログの非同期読み取り』 によって使用されるこの共用体には、ログ順序番号の定義が含まれます。ログ順序番号 (LSN) は、データベース・ログ内の相対バイト・アドレスを示します。すべてのログ・レコードは、この番号によって識別されます。この番号は、ログ・レコードのデータベース・ログの始まりからのバイト・オフセットを示します。

表 65. 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順序番号を指定します。
lsnWord	UNSIGNED SHORT の配列	3 メンバー文字配列のログ順序番号を指定します。

言語構文

C 構造

```
typedef union SQLU_LSN
{
unsigned char  lsnChar  [6] ;
unsigned short lsnWord  [3] ;
} SQLU_LSN;
```

SQLU-MEDIA-LIST

この構造は、以下の目的で使用されます。

- バックアップ・イメージの宛先 媒体のリストを保持する (345ページの『sqlubkp - データベースのバックアップ』を参照)。
- バックアップ・イメージのソース 媒体のリストを保持する (447ページの『sqlurestore - データベースの復元』を参照)。
- 408ページの『sqluload - ロード』に情報を渡す。

表 66. 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)	媒体のタイプを示す文字。
SESSIONS	INTEGER	この構造の *target* フィールドにより示される配列中の要素の数を示します。
TARGET	Union	このフィールドは、3 つの構造タイプのうちの 1 つを指すポインターです。示される構造のタイプは、*media_type* フィールドの値によって判別されます。このフィールドに提供する値の詳細については、適切な API を参照してください。

表 67. 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。

表 68. SQLU-MEDIA-ENTRY 構造のフィールド

フィールド名	データ・タイプ	説明
RESERVE_LEN	INTEGER	*media_entry* フィールドの長さ。 C 以外の言語用。
MEDIA_ENTRY	CHAR(215)	パックアップおよび復元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 69. 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 69. 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 70. 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。

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)。 *pDataFileList* パラメーターにのみ指定できます。

SQLU_TSM_MEDIA
TSM

SQLU_OTHER_MEDIA
ベンダー・ライブラリー

SQLU_USER_EXIT
ユーザー出口 (OS/2 のみ)

SQLU_PIPE_MEDIA
名前付きパイプ (ベンダー API のみ)

SQLU_DISK_MEDIA
ディスク (ベンダー API のみ)

SQLU_DISKETTE_MEDIA
ディスケット (ベンダー API のみ)

SQLU_TAPE_MEDIA
テープ (ベンダー API のみ)

言語構文

C 構造

```
/* File: sqlutil.h */
/* Structure: SQLU-MEDIA-LIST */
/* ... */
typedef SQL_STRUCTURE sqlu_media_list
{
  char            media_type;
  char            filler[3];
  sqlint32            sessions;
  union sqlu_media_list_targets target;
} sqlu_media_list;
/* ... */


/* File: sqlutil.h */
/* Structure: SQLU-MEDIA-LIST-TARGETS */
/* ... */
union sqlu_media_list_targets
{
  struct sqlu_media_entry        *media;
  struct sqlu_vendor             *vendor;
  struct sqlu_location_entry     *location;
};
/* ... */


/* File: sqlutil.h */
/* Structure: SQLU-MEDIA-ENTRY */
/* ... */
typedef SQL_STRUCTURE sqlu_media_entry
{
  sqluint32       reserve_len;
  char            media_entry[SQLU_DB_DIR_LEN+1];
} sqlu_media_entry;
/* ... */


/* File: sqlutil.h */
/* Structure: SQLU-VENDOR */
/* ... */
typedef SQL_STRUCTURE sqlu_vendor
{
  sqluint32       reserve_len1;
  char            shr_lib[SQLU_SHR_LIB_LEN+1];
  sqluint32       reserve_len2;
  char            filename[SQLU_SHR_LIB_LEN+1];
} sqlu_vendor;
/* ... */
```

```
/* File: sqlutil.h */
/* Structure: SQLU-LOCATION-ENTRY */
/* ... */
typedef SQL_STRUCTURE sqlu_location_entry
{
  sqluint32       reserve_len;
  char            location_entry[SQLU_MEDIA_LOCATION_LEN+1];
} sqlu_location_entry;
/* ... */
```

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 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.
*
```

SQLU-RLOG-INFO

この構造には、462ページの『sqlurlog - ログの非同期読み取り』への呼び出しに関する情報が入ります。ログ読み取りの情報構造には、呼び出しとデータベース・ログの状況に関する情報が入ります。

表 71. 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*接続* が発行された後でデータベースに書き込まれた最初のログ・レコードの LSN 値を指定します。 *SQLU_LSN* 構造の詳細については、 591ページの『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・バイトの LSN 値を指定します。
curActiveLSN	SQLU_LSN	現行のアクティブ・ログの LSN 値を指定します。
logRecsWritten	sqluint32	バッファーに書き込まれるログ・レコードの数を指定します。
logBytesWritten	sqluint32	バッファーに書き込まれるバイト数を指定します。

言語構文

C 構造

```
typedef SQL_STRUCTURE SQLU_RLOG_INFO
{
SQLU_LSN        initialLSN ;
SQLU_LSN        firstReadLSN ;
SQLU_LSN        lastReadLSN ;
SQLU_LSN        curActiveLSN ;
sqluint32       logRecsWritten ;
sqluint32       logBytesWritten ;
} SQLU_RLOG_INFO;
```

SQLU-TABLESPACE-BKRST-LIST

この構造は、表スペース名のリストを提供するのに使用されます。

表 72. SQLU-TABLESPACE-BKRST-LIST 構造のフィールド

フィールド名	データ・タイプ	説明
NUM_ENTRY	INTEGER	*tablespace* フィールドによって示されるリストにある項目の数。
TABLESPACE	ポインター	*sqlu_tablespace_entry* 構造を指すポインター。

表 73. SQLU-TABLESPACE-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	*tablespace_entry* フィールドで提供される文字ストリングの長さ。 C 以外の言語用。
TABLESPACE_ENTRY	CHAR(19)	表スペース名。

言語構文

C 構造

```
/* File: sqlutil.h */
/* Structure: SQLU-TABLESPACE-BKRST-LIST */
/* ... */
typedef SQL_STRUCTURE sqlu_tablespace_bkrst_list
{
  long            num_entry;
  struct sqlu_tablespace_entry *tablespace;
} sqlu_tablespace_bkrst_list;
/* ... */


/* File: sqlutil.h */
/* Structure: SQLU-TABLESPACE-ENTRY */
/* ... */
typedef SQL_STRUCTURE sqlu_tablespace_entry
{
  sqluint32       reserve_len;
  char            tablespace_entry[SQLU_MAX_TBS_NAME_LEN+1];
  char            filler[1];
} sqlu_tablespace_entry;
/* ... */
```

COBOL 構造

```
* File: sqlutil.cbl
01 SQLU-TABLESPACE-BKRST-LIST.
    05 SQL-NUM-ENTRY          PIC S9(9) COMP-5.
    05 SQL-TABLESPACE         USAGE IS POINTER.
*


* File: sqlutil.cbl
01 SQLU-TABLESPACE-ENTRY.
    05 SQL-TBSP-LEN           PIC 9(9) COMP-5.
    05 SQL-TABLESPACE-ENTRY   PIC X(18).
    05 FILLER                 PIC X.
    05 SQL-FILLER             PIC X(1).
*
```

SQLUEXPT-OUT

この構造は、 360ページの『sqluexpr - エクスポート』から情報を渡すのに使用されます。

表 74. SQLUEXPT-OUT 構造のフィールド

フィールド名	データ・タイプ	説明
SIZEOFSTRUCT	INTEGER	構造のサイズ。
ROWSEXPORTED	INTEGER	データベースからターゲット・ファイルにエクスポートされたレコードの数。

言語構文

C 構造

```
/* File: sqlutil.h */
/* Structure: SQL-UEXPT-OUT */
/* ... */
SQL_STRUCTURE sqluexpt_out
{
  sqluint32       sizeOfStruct;
  sqluint32       rowsExported;
};
/* ... */
```

COBOL 構造

```
* File: sqlutil.cbl
01 SQL-UEXPT-OUT.
    05 SQL-SIZE-OF-UEXPT-OUT  PIC 9(9) COMP-5 VALUE 8.
    05 SQL-ROWSEXPORTED       PIC 9(9) COMP-5 VALUE 0.
*
```

SQLUIMPT-IN

この構造は、 381ページの『sqluimpr - インポート』に情報を渡すのに使用されます。

表 75. SQLUIMPT-IN 構造のフィールド

フィールド名	データ・タイプ	説明
SIZEOFSTRUCT	INTEGER	この構造のサイズ (バイト単位)。
COMMITCNT	INTEGER	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。 COMMIT は、 *commitcnt* 個のレコードがインポートされるたびに実行されます。
RESTARTCNT	INTEGER	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。一部のレコードがデータベースにコミットされた後で、直前のレコードをインポートできない場合は、このパラメーターを使用してください。指定した値は、次のインポート操作の開始点を表します。

言語構文

C 構造

```
/* File: sqlutil.h */
/* Structure: SQLUIMPT-IN */
/* ... */
SQL_STRUCTURE sqluimpt_in
{
  sqluint32       sizeOfStruct;
  sqluint32       commitcnt;
  sqluint32       restartcnt;
};
/* ... */
```

COBOL 構造

```
* File: sqlutil.cbl
01 SQL-UIMPT-IN.
    05 SQL-SIZE-OF-UIMPT-IN   PIC 9(9) COMP-5 VALUE 12.
    05 SQL-COMMITCNT          PIC 9(9) COMP-5 VALUE 0.
    05 SQL-RESTARTCNT         PIC 9(9) COMP-5 VALUE 0.
*
```

SQLUIMPT-OUT

この構造は、 381ページの『sqluimpr - インポート』から情報を渡すのに使用されます。

表 76. SQLUIMPT-OUT 構造のフィールド

フィールド名	データ・タイプ	説明
SIZEOFSTRUCT	INTEGER	この構造のサイズ (バイト単位)。
ROWSREAD	INTEGER	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。
ROWSSKIPPED	INTEGER	挿入あるいは更新を開始する前にスキップしたレコードの数。
ROWSINSERTED	INTEGER	ターゲット表に挿入された行の数。
ROWSUPDATED	INTEGER	インポートされたレコード (1 次キーの値がすでに表内に存在するレコード) からの情報によって更新された、ターゲット表内の行数。
ROWSREJECTED	INTEGER	インポートできなかったレコードの数。
ROWSCOMMITTED	INTEGER	正常にインポートされ、データベースにコミットされたレコードの数。

言語構文

C 構造

```
/* File: sqlutil.h */
/* Structure: SQLUIMPT-OUT */
/* ... */
SQL_STRUCTURE sqluimpt_out
{
  sqluint32       sizeOfStruct;
  sqluint32       rowsRead;
  sqluint32       rowsSkipped;
  sqluint32       rowsInserted;
  sqluint32       rowsUpdated;
  sqluint32       rowsRejected;
  sqluint32       rowsCommitted;
};
/* ... */
```

COBOL 構造

```
* File: sqlutil.cbl
01 SQL-UIMPT-OUT.
    05 SQL-SIZE-OF-UIMPT-OUT  PIC 9(9) COMP-5 VALUE 28.
    05 SQL-ROWSREAD           PIC 9(9) COMP-5 VALUE 0.
    05 SQL-ROWSSKIPPED        PIC 9(9) COMP-5 VALUE 0.
    05 SQL-ROWSINSERTED       PIC 9(9) COMP-5 VALUE 0.
    05 SQL-ROWSUPDATED        PIC 9(9) COMP-5 VALUE 0.
    05 SQL-ROWSREJECTED       PIC 9(9) COMP-5 VALUE 0.
    05 SQL-ROWSCOMMITTED      PIC 9(9) COMP-5 VALUE 0.
*
```

SQLULOAD-IN

この構造は、 408ページの『sqluload - ロード』の呼び出し時に情報を入力するのに使用されます。

表 77. SQLULOAD-IN 構造のフィールド

フィールド名	データ・タイプ	説明
SIZEOFSTRUCT	sqluint32	この構造のサイズ (バイト単位)。
SAVECNT	sqluint32	一貫性ポイント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・サイズの間隔に切り上げられます。それぞれの一貫性ポイントでメッセージが発行されるため、 75ページの『db2LoadQuery - ロードの照会』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してください。 *savecnt* の値を大きく設定しておかないと、それぞれの一貫性ポイントで実行される活動の同期がとられるときにパフォーマンスに影響が及びます。 省略時値は `0` であり、これは、必要のない限り一貫性ポイントが確立されないことを意味します。
RESTARTCNT	sqluint32G	予約済み。
ROWCNT	sqluint32	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。
WARNINGCNT	sqluint32	*warningcnt* 個の警告後に、ロード操作を停止します。このパラメーターは、警告は予期されないが、正しいファイルと表が使用されていることの確認が望まれる場合に設定してください。 *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、最後の一貫性ポイントから自動的に続行します。あるいは、REPLACE モードで、入力ファイルの先頭からあらためてロード操作を開始できます。

表 77. SQLULOAD-IN 構造のフィールド (続き)

フィールド名	データ・タイプ	説明
DATA_BUFFER_SIZE	sqluint32	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 (並列処理度とは無関係)。指定された値がアルゴリズムの最小値よりも小さい場合には、必要最低限のページが使用され、警告は戻されません。 このメモリーは、ユーティリティー・ヒープから直接に割り振られます (ユーティリティー・ヒープのサイズは、 *util_heap_sz* データベース構成パラメーターを用いて修正できます)。 値を指定しないと、実行時にユーティリティーによって適切な省略時値が計算されます。省略時値は、ローダーのインスタンス生成時にユーティリティー・ヒープで使用可能な空きスペースの割合と、表の一部の特性に基づいて決まります。
SORT_BUFFER_SIZE	sqluint32	予約済み。
HOLD_QUIESCE	UNSIGNED SHORT	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。
RESTARTPHASE	CHAR(1)	予約済み。
STATSOPT	CHAR(1)	収集する統計の細分性。値については、以下を参照してください。
CPU_PARALLELISM	UNSIGNED SHORT	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、区画内並列処理を活用するために設計されています。特に、事前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省略時値を使用します。 **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。
DISK_PARALLELISM	UNSIGNED SHORT	ユーティリティーがデータを表スペース・コンテナーに書き込むために作成するプロセスつまりスレッドの数。値を指定しないと、ユーティリティーは、表スペース・コンテナーの数と表の特性に基づいて適切な省略時値を選択します。

表 77. SQLULOAD-IN 構造のフィールド (続き)

フィールド名	データ・タイプ	説明
NON_RECOVERABLE	UNSIGNED SHORT	ロード・トランザクションが回復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回復できない場合には、 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、そのような表は除去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保留状態になりません。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回復可能としてマークされる場合には、 SQLU_RECOVERABLE_LOAD に設定します。
INDEXING_MODE	UNSIGNED SHORT	ロード・ユーティリティーが索引を再作成するか、それとも索引を増分して拡張するかを指定します。値については、以下を参照してください。

STATSOPT に有効な値 (sqlutil で定義) は、以下のとおりです。

SQLU_STATS_NONE

SQL_STATS_EXTTABLE_ONLY

SQL_STATS_EXTTABLE_INDEX

SQL_STATS_INDEX

SQL_STATS_TABLE

SQL_STATS_EXTINDEX_ONLY

SQL_STATS_EXTINDEX_TABLE

SQL_STATS_ALL

SQL_STATS_BOTH

INDEXING_MODE に有効な値 (sqlutil で定義) は、以下のとおりです。

SQLU_INX_AUTOSELECT

SQLU_INX_REBUILD

SQLU_INX_INCREMENTAL

SQLU_INX_DEFERRED

これらの索引作成モードについては、 *コマンド解説書* の LOAD コマンドの説明を参照してください。

言語構文

C 構造

```
/* File: sqlutil.h */
/* Structure: SQLULOAD-IN */
/* ... */
SQL_STRUCTURE sqluload_in
{
  sqluint32       sizeOfStruct;
  sqluint32       savecnt;
  sqluint32       restartcnt;
  sqluint32       rowcnt;
  sqluint32       warningcnt;
  sqluint32       data_buffer_size;
  sqluint32       sort_buffer_size;  /* No longer used. */
  unsigned short  hold_quiesce;
  char            restartphase;
  char            statsopt;
  unsigned short  cpu_parallelism;
  unsigned short  disk_parallelism;
  unsigned short  non_recoverable;
  unsigned short  indexing_mode;
};
/* ... */
```

COBOL 構造

```
* File: sqlutil.cbl
01 SQLULOAD-IN.
    05 SQL-SIZE-OF-STRUCT     PIC 9(9) COMP-5 VALUE 40.
    05 SQL-SAVECNT            PIC 9(9) COMP-5.
    05 SQL-RESTARTCOUNT       PIC 9(9) COMP-5.
    05 SQL-ROWCNT             PIC 9(9) COMP-5.
    05 SQL-WARNINGCNT         PIC 9(9) COMP-5.
    05 SQL-DATA-BUFFER-SIZE   PIC 9(9) COMP-5.
    05 SQL-SORT-BUFFER-SIZE   PIC 9(9) COMP-5.  * No longer used.
    05 SQL-HOLD-QUIESCE       PIC 9(4) COMP-5.
    05 SQL-RESTARTPHASE       PIC X.
    05 SQL-STATSOPT           PIC X.
    05 SQL-CPU-PARALLELISM    PIC 9(4) COMP-5.
    05 SQL-DISK-PARALLELISM   PIC 9(4) COMP-5.
    05 SQL-NON-RECOVERABLE    PIC 9(4) COMP-5.
    05 SQL-INDEXING-MODE      PIC 9(4) COMP-5.
*
```

SQLULOAD-OUT

この構造は、 408ページの『sqluload - ロード』への呼び出しの後に情報を出力するのに使用されます。

表 78. SQLULOAD-OUT 構造のフィールド

フィールド名	データ・タイプ	説明
SIZEOFSTRUCT	sqluint32	この構造のサイズ (バイト単位)。
ROWSREAD	sqluint32	ロード操作中に読み取られたレコードの数。
ROWSSKIPPED	sqluint32	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。
ROWSLOADED	sqluint32	ターゲット表にロードされた行の数。
ROWSREJECTED	sqluint32	ロードできなかったレコードの数。
ROWSDELETED	sqluint32	削除された重複行の数。
ROWSCOMMITTED	sqluint32	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、スキップまたは拒否されたレコードの数の合計。

言語構文

C 構造

```
/* File: sqlutil.h */
/* Structure: SQLULOAD-OUT */
/* ... */
SQL_STRUCTURE sqluload_out
{
  sqluint32       sizeOfStruct;
  sqluint32       rowsRead;
  sqluint32       rowsSkipped;
  sqluint32       rowsLoaded;
  sqluint32       rowsRejected;
  sqluint32       rowsDeleted;
  sqluint32       rowsCommitted;
};
/* ... */
```

COBOL 構造

```
* File: sqlutil.cbl
01 SQLULOAD-OUT.
    05 SQL-SIZE-OF-STRUCT     PIC 9(9) COMP-5 VALUE 28.
    05 SQL-ROWS-READ          PIC 9(9) COMP-5.
    05 SQL-ROWS-SKIPPED       PIC 9(9) COMP-5.
    05 SQL-ROWS-LOADED        PIC 9(9) COMP-5.
    05 SQL-ROWS-REJECTED      PIC 9(9) COMP-5.
    05 SQL-ROWS-DELETED       PIC 9(9) COMP-5.
    05 SQL-ROWS-COMMITTED     PIC 9(9) COMP-5.
*
```

SQLUPI

この構造は、表の区分化マップや区分化キーなどの区分化情報を保管するのに使用されます。

表 79. 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	区分化マップの長さ (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	区分化マップ。
SQLD	INTEGER	使用された SQLPARTKEY 要素の数。つまり、区分化キー内のキー・パーツの数。
SQLPARTKEY	構造体	区分化キー内の区分化列の記述。区分化列の最大数は SQL_MAX_NUM_PART_KEYS です。

表80 に、SQLUPI データ構造の SQL データ・タイプおよび長さを示します。SQLTYPE 欄は、項目のデータ・タイプを表す数値を指定します。

表 80. SQLUPI 構造の SQL データ・タイプおよび長さ

データ・タイプ	SQLTYPE (ヌル使用不可)	SQLTYPE (ヌル使用可能)	SQLLEN	AIX
日付	384	385	無視	Yes
時刻	388	389	無視	Yes
タイム・スタンプ	392	393	無視	Yes
可変長文字列	448	449	文字列の長さ	Yes
固定長文字列	452	453	文字列の長さ	Yes
長文字列	456	457	無視	No
ヌル文字で終了する文字列	460	461	文字列の長さ	Yes
浮動小数点	480	481	無視	Yes
10 進	484	485	バイト 1 = 精度 バイト 2 = 位取り	Yes
長精度整数	496	497	無視	Yes
短精度整数	500	501	無視	Yes
可変長漢字ストリング	464	465	2 バイト文字の長さ	Yes
固定長漢字ストリング	468	469	2 バイト文字の長さ	Yes

表 80. 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 (ヌル使用不可)	SQLTYPE (ヌル使用可能)	SQLLEN	AIX
長漢字ストリング	472	473	無視	No

言語構文

C 構造

```
/* File: sqlutil.h */
/* Structure: SQLUPI */
/* ... */
SQL_STRUCTURE sqlupi
{
  unsigned short  pmaplen;
  SQL_PDB_NODE_TYPE pmap[SQL_PDB_MAP_SIZE];
  unsigned short  sqld;
  struct sqlpartkey sqlpartkey[SQL_MAX_NUM_PART_KEYS];
};
/* ... */


/* File: sqlutil.h */
/* Structure: SQLPARTKEY */
/* ... */
SQL_STRUCTURE sqlpartkey
{
  unsigned short  sqltype;
  unsigned short  sqllen;
};
/* ... */
```

SQLXA-RECOVER

トランザクション API により使用され、未確定トランザクションについての情報を戻します (619ページの『付録B. トランザクション API』を参照してください)。

表 81. SQLXA-RECOVER 構造のフィールド

フィールド名	データ・タイプ	説明
TIMESTAMP	INTEGER	トランザクションが、準備済み (未確定) 状態に入ったときのタイム・スタンプ。このタイム・スタンプは、ローカル時間帯の協定世界時からのずれを示す秒数です。
XID	CHAR(140)	グローバル・トランザクションを固有に識別する、トランザクション管理プログラムによって割り当てられた XA 識別子。
DBALIAS	CHAR(16)	未確定データベースが検索されるデータベースの別名。
APPLID	CHAR(30)	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識別子。
SEQUENCE_NO	CHAR(4)	データベース・マネージャーが *APPLID* への拡張部分として割り当てた順序番号。
AUTH_ID	CHAR(8)	トランザクションを実行したユーザーの ID。
LOG_FULL	CHAR(1)	このトランザクションによってログが完全な状態になったかどうかを示します。
CONNECTED	CHAR(1)	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。
INDOUBT_STATUS	CHAR(1)	有効な値については、以下を参照してください。
ORIGINATOR	CHAR(1)	区分データベース環境で、トランザクションが XA と DB2 のどちらで開始されたのかを示します。
RESERVED	CHAR(9)	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。

LOGFULL に有効な値 (sqlxa で定義) は、以下のとおりです。

SQLXA_TRUE
真。

SQLXA_FALSE
偽。

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
偽。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管理プログラムからの*再同期化* を待っています。

INDOUBT_STATUS に有効な値 (sqlxa で定義) は、以下のとおりです。

SQLXA_TS_PREP
準備済み

SQLXA_TS_HCOM
ヒューリスティック・コミット済み

SQLXA_TS_HROL
ヒューリスティック・ロールバック済み

SQLXA_TS_MACK
欠落したコミットの通知

SQLXA_TS_END
アイドル

言語構文

C 構造

```
/* File: sqlxa.h */
/* Structure: SQLXA-RECOVER */
/* ... */
typedef struct sqlxa_recover_t
{
  sqluint32      timestamp;
  SQLXA_XID      xid;
  _SQLOLDCHAR    dbalias[SQLXA_DBNAME_SZ];
  _SQLOLDCHAR    applid[SQLXA_APPLID_SZ];
  _SQLOLDCHAR    sequence_no[SQLXA_SEQ_SZ];
  _SQLOLDCHAR    auth_id[SQLXA_USERID_SZ];
  char           log_full;
  char           connected;
  char           indoubt_status;
  char           originator;
  char           reserved[8];
} SQLXA_RECOVER;
/* ... */
```

SQLXA-XID

トランザクション API により使用され、XA トランザクションを識別します (619ページの『付録B. トランザクション API』を参照してください)。

表 82. SQLXA-XID 構造のフィールド

フィールド名	データ・タイプ	説明
FORMATID	INTEGER	XA 形式 ID。
GTRID_LENGTH	INTEGER	グローバル・トランザクション ID の長さ。
BQUAL_LENGTH	INTEGER	ブランチ識別子の長さ。
DATA	CHAR[128]	BQUAL および後書きブランクが続く、合計 128 バイトの GTRID。
注: GTRID および BQUAL の最大サイズは、それぞれ 64 バイトです。		

言語構文

C 構造

```
/* File: sqlxa.h */
/* Structure: SQLXA-XID */
/* ... */
typedef struct sqlxa_xid_t SQLXA_XID;
/* ... */


/* File: sqlxa.h */
/* Structure: SQLXA-XID-T */
/* ... */
struct sqlxa_xid_t
{
  sqlint32 formatID;
  sqlint32 gtrid_length;
  sqlint32 bqual_length;
  char data[SQLXA_XIDDATASIZE];
};
/* ... */
```

付録A. 命名規則

データベースや表、認証 ID などのデータベース・マネージャー・オブジェクトの命名の際に適用される規則について説明します。

- データベース・マネージャー・オブジェクトの名前を表示する文字ストリングには、次のいずれかを含めることができます。 a ～ z、A ～ Z、0 ～ 9、@、 #、および $。
- ストリングの最初の文字は、アルファベット文字、@、 #、または $ にする必要があります。先頭文字に数字または SYS、 DBM、または IBM のストリングを使用することはできません。
- 特に注記がなければ、名前は小文字で入力して構いません。ただし、データベース・マネージャーはそれを大文字とみなして処理します。

  ただし、システム・ネットワーク体系 (SNA) 下の名前を表す文字ストリングは例外です。論理単位名 (partner_lu および local_lu) など、多くの値では大文字小文字が区別されます。こうした名前は、それらの用語に対応する SNA 定義に出ているとおりに入力してください。
- データベース名やデータベース別名は、前に説明した集合内の 1 つから 8 つの文字、数字、キーボード文字を含む固有の文字ストリングです。

  データベースはシステム内にカタログ化されており、ローカル・データベース・ディレクトリーの別名が 1 つのフィールドに、元の名前が別のフィールドに入っています。ほとんどの機能の場合、データベース・マネージャーは、データベース・ディレクトリーの別名フィールドに入力された名前を使用します。 (ただし、CHANGE DATABASE COMMENT および CREATE DATABASE は例外です。この場合は、ディレクトリー・パスを指定しなければなりません。)
- 表または視点の名前や別名は、SQL 識別子です。これは、長さが 1 ～ 128 文字の固有な文字ストリングです。列名の長さは、1 ～ 30 文字です。

  完全修飾表名は、*schema.tablename* で構成されます。スキーマは固有のユーザー ID で、その下に表が作成されます。宣言された一時表のスキーマ名は SESSION でなければなりません。
- 認証 ID の長さは、Windows 32 ビット・オペレーティング・システムで 30 文字以内に、他のすべてのオペレーティング・システムでは 8 文字以内に制限されています。
- グループ ID の長さは、8 文字以内に制限されています。

© Copyright IBM Corp. 1993, 2000

- ノード・ディレクトリー内でカタログ化するリモート・ノードのローカル別名の長さは、 8 文字より長くはできません。

命名規則の詳細については、*管理の手引き* を参照してください。すべての DB2 識別子の長さの制限については、*SQL 解説書* を参照してください。

付録B. トランザクション 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。このトピックとヒューリスティック操作の詳細については、 *管理の手引き* を参照してください。

ヒューリスティック API

リソース所有者 (たとえばデータベース管理者) がトランザクション管理プログラム (TM) による*再同期* アクションの実行を待てないときに、ツール作成者が未確定トランザクションに対してヒューリスティック関数を実行するための一連の API が用意されています。この状態は、たとえば通信回線が故障し、未確定トランザクションが必要なリソースを拘束している場合などに発生する可能性があります。データベース・マネージャーの場合、これらのリソースには、トランザクションによって使用されている表と索引、ログ・スペース、および記憶域に対するロックが含まれます。さらに、未確定トランザクションが 1 つ増えるたびに、データベース・マネージャーが処理できる並行トランザクションの最大数が 1 つずつ減少します。

ヒューリスティック API では、未確定トランザクションを照会、コミット、およびロールバックすることができます。さらにログ・レコードを除去し、ログ・ページを解放することにより、手動操作によりコミットあるいはロールバックされたトランザクションを取り消すことができます。

考慮事項: ヒューリスティック API は、慎重に、そして最終手段としてのみ使用してください。 TM は、再同期イベントを開始します。 TM に再同期化アクションを開始するためのオペレーター・コマンドがある場合は、それが使用されます。ユーザーが TM によって開始される再同期を待てない場合は、ヒューリスティック・アクションが必要となります。

これらのアクションを実行する絶対確実な方法というものは存在しませんが、以下の指針を参考にしてください。

- 未確定トランザクションを表示するには、 **sqlxphqr** 関数を使用してください。それらの未確定トランザクションは、状態 = 'P' (準備済み) であり、接続されていません。 *xid* の *gtrid* 部分は、グローバル・トランザクションに参加する、他のリソース・マネージャー (RM) と同一のグローバル・トランザクション ID です。

© Copyright IBM Corp. 1993, 2000

- アプリケーションと操作環境の知識を利用して、参加している他の RM を識別してください。
- トランザクション管理プログラムが CICS で、 RM だけが CICS リソースである場合は、ヒューリスティック・ロールバックを実行してください。
- トランザクション管理プログラムが CICS でない場合は、それを使用して、未確定トランザクションと同じ *gtrid* を持つトランザクションの状況を判別してください。
- 1 つでも RM がコミットあるいはロールバックされた場合は、ヒューリスティック・コミットまたはロールバックを実行してください。
- それらすべてが準備済み状態にある場合は、ヒューリスティック・ロールバックを実行してください。
- 1 つでも利用不能な RM がある場合は、ヒューリスティック・ロールバックを実行してください。

トランザクション管理プログラムが利用可能であり、未確定トランザクションが、第 2 フェーズまたは以前の再同期において RM が使用可能でなかったことが原因で発生している場合、 DBA は、TM のログから、他の RM に対して行われたアクションを判別し、同じことを行わなければなりません。 *gtrid* は、TM と RM 間の突き合わせキーです。

ヒューリスティックにコミットまたはロールバックされたトランザクションがログ満杯の状態を引き起こさない限り、 621ページの『sqlxhfrg - トランザクション状況の忘却』を実行しないでください。この forget 関数は、この未確定トランザクションにより占有されているログ・スペースを解放します。トランザクション管理プログラムがこの未確定トランザクションに対して再同期アクションを実行する場合、 TM は、この RM 内にレコードが検索されないために、間違った判断を下し、他の RM をコミットまたはロールバックしてしまう可能性があります。一般に、レコードの脱落は、RM がロールバックされたことを示唆します。

sqlxhfrg - トランザクション状況の忘却

RM に、ヒューリスティックに完了した (ヒューリスティックにコミットあるいはロールバックされた) トランザクションの知識を消去させます。

許可

以下のいずれかです。

- *sysadm*
- *dbadm*

必須接続

データベース

API 組み込みファイル

sqlxa.h

C API 構文

```
/* File: sqlxa.h */
/* API: Forget Transaction Status */
/* ... */
extern int SQL_API_FN sqlxhfrg(
   SQLXA_XID            *pTransId,
   struct sqlca         *pSqlca
   );
/* ... */
```

API パラメーター

pTransId

入力。ヒューリスティックに忘却される、またはデータベース・ログから除去されるトランザクションの XA 識別子です。

pSqlca

出力。 *sqlca* 構造を指すポインターです。 この構造に関する詳細は、520ページの『SQLCA』を参照してください。

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。

SQLXA_XID 構造の詳細については、 616ページの『SQLXA-XID』を参照してください。

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。

効力範囲

この API は、それが発行されたノードにのみ影響を与えます。

許可

以下のいずれかです。

- *sysadm*
- *dbadm*

必須接続

データベース

API 組み込みファイル

sqlxa.h

C API 構文

```
/* File: sqlxa.h */
/* API: Commit an Indoubt Transaction */
/* ... */
extern int SQL_API_FN sqlxphcm(
   int                  exe_type,
   SQLXA_XID            *pTransId,
   struct sqlca         *pSqlca
   );
/* ... */
```

API パラメーター

exe_type
入力。 EXE_THIS_NODE が指定されると、操作はこのノードでのみ実行されます。

pTransId
入力。ヒューリスティックにコミットされるトランザクションの XA 識別子です。

pSqlca

出力。 *sqlca* 構造を指すポインターです。 この構造に関する詳細は、520ページの『SQLCA』を参照してください。

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、621ページの『sqlxhfrg - トランザクション状況の忘却』が発行されるまで、トランザクションの状態を記憶します。

SQLXA_XID 構造の詳細については、 616ページの『SQLXA-XID』を参照してください。

sqlxphqr - 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入手します。

効力範囲

この API は、それが発行されたノードにのみ影響を与えます。

許可

以下のいずれかです。

- *sysadm*
- *dbadm*

必須接続

データベース

API 組み込みファイル

sqlxa.h

C API 構文

```
/* File: sqlxa.h */
/* API: List Indoubt Transactions */
/* ... */
extern int SQL_API_FN sqlxphqr(
   int                  exe_type,
   SQLXA_RECOVER        **ppIndoubtData,
   sqlint32             *pNumIndoubts,
   struct sqlca         *pSqlca
   );
/* ... */
```

API パラメーター

exe_type
入力。 EXE_THIS_NODE が指定されると、操作はこのノードでのみ実行されます。

ppIndoubtData
出力。未確定トランザクションを保持する *SQLXA_RECOVER* 構造へのポインターのアドレスを提供します。この API は、未確定トランザク

ションのリストを保持するのに十分なスペースを割り振り、ポインターをこのスペースに戻します。スペースは、処理の終了時にのみ解放されます。このメモリーには解放されない、動的に割り振られた他の構造へのポインターを含むため、 231ページの『sqlefmem - メモリーの解放』を用いて解放しないでください。詳細については、613ページの『SQLXA-RECOVER』を参照してください。

pNumIndoubts

出力。 API は、 *ppIndoubtData* で戻された未確定トランザクションの数を戻します。

pSqlca

出力。 *sqlca* 構造を指すポインターです。 この構造に関する詳細は、520ページの『SQLCA』を参照してください。

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。

効力範囲

この API は、それが発行されたノードにのみ影響を与えます。

許可

以下のいずれかです。

- *sysadm*
- *dbadm*

必須接続

データベース

API 組み込みファイル

sqlxa.h

C API 構文

```
/* File: sqlxa.h */
/* API: Roll Back an Indoubt Transaction */
/* ... */
extern int SQL_API_FN sqlxphrl(
   int                  exe_type,
   SQLXA_XID            *pTransId,
   struct sqlca         *pSqlca
   );
/* ... */
```

API パラメーター

exe_type
入力。 EXE_THIS_NODE が指定されると、操作はこのノードでのみ実行されます。

pTransId
入力。ヒューリスティックにロールバックされる、トランザクションの XA 識別子です。

pSqlca

出力。 *sqlca* 構造を指すポインターです。 この構造に関する詳細は、520ページの『SQLCA』を参照してください。

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、621ページの『sqlxhfrg - トランザクション状況の忘却』 が発行されるまで、トランザクションの状態を記憶します。

SQLXA_XID 構造の詳細については、 616ページの『SQLXA-XID』を参照してください。

付録C. プリコンパイラーのカスタマイズ API

プリコンパイラーをカスタマイズするために使用できる、プリコンパイラー・サービス API のセットが用意されています。これらの API に関する情報と、その使用方法は、匿名 FTP サイト **ftp://ftp.software.ibm.com** から入手可能です。 `prepapi.psbin` という PostScript ファイルが、ディレクトリー `/ps/products/db2/info` にあります。このファイルはバイナリー形式です。

この電子フォーラムにアクセスできず、この資料のコピーを望まれる方は、サービス・インフォメーションのお知らせ で記述されているように IBM サービスにご連絡ください。

インターネットで入手できる情報の詳細と、それにアクセスする方法については、 759ページの『IBM と連絡をとる』を参照してください。

© Copyright IBM Corp. 1993, 2000

付録D. ベンダー製品用のバックアップおよび復元 API

DB2 では、サード・パーティーの媒体管理製品が、バックアップおよび復元操作のデータを保管および検索するために使用できるインターフェースが用意されています。この機能は、DB2 の標準部分としてサポートされている、ディスケット、ディスク、テープ、および Tivoli Storage Manager のバックアップおよび復元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・パーティーの媒体管理製品は、この付録ではベンダー製品と呼ばれています。

DB2 では、多くのベンダーが使用できる汎用データ・インターフェースを提供する関数プロトタイプのセットが定義されており、これらを用いてバックアップと復元を行います。これらの関数は、共用ライブラリー (UNIX ベースのシステムの場合) または DLL (OS/2 または Windows オペレーティング・システムの場合) において、ベンダーにより提供されます。関数が DB2 によって呼び出されると、バックアップまたは復元呼び出しルーチンによって指定された共用ライブラリーまたは DLL がロードされ、必要なタスクを実行するためにベンダー提供の関数が呼び出されます。

この付録は、次の 4 つの部分に分かれています。

- ベンダー製品と DB2 との対話の操作概要。
- DB2 のベンダー API の詳細記述。
- API 呼び出しで使用されるデータ構造についての情報。
- ベンダー製品を使用したバックアップおよび復元の呼び出しについての詳細。

操作概要

DB2 とベンダー製品の間のインターフェースのために、 5 つの関数が定義されています。

- sqluvint - 初期設定と装置へのリンク
- sqluvget - 装置からのデータの読み取り
- sqluvput - 装置へのデータの書き込み
- sqluvend - 装置へのリンク解除
- sqluvdel - コミット済みセッションの削除

© Copyright IBM Corp. 1993, 2000

DB2 がこれらの関数を呼び出したら、これらの関数は共用ライブラリー (UNIX ベースのシステムの場合) または DLL (OS/2 または Windows オペレーティング・システムの場合) において、ベンダー製品から提供されなければなりません。

注: 共用ライブラリーまたは DLL コードは、データベース・エンジン・コードの一部として実行されます。したがって、再入可能でなければならず、徹底的にデバッグされなければなりません。関数に誤りがあると、データベースのデータ保全性を危険にさらす可能性があります。

特定のバックアップまたは復元セッションで DB2 が呼び出す関数の順序は、以下の要因によって異なります。

- 使用されるセッションの数 (1 つまたは複数)。
- バックアップと復元のどちらが行われるか。
- バックアップまたは復元で指定された PROMPTING モード。
- データが格納されている装置の特性。
- 操作中に生じたエラー。

セッションの数

DB2 は、1 つまたは複数のデータ・ストリームまたはセッションを使用したデータベース・オブジェクトのバックアップおよび復元をサポートしています。バックアップまたは復元に 3 つのセッションを使用しているときには、 3 つの物理装置または論理装置が使用できなければなりません。ベンダー装置サポートが使用されているときには、それぞれの物理装置または論理装置へのインターフェースを管理するのはベンダーの関数です。 DB2 の側では、ベンダー提供の関数との間でデータ・バッファーの送受信を行うだけです。

使用されるセッションの数は、データベースのバックアップまたは復元関数を呼び出すアプリケーションによってパラメーターとして指定されます。この値は、**sqluvint** によって使用される INIT-INPUT 構造で提供されます (641ページの『sqluvint - 初期設定と装置へのリンク』を参照してください)。

DB2 は、指定された数値に達するか、または **sqluvint** 呼び出しから SQLUV_MAX_LINK_GRANT 警告戻りコードを受信するまで、セッションの初期設定を継続します。サポートされているセッションの最大数に達したことを DB2 へ警告するために、ベンダー製品には活動セッションの数を追跡するためのコードが必要です。 DB2 への警告が失敗すると、DB2 のセッションの初期設定要求が失敗し、結果としてすべてのセッションが終了し、バックアップまたは復元操作全体が失敗に終わる可能性があります。

この操作がバックアップであれば、DB2 は各セッションの開始時に媒体ヘッダー・レコードを書き込みます。このレコードには、DB2 が復元時にセッションを識別するために使用する情報が入ります。 DB2 は、バックアップの名前に順序番号を付加することによって、各セッションを固有に識別します。この番号は、最初のセッションを表す 1 で始まり、バックアップまたは復元操作用の **sqluvint** 呼び出しで他のセッションが開始されるたびに 1 ずつ増加します。詳細については、663ページの『INIT-INPUT』を参照してください。

バックアップが正常に完了すると、DB2 はクローズする最後のセッションに媒体トレーラーを書き込みます。このトレーラーには、バックアップを実行するのに使用したセッションの数を DB2 へ知らせる情報が含まれます。この情報は、復元時に、すべてのセッションまたはデータ・ストリームが復元されたことを確認するために使用されます。

エラー、警告、またはプロンプトなしの操作

バックアップの場合、DB2 は、**それぞれの**セッションごとに次の順序の呼び出しを発行します。

```
sqluvint, action = SQLUV_WRITE
```

続いて、1 個 ～ n 個の

```
sqluvput
```

続いて、1 個の

```
sqluvend, action = SQLUV_COMMIT
```

DB2 が **sqluvend** 呼び出し (アクション SQLUV_COMMIT) を発行するときには、ベンダー製品が出力データを適切に保管することを予期します。 DB2 へ SQLUV_OK の戻りコードが戻されれば、成功です。

sqluvint 呼び出しで使用される DB2-INFO 構造には、バックアップを識別するのに必要な情報が含まれます (657ページの『DB2-INFO』を参照してください)。順序番号が提供されます。ベンダー製品は、この情報を保管することを選択する場合があります。 DB2 は、この情報を復元時に使用し、復元されるバックアップを識別します。

復元の場合、セッションごとの呼び出しの順序は次のとおりです。

```
sqluvint, action = SQLUV_READ
```

続いて、1 個 ～ n 個の

```
sqluvget
```

続いて、1 個の

```
sqluvend, action = SQLUV_COMMIT
```

sqluvint 呼び出しで使用される DB2-INFO 構造内の情報には、バックアップを識別するのに必要な情報が含まれます。順序番号は提供されません。 DB2 はすべてのバックアップ・オブジェクト (バックアップ時にコミットされたセッション出力) が戻されることを予期します。最初に戻されるバックアップ・オブジェクトは、順序番号 1 で生成されたオブジェクトです。他のすべてのオブジェクトは順番に関係なく復元されます。 DB2 は、すべてのオブジェクトが処理されたかどうか確認するために、媒体の末尾を検査します。

注: すべてのベンダー製品が、バックアップ・オブジェクトの名前のレコードを保持するわけではありません。これは、テープや、容量が限られているその他の媒体にバックアップが行われる場合に特に当てはまります。復元セッションの初期設定時に、識別情報を利用して、必要なバックアップ・オブジェクトを、それらが必要とされるときに使用可能になるように計画することができます。これは、バックアップの保管にジュークボックスまたはロボット・システムを使用する場合に最も役立ちます。 DB2 は、必ず正しいデータが復元されるようにするために、必ず媒体ヘッダー (各セッションの出力の先頭レコード) をチェックします。

PROMPTING モード

バックアップまたは復元の開始時には、次の 2 つのプロンプト・モードが使用可能です。

- ベンダー製品がユーザーに対してメッセージを書き込んだり、ユーザーがそれらに応答する機会のない WITHOUT PROMPTING または NOINTERRUPT。
- ユーザーがベンダー製品からメッセージを受け取り、それに応答することができる PROMPTING または INTERRUPT。

PROMPTING モードの場合、バックアップおよび復元について次の 3 つの応答が可能です。

- 継続

  装置へのデータの書き込みまたは読み取りの操作が再開されます。
- 装置終了

  装置が追加のデータを受信せず、セッションが終了します。
- 終了

  バックアップまたは復元操作全体が終了します。

PROMPTING および WITHOUT PROMPTING モードの使用法については、以下のセクションで説明されています。

装置の特性

ベンダー装置サポート API のために、 2 つの一般タイプの装置が定義されています。

- 媒体を交換するためのユーザー・アクションが必要とされる、容量が限られている装置。たとえばテープ・ドライブ、ディスケット、または CD-ROM ドライブ。
- 通常の操作ではユーザーが媒体を扱う必要のない大容量の装置。たとえば、ジュークボックスや、高機能のロボット媒体処理装置。

容量が限られている装置では、バックアップまたは復元操作時に、追加媒体をロードするようユーザーに指示することが必要になる可能性があります。通常、DB2 では、バックアップまたは復元のいずれにおいても、媒体がロードされる順序は重要ではありません。また、DB2 では、ベンダー媒体処理メッセージをユーザーに渡すための機能も用意されています。このプロンプトでは、PROMPTING をオンにしてバックアップまたは復元操作を開始することが必要です。媒体処理メッセージのテキストは、戻りコード構造の記述フィールドで指定されます。

PROMPTING がオンになっており、 DB2 が **sqluvput** (書き込み) または **sqluvget** (読み取り) 呼び出しから SQLUV_ENDOFMEDIA または SQLUV_ENDOFMEDIA_NO_DATA の戻りコードを受信すると、DB2 は次のことを行います。

- 呼び出しが **sqluvput** であれば、セッションに送信された最後のバッファーが再送されるようにマークする。これは、後でセッションに置かれます。
- **sqluvend** (アクション = SQLUV_COMMIT) を用いてセッションを呼び出す。成功すると (SQLUV_OK 戻りコード)、DB2 は次のことを行います。
  - 媒体終端を示している戻りコード構造からのベンダー媒体処理メッセージを含むメッセージをユーザーに書き込む。
  - 継続、装置終了、または終了の応答をするようユーザーに指示する。

ユーザーの応答に基づき、DB2 は次のことを行います。

- **継続**の場合、 DB2 は **sqluvint** 呼び出しを使用して別のセッションを初期設定します。成功すると、セッションへのデータの書き込みまたはセッションからのデータの読み取りを開始します。書き込み時にセッションを固有に識別するために、 DB2 は順序番号を増加させます。順序番号は、**sqluvint**

で使用される DB2-INFO 構造内で使用できます。この番号は、セッションへ送信される最初のデータ・レコードである媒体ヘッダー・レコード内にあります。

DB2 は、バックアップまたは復元の開始時に要求された数、または **sqluvint** 時に SQLUV_MAX_LINK_GRANT 警告を出してベンダー製品が表示した数よりも多くのセッションを開始しません。

- **装置終了**の場合、DB2 は別のセッションを初期設定せず、活動セッションの数が 1 減ります。 DB2 は、装置終了の応答によってすべてのセッションを終了させることはありません。バックアップまたは復元操作が完了する (たとえば、すべてのデータが処理される) まで、少なくとも 1 つのセッションが活動状態のままでなければなりません。
- **終了**の場合、DB2 はバックアップまたは復元操作を終了させます。セッションを終了させるときの DB2 の処理については、 637ページの『エラー条件が DB2 に戻される場合』を参照してください。

バックアップまたは復元時のパフォーマンスは、使用している装置の数によって異なってくるため、処理の並列性を保持しておくことが重要になります。バックアップの場合、残りの活動セッションで、書き込まれるデータが保持されることが分かっていない限り、プロンプトに対して継続の応答を行うようお勧めします。復元の場合、すべての媒体が処理されたか、または処理中である (たとえば、すべてのテープが読み取られたか、または読み取り中である) まで、継続の応答を使用しなければなりません。

バックアップまたは復元モードが **WITHOUT PROMPTING** であり、 DB2 がセッションから SQLUV_ENDOFMEDIA または SQLUV_ENDOFMEDIA_NO_DATA の戻りコードを受信すると、 DB2 はセッションを終了し、別のセッションをオープンしません。バックアップまたは復元が完了する前に、すべてのセッションが DB2 へ媒体終端を戻すと、バックアップまたは復元操作が失敗します。このため、容量が限られた装置で WITHOUT PROMPTING を使用する際には注意が必要です。ただし、大容量の装置にとっては、このモードで稼働することは有意義です。

ベンダー製品では、装置に限りなく容量があるように見えるように、媒体の装てんおよび交換アクションを DB2 から隠すことができます。一部の大容量装置は、このモードで稼働します。そのような場合は、バックアップされたすべてのデータが復元操作の進行中に同じ順序で DB2 に戻されることが重要です。そうでなければ、データが消失する可能性がありますが、 DB2 は消失したデータを検出する方法を持たないため、復元操作が成功したものとみなします。

DB2 は、各バッファーが 1 つの媒体 (たとえば、テープ) に入れられるということを前提にして、データをベンダー製品へ書き込みます。ベンダー製品は、DB2 側には知らせずに、バッファーを複数の媒体に分割することができます。このような場合、複数の媒体から再構成したバッファーを DB2 に戻すのはベンダー製品の責任であるため、復元中に媒体が処理される順序は重要になります。順序が正しくなければ、復元操作は失敗します。

エラー条件が DB2 に戻される場合

バックアップまたは復元操作時に、DB2 は、すべてのセッションが正常に完了するか、そうでなければバックアップまたは復元操作全体が失敗することを予期します。セッションは、**sqluvend** (アクション = SQLUV_COMMIT) の呼び出し時に、 SQLUV_OK 戻りコードを用いて DB2 に正しい完了 (たとえば、コミット済み) を知らせます。

回復不能エラーが検出されると、セッションは DB2 によって終了されます。これらのエラーは DB2 エラーとなるか、ベンダー製品から DB2 へ戻されるエラーとなります。バックアップまたは復元が完了するにはすべてのセッションが正常にコミットされなければならないため、 1 つが失敗すると、DB2 はその操作と関連した他のセッションも終了させてしまいます。

ベンダー製品が DB2 からの呼び出しに対して回復不能を示す戻りコードで応答することを決定する場合、ベンダー製品は、オプションで、RETURN-CODE 構造の記述フィールドに置かれたメッセージ・テキストを使用して、ユーザーに追加情報を提供することができます。このメッセージ・テキストは、訂正の処置をとれるように DB2 情報と共に表示されます。

バックアップのシナリオとして、あるセッションが正常にコミットされたものの、バックアップ操作に関連した他のセッションで回復不能エラーが発生したというケースが考えられます。バックアップ操作が成功するためにはすべてのセッションが正常に完了しなければならないため、 DB2 はコミットされたセッション内の出力データを削除する必要があります。 DB2 は、**sqluvdel** 呼び出しを発行して、オブジェクトの削除を要求します。この呼び出しは入出力セッションとはみなされず、バックアップ・オブジェクトの削除に必要な接続を初期化したり終了したりする機能を果たします。

DB2-INFO 構造内の情報には順序番号は含まれていません。 **sqluvdel** は、DB2-INFO 構造内の残りのパラメーターと一致するバックアップ・オブジェクトをすべて削除します。

警告条件

DB2 は、ベンダー製品から警告戻りコードを受信する可能性があります。たとえば、装置が作動不能である場合や、その他の訂正可能条件が生じた場合などです。これは、読み取りと書き込みの両方の操作に当てはまります。

sqluvput および **sqluvget** の呼び出し時に、ベンダーは戻りコードを SQLUV_WARNING に設定することができます。さらに、オプションで、戻りコード構造の記述フィールドに置かれたメッセージ・テキストを使用して、ユーザーに追加情報を提供することができます。このメッセージ・テキストは、訂正の処置をとれるように表示されます。ここでも、ユーザーは、3 つの方法 (継続、装置終了、または終了) の 1 つで応答することができます。ユーザーとの通信を確立するのに使用されるメカニズムは、媒体終端条件の場合と同じです。

DB2 のアクションは、次のとおりです。

- 継続の場合、操作がバックアップであれば、 DB2 は **sqluvput** を使用してバッファーの再書き込みを試みます。操作が復元であれば、DB2 は **sqluvget** 呼び出しを発行して、次のバッファーを読み取ります。
- 装置終了または終了の場合、DB2 は回復不能エラーが生じたときと同じ方法で、バックアップまたは復元全体を終了させます (たとえば、活動セッションを終了し、コミット済みセッションを削除する)。

それぞれの関数呼び出しについての可能な戻りコードと DB2 の反応は、以下の API セクションで詳細に記述されています。

操作上のヒント

このセクションでは、ベンダー製品の作成時のヒントをいくつか提供します。

回復活動記録ファイル

回復活動記録ファイルは、データベース回復操作に役立てることができます。このファイルは、各データベースに関連しており、バックアップまたは復元操作が行われるたびに自動的に更新されます。ファイルの概要は、*管理の手引き* で示されています。ファイル内の情報は、以下の機能を使用して表示、更新、および除去することができます。

- コントロール・センター
- コマンド行プロセッサー
  - LIST HISTORY
  - PRUNE HISTORY

  - UPDATE RECOVERY HISTORY FILE
- API
  - sqluhcls、sqluhgne、slquhops、sqluhprn、および sqluhupd

ファイルのレイアウトについては、 493ページの『db2HistData』 を参照してください。

バックアップ操作が完了すると、 1 つまたは複数のレコードがファイルへ書き込まれます。バックアップ操作の出力がベンダー装置に送られた場合、活動記録レコード内の DEVICE フィールドには 0 が入り、 LOCATION フィールドには以下のいずれかが入ります。

- バックアップが呼び出されたときに提供されたベンダー・ファイル名。
- バックアップが呼び出されたときに提供されたベンダー・ファイル名がなければ、共用ライブラリーの名前。

このオプションの指定に関する詳細については、 668ページの『ベンダー製品を使用してバックアップ / 復元を呼び出す』を参照してください。ベンダー・ファイル名が指定されていなければ、LOCATION はブランクになります。

上記の機能を使用して、LOCATION フィールドを更新できます。バックアップを保存するのに容量の限られた装置 (たとえば、取り外し可能媒体) が使用されており、その媒体が異なる記憶場所 (たとえば、オフ・サイト) へ物理的に移動された場合、この機能を使用してバックアップ情報のロケーションを更新することができます。これを行えば、回復が必要となったときに、このファイルを使用してバックアップを見つけることができます。

関数およびデータ構造

以下のセクションでは、ベンダー製品で使用できる汎用関数とデータ構造について説明します。

ベンダー製品用の API は、以下のとおりです。

- 641ページの『sqluvint - 初期設定と装置へのリンク』
- 646ページの『sqluvget - 装置からのデータの読み取り』
- 649ページの『sqluvput - 装置へのデータの書き込み』
- 652ページの『sqluvend - 装置のリンク解除および資源の解放』
- 655ページの『sqluvdel - コミット済みセッションの削除』

ベンダー API によって使用されるデータ構造は、以下のとおりです。

sqluvint - 初期設定と装置へのリンク

この関数は、 DB2 とベンダー製品との間の論理リンクの初期設定および確立に関する情報を提供するために呼び出されます。

許可

以下のいずれかです。

- *sysadm*
- *dbadm*

必須接続

データベース

API 組み込みファイル

sql.h

C API 構文

```
/* File: sqluvend.h */
/* API: Initialize and Link to Device */
/* ... */
int sqluvint (
  struct Init_input   *,
  struct Init_output  *,
  struct Return_code  *);
/* ... */
```

API パラメーター

Init_input
入力。ベンダー装置との論理リンクを確立するための、 DB2 が提供する情報を含む情報。

Init_output
出力。ベンダー装置が戻した出力を含む構造。

Return_code
出力。 DB2 へ渡される戻りコードと短いテキスト記述を含む構造。

使用上の注意

それぞれの媒体入出力セッションごとに、 DB2 はこの関数を呼び出して装置ハンドルを取得します。何かの理由で初期設定時にベンダー関数にエラーが発生すると、戻りコードを通してそのことを知らせます。エラーを示す戻りコードであれば、DB2 は、 **sqluvend** 関数を呼び出して操作を終了させることがあります。可能な戻りコードと、これらのそれぞれに対する DB2 の反応については、戻りコード表 (643ページの表83 を参照) に記載されています。

INIT-INPUT 構造には、ベンダー製品がバックアップまたは復元を続行できるかどうかを判別するために使用できる要素が含まれます。

- 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媒体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・レベルと関連づけて使用して、バックアップまたは復元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、取り外し可能媒体の変更 (たとえば、テープ・ドライブへ他のテープを入れる) などのアクションを指示できるかどうかをベンダーに示します。これは、ユーザーへの指示方法がないために操作が続行できないことを示唆する場合があります。

  プロンプト・レベルが WITHOUT PROMPTING であり、取り外し可能媒体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 (詳細については、 634ページの『PROMPTINGモード』および 635ページの『装置の特性』を参照してください)。

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復元を指定します。アクション = SQLUV_READ の場合、ベンダー製品は指定されたオブジェクトの存在を調べなければなりません。見つからなければ、DB2 が適切な処置をとれるように戻りコードが SQLUV_OBJ_NOT_FOUND に設定される必要があります。

初期設定が正常に完了した後、DB2 は他のデータ転送機能を発行して継続しますが、 **sqluvend** 呼び出しでいつでもセッションを終了させることができます。

戻りコード

表 83. sqluvint についての有効な戻りコードと結果の DB2 アクション

ヘッダー・ファイルのリテラル	説明	次の呼び出し	その他のコメント
SQLUV_OK	操作が成功した。	sqluvput、sqluvget (コメント参照)	アクション = SQLUV_WRITE であれば、次の呼び出しは sqluvput (データのバックアップ) です。アクション = SQLUV_READ であれば、SQLUV_OK を戻す前に、指定されたオブジェクトの存在を確認します。次の呼び出しは、sqluvget (データの復元) になります。
SQLUV_LINK_EXIST	セッションがすでに活動化されている。	ありません	セッションの開始に失敗しました。セッションに割り振られていたメモリーを解放し、終了します。セッションが確立されなかったため、sqluvend 呼び出しは受信されません。
SQLUV_COMM_ERROR	装置との通信エラー	ありません	セッションの開始に失敗しました。セッションに割り振られていたメモリーを解放し、終了します。セッションが確立されなかったため、sqluvend 呼び出しは受信されません。
SQLUV_INV_VERSION	DB2 とベンダー製品に互換性がない。	ありません	セッションの開始に失敗しました。セッションに割り振られていたメモリーを解放し、終了します。セッションが確立されなかったため、sqluvend 呼び出しは受信されません。
SQLUV_INV_ACTION	無効なアクションが要求された。これは、パラメーターの組み合わせにより不可能な操作が生じたことを示すためにも使用される。	ありません	セッションの開始に失敗しました。セッションに割り振られていたメモリーを解放し、終了します。セッションが確立されなかったため、sqluvend 呼び出しは受信されません。
SQLUV_NO_DEV_AVAIL	現在使用できる装置がない。	ありません	セッションの開始に失敗しました。セッションに割り振られていたメモリーを解放し、終了します。セッションが確立されなかったため、sqluvend 呼び出しは受信されません。
SQLUV_OBJ_NOT_FOUND	指定されたオブジェクトが見つからない。これは、sqluvint 呼び出しでのアクションが 'R' (読み取り) であり、DB2-INFO 構造で指定された基準に基づいて要求されたオブジェクトが見つからないときに使用される。	ありません	セッションの開始に失敗しました。セッションに割り振られていたメモリーを解放し、終了します。セッションが確立されなかったため、sqluvend 呼び出しは受信されません。

表 83. sqluvint についての有効な戻りコードと結果の DB2 アクション (続き)

ヘッダー・ファイルのリテラル	説明	次の呼び出し	その他のコメント
SQLUV_OBJS_FOUND	2 つ以上のオブジェクトが、指定された基準に一致する。これは、sqluvint 呼び出しでのアクションが 'R' (読み取り) であり、 DB2-INFO 構造内の基準と一致するオブジェクトが 2 つ以上あるときに発生する。	ありません	セッションの開始に失敗しました。セッションに割り振られていたメモリーを解放し、終了します。セッションが確立されなかったため、sqluvend 呼び出しは受信されません。
SQLUV_INV_USERID	指定されたユーザー ID が無効である。	ありません	セッションの開始に失敗しました。セッションに割り振られていたメモリーを解放し、終了します。セッションが確立されなかったため、sqluvend 呼び出しは受信されません。
SQLUV_INV_PASSWORD	提供されたパスワードが無効である。	ありません	セッションの開始に失敗しました。セッションに割り振られていたメモリーを解放し、終了します。セッションが確立されなかったため、sqluvend 呼び出しは受信されません。
SQLUV_INV_OPTIONS	ベンダー・オプション・フィールド内で無効なオプションが検出された。	ありません	セッションの開始に失敗しました。セッションに割り振られていたメモリーを解放し、終了します。セッションが確立されなかったため、sqluvend 呼び出しは受信されません。
SQLUV_INIT_FAILED	初期設定が失敗し、セッションが終了される。	ありません	セッションの開始に失敗しました。セッションに割り振られていたメモリーを解放し、終了します。セッションが確立されなかったため、sqluvend 呼び出しは受信されません。
SQLUV_DEV_ERROR	装置エラー	ありません	セッションの開始に失敗しました。セッションに割り振られていたメモリーを解放し、終了します。セッションが確立されなかったため、sqluvend 呼び出しは受信されません。
SQLUV_MAX_LINK_GRANT	最大数のリンクが確立された。	sqluvput、sqluvget (コメント参照)	これは、DB2 からの警告として扱われます。この警告は、サポート可能なセッションの最大数に達しているので、これ以上ベンダー製品とのセッションをオープンしないよう DB2 に知らせるものです (注意: これは、装置の可用性が原因となっている可能性もあります)。アクション = SQLUV_WRITE (BACKUP) であれば、次の呼び出しは sqluvput です。アクション = SQLUV_READ であれば、SQLUV_MAX_LINK_GRANT を戻す前に、指定されたオブジェクトの存在を確認します。次の呼び出しは、 sqluvget (データの復元) になります。
SQLUV_IO_ERROR	入出力エラー	ありません	セッションの開始に失敗しました。セッションに割り振られていたメモリーを解放し、終了します。セッションが確立されなかったため、sqluvend 呼び出しは受信されません。

表 83. sqluvint についての有効な戻りコードと結果の DB2 アクション (続き)

ヘッダー・ファイルのリテラル	説明	次の呼び出し	その他のコメント
SQLUV_NOT_ENOUGH_SPACE	バックアップ・イメージ全体を格納するための十分なスペースがない (サイズの見積もりは 64 ビットのバイト数で提供される)。	ありません	セッションの開始に失敗しました。セッションに割り振られていたメモリーを解放し、終了します。セッションが確立されなかったため、sqluvend 呼び出しは受信されません。

sqluvget - 装置からのデータの読み取り

初期設定の後、この関数を呼び出して装置からデータを読み取ることができます。

許可

以下のいずれかです。

- *sysadm*
- *dbadm*

必須接続

データベース

API 組み込みファイル

sqluvend.h

C API 構文

```
/* File: sqluvend.h */
/* API: Reading Data from Device */
/* ... */
int sqluvget (
  void * pVendorCB,
  struct Data         *,
  struct Return_code  *);
/* ... */
typedef struct Data
}
   sqlint32  obj_num;
   sqlint32  buff_size;
   sqlint32  actual_buff_size;
   void      *dataptr;
   void      *reserve;
{ Data;
```

API パラメーター

pVendorCB
入力。 DATA 構造 (データ・バッファー) および Return_code 用に割り振られるスペースを指すポインター。

Data 入出力。データ 構造を指すポインター。

Return_code
出力。 API 呼び出しからの戻りコード。

obj_num
検索するバックアップ・オブジェクトを指定します。

buff_size
使用するバッファー・サイズを指定します。

actual_buff_size
読み取られる、または書き込まれる実際のバイト数を指定します。この値は、実際に読み取られたデータのバイト数を示す出力に設定してください。

dataptr
データ・バッファーを指すポインター。

reserve
将来の利用のために予約されています。

使用上の注意

これは復元関数で使用されます。

戻りコード

表 84. sqluvget についての有効な戻りコードと結果の DB2 アクション

ヘッダー・ファイルのリテラル	説明	次の呼び出し	その他のコメント
SQLUV_OK	操作が成功した。	sqluvget	DB2 はデータを処理します。
SQLUV_COMM_ERROR	装置との通信エラー	sqluvend、アクション = SQLU_ABORT[a]	セッションは終了します。
SQLUV_INV_ACTION	無効なアクションが要求された。	sqluvend、アクション = SQLU_ABORT[a]	セッションは終了します。
SQLUV_INV_DEV_HANDLE	無効な装置ハンドル	sqluvend、アクション = SQLU_ABORT[a]	セッションは終了します。
SQLUV_INV_BUFF_SIZE	無効なバッファー・サイズが指定された。	sqluvend、アクション = SQLU_ABORT[a]	セッションは終了します。
SQLUV_DEV_ERROR	装置エラー	sqluvend、アクション = SQLU_ABORT[a]	セッションは終了します。
SQLUV_WARNING	警告。これは、DB2 に媒体終端を示すためには使用されない (その目的では、SQLUV_ENDOFMEDIA または SQLUV_ENDOFMEDIA_NO_DATA が使用される)。ただし、装置の作動不能条件がこの戻りコードによって示される可能性がある。	sqluvget、または sqluvend、アクション =SQLU_ABORT	警告に対する DB2 の扱いについての説明 (638ページの『警告条件』) を参照してください。

表 84. sqluvget についての有効な戻りコードと結果の DB2 アクション (続き)

ヘッダー・ファイルのリテラル	説明	次の呼び出し	その他のコメント
SQLUV_LINK_NOT_EXIST	リンクが現在存在していない。	sqluvend、アクション = SQLU_ABORT[a]	セッションは終了します。
SQLUV_MORE_DATA	操作が成功し、さらにデータが使用可能である。	sqluvget	
SQLUV_ENDOFMEDIA_NO_DATA	媒体が終了し、0 バイトが読み取られた (たとえば、テープの終わり)。	sqluvend	媒体終端条件に対する DB2 の扱いについての説明は、 634ページの『PROMPTING モード』および 635ページの『装置の特性』を参照してください。
SQLUV_ENDOFMEDIA	媒体が終了し、> 0 バイトが読み取られた (たとえば、テープの終わり)。	sqluvend	DB2 はデータを処理し、634ページの『PROMPTING モード』および 635ページの『装置の特性』で説明されているように媒体終端条件を処理します。
SQLUV_IO_ERROR	入出力エラー	sqluvend、アクション = SQLU_ABORT[a]	セッションは終了します。

次の呼び出し:

- [a] 次の呼び出しが sqluvend、アクション = SQLU_ABORT である場合には、このセッションは終了されます。さらに、sqluvend、アクション = SQLU_ABORT では、他のすべての活動セッションが終了されます。

sqluvput - 装置へのデータの書き込み

初期設定の後、この関数を使用して装置へデータを書き込むことができます。

許可

以下のいずれかです。

- *sysadm*
- *dbadm*

必須接続

データベース

API 組み込みファイル

sqluvend.h

C API 構文

```
/* File: sqluvend.h */
/* API: Writing Data to Device */
/* ... */
int sqluvput (
  void * pVendorCB,
  struct Data  *,
  struct Return_code  *);
/* ... */
typedef struct Data
}
   sqlint32  obj_num;
   sqlint32  buff_size;
   sqlint32  actual_buff_size;
   void      *dataptr;
   void      *reserve;
{ Data;
```

API パラメーター

pVendorCB
入力。 DATA 構造 (データ・バッファー) および Return_code 用に割り振られたスペースを指すポインター。

Data 出力。書き込まれるデータが入れられたデータ・バッファー。

Return_code
出力。 API 呼び出しからの戻りコード。

obj_num
検索するバックアップ・オブジェクトを指定します。

buff_size
使用するバッファー・サイズを指定します。

actual_buff_size
読み取られる、または書き込まれる実際のバイト数を指定します。この値は、実際に読み取られたデータのバイト数を示す出力に設定してください。

dataptr
データ・バッファーを指すポインター。

reserve
将来の利用のために予約されています。

使用上の注意

これはバックアップ関数で使用されます。

戻りコード

表 85. sqluvput についての有効な戻りコードと結果の DB2 アクション

ヘッダー・ファイルのリテラル	説明	次の呼び出し	その他のコメント
SQLUV_OK	操作が成功した。	完了 (たとえば、DB2 にこれ以上データがなくなった) 後、sqluvput または sqluvend	他のプロセスに操作の成功を通知します。
SQLUV_COMM_ERROR	装置との通信エラー	sqluvend、アクション = SQLU_ABORT[a]	セッションは終了します。
SQLUV_INV_ACTION	無効なアクションが要求された。	sqluvend、アクション = SQLU_ABORT[a]	セッションは終了します。
SQLUV_INV_DEV_HANDLE	無効な装置ハンドル	sqluvend、アクション = SQLU_ABORT[a]	セッションは終了します。
SQLUV_INV_BUFF_SIZE	無効なバッファー・サイズが指定された。	sqluvend、アクション = SQLU_ABORT[a]	セッションは終了します。
SQLUV_ENDOFMEDIA	媒体が終了した (たとえば、テープの終了)。	sqluvend	媒体終端条件に対する DB2 の扱いについての説明は、 634ページの『PROMPTING モード』および 635ページの『装置の特性』を参照してください。
SQLUV_DATA_RESEND	装置が再びバッファーを送信するよう要求した。	sqluvput	DB2 は最新のバッファーを再度送信します。これは一度だけ行われます。
SQLUV_DEV_ERROR	装置エラー	sqluvend、アクション = SQLU_ABORT[a]	セッションは終了します。

表 85. sqluvput についての有効な戻りコードと結果の DB2 アクション (続き)

ヘッダー・ファイルのリテラル	説明	次の呼び出し	その他のコメント
SQLUV_WARNING	警告。これは、DB2 に媒体終端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	警告に対する DB2 の扱いについての説明 (638ページの『警告条件』) を参照してください。
SQLUV_LINK_NOT_EXIST	リンクが現在存在していない。	sqluvend、アクション = SQLU_ABORT[a]	セッションは終了します。
SQLUV_IO_ERROR	入出力エラー	sqluvend、アクション = SQLU_ABORT[a]	セッションは終了します。

次の呼び出し:

- [a] 次の呼び出しが sqluvend、アクション = SQLU_ABORT である場合には、このセッションは終了されます。さらに、sqluvend、アクション = SQLU_ABORT では、他のすべての活動セッションが終了されます。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 (637ページの『エラー条件が DB2 に戻される場合』を参照してください)。

sqluvend - 装置のリンク解除および資源の解放

装置を終了またはリンク解除し、関連した資源をすべて解放します。ベンダーは DB2 へ戻る前に、使用していない資源 (たとえば、割り振られたスペースやファイル・ハンドル) を解放する必要があります。

許可

以下のいずれかです。

- *sysadm*
- *dbadm*

必須接続

データベース

API 組み込みファイル

sql.h

C API 構文

```
/* File: sqluvend.h */
/* API: Unlink the Device and Release its Resources */
/* ... */
int sqluvend (
  sqlint32 action,
  void * pVendorCB,
  struct Init_output  *,
  struct Return_code  *);
/* ... */
```

API パラメーター

action 入力。セッションのコミットまたは打ち切りに使用されます。

- SQLUV_COMMIT (0 = コミット)
- SQLUV_ABORT (1 = 打ち切り)

pVendorCB
入力。 Init_output 構造を指すポインター。

Init_output
出力。割り振り解除された Init_output 用のスペース。アクションがコ

ミットであれば、データはバックアップのために保管用の記憶域へコミットされています。アクションが打ち切りであれば、データはバックアップのために除去されます。

Return code
出力。 API 呼び出しからの戻りコード。

使用上の注意

この関数は、セッションがオープンされるたびに呼び出されます。

可能なアクション・コードは、次の 2 つがあります。

- コミット

  このセッションへのデータの出力またはセッションからのデータの読み取りが完了します。

  書き込み (BACKUP) 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 (RESTORE) 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 はバックアップまたは復元全体に問題があると判断します。すべての活動セッションは、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 (637ページの『エラー条件が DB2 に戻される場合』を参照してください)。

- 打ち切り

  DB2 によって問題が生じているときには、セッションではデータの読み取りまたはデータの書き込みは行われません。

  書き込み (BACKUP) 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。また、DB2 はバックアップ全体に問題があると判断します。すべての活動セッションは、アクション = 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 (637ページの『エラー条件が DB2 に戻される場合』を参照してください)。

読み取り (RESTORE) 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、アクション = SQLUV_ABORT の sqluvend 呼び出しですべての復元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ず、後続の障害は無視されます。この場合、アクション = SQLUV_ABORT の sqluvend 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 86. sqluvend についての有効な戻りコードと結果の DB2 アクション

ヘッダー・ファイルのリテラル	説明	次の呼び出し	その他のコメント
SQLUV_OK	操作が成功した。	ありません	このセッションに割り振られていたメモリーをすべて解放し、終了します。
SQLUV_COMMIT_FAILED	コミット要求が失敗した。	ありません	このセッションに割り振られていたメモリーをすべて解放し、終了します。
SQLUV_ABORT_FAILED	打ち切り要求が失敗した。	ありません	

sqluvdel - コミット済みセッションの削除

コミット済みセッションを削除します。

許可

以下のいずれかです。

- *sysadm*
- *dbadm*

必須接続

データベース

API 組み込みファイル

sqluvend.h

C API 構文

```
/* File: sqluvend.h */
/* API: Delete Committed Session */
/* ... */
int sqluvdel (
  struct Init_input   *,
  struct Init_output  *,
  struct Return_code  *);
/* ... */
```

API パラメーター

Init_input
入力。 Init_input および Return_code 用に割り振られたスペース。

Return_code
出力。 API 呼び出しからの戻りコード。 Init_input 構造によって指示されたオブジェクトは削除されます。

使用上の注意

複数のセッションがオープンしていて、いくつかのセッションはコミットされたが 1 つが失敗したというような場合、この関数が呼び出されて、コミット済みセッションを削除します。順序番号は指定されません。特定のバックアップ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 はこのエラーを呼び出し側へ通知しません。 DB2 は最初の致命的な障害は戻し、その後に続く障害は無視するという方式なので、このように行います。この場合、**sqluvdel** を呼び出した DB2 に関して、最初の致命的エラーが発生しています。

戻りコード

表 87. sqluvdel についての有効な戻りコードと結果の DB2 アクション

ヘッダー・ファイルのリテラル	説明	次の呼び出し	その他のコメント
SQLUV_OK	操作が成功した。	ありません	
SQLUV_DELETE_FAILED	削除要求が失敗した。	ありません	

DB2-INFO

この構造には、DB2 が提供する情報が含まれ、この情報でベンダー装置が DB2 を識別します。

注: フィールドはすべて、ヌル文字で終了するストリングです。

表 88. 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 製品を表す識別子。指示する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。指示する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。重要性がなければヌルに設定します。指示する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。重要性がなければヌルに設定します。指示する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。指示する文字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。ヌルであれば、*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。指示する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名。指示する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。指示するストリングの最大長は 8 文字です。

表 88. 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、または実行する復元のタイプを指定します。以下の値を指定することができます。 アクションが SQLUV_WRITE の場合: 0 - データベースの全バックアップ 3 - 表スペース・レベルのバックアップ アクションが SQLUV_READ の場合: 0 - 全復元 3 - オンラインの表スペース復元 4 - 表スペース復元 5 - 活動記録ファイル復元
dbname	char	バックアップまたは復元するデータベースの名前。指示する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復元するデータベースの別名。指示する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。指示する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、値は常にゼロです。指示するストリングの最大長は 3 文字です。
obj_list	struct sqlu_gen_list	バックアップ・イメージ内のオブジェクトをリストします。これは、情報としてのみベンダーに提供されます。
max_bytes_per_txn	sqlint32	ユーザーによって指定された転送バッファー・サイズをバイト単位でベンダーに指定します。
image_filename	char	将来の利用のために予約されています。
reserve	void	将来の利用のために予約されています。
nodename	char	バックアップが生成されたノードの名前。
password	char	バックアップが生成されたノードのパスワード。
owner	char	バックアップの開始元の ID。
mcNameP	char	管理クラス。

表 88. 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
nodeNum	SQL_PDB_NODE_TYPE	ノード番号。ベンダー・インターフェースでは、255 より大きい番号がサポートされます。

バックアップ・イメージは、*filename*、または *server_id*、 *db2instance*、*type*、*dbname* と *timestamp* によって固有に識別されます。 *seq* によって指定される順序番号は、ファイル拡張子を識別します。バックアップ・イメージを復元するときには、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ヌルに設定されたり、その逆が起こることがあります。

言語構文

C 構造

```
/* File: sqluvend.h */
/* ... */
typedef struct DB2_info
{
  char                   *DB2_id;
  char                   *version;
  char                   *release;
  char                   *level;
  char                   *action;
  char                   *filename;
  char                   *server_id;
  char                   *db2instance;
  char                   *type;
  char                   *dbname;
  char                   *alias;
  char                   *timestamp;
  char                   *sequence;
  struct sqlu_gen_list   *obj_list;
  long                   max_bytes_per_txn;
  char                   *image_filename;
  void                   *reserve;
  char                   *nodename;
  char                   *password;
  char                   *owner;
  char                   *mcNameP;
  SQL_PDB_NODE_TYPE      nodeNum;
} DB2_info;
/* ... */
```

VENDOR-INFO

この構造には、ベンダーと、使用される装置のバージョンを識別するための情報が含まれます。

注: フィールドはすべて、ヌル文字で終了するストリングです。

表 89. 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	ベンダーを表す識別子。指示する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。指示する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。重要性がなければヌルに設定します。指示する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。重要性がなければヌルに設定します。指示する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名。指示するストリングの最大長は 8 文字です。
max_bytes_per_txn	sqlint32	サポートされる最大の転送バッファー・サイズ。ベンダーが指定します (バイト単位)。これは、ベンダーの初期設定関数からの戻りコードが SQLUV_BUFF_SIZE (無効なバッファー・サイズが指定されたことを示す) である場合にのみ使用されます。
num_objects_in_backup	sqlint32	あるバックアップを完了するために使用されたセッションの数。これは、復元時に、すべてのバックアップ・イメージが処理されたかどうかを判別するために使用されます。
reserve	void	将来の利用のために予約されています。

言語構文

C 構造

```
typedef struct Vendor_info
{
  char       *vendor_id;
  char       *version;
  char       *release;
  char       *level;
  char       *server_id;
  sqlint32   max_bytes_per_txn;
  sqlint32   num_objects_in_backup;
  void       *reserve;
} Vendor_info;
```

INIT-INPUT

この構造には、ベンダー装置との論理リンクを設定し、確立するために DB2 が提供する情報が含まれます。

注: フィールドはすべて、ヌル文字で終了するストリングです。

表 90. 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	options フィールドの長さ。 sqlubkp および sqlurestore を使用している場合、このフィールドのデータは *VendorOptionsSize* パラメーターから直接渡されます。
size_HI_order	sqluint32	バイト単位で見積もられた DB サイズの高位 32 ビット。合計サイズは 64 ビットです。
size_LOW_order	sqluint32	バイト単位で見積もられた DB サイズの低位 32 ビット。合計サイズは 64 ビットです。
options	void	この情報は、バックアップまたは復元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、バイト逆転が行われず、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。 sqlubkp および sqlurestore 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	バックアップまたは復元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。指示する文字ストリングの最大長は 1 文字です。
num_sessions	unsigned short	バックアップまたは復元を呼び出したときにユーザーが要求したセッションの数。

言語構文

C 構造

```
typedef struct Init_input
{
  struct DB2_info  *DB2_session;
  unsigned short   size_options;
  sqluint32        size_HI_order;
  sqluint32        size_LOW_order;
  void             *options;
  void             *reserve;
  char             *prompt_lvl;
  unsigned short   num_sessions;
} Init_input;
```

INIT-OUTPUT

この構造には、ベンダー装置が戻した出力が含まれます。

表 91. INIT-OUTPUT 構造のフィールド

フィールド名	データ・タイプ	説明
vendor_session	struct Vendor_info	ベンダーを DB2 に識別させるための情報が含まれます。
pVendorCB	void	ベンダーの制御ブロック。
reserve	void	将来の利用のために予約されています。

言語構文

C 構造

```
typedef struct Init_output
{
  struct Vendor_info  *vendor_session;
  void                *pVendorCB;
  void                *reserve;
} Init_output;
```

DATA

この構造には、DB2 とベンダー装置の間で転送されるデータ (読み取りおよび書き込み) が含まれます。

表 92. DATA 構造のフィールド

フィールド名	データ・タイプ	説明
obj_num	sqlint32	バックアップ時に DB2 によって割り当てられる順序番号。
buff_size	sqlint32	バッファーのサイズ。
actual_buf_size	sqlint32	送受信された実際のバイト数。これは *buff_size* を超えてはなりません。
dataptr	void	データ・バッファーを指すポインター。 DB2 はこのバッファーにスペースを割り振ります。
reserve	void	将来の利用のために予約されています。

言語構文

C 構造

```
typedef struct Data
{
  sqlint32  obj_num;
  sqlint32  buff_size;
  sqlint32  actual_buff_size;
  void      *dataptr;
  void      *reserve;
} Data;
```

RETURN-CODE

この構造には、DB2 へ戻される戻りコードとエラーの短いテキスト説明が含まれます。

表 93. RETURN-CODE 構造のフィールド

フィールド名	データ・タイプ	説明
return_code[a]	sqlint32	ベンダー関数からの戻りコード。
説明	char	戻りコードの短いテキスト記述。
reserve	void	将来の利用のために予約されています。
注: [a]これはベンダー固有の戻りコードで、さまざまな API で戻り値として使用されるものとは異なります。		

言語構文

C 構造

```
typedef struct Return_code
{
  sqlint32  return_code,
  char      description[60],
  void      *reserve,
} Return_code;
```

ベンダー製品から受け入れられる有効な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。

ベンダー製品を使用してバックアップ / 復元を呼び出す

以下のインターフェースでは、パラメーターにより、バックアップおよび復元にベンダー製品を使用することを指定できます。

- コントロール・センターのバックアップおよび復元ツール
- コマンド行プロセッサー (CLP) の BACKUP および RESTORE コマンド
- バックアップおよび復元 API 関数呼び出し

コントロール・センター

コントロール・センターは、 DB2 と共に出荷されるデータベース管理用の GUI インターフェースです。コントロール・センターの呼び出しに関する情報は、 *コマンド解説書* に記載されています。

その使用法は、インターフェースと一緒に提供されるヘルプ・パネルで説明されます。コントロール・センターの一部であるバックアップおよび復元ツールについて理解するには、この情報をよく検討する必要があります。

次のパラメーターで、ベンダー装置サポートを使用することを指定します。

指定内容	コントロール・センターの入力変数 (バックアップと復元の両方)
ベンダー装置を使用することと、ライブラリー名	*Use Library* を選択し、ライブラリー名 (UNIX ベースのシステムの場合) または DLL 名 (OS/2 または Windows オペレーティング・システムの場合) を指定します。
セッションの数	*Sessions*
ベンダー・オプション	サポートされていません
ベンダー・ファイル名	サポートされていません
転送バッファー・サイズ	バックアップの場合: *Size of each Buffer*、復元の場合: 適用されません

コマンド行プロセッサー

コマンド行プロセッサー (CLP) は、DB2 と共に出荷される非 GUI ツールです。これはデータベース管理および他のタスクのために使用できます。
BACKUP DATABASE および RESTORE DATABASE CLP コマンドについては、 *コマンド解説書* に記載されています。

ベンダー装置サポートの指定は、次のパラメーターを使用して行います。

指定内容	コマンド行プロセッサーのパラメーター	
	バックアップの場合	復元の場合
ベンダー装置を使用することと、ライブラリー名	library-name	shared-library
セッションの数	num-sessions	num-sessions
ベンダー・オプション	サポートされていません	サポートされていません
ベンダー・ファイル名	サポートされていません	サポートされていません
転送バッファー・サイズ	buffer-size	buffer-size

バックアップおよび復元 API 関数呼び出し

バックアップと復元をサポートするために、バックアップ用の **sqlubkp** (345ページの『sqlubkp - データベースのバックアップ』 を参照) と復元用の **sqlurestore** (447ページの『sqlurestore - データベースの復元』を参照) の 2 つの API 関数呼び出しが用意されています。

これらの API 呼び出しの多くのパラメーターでは、データを呼び出し、ベンダー装置サポート関数に渡すことがサポートされます。

指定内容	API のパラメーター (sqlubkp と sqlurst の両方)
ベンダー装置を使用することと、ライブラリー名	構造 sqlu_media_list に SQLU_OTHER_MEDIA の媒体タイプを指定し、構造 sqlu_vendor に shr_lib に共用ライブラリーまたは DLL を指定します。
セッションの数	構造 sqlu_media_list にセッション数を指定します。
ベンダー・オプション	PVendorOptions
ベンダー・ファイル名	構造 sqlu_media_list に SQLU_OTHER_MEDIA の媒体タイプを指定し、構造 sqlu_vendor に filename を使用してファイル名を指定します。
転送バッファー・サイズ	BufferSize

付録E. 並行アクセスを伴うスレッド化アプリケーション

DB2 データベースに対するスレッド化アプリケーションの省略時のインプリメンテーションでは、データベースへのアクセスの逐次化はデータベース API によって実施されます。あるスレッドが、何らかの理由でブロックされている (つまり、表がすでに排他使用されている場合) データベース呼び出しを実行する場合、他のすべてのスレッドも同様にブロックされます。さらに、プロセス内のすべてのスレッドがコミット有効範囲を共用します。本当の意味でのデータベースへの並行アクセスは、独立したプロセスによって、あるいはこのセクションで説明されている API を用いることによってのみ達成できます。

このセクションでは、データベース API および組み込み SQL を使用するための独立した環境 (コンテキスト) を割り振り、操作するのに使用できる API について説明します。各コンテキストは独立したエンティティーであり、あるコンテキストを使用した接続は、それ以外のコンテキスト (したがって、プロセス内の他のすべての接続) から独立します。あるコンテキストで作業が行われるためには、まず、そのコンテキストがスレッドに関連付けられなければなりません。スレッドは、データベース API 呼び出しを行うとき、または組み込み SQL を使用するときには、必ずコンテキストを所有していなければなりません。コンテキストを操作するのに以下の API を使用していないと、プロセス内のすべてのスレッドが同じコンテキストを共用します。以下の API を使用すると、各スレッドが独自のコンテキストを所有することができます。各スレッドはデータベースまたはインスタンスにそれぞれ別個に接続しており、独自のコミット有効範囲を持ちます。

接続の期間中、コンテキストはある特定のスレッドに関連付けられていなくても構いません。あるスレッドがあるコンテキストに接続し、データベースに接続し、そのコンテキストから切り離されたら、2 番目のスレッドがそのコンテキストに接続し、既存のデータベース接続を用いて作業を継続することが可能です。コンテキストは 1 つのプロセス内のスレッド間で受け渡しすることはできますが、複数のプロセス間で受け渡しすることはできません。

新しい API を使用しない場合には、古い動作が有効であり、既存のアプリケーションを変更する必要はありません。

新しい API を使用する場合でも、以下の API は引き続き逐次化されます。

- sqlabndx - バインド

© Copyright IBM Corp. 1993, 2000

- sqlaprep - プログラムのプリコンパイル
- sqluexpr - エクスポート
- sqluimpr - インポート

新しい API は、組み込み SQL およびトランザクション API と一緒に使用することができます。

以下の API は、アプリケーションのスレッド化をサポートしないプラットフォームでは無効です (つまり、操作を行いません)。

注:

1. CLI は、自動的に新しいスキーマを使用します (着信接続ごとに新しいコンテキストを作成します)。これを明示的に使用不可にするかどうかはユーザーの選択によります。詳細については、*コール・レベル・インターフェースの手引きおよび解説書* を参照してください。
2. 省略時解釈では、AIX で 1 つのプロセスにつき許可される共用メモリー・セグメントの数は 10 以下です。したがって、1 プロセスあたりのローカル DB2 接続の数は 10 に制限されています。この制限に到達すると、DB2 は SQL CONNECT で SQLCODE -1224 を戻します。 DB2 コネクトを使用している場合で、ローカル・ユーザーが SNA を介して 2 フェーズ・コミットを実行している場合や、 TP モニター (SNA または TCP/IP) との 2 フェーズ・コミットを実行している場合にも、接続数は 10 に制限されます。

   AIX バージョン 4.2.1 以降では、環境変数 **EXTSHM** (=ON) を使用すれば、プロセスに接続する共用メモリー領域の数を増やすことができます。

   バージョン 4.2.1 よりも前の AIX では、オペレーティング・システム・ベースの解決策はありません。ローカル・データベースまたは DB2 コネクトを別のマシンに移動してそれにリモートからアクセスするか、 TCP/IP ループバックによってローカル・データベースまたは DB2 コネクト・データベースにアクセスできるようにするため、ローカル・マシンの TCP/IP アドレスを持つリモート・ノードとしてそのデータベースをカタログするのが代替の手段です。

sqleAttachToCtx - コンテキストへの接続

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接続されると、これらのスレッドについてはアクセスが逐次化され、これらのスレッドはコミット有効範囲を共用します。

効力範囲

この API の有効範囲は、即時プロセスに限定されます。

許可

ありません

必須接続

ありません

API 組み込みファイル

sql.h

C API 構文

```
int sqleAttachToCtx (
void           *pCtx,
void           *reserved,
struct sqlca   *pstSqlca);
```

API パラメーター

pCtx 入力。 674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接続』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。

reserved
将来の利用のために予約されています。ヌルに設定しなければなりません。

pstSqlca
出力。 *sqlca* 構造を指すポインター。この構造の詳細については、520ページの『SQLCA』を参照してください。

sqleBeginCtx - アプリケーション・コンテキストの作成および接続

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、それに接続します。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、別個のスレッドを接続できます (673ページの『sqleAttachToCtx - コンテキストへの接続』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、すべて相互に逐次化されます。

効力範囲

この API の有効範囲は、即時プロセスに限定されます。

許可

ありません

必須接続

ありません

API 組み込みファイル

sql.h

C API 構文

```
int sqleBeginCtx (
void           **ppCtx,
sqlint32       lOptions,
void           *reserved,
struct sqlca   *pstSqlca);
```

API パラメーター

ppCtx 出力。コンテキスト情報を格納するために私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、接続は行われません。

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 673ページの『sqleAttachToCtx - コンテキストへの接続』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* パラメーターをヌルにすることができます。スレッドがすでにコンテキストに接続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。ヌルに設定しなければなりません。

pstSqlca
出力。 *sqlca* 構造を指すポインター。この構造の詳細については、520ページの『SQLCA』を参照してください。

sqleDetachFromCtx - コンテキストからの切り離し

現行のスレッドによって使用されているコンテキストを切り離します。コンテキストが切り離されるのは、そのコンテキストに以前に接続していた場合に限ります。

効力範囲

この API の有効範囲は、即時プロセスに限定されます。

許可

ありません

必須接続

ありません

API 組み込みファイル

sql.h

C API 構文

```
int sqleDetachFromCtx (
void           *pCtx,
void           *reserved,
struct sqlca   *pstSqlca);
```

API パラメーター

pCtx 入力。 674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接続』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。

reserved
将来の利用のために予約されています。ヌルに設定しなければなりません。

pstSqlca
出力。 *sqlca* 構造を指すポインター。この構造の詳細については、520ページの『SQLCA』を参照してください。

sqleEndCtx - アプリケーション・コンテキストの切り離しおよび破棄

特定のコンテキストに関連付けられているすべてのメモリーを解放します。

効力範囲

この API の有効範囲は、即時プロセスに限定されます。

許可

ありません

必須接続

ありません

API 組み込みファイル

sql.h

C API 構文

```
int sqleEndCtx (
void           **ppCtx,
sqlint32       lOptions,
void           *reserved,
struct sqlca   *pstSqlca);
```

API パラメーター

ppCtx 出力。私用メモリー (コンテキスト情報を格納するために使用されている) 内で解放されるデータ域。

lOptions

入力。有効な値は以下のとおりです。

SQL_CTX_FREE_ONLY

コンテキスト・メモリーは、以前に切り離しが行われている場合にのみ解放されます。

注: *pCtx* は、 674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接続』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 676ページの『sqleDetachFromCtx - コンテキストからの切り離し』の呼び出しが行われます。

注: 切り離しは、コンテキストがまだ使用中である場合でも行われます。このオプションを使用する場合には、 *ppCtx* パラメーターをヌル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、 674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接続』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)。 679ページの『sqleGetCurrentCtx - 現行コンテキストの入手』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。ヌルに設定しなければなりません。

pstSqlca
出力。 *sqlca* 構造を指すポインター。この構造の詳細については、520ページの『SQLCA』を参照してください。

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 (たとえば、 327ページの『sqlfxdb - データベース構成の入手』) を呼び出す場合には、 **sqleEndCtx** を呼び出す前に、 229ページの『sqledtin - 切断』を用いてコンテキストをインスタンスから切り離すことが必要です。

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。

効力範囲

この API の有効範囲は、即時プロセスに限定されます。

許可

ありません

必須接続

ありません

API 組み込みファイル

sql.h

C API 構文

```
int sqleGetCurrentCtx (
void           **ppCtx,
void           *reserved,
struct sqlca   *pstSqlca);
```

API パラメーター

ppCtx 出力。コンテキスト情報を格納するために私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。ヌルに設定しなければなりません。

pstSqlca
出力。 *sqlca* 構造を指すポインター。この構造の詳細については、520ページの『SQLCA』を参照してください。

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。

効力範囲

この API の有効範囲は、即時プロセスに限定されます。

許可

ありません

必須接続

データベース

API 組み込みファイル

sql.h

C API 構文

```
int sqleInterruptCtx (
void           *pCtx,
void           *reserved,
struct sqlca   *pstSqlca);
```

API パラメーター

pCtx 入力。 674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接続』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。

reserved
将来の利用のために予約されています。ヌルに設定しなければなりません。

pstSqlca
出力。 *sqlca* 構造を指すポインター。この構造の詳細については、520ページの『SQLCA』を参照してください。

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。

効力範囲

この API の有効範囲は、即時プロセスに限定されます。

許可

ありません

必須接続

ありません

API 組み込みファイル

sql.h

C API 構文

```
int sqleSetTypeCtx (
sqlint32   lOptions);
```

API パラメーター

lOptions

入力。有効な値は以下のとおりです。

SQL_CTX_ORIGINAL

すべてのスレッドが同じコンテキストを使用し、並行アクセスはブロックされます。これは、どの API も呼び出されない場合の省略時値です。

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。

- 674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接続』
- 673ページの『sqleAttachToCtx - コンテキストへの接続』

- 676ページの『sqleDetachFromCtx - コンテキストからの切り離し』
- 677ページの『sqleEndCtx - アプリケーション・コンテキストの切り離しおよび破棄』

このオプションが使用されるときには、以下の制限 / 変更が適用されます。

- 正常終了の場合、プロセス終了時の自動 COMMIT は使用不可になります。未解決トランザクションはすべてロールバックされます。すべての COMMIT を明示的に行わなければなりません。
- 257ページの『sqleintr - 割り込み』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、680ページの『sqleInterruptCtx - コンテキストへの割り込み』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。

付録F. DB2 共通サーバー・ログ・レコード

このセクションでは、462ページの『sqlurlog - ログの非同期読み取り』によって戻される DB2 共通サーバー・ログ・レコードの構造を記述します。

すべての DB2 共通サーバー・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、688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、ログ順序番号 (LSN) によって固有に識別されます。 LSN は、データベース・ログにおける、ログ・レコードの最初のバイトに対応する相対バイト・アドレスを表します。データベース・ログの始まりからのログ・レコードのオフセットをマークします。

単一のトランザクションで書き込まれたログ・レコードは、ログ・レコード・ヘッダーにあるフィールドによって固有に識別することができます。固有なトランザクション識別子は、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、同じ識別子が含まれます。

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、トランザクション管理プログラムのログ・レコードを伝搬可能としてマークします。

表 94. DB2 共通サーバー・ログ・レコード

データ・マネージャー	
692ページの『表の初期設定』	新しい永続表の作成。
694ページの『インポート置換 (切り捨て)』	インポート置換活動。
694ページの『挿入のロールバック』	ロールバック行の挿入。
695ページの『表の再編成』	REORG のコミット。
695ページの『索引の作成、索引の除去』	索引活動。

© Copyright IBM Corp. 1993, 2000

表 94. DB2 共通サーバー・ログ・レコード (続き)

696ページの『表の作成、表の除去、表作成のロールバック、表除去のロールバック』	表活動。
696ページの『表変更属性』	伝搬、検査保留、および付加モード活動。
697ページの『表の更新による列の追加、列追加のロールバック』	既存の表への列の追加。
698ページの『レコードの挿入、レコードの削除、レコード削除のロールバック、レコード更新のロールバック』	表レコード活動。
702ページの『レコードの更新』	記憶場所が変更されない場合の行更新。
長フィールド・マネージャー	
704ページの『長フィールド・レコードの追加 / 削除 / 非更新』	長フィールド・レコード活動。
LOB マネージャー	
706ページの『LOB データの挿入ログ・レコード (AFIM_DATA)』	ロギングを伴う LOB データの追加。
706ページの『LOB データの挿入ログ・レコード (AFIM_AMOUNT)』	ロギングなしの LOB データの追加。
トランザクション管理プログラム	
707ページの『通常コミット』	トランザクションのコミット。
707ページの『ヒューリスティック・コミット』	未確定トランザクションのコミット。
708ページの『MPP 調整ノード・コミット』	トランザクションのコミット。これは、少なくとも 1 つの従属ノードで更新を実行するアプリケーションの調整ノードに書き込まれます。
708ページの『MPP 従属ノード・コミット』	トランザクションのコミット。これは、従属ノードに書き込まれます。
708ページの『通常打ち切り』	トランザクションの打ち切り。
709ページの『ヒューリスティック打ち切り』	未確定トランザクションの打ち切り。
709ページの『ローカル保留リスト』	保留リストが存在する場合のトランザクションのコミット。
710ページの『グローバル保留リスト』	保留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)。

表 94. DB2 共通サーバー・ログ・レコード (続き)

710ページの『XA 準備』	2 フェース・コミット環境での XA トランザクションの準備。
711ページの『MPP 従属ノード準備』	2 フェース・コミット環境での MPP トランザクションの準備。このログ・レコードは、従属ノードにのみ存在します。
712ページの『バックアウト解放』	バックアウト解放インターバル終了のマーク。バックアウト解放インターバルは、トランザクションが打ち切られたときに補正されていないログ・レコードのセットです。
ユーティリティー・マネージャー	
712ページの『移行の開始』	カタログ移行の開始。
713ページの『移行の終了』	カタログ移行の完了。
713ページの『ロードの開始』	表ロードの開始。
713ページの『表ロードの削除開始』	ロード削除フェーズの開始。
714ページの『ロード削除開始の補正』	ロード削除フェーズの終了。
714ページの『ロード保留リスト』	表ロードの完了。
714ページの『バックアップの終了』	バックアップ活動の完了。
714ページの『表スペースのロールフォーワード』	表スペース・ロールフォワードの完了。
715ページの『PIT への表スペース・ロールフォワードの開始』	特定の時点への表スペース・ロールフォワードの開始。
715ページの『PIT への表スペース・ロールフォワードの終了』	特定の時点への表スペース・ロールフォワードの終了。
データ・リンク・マネージャー	
716ページの『ファイルのリンク』	DATALINK 列を持つ表に対して挿入または更新が行われた結果、ファイルへのリンクが作成されたときに書き込まれます。
717ページの『ファイルのリンク解除』	DATALINK 列を持つ表に対して削除または更新が行われた結果、ファイルへのリンクが除去されたときに書き込まれます。
718ページの『グループの削除』	DATALINK 列を持つ表 (ファイルのリンク制御属性を持つ) が除去されたときに書き込まれます。
719ページの『p グループの削除』	表スペースが除去されたときに書き込まれます。

表 94. DB2 共通サーバー・ログ・レコード (続き)

719ページの『DLFM 準備』	DB2 データ・リンク・マネージャーに関係したトランザクションに 2 フェーズ・コミットが使用されている場合の準備段階で書き込まれます。

ログ・マネージャー・ヘッダー

すべての DB2 共通サーバー・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。

表 95. ログ・マネージャーのログ・レコード・ヘッダー

説明	種類	オフセット (バイト)
ログ・レコード全体の長さ	int	0(4)
ログ・レコードのタイプ[a]	short	4(2)
ログ・レコードの汎用フラグ[b]	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・レコードのログ順序番号。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[c]	8(6)
固有のトランザクション識別子	SQLU_TID[d]	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・レコードのログ順序番号。 (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・レコードのログ順序番号。 (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*		

表 95. ログ・マネージャーのログ・レコード・ヘッダー (続き)

説明	種類	オフセット (バイト)
定義および値		

[a] 有効なログ・レコード・タイプ

a データ・リンク・マネージャーのログ・レコード	o バックアップの開始
A 通常打ち切り	O バックアップの終了
B バックアウト解放	p PIT への表スペース・ロールフォワードの開始
c MPP 調整ノード・コミット	P 表の静止
C 補正	q PIT への表スペース・ロールフォワードの終了
D 表スペースのロールフォワード	Q グローバル保留リスト
E ローカル保留リスト	R 再実行
F トランザクションの忘却	s MPP 従属ノード・コミット
g MPP ログ同期	S 必要な補正
G ロード保留リスト	T 部分打ち切り
H 表ロードの削除開始	U やり直し
i 伝搬のみ	V 移行の開始
I ヒューリスティック打ち切り	W 移行の終了
J ロードの開始	X TM 準備
K ロード削除開始の補正	Y ヒューリスティック・コミット
L ロック記述	z MPP 準備
M 通常コミット	Z XA 準備
N 通常	

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、および破損回復時に、DB2 から無視されます。

[b] 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
条件付き回復可能	0x0080

[c] ログ順序番号 (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・レコード識別子。

```
SQLU_LSN: union {  char  [6] ;
                   short [3] ;
                 }
```

[d] トランザクション識別子 (TID)

トランザクションを示す固有なログ・レコード識別子。

```
SQLU_TID: union {  char  [6] ;
                   short [3] ;
                 }
```

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。ヘッダーの構成要素識別子が 1 になります。
- データ・オブジェクト・マネージャー (DOM) ログ。ヘッダーの構成要素識別子が 4 になります。

表 96. DMS ログ・レコード・ヘッダーの構造 (DMSLogRecordHeader)

説明	種類	オフセット (バイト)
構成要素識別子 (=1)	unsigned char	0(1)
機能識別子[a]	unsigned char	1(1)
表識別子 表スペース識別子 表識別子	unsigned short unsigned short	2(2) 4(2)
全長: 6 バイト		
値および定義 [a] 有効な機能識別子の値 102 表に列を追加する 104 列の追加を取り消す 106 レコードを削除する 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 列の長さを変更する 115 列の長さ変更を取り消す 118 レコードを挿入する 120 レコードを更新する 124 表属性を変更する 128 表を初期設定する		

表 97. DOM ログ・レコード・ヘッダーの構造 (DOMLogRecordHeader)

説明	種類	オフセット (バイト)
構成要素識別子 (=4)	unsigned char	0(1)
機能識別子[a]	unsigned char	1(1)

表 97. DOM ログ・レコード・ヘッダーの構造 (DOMLogRecordHeader) (続き)

説明	種類	オフセット (バイト)
オブジェクト識別子 表スペース識別子 オブジェクト識別子	unsigned short unsigned short	2(2) 4(2)
表識別子 表スペース識別子 表識別子	unsigned short unsigned short	6(2) 8(2)
オブジェクト・タイプ	unsigned char	10(1)
フラグ	unsigned char	11(1)
全長: 12 バイト		
値および定義 [a] 有効な機能識別子の値 2 索引を作成する 3 索引を除去する 4 表を除去する 11 表を切り捨てる (インポート置換) 35 表を再編成する 101 表を作成する 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。

機能識別子の省略名が UNDO で始まるすべてのログ・レコードは、当該アクションの UNDO または ROLLBACK 中に書き込まれたログ・レコードです。

ROLLBACK は、次の結果として起こる可能性があります。

- ユーザーが発行したトランザクションの ROLLBACK ステートメント
- 選択されたトランザクションの ROLLBACK を引き起こすデッドロック
- 破損の回復に続く、コミットされていないトランザクションの ROLLBACK
- ログの RESTORE および ROLLFORWARD に続く、コミットされていないトランザクションの ROLLBACK

表の初期設定

表の初期設定ログ・レコードは、新しい永続表が作成されるときに書き込まれ、表の初期設定を示します。このレコードは、DATA 記憶オブジェクト作成ログ・レコードの後、 LF および LOB 記憶オブジェクト作成ログ・レコードの前に入れられます。これは再実行ログ・レコードです。

表 98. 表の初期設定ログ・レコードの構造

説明	種類	オフセット (バイト)
ログ・ヘッダー	DMSLogRecordHeader	0(6)
ファイル作成 LSN	SQLU_LSN	6(6)
表ディレクトリー・レコード	variable	12(72)
レコード・タイプ	unsigned char	12(1)
予約済み	char	13(1)
索引フラグ	unsigned short	14(2)
索引ルート・ページ	sqluint32	16(4)
TDESC レコード ID	sqlint32	20(4)
予約済み	char	24(56)
フラグ[a]	sqluint32	80(4)
表記述の長さ		84(4)
表記述レコード	variable	88 (可変)
レコード・タイプ	unsigned char	88(1)
予約済み	char	89(1)
列の数	unsigned short	90(2)
配列	variable long	92 (可変)
全長: 88 バイト + 表記述レコードの長さ		
[a] ビット 0x00000020 は、表が NOT LOGGED INITIALLY オプションを指定して作成されたことと、表を作成したトランザクションがコミットされるまではこの表に関する DML 活動が記録されないことを示します。		

表 98. 表の初期設定ログ・レコードの構造 (続き)

説明	種類	オフセット (バイト)
表記述レコード: 列記述子配列 (列の数) * 8、配列の各要素の内容は次のとおりです。 • フィールド・タイプ (unsigned short、2 バイト) SMALLINT 0x0000 / INTEGER 0x0001 / DECIMAL 0x0002 / DOUBLE 0x0003 / REAL 0x0004 / BIGINT 0x0005 CHAR 0x0100 / VARCHAR 0x0101 / LONG VARCHAR 0x0104 / DATE 0x0105 / TIME 0x0106 / TIMESTAMP 0x0107 / BLOB 0x0108 / CLOB 0x0109 / DATALINK 0x010E GRAPHIC 0x0200 / VARGRAPH 0x0201 / LONG VARG 0x0202 / DBCLOB 0x0203 • 長さ (2 バイト) – BLOB、CLOB、または DBCLOB の場合、このフィールドは使用されません。このフィールドの最大長については、列記述子配列に続く配列を参照してください。 – DECIMAL でない場合、長さはフィールドの最大長 (short) になります。 – PACKED DECIMAL の場合、バイト 1 は unsigned char、精度 (全長)、バイト 2 は unsigned char、位取り (小数部の桁数) になります。 • ヌル・フラグ (unsigned short、2 バイト) – 相互に排他的: ヌルを許可するか、あるいはヌルを許可しない – 有効なオプション: 省略時値なし、省略時値入力、またはユーザー設定省略時値 ISNULL 0x01 NONULLS 0x02 TYPE_DEFAULT 0x04 USER_DEFAULT 0x08 • フィールド・オフセット (unsigned short、2 バイト)。これは、形式化レコードの始まりからフィールドの固定値が見つかる位置までのオフセットです。		

表 98. 表の初期設定ログ・レコードの構造 (続き)

説明	種類	オフセット (バイト)
表記述レコード: LOB 記述子配列 (LOB、CLOB、および DBCLOB フィールドの数) * 12、配列の各要素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要素を使用します。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 番目の要素を使用し、以下同様に続きます。		

インポート置換 (切り捨て)

インポート置換 (切り捨て) ログ・レコードは、 IMPORT REPLACE アクションが実行されるときに書き込まれます。このレコードは、表の再初期設定を示します (ユーザー・レコードを伴わず、新しい LSN を持ちます)。プールおよびオブジェクト ID の 2 番目のセットにより、切り捨て (IMPORT REPLACE) の対象である表が識別されます。これは再実行ログ・レコードです。

表 99. インポート置換 (切り捨て) ログ・レコードの構造

説明	種類	オフセット (バイト)
ログ・ヘッダー	DOMLogRecordHeader	0(12)
内部	variable	12 (可変)
全長: 12 バイト + 可変長部分		

挿入のロールバック

挿入のロールバック・ログ・レコードは、行の挿入アクション (INSERT RECORD) がロールバックされるときに書き込まれます。これは補正ログ・レコードです。

表 100. 挿入のロールバック・ログ・レコードの構造

説明	種類	オフセット (バイト)
ログ・ヘッダー	DMSLogRecordHeader	0(6)
埋め込み	char[ ]	6(2)
RID	sqlint32	8(4)

表 100. 挿入のロールバック・ログ・レコードの構造 (続き)

説明	種類	オフセット (バイト)
レコード長	unsigned short	12(2)
空きスペース	unsigned short	14(2)
全長: 16 バイト		

表の再編成

表の再編成ログ・レコードは、表の再編成を完了させるために REORG ユーティリティーがコミットされるときに書き込まれます。これは通常ログ・レコードです。

表 101. 表の再編成ログ・レコードの構造

説明	種類	オフセット (バイト)
ログ・ヘッダー	DOMLogRecordHeader	0(12)
内部	variable	12(252)
索引トークン[a]	unsigned short	2(264)
一時表スペース ID[b]	unsigned short	2(266)
全長: 268 バイト		
[a] 0 でない場合、これは再編成のクラスター化に使用された索引 (クラスター化索引) です。 [b] 0 でない場合、これは再編成の組み立てに使用されたシステム一時表スペースです。		

索引の作成、索引の除去

これらのログ・レコードは、索引が作成または除去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 つの要素があります。

- 索引ルート・ページ。これは内部識別子です。
- 索引トークン。これは、SYSIBM.SYSINDEXES 内の IID 列と同じです。この要素の値が 0 でない場合、ログ・レコードは内部索引に対するアクションを表しており、ユーザー索引とは関連していません。

これはやり直しログ・レコードです。

表 102. 索引の作成、索引の除去ログ・レコードの構造

説明	種類	オフセット (バイト)
ログ・ヘッダー	DOMLogRecordHeader	0(12)
埋め込み	char[ ]	12(2)

表 102. 索引の作成、索引の除去ログ・レコードの構造 (続き)

説明	種類	オフセット (バイト)
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、表の除去、表作成のロールバック、表除去のロールバック

これらのログ・レコードは、永続表の DATA オブジェクトが作成または除去されるときに作成されます。 DATA オブジェクトは、CREATE TABLE 中に、表の初期設定 (Initialize Table) に先立って作成されます。表の作成および表の除去は、通常ログ・レコードです。表作成のロールバックおよび表除去のロールバックは、補正ログ・レコードです。

表 103. 表の作成、表の除去、表作成のロールバック、表除去のロールバック・ログ・レコードの構造

説明	種類	オフセット (バイト)
ログ・ヘッダー	DOMLogRecordHeader	0(12)
内部	variable	12(56)
全長: 68 バイト		

表変更属性

表変更ログ・レコードは、表の状態が ALTER TABLE ステートメントを介して変更されたときや、制約の追加または妥当性検査の結果として変更されたときに書き込まれます。

表 104. 表変更属性、表変更の取り消し属性

説明	種類	オフセット (バイト)
ログ・ヘッダー	DMSLogRecordHeader	0(6)
埋め込み	char[ ]	6(2)
ビット変更 (属性) マスク	int	8(4)
ビット変更 (属性) 値	int	12(4)
全長: 16 バイト		

表 104. 表変更属性、表変更の取り消し属性 (続き)

説明	種類	オフセット (バイト)
属性ビット: 0x00000001 伝搬 0x00000002 検査保留 0x00010000 付加モード 0x00200000 LF 伝搬 0x00400000 LOB 伝搬 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。		

表の更新による列の追加、列追加のロールバック

表の更新による列の追加ログ・レコードは、ユーザーが ALTER TABLE ステートメントを用いて既存の表に列を追加するときに書き込まれます。以前の列と新しい列についての完全な情報が記録されます。

- 列カウント要素は、列の古い数と、列の新しい合計数を示します。
- 平行配列には、表で定義されている列に関する情報が入ります。古い平行配列では、ALTER TABLE ステートメントの前の表が定義され、新しい平行配列では、ALTER TABLE ステートメントの結果の表が定義されます。
- 各平行配列は、次のものから構成されます。
  - 表記述レコード内の列記述子配列と等しい配列 (692ページの『表の初期設定』を参照)。
  - 表記述レコード内の LOB 記述子配列と等しい 2 番目の配列。ただし、この配列は最初の配列に対する平行配列であるため、使用される要素は、最初の配列内の対応する要素がタイプ BLOB、CLOB、または DBCLOB であるものだけです。

表の更新による列の追加は通常ログ・レコードです。列追加のロールバックは補正ログ・レコードです。

表 105. 表の更新による列の追加、列追加のロールバック・ログ・レコードの構造

説明	種類	オフセット (バイト)
ログ・ヘッダー	DMSLogRecordheader	0(6)
埋め込み	char[ ]	6(2)
古い列カウント	int	8(4)
新しい列カウント	int	12(4)
古い平行配列[a]	variable	16 (可変)

表 105. 表の更新による列の追加、列追加のロールバック・ログ・レコードの構造 (続き)

説明	種類	オフセット (バイト)
新しい平行配列[b]	variable	variable
*全長: 40 バイト + 2 セットの平行配列。配列のサイズは、(古い / 新しい列カウント) * 20 です。*		
配列要素 <\p> [a] この配列内の各要素は 8 バイトです。 [b] この配列内の各要素は 12 バイトです。 列記述子配列または LOB 記述子配列の詳細については、 692ページの表98 を参照してください。		

レコードの挿入、レコードの削除、レコード削除のロールバック、レコード更新のロールバック

これらのログ・レコードは、表の行が挿入または削除されるときに書き込まれます。レコードの挿入およびレコードの削除ログ・レコードは、更新中に、修正されたレコード・データに合わせてレコードの位置を変更しなければならない場合に生成されます。レコードの挿入およびレコードの削除は、通常ログ・レコードです。レコード削除のロールバックおよびレコード更新のロールバックは、補正ログ・レコードです。

表 106. レコードの挿入、レコードの削除、レコード削除のロールバック、レコード更新のロールバック・ログ・レコードの構造

説明	種類	オフセット (バイト)
ログ・ヘッダー	DMSLogRecordHeader	0(6)
埋め込み	char[ ]	6(2)
RID	sqlint32	8(4)
レコード長	unsigned short	12(2)
空きスペース	unsigned short	14(2)
レコード・オフセット	unsigned short	16(2)
レコードのヘッダーとデータ	variable	18 (可変)
全長: 18 バイト + レコード長		

表 106. レコードの挿入、レコードの削除、レコード削除のロールバック、レコード更新のロールバック・ログ・レコードの構造 (続き)

説明	種類	オフセット (バイト)
レコード・ヘッダーおよびデータの詳細:		
レコード・ヘッダー 4 バイト • レコード・タイプ[a] (unsigned char、1 バイト)。レコードには、次の 2 つのクラスがあります。 – 更新可能 – 特殊制御 0 または 4 の値は、レコードが表示できることを示します。 各クラスには、次の 3 つのタイプがあります。 – 通常 – ポインター – オーバーフロー • 予約済み (char、1 バイト) • レコード長 (unsigned short、2 バイト)		
レコード variable • レコード・タイプ (unsigned char、1 バイト)。更新可能レコードには、次の 2 つのタイプがあります。 – 内部制御 – 形式化ユーザー・データ 1 の値は、形式化ユーザー・データ・レコードを示します。 • 予約済み (char、1 バイト) • レコードの残りの部分は、レコード・タイプと、表に関して定義されている表記述子レコードによって異なります。レコード・タイプが内部制御である場合には、データは表示できません。ユーザー・データ・レコードには、次のフィールドが適用されます。 – 固定長 (unsigned short、2 バイト)。これは、データ行の固定部分すべての長さです。 – 形式化レコード (固定長および可変長)。形式化レコードの詳細については、「形式化ユーザー・データ・レコード」を参照してください。		

[a] レコード・データは、レコード・タイプ (レコード・ヘッダー内で指定される) が更新可能 (つまり、特殊制御*ではない*) 場合にのみ表示できます。

形式化ユーザー・データ・レコード

形式化ユーザー・データ・レコードは、固定長データと可変長データの組み合わせにすることができます。すべてのフィールドには、固定長部分が含まれます。さらに、可変長部分を持つ 8 つのフィールド・タイプがあります。

- VARCHAR
- LONG VARCHAR
- DATALINK
- BLOB
- CLOB
- VARGRAPHIC
- LONG VARG
- DBCLOB

フィールド長

各フィールド・タイプの固定長部分の長さは、次のように判別できます。

- DECIMAL

  このフィールドは、*nnnnnn...s* の形式の標準のパック 10 進数です。フィールドの長さは、(精度 + 2)/2 です。符号ニブルは、正 (+) を表す xC と、負 (-) を表す xD または xB です。

- SMALLINT INTEGER BIGINT DOUBLE REAL CHAR GRAPHIC

  表記述子レコード内のこの列の要素にある長さフィールドは、フィールドの固定長サイズを含んでいます。

- DATE

  このフィールドは、*yyyymmdd* の形式の 4 バイトのパック 10 進数です。たとえば、1996 年 4 月 3 日は x'19960403' として表されます。

- TIME

  このフィールドは、*hhmmss* の形式の 3 バイトのパック 10 進数です。たとえば、1:32PM は x'133200' として表されます。

- TIMESTAMP

  このフィールドは、*yyyymmddhhmmssuuuuuu* (日付 | 時刻 | ミリ秒) の形式の 10 バイトのパック 10 進数です。

- VARCHAR LONG VARCHAR DATALINK BLOB CLOB VARGRAPHIC LONG VARG DBCLOB

  すべての可変長フィールドの固定長部分の長さは 4 です。

注: 要素アドレスについては、692ページの表98 を参照してください。

フィールド・タイプの詳細については、*SQL 解説書* を参照してください。

以下のセクションでは、形式化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

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、その要素はフィールド・タイプ、フィールド長、ヌル・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、形式化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 107. 表記述子レコードの構造

表記述子レコード			
レコード・タイプ	列の数	列の構造 • フィールド・タイプ • 長さ • ヌル・フラグ • フィールド・オフセット	LOB 情報
注: 詳細については、692ページの表98 を参照してください。			

ヌル可能 (ヌル・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)

ヌル可能の列について形式化レコード内のヌル・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。ヌル・フラグ値が 0x01 である場合、データ・フィールド値はヌルです。

形式化ユーザー・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 108. 形式化ユーザー・データ・レコードの構造

形式化ユーザー・データ・レコード			
レコード・タイプ	固定長セクションの長さ	固定長セクション	可変長データ・セクション
注: 詳細については、698ページの表106 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、列がヌル可能であればヌル・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。

レコードの更新

レコードの更新ログ・レコードは、行が更新され、その記憶場所が変わらない場合に書き込まれます。ログ・レコードの形式は 2 つあり、それらはレコードの挿入およびレコードの削除ログ・レコードと同じです (698ページの『レコードの挿入、レコードの削除、レコード削除のロールバック、レコード更新のロールバック』を参照)。一方は、更新される行の更新*前* のイメージを含み、もう一方は更新される行の更新*後* のイメージを含みます。これは通常ログ・レコードです。

表 109. レコードの更新ログ・レコードの構造

説明	**種類**	**オフセット (バイト)**
ログ・ヘッダー	DMSLogRecordHeader	0(6)
埋め込み	char[ ]	6(2)
RID	sqlint32	8(4)
新しいレコード長	unsigned short	12(2)
空きスペース	unsigned short	14(2)
レコード・オフセット	unsigned short	16(2)
古いレコードのヘッダーとデータ	variable	18 (可変)
ログ・ヘッダー	DMSLogRecordHeader	可変 (6)
埋め込み	char[ ]	可変 (2)
RID	sqlint32	可変 (4)
古いレコード長	unsigned short	可変 (2)
空きスペース	unsigned short	可変 (2)

表 109. レコードの更新ログ・レコードの構造 (続き)

説明	種類	オフセット (バイト)
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: 36 バイト + 2 つのレコード長		

長フィールド・マネージャーのログ・レコード

長フィールド・マネージャーのログ・レコードが書き込まれるのは、 LOG RETAIN がオンであるかまたは USEREXITS が使用可能の状態でデータベースが構成されている場合のみです。これらのログ・レコードは、長フィールド・データが挿入、削除、または更新されるたびに書き込まれます。

ログ・スペースを節約するために、データベースが循環ロ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 列を取り込むように表が変更された (ALTER TABLE で INCLUDE LONGVAR COLUMNS が指定された) 場合、

- 長フィールド・マネージャーは適切な長フィールド・ログ・レコードを書き込みます。
- 長フィールド・データが更新されるときには、その更新は、古い長フィールド値の削除と、新しい値の挿入として扱われます。
- 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 110. 長フィールド・マネージャーのログ・レコード・ヘッダー (LongFieldLogRecordHeader)

説明	種類	オフセット (バイト)
発信元コード (構成要素識別子 = 3)	unsigned char	0(1)
操作タイプ[a]	unsigned char	1(1)
プール識別子	unsigned short	2(2)
オブジェクト識別子	unsigned short	4(2)
親プール識別子[b]	unsigned short	6(2)
親オブジェクト識別子[c]	unsigned short	8(2)
全長: 10 バイト		

[a] 有効な操作値のタイプと定義は以下のとおりです。

操作タイプ値	長フィールド・ログ・レコード・タイプ
110	長フィールド・レコードの追加
111	長フィールド・レコードの削除
112	長フィールド・レコードの非更新

[b] データ・オブジェクトのプール ID

[c] データ・オブジェクトのオブジェクト ID

長フィールド・レコードの追加 / 削除 / 非更新

これらのログ・レコードは、長フィールド・データが挿入、削除、または更新されるたびに書き込まれます。データの長さは、次の 512 バイト境界に切り上げられます。

表 111. 長フィールド・レコードの追加 / 削除/ 非更新ログ・レコードの構造

説明	種類	オフセット (バイト)
ログ・ヘッダー	LongFieldLogRecordHeader	0(10)
長フィールドの長さ[a]	unsigned short	10(2)
ファイル・オフセット[b]	sqluint32	12(4)
長フィールド・データ	char[ ]	16 (可変)

[a] 長フィールド・データの長さは、 512 バイト単位です (実際のデータ長は記録されません)。このフィールドは常に正の値です。ゼロの長さの長フィールドが挿入、削除、または更新された場合は、長フィールド・マネージャーはログ・レコードを作成しません。

[b] 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。

LOB マネージャーのログ・レコード

LOB マネージャーのログ・レコードが書き込まれるのは、 LOG RETAIN がオンであるかまたは USEREXITS が使用可能の状態でデータベースが構成されている場合のみです。このログ・レコードは、LOB データが表に挿入されるたびに書き込まれます。 LOB データが更新されるときには、その更新は、古い LOB 値の削除と、新しい値の挿入として扱われます。 LOB マネージャーが、その新しい値が古い値に新しいデータを付加しただけのものであると判別できる場合には、新しいデータが古いデータに付加されます。この場合、新しいデータだけが記録されます。

NOT LOGGED オプションを指定して作成された LOB 列の場合でも、データベースが順方向回復可能であれば、ログ・レコードが書き込まれます。ただし、実際のデータが記録されるわけではなく、データの量と LOB オブジェクトにおけるその位置だけが記録されます。順方向回復時に、LOB オブジェクトにはゼロ (ユーザー・データではなく) が書き込まれます。

挿入された LOB 値について、複数の LOB レコードが書き込まれる可能性があります。 1 つの LOB レコードには、32 768 バイトよりも多くのデータは含まれません。

ログ・スペースを節約するために、データベースが循環ログ用に構成されている場合には、表に挿入された LOB データは記録されません。さらに、LOB 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。

LOB マネージャーのすべてのログ・レコードは、ログ・レコード・ヘッダーで始まります。

LOB 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。

表 112. LOB マネージャーのログ・レコード・ヘッダーの構造

説明	種類	オフセット (バイト)
発信元コード (構成要素識別子 = 5)	unsigned char	0(1)
操作識別子	unsigned char	1(1)
プール識別子	unsigned short	2(2)
オブジェクト識別子	unsigned short	4(2)
親プール識別子	unsigned short	6(2)
親オブジェクト識別子	unsigned short	8(2)

表 112. LOB マネージャーのログ・レコード・ヘッダーの構造 (続き)

説明	種類	オフセット (バイト)
オブジェクト・タイプ	unsigned char	10(1)
全長: 11 バイト		

LOB データの挿入ログ・レコード (AFIM_DATA)

このログ・レコードは、データのロギングが指定されている場合に、 LOB データが LOB 列に挿入されるか、または既存の LOB 値に付加されるときに書き込まれます。

表 113. LOB データの挿入ログ・レコード (AFIM_DATA)

説明	種類	オフセット (バイト)
ログ・ヘッダー	LOBLogRecordHeader	0(11)
埋め込み	char	11(1)
データ長	sqluint32	12(4)
オブジェクト内のバイト・アドレス	double	16(8)
LOB データ	variable	24 (可変)
全長: 24 バイト + LOB データ		

LOB データの挿入ログ・レコード (AFIM_AMOUNT)

このログ・レコードは、LOB 列のロギングがオフにされている場合に、AFIM_DATA ログ・レコードの代わりに書き込まれます。

表 114. LOB データの挿入ログ・レコード (AFIM_AMOUNT)

説明	種類	オフセット (バイト)
ログ・ヘッダー	LOBLogRecordHeader	0(11)
埋め込み	char	11(1)
データ長	sqluint32	12(4)
オブジェクト内のバイト・アドレス	double	16(8)
全長: 24 バイト		

トランザクション管理プログラムのログ・レコード

トランザクション管理プログラムは、トランザクション・イベント (たとえば、コミットまたはロールバック) の完了を示すログ・レコードを生成します。ログ・レコード内のタイム・スタンプは、協定世界時 (CUT) であり、1970 年 1 月 1 日から経過した時間 (秒単位) を示します。

通常コミット

このログ・レコードは、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 にある調整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、以下のいずれかのイベントの後で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。

- ユーザーが COMMIT を発行した
- CONNECT RESET 中に暗黙的なコミットが起こった

表 115. 通常コミット・ログ・レコードの構造

説明	種類	オフセット (バイト)
ログ・ヘッダー	LogManagerLogRecordHeader	0(20)
トランザクションがコミットされた時刻	sqluint32	20(4)
アプリケーションの許可 ID[a]	char[ ]	24 (可変)
全長: 24 バイト + 伝搬可能な変数 (伝搬不可能な 24 バイト)		
[a] ログ・レコードが伝搬可能としてマークされている場合		

ヒューリスティック・コミット

このログ・レコードは、未確定トランザクションがコミットされるときに書き込まれます。

表 116. ヒューリスティック・コミット・ログ・レコードの構造

説明	種類	オフセット (バイト)
ログ・ヘッダー	LogManagerLogRecordHeader	0(20)
トランザクションがコミットされた時刻	sqluint32	20(4)
アプリケーションの許可 ID[a]	char[ ]	24 (可変)
全長: 24 バイト + 伝搬可能な変数 (伝搬不可能な 24 バイト)		
[a] ログ・レコードが伝搬可能としてマークされている場合		

MPP 調整ノード・コミット

このログ・レコードは、少なくとも 1 つの従属ノードで更新を実行するアプリケーションについて調整ノードに書き込まれます。

表 117. ヒューリスティック MPP 調整ノード・コミット・ログ・レコードの構造

説明	種類	オフセット (バイト)
ログ・ヘッダー	LogManagerLogRecordHeader	0(20)
トランザクションがコミットされた時刻	sqluint32	20(4)
トランザクションの MPP 識別子	SQLP_GXID	24(20)
最大ノード番号	unsigned short	44(2)
TNL	unsigned char [ ]	46(最大ノード番号/8 + 1)
アプリケーションの許可 ID[a]	char[ ]	可変 (可変)
全長: 変数		
[a] ログ・レコードが伝搬可能としてマークされている場合		

MPP 従属ノード・コミット

このログ・レコードは、MPP 内の従属ノードで書き込まれます。

表 118. MPP 従属ノード・コミット・ログ・レコードの構造

説明	種類	オフセット (バイト)
ログ・ヘッダー	LogManagerLogRecordHeader	0(20)
トランザクションがコミットされた時刻	sqluint32	20(4)
トランザクションの MPP 識別子	SQLP_GXID	24(20)
許可識別子[a]	char[ ]	44(可変)
全長: 44 バイト + 伝搬可能な変数 (伝搬不可能な 44 バイト)		
[a] ログ・レコードが伝搬可能としてマークされている場合		

通常打ち切り

このログ・レコードは、以下のいずれかのイベントの後でトランザクションが打ち切られるときに書き込まれます。

- ユーザーが ROLLBACK を発行した
- デッドロックが発生した
- 破損の回復中に暗黙的なロールバックが起こった
- ROLLFORWARD 回復中に暗黙的なロールバックが起こった

表 119. 通常打ち切りログ・レコードの構造

説明	種類	オフセット (バイト)
ログ・ヘッダー	LogManagerLogRecordHeader	0(20)
アプリケーションの許可 ID[a]	char[ ]	20(可変)
全長: 20 バイト + 変数 (伝搬不可能な 20 バイト)		
[a] ログ・レコードが伝搬可能としてマークされている場合		

ヒューリスティック打ち切り

このログ・レコードは、未確定トランザクションが打ち切られるときに書き込まれます。

表 120. ヒューリスティック打ち切りログ・レコードの構造

説明	種類	オフセット (バイト)
ログ・ヘッダー	LogManagerLogRecordHeader	0(20)
アプリケーションの許可 ID[a]	char[ ]	20(可変)
全長: 20 バイト + 変数 (伝搬不可能な 20 バイト)		
[a] ログ・レコードが伝搬可能としてマークされている場合		

ローカル保留リスト

このログ・レコードは、トランザクションがコミットされるときに、保留リストが存在する場合に書き込まれます。保留リストは、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回復不能操作 (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、保留リストの項目が含まれます。

表 121. ローカル保留リスト・ログ・レコードの構造

説明	種類	オフセット (バイト)
ログ・ヘッダー	LogManagerLogRecordHeader	0(20)
トランザクションがコミットされた時刻	sqluint32	20(4)
許可 ID の長さ[a]	unsigned short	24(2)
アプリケーションの許可 ID[a]	char[ ]	26(可変)[b]
保留リストの項目	variable	可変 (可変)
全長: 26 バイト + 伝搬可能な変数 (24 バイト + 伝搬不可能な保留リスト項目)		
[a] ログ・レコードが伝搬可能としてマークされている場合 [b] 許可 ID の長さに基づく変数		

グローバル保留リスト

このログ・レコードは、2 フェーズ・コミットに必要なトランザクションがコミットされるときに、保留リストが存在する場合に書き込まれます。保留リストには、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回復不能操作 (ファイルの削除など) が含まれます。この可変長構造には、保留リストの項目が含まれます。

表 122. グローバル保留リスト・ログ・レコードの構造

説明	種類	オフセット (バイト)
ログ・ヘッダー	LogManagerLogRecordHeader	0(20)
許可 ID の長さ[a]	unsigned short	20(2)
アプリケーションの許可 ID[a]	char[ ]	22(可変)[b]
グローバル保留リストの項目	variable	可変 (可変)
全長: 22 バイト + 伝搬可能な変数 (20 バイト + 伝搬不可能な保留リスト項目)		
[a] ログ・レコードが伝搬可能としてマークされている場合 [b] 許可 ID の長さに基づく変数		

XA 準備

このログ・レコードは、単一ノード環境にある XA トランザクションについて、あるいは MPP にある調整ノードに書き込まれます。これは、XA アプリケーション専用です。このログ・レコードは、トランザクションの準備を 2 フェーズ・コミットの一部としてマークするために書き込まれます。 XA 準備ログ・レコードは、トランザクションを開始したアプリケーションを記述し、未確定トランザクションの再作成に使用されます。

表 123. XA 準備ログ・レコードの構造

説明	種類	オフセット (バイト)
ログ・ヘッダー	LogManagerLogRecordHeader	0(20)
トランザクションが準備された時刻	sqluint32	20(4)
トランザクションによって使用されたログ・スペース	sqluint64	24(8)
トランザクション・ノード・リストのサイズ	sqluint32	32(4)
トランザクション・ノード・リスト	unsigned char [ ]	36(可変)
トランザクションの XA 識別子	SQLXA_XID	可変 (140)
アプリケーション情報の長さ	sqluint32	可変 (4)

表 123. XA 準備ログ・レコードの構造 (続き)

説明	種類	オフセット (バイト)
コード・ページ識別子	sqluint32	可変 (4)
トランザクションの開始時刻	sqluint32	可変 (4)
アプリケーション名	char[ ]	可変 (20)
アプリケーション識別子	char[ ]	可変 (32)
順序番号	char[ ]	可変 (4)
クライアントによって使用されたデータベース別名	char[ ]	240(20)
許可識別子	char[ ]	可変 (可変)
同期ログ情報	variable	可変 (可変)
全長: 264 バイト + 変数		

MPP 従属ノード準備

このログ・レコードは、従属ノード上の MPP トランザクションについて書き込まれます。このログ・レコードは、トランザクションの準備を 2 フェーズ・コミットの一部としてマークするために書き込まれます。 MPP 従属ノード準備ログ・レコードは、トランザクションを開始したアプリケーションを記述し、未確定トランザクションの再作成に使用されます。

表 124. MPP 従属ノード準備ログ・レコードの構造

説明	種類	オフセット (バイト)
ログ・ヘッダー	LogManagerLogRecordHeader	0(20)
トランザクションが準備された時刻	sqluint32	20(4)
トランザクションによって使用されたログ・スペース	sqluint64	24(8)
調整ノード LSN	SQLP_LSN	32(6)
埋め込み	char[ ]	38(2)
トランザクションの MPP 識別子	SQLP_GXID	40(20)
アプリケーション情報の長さ	sqluint32	60(4)
コード・ページ	sqluint32	64(4)
トランザクションの開始時刻	sqluint32	68(4)
アプリケーション名	char[ ]	72(20)
アプリケーション識別子	char[ ]	92(32)
順序番号	char[ ]	124(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[ ]	128(20)

表 124. MPP 従属ノード準備ログ・レコードの構造 (続き)

説明	種類	オフセット (バイト)
許可識別子	char[ ]	148(可変)
全長: 148 バイト + 変数		

バックアウト解放

このログ・レコードは、バックアウト解放インターバルの終了をマークするために使用されます。バックアウト解放インターバルは、トランザクションが打ち切られたときに補正されていないログ・レコードのセットです。このログ・レコードには、6 バイトのログ順序番号 (*complsn*、オフセット 20 から始まるログ・レコード・ヘッダーに格納される) だけが含まれます。このログ・レコードを (打ち切られたトランザクションの後の) ロールバック時に読み取ると、 *complsn* が次に補正すべきログ・レコードをマークします。

表 125. 移行の開始ログ・レコードの構造

説明	種類	オフセット (バイト)
ログ・ヘッダー	LogManagerLogRecordHeader	0(20)
Complsn	SQLP_LSN	20(6)
全長: 26 バイト		

ユーティリティー・マネージャーのログ・レコード

ユーティリティー・マネージャーは、次の DB2 共通サーバー・ユーティリティーに関連するログ・レコードを生成します。

- 移行
- ロード
- バックアップ
- 表スペースのロールフォワード

ログ・レコードは、要求された活動の始まりと終わりを示します。ユーティリティー・マネージャーのすべてのログ・レコードは、それらが影響を与える表に関係なく、伝搬可能としてマークされます。

移行の開始

このログ・レコードは、カタログ移行の開始と関連しています。

表 126. 移行の開始ログ・レコードの構造

説明	種類	オフセット (バイト)
ログ・ヘッダー	LogManagerLogRecordHeader	0(20)

表 126. 移行の開始ログ・レコードの構造 (続き)

説明	種類	オフセット (バイト)
移行の開始時刻	char[ ]	20(10)
移行元のリリース	unsigned short	30(2)
移行先のリリース	unsigned short	32(2)
全長: 34 バイト		

移行の終了

このログ・レコードは、カタログ移行の正常終了と関連しています。

表 127. 移行の終了ログ・レコードの構造

説明	種類	オフセット (バイト)
ログ・ヘッダー	LogManagerLogRecordHeader	0(20)
移行の終了時刻	char[ ]	20(10)
移行先のリリース	unsigned short	30(2)
全長: 32 バイト		

ロードの開始

このログ・レコードは、ロードの開始と関連しています。

表 128. ロードの開始ログ・レコードの構造

説明	種類	オフセット (バイト)
ログ・ヘッダー	LogManagerLogRecordHeader	0(20)
ログ・レコード識別子	sqluint32	20(4)
プール識別子	unsigned short	24(2)
オブジェクト識別子	unsigned short	26(2)
フラグ	unsigned char	28(1)
オブジェクト・プールのリスト	variable	29 (可変)
全長: 29 バイト + 可変長部分		

表ロードの削除開始

このログ・レコードは、ロード操作における削除フェーズの開始と関連しています。削除フェーズが開始されるのは、1 次キー値が重複して存在する場合だけです。

表 129. 表ロードの削除開始ログ・レコードの構造

説明	種類	オフセット (バイト)
ログ・ヘッダー	LogManagerLogRecordHeader	0(20)
全長: 20 バイト		

ロード削除開始の補正

このログ・レコードは、ロード操作における削除フェーズの終了と関連しています。

表 130. ロード削除開始の補正ログ・レコードの構造

説明	種類	オフセット (バイト)
ログ・ヘッダー	LogManagerLogRecordHeader	0(20)
全長: 20 バイト		

ロード保留リスト

このログ・レコードは、ロード・トランザクションのコミット時に書き込まれます。保留リストは、トランザクションがコミットするまで据え置かれる回復不能操作のリンク・リストです。このトランザクションの後には、コミット・ログ・レコードは書き込まれません。

表 131. ロード保留リスト・ログ・レコードの構造

説明	種類	オフセット (バイト)
ログ・ヘッダー	LogManagerLogRecordHeader	0(20)
トランザクションがコミットされた時刻	sqluint32	20(4)
アプリケーションの許可 ID[a]	char[ ]	24(9)
保留リストの項目	variable	33 (可変)
全長: 33 バイト + 伝搬可能な保留リスト項目 (24 バイト + 伝搬不可能な保留リスト項目)		
[a] ログ・レコードが伝搬可能としてマークされている場合		

バックアップの終了

このログ・レコードは、バックアップの正常終了と関連しています。

表 132. バックアップの終了ログ・レコードの構造

説明	種類	オフセット (バイト)
ログ・ヘッダー	LogManagerLogRecordHeader	0(20)
バックアップの終了時刻	sqluint32	20(4)
全長: 24 バイト		

表スペースのロールフォーワード

このログ・レコードは、表スペースの ROLLFORWARD 回復と関連しています。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 133. 表スペースのロールフォーワード・ログ・レコードの構造

説明	種類	オフセット (バイト)
ログ・ヘッダー	LogManagerLogRecordHeader	0(20)
表スペース識別子	unsigned short	20(2)
全長: 22 バイト		

PIT への表スペース・ロールフォワードの開始

このログ・レコードは、表スペースの ROLLFORWARD 回復と関連しています。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 134. PIT への表スペース・ロールフォワードの開始ログ・レコードの構造

説明	種類	オフセット (バイト)
このログ・レコードのタイム・スタンプ	sqluint32	0(4)
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	4(4)
ロールフォワードされるプールの数	unsigned short	8(2)
ロールフォワードされるプール ID の整数リスト	int*numpools	10 (可変)
全長: 10 バイト + 可変長部分		

PIT への表スペース・ロールフォワードの終了

このログ・レコードは、表スペースの ROLLFORWARD 回復と関連しています。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 135. PIT への表スペース・ロールフォワードの終了ログ・レコードの構造

説明	種類	オフセット (バイト)
このログ・レコードのタイム・スタンプ	sqluint32	0(4)
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	4(4)

表 135. PIT への表スペース・ロールフォワードの終了ログ・レコードの構造 (続き)

説明	種類	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	int	8(4)
全長: 12 バイト		

データ・リンク・マネージャーのログ・レコード

データ・リンク・マネージャーのログ・レコードは、DDL、DML、または DATALINK 列に関係したトランザクション・イベント完了の結果です。これらのログ・レコードが書き込まれるのは、 DLL または DML がファイルのリンク制御属性を持つ DATALINK 列に関連している場合だけです。

表 136. データ・リンク・マネージャーのログ・レコード・ヘッダーの構造 (DLMLogRecordHeader)

説明	種類	オフセット (バイト)
構成要素識別子 (=8)	unsigned char	0(1)
機能識別子[a]	unsigned char	1(1)
埋め込み	char[ ]	2(2)
全長: 6 バイト		
定義および値 [a] 有効な機能識別子の値 LINK_FILE 33 ファイルのリンク UNLINK_FILE 34 ファイルのリンク解除 DELETE_GROUP 35 グループの削除 DELETE_PGROUP 36 p グループの削除 DLFM_PREPARE 37 DLFM 準備		

ファイルのリンク

ファイルのリンク・ログ・レコードは、DATALINK 列を持つ表に対して挿入または更新が行われた結果、ファイルへのリンクが作成されたときに書き込まれます。新しいリンクが 1 つ作成されるごとに、ログ・レコードが 1 つずつ書き込まれます。このログ・レコードは、やり直しのためにのみ使用されます。

表 137. ファイルのリンク・ログ・レコードの構造

説明	種類	オフセット (バイト)
ログ・ヘッダー	DLMLogRecordHeader	0(4)
サーバー ID	sqlint32	4(4)
読み取り専用	int	8(4)
許可 ID	char[ ]	12(8)
グループ ID	char[ ]	20(17)
埋め込み	char[ ]	37(1)
アクセス制御	unsigned short	38(2)
接頭部 ID	char[ ]	40(9)
埋め込み	char[ ]	49(3)
回復 ID	char[ ]	52(7)
埋め込み	char[ ]	59(1)
タイム・スタンプ	sqluint32	60(4)
語幹名の長さ	sqluint32	64(4)
語幹名	variable	68(可変)
サーバー名の長さ[a]	sqluint32	可変 (4)
接頭部名の長さ[a]	sqluint32	可変 (4)
サーバー名の接頭部名[a]	variable	可変 (可変)
全長: 伝搬不可能の場合、 68 + 語幹名の長さ (伝搬可能の場合、 76 + 語幹名の長さ + サーバー名の長さ + 接頭部名の長さ)		
注: [a] ログ・レコードが伝搬可能な場合。		

ファイルのリンク解除

ファイルのリンク解除ログ・レコードは、DATALINK 列を持つ表に対して削除または更新が行われた結果、ファイルへのリンクが除去されたときに書き込まれます。リンクが 1 つ除去されるごとに、ログ・レコードが 1 つずつ書き込まれます。このログ・レコードは、やり直しのためにのみ使用されます。

表 138. ファイルのリンク解除ログ・レコードの構造

説明	種類	オフセット (バイト)
ログ・ヘッダー	DLMLogRecordHeader	0(4)
サーバー ID	sqlint32	4(4)
接頭部 ID	char[ ]	8(9)
埋め込み	char[ ]	17(3)

表 138. ファイルのリンク解除ログ・レコードの構造 (続き)

説明	種類	オフセット (バイト)
回復 ID	char[ ]	20(7)
埋め込み	char[ ]	27(1)
タイム・スタンプ	sqluint32	28(4)
語幹名の長さ	sqluint32	32(4)
語幹名	variable	36(可変)
プール ID[a]	unsigned short	可変 (2)
オブジェクト ID[a]	unsigned short	可変 (2)
列番号[a]	unsigned short	可変 (2)
埋め込み[a]	char[ ]	可変 (2)
サーバー名の長さ[a]	sqluint32	可変 (4)
接頭部名の長さ[a]	sqluint32	可変 (4)
サーバー名の接頭部名[a]	variable	可変 (可変)
全長: 伝搬不可能の場合、 36 + 語幹名の長さ (伝搬可能の場合、 52 + 語幹名の長さ + サーバー名の長さ + 接頭部名の長さ)		
注: [a] ログ・レコードが伝搬可能な場合。		

グループの削除

グループの削除ログ・レコードは、DATALINK 列を持つ表 (ファイルのリンク制御属性を持つ) が除去されたときに書き込まれます。データベースに DB2 データ・リンク・マネージャーが構成されるごとに、そのような DATALINK 列についてログ・レコードが 1 つずつ書き込まれます。ある DB2 データ・リンク・マネージャーに関係してログ・レコードが書き込まれるのは、表が除去されたときに、その DB2 データ・リンク・マネージャーにグループが定義されていた場合です。このログ・レコードは、やり直しのためにのみ使用されます。

表 139. グループの削除ログ・レコードの構造

説明	種類	オフセット (バイト)
ログ・ヘッダー	DLMLogRecordHeader	0(4)
サーバー ID	sqlint32	4(4)
回復 ID	char[ ]	8(7)
埋め込み	char[ ]	15(1)
グループ ID	char[ ]	16(17)

表 139. グループの削除ログ・レコードの構造 (続き)

説明	種類	オフセット (バイト)
埋め込み	char[ ]	33(3)
全長: 36 バイト		

p グループの削除

p グループの削除ログ・レコードは、表スペースが除去されるときに書き込まれます。データベースに DB2 データ・リンク・マネージャーが 1 つ構成されるごとに、ログ・レコードが 1 つずつ書き込まれます。ある DB2 データ・リンク・マネージャーに関係してログ・レコードが書き込まれるのは、表スペースが除去されたときに、その DB2 データ・リンク・マネージャーに p グループが定義されていた場合です。このログ・レコードは、やり直しのためにのみ使用されます。

表 140. p グループの削除ログ・レコードの構造

説明	種類	オフセット (バイト)
ログ・ヘッダー	DLMLogRecordHeader	0(4)
サーバー ID	sqlint32	4(4)
poolLifeLSN	SQLU_LSN	8(6)
プール ID	unsigned short	14(2)
回復 ID	char[ ]	16(7)
埋め込み	char[ ]	23(1)
全長: 24 バイト		

DLFM 準備

DLFM 準備ログ・レコードは、DB2 データ・リンク・マネージャーに関係したトランザクションに 2 フェーズ・コミットが使用されている場合の準備段階で書き込まれます。 DLFM 準備ログ・レコードは、未確定な DB2 データ・リンク・マネージャーに関係したトランザクションを再作成するために使用されます。

表 141. DLFM 準備ログ・レコードの構造

説明	種類	オフセット (バイト)
ログ・ヘッダー	DLMLogRecordHeader	0(4)
DLFM の数	unsigned short	4(4)
サーバー ID	variable	8(可変)
*全長: 8 バイト + (DLFM の数 * 4)*		

アプリケーション移行時の考慮事項

このセクションでは、アプリケーションをバージョン 7 に移行する際に考慮しなければならない問題について説明します。

可能な操作のシナリオが 4 つあります。

1. 移行されていないデータベースに対してバージョン 7 以前のアプリケーションを実行する
2. 移行されたデータベースに対してバージョン 7 以前のアプリケーションを実行する
3. バージョン 7 の API でアプリケーションを更新する
4. 移行されたデータベースに対してバージョン 7 のアプリケーションを実行する

1 番目と 4 番目は、修正を必要としない、一貫した操作環境です。

データベースのみが移行されている 2 番目の操作環境は、アプリケーションに変更を加えなくても機能するはずです (バック・レベルのアプリケーションがサポートされているため)。ただし、新しいバージョンについては、若干の非互換性が生じる可能性があります。それについては、*管理の手引き* で説明されています。

アプリケーションがバージョン 7 の API で更新される 3 番目のシナリオでは、以下の点を考慮しなければなりません。

- バージョン 7 で廃止されているバージョン 7 以前のすべての API は、バージョン 7 のヘッダー・ファイルで依然として定義されています。それにより、古いアプリケーションがバージョン 7 のヘッダーを用いてコンパイルおよびリンクすることができます。
- 廃止された API はできるだけ早くアプリケーションから除去して、アプリケーションがバージョン 7 で利用可能な新しい関数を十分に活用できるようにし、将来の拡張に備えておくべきです。
- 以下にリストされている API の名前は、バージョン 7 の新機能のために変更されています。ユーザーはアプリケーションのソース・コード内でこれらの名前を走査して、バージョン 7 への移行に続いて必要になる変更を識別することが必要です。

  ここにリストされていない API には、アプリケーションの移行に続く変更は必要ありません。

アプリケーションには、使用中のアプリケーション・プログラミング言語によって、 API 呼び出しの汎用バージョンが含まれることがあります。すべてのケースで、汎用バージョンの API 名は、 4 番目の文字が常に **g** である点を除き、 C バージョンの名前と同一です。

変更された API およびデータ構造

表 142. バック・レベル・サポートが存在する API

API (バージョン)	記述名	新しい API (バージョン)
sqlbftsq (V2)	表スペース照会の取り出し	sqlbftpq (V5)
sqlbstsq (V2)	単一表スペースの照会	sqlbstpq (V5)
sqlbtsq (V2)	表スペースの照会	sqlbmtsq (V5)
sqlectdd (V2)	データベースのカタログ	sqlecadb (V5)
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)	sqlepstart (V5)
sqlestar (V2)	データベース・マネージャーの開始 (DB2 バージョン 2)	sqlepstart (V5)
sqlestop (V2)	データベース・マネージャーの開始	sqlepstp (V5)
sqlerstd (V5)	データベースの再始動	db2DatabaseRestart (V6)
sqlmon (V6)	モニター・スイッチの入手 / 更新	db2MonitorSwitches (V7)
sqlmonss (V5)	スナップショットの入手	db2GetSnapshot (V6)
sqlmonsz (V6)	sqlmonss() 出力バッファーに必要なサイズの見積もり	db2GetSnapshotSize (V7)
sqlmrset (V6)	モニターのリセット	db2ResetMonitorData (V7)
sqlubkup (V2)	データベースのバックアップ	sqlubkp (V5)
sqlugrpi (V2)	行の区分化情報の入手 (DB2 パラレル・エディション バージョン 1.x)	sqlugrpn (V5)
sqluhcls (V5)	回復活動記録ファイルの走査のクローズ	db2HistoryCloseScan (V6)
sqluhget (V5)	活動記録ファイルからの DDL 情報の検索	db2HistoryGetEntry (V6)
sqluhgne (V5)	回復活動記録ファイルの次項目の入手	db2HistoryGetEntry (V6)
sqluhops (V5)	回復活動記録ファイルの走査のオープン	db2HistoryOpenScan (V6)
sqluhprn (V5)	回復活動記録ファイルの枝取り	db2Prune (V6)
sqluhupd (V5)	回復活動記録ファイルの更新	db2HistoryUpdate (V6)
sqluqry (V5)	ロードの照会	db2LoadQuery (V6)
sqlursto (V2)	RESTORE DATABASE	sqlurst (V5)
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)

表 142. バック・レベル・サポートが存在する API (続き)

API (バージョン)	記述名	新しい API (バージョン)
SQLB-TBSQRY-DATA (V2)	表スペース・データ構造	SQLB-TBSPQRY-DATA (V5)
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)	SQLE-START-OPTIONS (V5)
SQLUHINFO and SQLUHADM (V5)	活動記録ファイルのデータ構造	db2HistData (V6)

表 143. バック・レベル・サポートが存在しない API

名前	記述名	V7 でサポートされている API
sqlufrol/sqlgfrol	データベースのロールフォワード (DB2 バージョン 1.1)	sqluroll
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)	sqluroll
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)	sqluroll
sqlurllf/sqlgrfwd	データベースのロールフォワード (DB2 バージョン 2)	sqluroll

DB2 ライブラリーの使用法

DB2 ユニバーサル・データベース ライブラリーは、オンライン・ヘルプ、ブック (PDF および HTML)、および HTML 形式のサンプル・プログラムから成っています。このセクションでは、ユーザーに提供される情報について紹介し、その入手方法を示します。

オンライン製品情報をご利用になるには、インフォメーション・センターを使用することができます。詳細については、738ページの『インフォメーション・センターを使用した情報へのアクセス』を参照してください。ここではタスク情報、DB2 ブック、トラブルシューティング情報、サンプル・プログラム、および Web の DB2 情報を見ることができます。

DB2 PDF ファイルおよびハードコピー版資料

DB2 情報

以下に示す表では、DB2 ブックを 4 つのカテゴリーに分類しています。

DB2 の手引きおよび解説書
これらの資料は、すべてのプラットフォームに共通の DB2 情報を含んでいます。

DB2 のインストールおよび構成の情報
これらの資料は、特定のプラットフォーム上の DB2 ごとに用意されています。たとえば、 OS/2、Windows、および UNIX ベースのプラットフォームで稼働するそれぞれの DB2 用に、別個の*概説およびインストール* 資料が用意されています。

プラットフォーム共通のサンプル・プログラム (HTML 形式)
これらのサンプルは、アプリケーション開発クライアントとともにインストールされるサンプル・プログラムの HTML 版です。これらのサンプルは参考用であり、実際のプログラムに代わるものではありません。

リリース情報
これらのファイルには、DB2 ブックには含められなかった最新の情報が記載されています。

インストール情報、リリース情報、およびチュートリアルは、製品 CD-ROM から HTML 形式で参照することができます。ほとんどの資料は、製品 CD-ROM から HTML 形式で表示できますし、 DB2 の資料 CD-ROM から Adobe Acrobat (PDF) 形式で表示し印刷することができます。 IBM にハードコピー版の資料を注文したい場合は、 734ページの『印刷資料の注文方法』を参照してください。注文可能な資料については、以下の表をご覧ください。

OS/2 および Windows プラットフォームの場合、 HTML ファイルは `sqllib¥doc¥html` ディレクトリーにインストールできます。 DB2 情報はいくつかの言語で提供されています。しかし、すべての言語に翻訳されているわけではありません。ある言語で情報が提供されていない場合は、英語版の情報が提供されます。

UNIX プラットフォームの場合、言語ごとに異なる複数の HTML ファイルを `doc/%L/html` ディレクトリーにインストールできます。ここで、*%L* は地域を表しています。詳細については、適切な*概説およびインストールの手引き* を参照してください。

DB2 ブックを入手して情報を利用するには、次のようなさまざまな方法があります。

- 737ページの『オンライン情報の表示』
- 742ページの『オンライン情報の検索』
- 734ページの『印刷資料の注文方法』

- 734ページの『PDF 資料の印刷』

表 144. DB2 情報

資料名	説明	資料番号 PDF ファイル名	HTML ディレクトリー
	DB2 の手引きおよび解説書情報		
管理の手引き	*管理の手引き: 計画* は、データベース概念について概説し、設計 (たとえば、論理および物理データベース設計) に関する情報を提供し、高い可用性について解説しています。 *管理の手引き: インプリメンテーション* は、設計、データベースへのアクセス、監査、バックアップ、および回復などのインプリメンテーションについて説明しています。 *管理の手引き: パフォーマンス* は、データベース環境について解説し、さらにアプリケーションのパフォーマンスの評価と調整の方法について説明しています。	第 1 巻 SC88-8513 db2d1x70 第 2 巻 SC88-8511 db2d2x70 第 3 巻 SC88-8512 db2d3x70	db2d0
管理 API 解説書	データベースの管理に使用できる DB2 アプリケーション・プログラミング・インターフェース (API) およびデータ構造について説明します。また、この資料は、アプリケーションから API を呼び出す方法も示します。	SC88-8514 db2b0x70	db2b0
アプリケーション構築の手引き	環境設定に関する情報を提供し、Windows、OS/2、および UNIX ベースのプラットフォームでの DB2 アプリケーションのコンパイル、リンク、実行の各ステップについて説明します。	SC88-8515 db2axx70	db2ax
APPC, CPI-C, and SNA Sense Codes	DB2 ユニバーサル・データベース製品をご使用中に発生する可能性のあるセンス・コード APPC、CPI-C、および SNA についての一般情報を提供します。 HTML 形式でのみご利用いただけます。	資料番号なし db2apx70	db2ap

表 144. DB2 情報 (続き)

資料名	説明	資料番号 PDF ファイル名	HTML ディレクトリー
アプリケーション開発の手引き	DB2 データベースにアクセスするアプリケーションを、組み込み SQL または Java (JDBC および SQLJ) を使用して開発する方法について説明します。さらに、ストアード・プロシージャーの作成方法、ユーザー定義関数の作成方法、ユーザー定義タイプの作成方法、トリガーの使用方法、区画化されている環境または統合されているシステムでのアプリケーションの開発方法などについて解説されています。	SC88-8516 db2a0x70	db2a0
コール・レベル・インターフェースの手引きおよび解説書	DB2 データベースにアクセスするアプリケーションを、 DB2 コール・レベル・インターフェース (Microsoft ODBC 仕様互換の呼び出し可能 SQL) を使用して開発する方法について説明します。	SC88-8517 db2l0x70	db2l0
コマンド解説書	コマンド行プロセッサーの使用法について説明し、データベースの管理に使用できる DB2 コマンドについて解説しています。	SC88-8518 db2n0x70	db2n0
コネクティビティー 補足	DB2 (AS/400 版)、DB2 (OS/390 版)、DB2 (MVS 版)、または DB2 (VM 版) を DRDA アプリケーション・リクエスターとして DB2 ユニバーサル・データベースとともに使用するためのセットアップ情報および参照情報を提供します。また、この資料は DRDA アプリケーション・サーバーを DB2 コネクト アプリケーション・リクエスターとともに使用する方法の詳細を示します。 HTML と PDF でのみ利用可能	資料番号なし db2h1x70	db2h1
データ移動ユーティリティー 手引きおよび解説書	データの移動を行う DB2 ユーティリティー (インポート、エクスポート、ロード、AutoLoader、および DPROP など) の使用法について説明しています。	SC88-8522 db2dmx70	db2dm

表 144. DB2 情報 (続き)

資料名	説明	資料番号 PDF ファイル名	HTML ディレクトリー
データウェアハウスセンター 管理の手引き	データウェアハウスセンターを使用してデータウェアハウスを構築および保守する方法を説明します。	SC88-8545 db2ddx70	db2dd
データウェアハウスセンター アプリケーション統合の手引き	プログラマーがアプリケーションをデータウェアハウスセンターおよび情報カタログ・マネージャーと統合するのに役立つ情報を提供します。	SC88-8546 db2adx70	db2ad
DB2 コネクト 使用者の手引き	DB2 コネクト製品の概念、プログラミング、および一般的な使用方法に関する情報を提供します。	SC88-8521 db2c0x70	db2c0
DB2 クエリー・パトローラー 管理の手引き	DB2 クエリー・パトローラー・システムの運用の概説を行い、運用および管理に関する詳細情報、および管理用グラフィカル・ユーザー・インターフェース・ユーティリティーについてのタスク情報を提供します。	SC88-8525 db2dwx70	db2dw
DB2 クエリー・パトローラー 使用者の手引き	DB2 クエリー・パトローラーのツールや関数の使用方法を説明します。	SC88-8527 db2wwx70	db2ww
用語集	DB2 およびその構成要素で使用される用語の定義を示します。 HTML 形式と *SQL 解説書* で利用可能	資料番号なし db2t0x70	db2t0
イメージ、オーディオ、およびビデオ・エクステンダー 管理およびプログラミングの手引き	DB2 エクステンダーの一般情報について提供し、画像、音声、およびビデオ (IAV) エクステンダーの管理と構成について、および IAV エクステンダーを使用したプログラミングについて説明しています。さらに、参照情報、診断情報 (メッセージ解説)、およびサンプルも収録されています。	SC88-8609 dmbu7x70	dmbu7
情報カタログ・マネージャー 管理の手引き	情報カタログを管理するためのガイドです。	SC88-8547 db2dix70	db2di
情報カタログ・マネージャー プログラミングの手引きおよび解説書	情報カタログ・マネージャー用の体系化されたインターフェースの定義を示します。	SC88-8549 db2bix70	db2bi

表 144. DB2 情報 (続き)

資料名	説明	資料番号 PDF ファイル名	HTML ディレクトリー
情報カタログ・マネージャー 使用者の手引き	情報カタログ・マネージャー・ユーザー・インターフェースの使用に関する情報を提供します。	SC88-8548 db2aix70	db2ai
インストールおよび構成補足	プラットフォーム固有の DB2 クライアントの計画、インストール、およびセットアップのガイドです。この補足資料には、バインド、クライアント / サーバー通信の設定、 DB2 GUI ツール、DRDA AS、分散インストール、分散要求の構成、および異種データ・ソースへのアクセスについても説明されています。	GC88-8524 db2iyx70	db2iy
メッセージ解説書	DB2、情報カタログ・マネージャー、およびデータウェアハウスセンターから出されるメッセージとコードをリストし、取るべき処置を解説しています。	第 1 巻 GC88-8543 db2m1x70 第 2 巻 GC88-8544 db2m2x70	db2m0
OLAP Integration Server Administration Guide	OLAP Integration Server の Administration Manager 構成要素の使用方法を説明します。	SC27-0782 db2dpx70	n/a
OLAP Integration Server Metaoutline User's Guide	標準の OLAP Metaoutline インターフェースを使用して (Metaoutline Assistant を使用するのではなく) OLAP metaoutline を作成しデータを取り込む方法を説明しています。	SC27-0784 db2upx70	n/a
OLAP Integration Server Model User's Guide	(Model Assistant ではなく) 標準的な OLAP Model Interface を使用して OLAP モデルを作成する方法を説明します。	SC27-0783 db2lpx70	n/a
OLAP Setup and User's Guide	OLAP Starter Kit の構成およびセットアップに関する情報を提供します。	SC27-0702 db2ipx70	db2ip
OLAP Spreadsheet Add-in User's Guide for Excel	Excel 作表計算プログラムを使用して OLAP データを分析する方法を説明します。	SC27-0786 db2epx70	db2ep

表 144. DB2 情報 (続き)

資料名	説明	資料番号 PDF ファイル名	HTML ディレクトリー
OLAP Spreadsheet Add-in User's Guide for Lotus 1-2-3	ロータス 1-2-3 作表計算プログラムを使用して OLAP データを分析する方法を説明します。	SC27-0785 db2tpx70	db2tp
レプリケーションの手引きおよび解説書	DB2 に付属の IBM レプリケーション・ツールの計画、構成、管理、および使用方法に関する情報を提供します。	SC88-8550 db2e0x70	db2e0
地理情報エクステンダー使用者の手引きおよび解説書	地理情報エクステンダーのインストール、構成、管理、プログラミング、およびトラブルシューティングに関する情報を提供します。また、地理情報データの概念についての重要事項を示し、地理情報エクステンダー固有の参照情報 (メッセージおよび SQL) を提供します。	SC88-8624 db2sbx70	db2sb
SQL 概説	SQL の概念を紹介し、構造体とタスクの例を多数提供しています。	SC88-8539 db2y0x70	db2y0
SQL 解説書	SQL の構文、セマンティクス、および言語規則について説明します。また、この資料には、各リリース間の互換性、製品の制限事項、およびカタログ・ビューも含まれます。	第 1 巻 SC88-8540 db2s1x70 第 2 巻 SC88-8657 db2s2x70	db2s0
システム・モニター 手引きおよび解説書	データベースおよびデータベース・マネージャーに関連したさまざまな情報を収集する方法を示します。この資料は、この情報を利用して、データベース活動の把握、パフォーマンス向上、および問題原因の判別を行う方法を説明しています。	SC88-8523 db2f0x70	db2f0

表 144. DB2 情報 (続き)

資料名	説明	資料番号 PDF ファイル名	HTML ディレクトリー
テキスト・エクステンダー *管理およびプログラミング*	DB2 エクステンダーの一般情報、テキスト・エクステンダーの管理および構成情報、およびテキスト・エクステンダーを使用したプログラミングの方法について解説します。この資料には、参照情報、診断情報 (メッセージ解説)、およびサンプルが含まれています。	SC88-8610 desu9x70	desu9
問題判別の手引き	エラーの原因の判別、問題からの回復、および DB2 カスタマー・サービスの支援の下での診断ツールの使用法を記載しています。	GD88-7271 db2p0x70	db2p0
新機能	DB2 ユニバーサル・データベース バージョン 7 の新しい機能および拡張機能について説明します。	SC88-8541 db2q0x70	db2q0
DB2 のインストールおよび構成の情報			
DB2 コネクト エンタープライズ・エディション (OS/2 および Windows 版) 概説およびインストール	OS/2 および Windows 32 ビット・オペレーティング・システム版の DB2 コネクト エンタープライズ・エディションで、計画、移行、インストール、および構成を行う場合の情報を提供します。また、この資料はサポートされている多数のクライアントのインストールおよびセットアップについても説明します。	GC88-8520 db2c6x70	db2c6
DB2 コネクト エンタープライズ・エディション (UNIX 版) 概説およびインストール	UNIX ベースのプラットフォームでの DB2 コネクト エンタープライズ・エディションの計画、移行、インストール、構成、およびタスクに関する情報を提供します。また、この資料はサポートされている多数のクライアントのインストールおよびセットアップについても説明します。	GC88-8519 db2cyx70	db2cy

表 144. DB2 情報 (続き)

資料名	説明	資料番号 PDF ファイル名	HTML ディレクトリー
DB2 コネクト パーソナル・エディション 概説およびインストール	OS/2 および Windows 32 ビット・オペレーティング・システムの DB2 コネクト パーソナル・エディションで、計画、移行、インストール、および構成を行う場合のタスク情報を提供します。また、この資料はサポートされているすべてのクライアントのインストールおよびセットアップについても説明します。	GC88-8533 db2c1x70	db2c1
DB2 コネクト パーソナル・エディション (Linux 版) 概説およびインストール	サポートされる Linux 配布プログラムの DB2 コネクト パーソナル・エディションで、計画、インストール、移行、および構成を行う場合の情報を提供します。	GC88-8528 db2c4x70	db2c4
DB2 データ・リンク・マネージャー (Windows 版) 概説およびインストール	AIX および Windows 32 ビット・オペレーティング・システムの DB2 データ・リンク・マネージャーで、計画、インストール、構成を行う場合の情報を提供します。	GC88-8532 db2z6x70	db2z6
DB2 エンタープライズ拡張エディション (UNIX 版) 概説およびインストール	UNIX ベースのプラットフォームでの DB2 エンタープライズ拡張エディションの計画、インストール、および構成に関する情報を提供します。また、この資料はサポートされている多数のクライアントのインストールおよびセットアップについても説明します。	GC88-8530 db2v3x70	db2v3
DB2 エンタープライズ拡張エディション (Windows 版) 概説およびインストール	Windows 32 ビット・オペレーティング・システムの DB2 エンタープライズ拡張エディションで、計画、インストール、および構成を行う場合の情報を提供します。また、この資料はサポートされている多数のクライアントのインストールおよびセットアップについても説明します。	GC88-8529 db2v6x70	db2v6

表 144. DB2 情報 (続き)

資料名	説明	資料番号 PDF ファイル名	HTML ディレクトリー
DB2 ユニバーサル・データベース (OS/2 版) 概説およびインストール	OS/2 オペレーティング・システムでの DB2 ユニバーサル・データベースの計画、インストール、移行、および構成に関する情報を提供します。また、この資料はサポートされている多数のクライアントのインストールおよびセットアップについても説明します。	GC88-8534 db2i2x70	db2i2
DB2 ユニバーサル・データベース (UNIX 版) 概説およびインストール	UNIX ベースのプラットフォームでの DB2 ユニバーサル・データベースの計画、インストール、移行、および構成に関する情報を提供します。 また、この資料はサポートされている多数のクライアントのインストールおよびセットアップについても説明します。	GC88-8536 db2ixx70	db2ix
DB2 ユニバーサル・データベース (Windows 版) 概説およびインストール	Windows 32 ビット・オペレーティング・システムの DB2 ユニバーサル・データベースで、計画、インストール、移行、および構成を行う場合の情報を提供します。また、この資料はサポートされている多数のクライアントのインストールおよびセットアップについても説明します。	GC88-8537 db2i6x70	db2i6
DB2 パーソナル・エディション 概説およびインストール	OS/2 および Windows 32 ビット・オペレーティング・システム版の DB2 ユニバーサル・データベース パーソナル・エディションで、計画、インストール、移行、および構成を行う場合の情報を提供します。	GC88-8535 db2i1x70	db2i1
DB2 パーソナル・エディション (Linux 版) 概説およびインストール	サポートされる Linux 配布プログラムの DB2 ユニバーサル・データベース・パーソナル・エディションで、計画、インストール、移行、および構成を行う場合の情報を提供します。	GC88-8538 db2i4x70	db2i4
DB2 クエリー・パトローラー インストールの手引き	DB2 クエリー・パトローラーのインストール情報を提供します。	GC88-8526 db2iwx70	db2iw

表 144. DB2 情報 (続き)

資料名	説明	資料番号 PDF ファイル名	HTML ディレクトリー
ウェアハウス・マネージャー *インストールの手引き*	ウェアハウス・エージェント、ウェアハウス・トランスフォーマー、および情報カタログ・マネージャーのインストール情報を提供します。	GC88-8572 db2idx70	db2id
プラットフォーム共通のサンプル・プログラム (HTML 形式)			
サンプル・プログラム (HTML)	DB2 のサポートするすべてのプラットフォームでのプログラム言語用に、サンプル・プログラム (HTML 形式) を提供します。これらのサンプル・プログラムは、参照用としてのみ提供されています。サンプルは、すべてのプログラミング言語で利用できるわけではありません。HTML サンプルが利用できるのは、 DB2 アプリケーション開発クライアントがインストールされている場合だけです。 プログラムの詳細については、*アプリケーション構築の手引き* を参照してください。	資料番号なし	db2hs
リリース情報			
DB2 コネクト リリース情報	DB2 コネクトの資料には含められなかった最新の情報が収録されています。	注 #2 を参照してください。	db2cr
DB2 インストール情報	DB2 ブックには含められなかったインストールに関する最新の情報が収録されています。	製品 CD-ROM からのみ利用できます。	
DB2 リリース情報	DB2 ブックには含められなかった DB2 製品とその機能に関する最新の情報が収録されています。	注 #2 を参照してください。	db2ir

注:

1. ファイル名の 6 桁目の文字 *x* は、その資料の言語を表します。たとえば、ファイル名 `db2d0e70` は、*管理の手引き* の英語版であることを示し、ファイル名 `db2d0f70` は同じ資料のフランス語版を示します。資料の言語を表すためにファイル名の 6 桁目で使用されている文字は以下のとおりです。

言語	識別子
ブラジル・ポルトガル語	b
ブルガリア語	u
チェコ語	x
デンマーク語	d
オランダ語	q
英語	e
フィンランド語	y
フランス語	f
ドイツ語	g
ギリシャ語	a
ハンガリー語	h
イタリア語	i
日本語	j
韓国語	k
ノルウェー語	n
ポーランド語	p
ポルトガル語	v
ロシア語	r
簡体字中国語	c
スロベニア語	l
スペイン語	z
スウェーデン語	s
繁体字中国語	t
トルコ語	m

2. DB2 ブックには含められなかった最新の情報が、「リリース情報」で HTML 形式および ASCII ファイルとして利用できます。 HTML 版は、インフォメーション・センターおよび製品 CD-ROM からご利用になれます。 ASCII ファイルの参照方法:
   - UNIX ベースのプラットフォームでは、ファイル `Release.Notes` を参照してください。このファイルは `DB2DIR/Readme/%L` ディレクトリーにあります。ここで *%L* は地域名を、 `DB2DIR` は以下のものを表します。
     - `/usr/lpp/db2_07_01` (AIX の場合)
     - `/opt/IBMdb2/V7.1` (HP-UX、DYNIX/ptx、Solaris、および Silicon Graphics IRIX の場合)
     - `/usr/IBMdb2/V7.1` (Linux の場合)
   - これ以外のプラットフォームでは、ファイル `RELEASE.TXT` を参照してください。このファイルは、製品がインストールされているディレクトリーにあります。 OS/2 プラットフォームでは、 **IBM DB2** フォルダーをダブルクリックし、 **Release Notes** アイコンをダブルクリックすることもできます。

PDF 資料の印刷

資料のハードコピー版が必要な場合、 DB2 の資料 CD-ROM にある PDF ファイルを印刷することができます。 Adobe Acrobat Reader を使用すれば、資料全体または特定のページを印刷することができます。ライブラリー内の各資料のファイルについては、724ページの表144 を参照してください。

Adobe Acrobat Reader の最新版は、Adobe の Web サイト http://www.adobe.com から入手できます。

PDF ファイルは、DB2 の資料 CD-ROM に収録されており、ファイル拡張子 PDF が付いています。 PDF ファイルにアクセスするには以下のようにします。

1. DB2 の資料 CD-ROM を挿入します。 UNIX ベースのプラットフォームの場合は、 DB2 資料 CD-ROM をマウントします。マウントの手順については、*概説およびインストール* を参照してください。
2. Acrobat Reader を起動します。
3. 以下に示すいずれかの位置から必要な PDF ファイルを開きます。
   - OS/2 および Windows プラットフォームでは:

     x:¥doc¥*language* ディレクトリー。ここで、*x* は CD-ROM ドライブを、*language* は 2 桁の言語を表す国コード (たとえば、EN は英語) を示します。
   - UNIX ベースのプラットフォームでは:

     CD-ROM の */cdrom*/doc/*%L* ディレクトリー。ここで、*/cdrom* は CD-ROM のマウント・ポイントを、 *%L* は地域名を表します。

さらに、PDF ファイルを CD-ROM からローカル・ドライブまたはネットワーク・ドライブにコピーし、そこから参照することもできます。

印刷資料の注文方法

ハードコピー版の DB2 ブックは、個別に注文することができます。資料を注文するには、IBM 承認の販売業者または営業担当員に連絡してください。

DB2 オンライン文書

オンライン・ヘルプへのアクセス

すべての DB2 構成要素で、オンライン・ヘルプを利用できます。以下の表に、さまざまな種類のヘルプを示します。

ヘルプの種類	内容	利用方法
コマンド・ヘルプ	コマンド行プロセッサーのコマンド構文について説明します。	コマンド行プロセッサーの対話モードから、次のように入力します。 `? command` ここで *command* はキーワードまたはコマンド全体を表します。 たとえば、`? catalog` と入力すると、すべての CATALOG コマンドに関するヘルプが表示され、`? catalog database` と入力すると、 CATALOG DATABASE コマンドのヘルプが表示されます。
クライアント構成アシスタントのヘルプ コマンド・センターのヘルプ コントロール・センターのヘルプ データウェアハウスセンターのヘルプ イベント・アナライザーのヘルプ 情報カタログ・マネージャーのヘルプ サテライト管理センターのヘルプ スクリプト・センターのヘルプ	そのウィンドウまたはノートブックで実行できるタスクについて説明します。このヘルプは、知っておく必要のある概説および前提条件に関する情報を含みます。また、ウィンドウやノートブックの制御の使用方法を示します。	ウィンドウまたはノートブックから、「**ヘルプ (Help)**」押しボタンをクリックするか、または **F1** キーを押します。

ヘルプの種類	内容	利用方法

メッセージ・ヘルプ	メッセージの原因、および取るべき処置を説明します。	コマンド行プロセッサーの対話モードから、次のように入力します。
		`? XXXnnnnn`
		ここで、*XXXnnnnn* は有効なメッセージ識別子を表します。
		たとえば、`? SQL30081` と入力すると、メッセージ SQL30081 に関するヘルプを表示します。
		一度に 1 画面分のメッセージ・ヘルプを表示させるには、次のように入力します。
		`? XXXnnnnn | more`
		メッセージ・ヘルプをファイルに保管するには、次のように入力します。
		`? XXXnnnnn > filename.ext`
		ここで、*filename.ext* はメッセージ・ヘルプを保管するファイルを表します。

SQL ヘルプ	SQL ステートメントの構文について説明します。	コマンド行プロセッサーの対話モードから、次のように入力します。 `help statement` ここで、*statement* は SQL ステートメントを表します。 たとえば、`help SELECT` と入力すると、 SELECT ステートメントのヘルプが表示されます。 **注:** UNIX ベースのプラットフォームでは、SQL ヘルプを利用できません。
SQLSTATE ヘルプ	SQL 状態およびクラス・コードについて説明します。	コマンド行プロセッサーの対話モードから、次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。 たとえば、`? 08003` によって SQL 状態 08003 のヘルプが表示され、 `? 08` によってクラス・コード 08 のヘルプが表示されます。

オンライン情報の表示

この製品に付属のブックは、ハイパーテキスト・マークアップ言語 (HTML) ソフトコピー形式です。ソフトコピー形式では情報を検索または表示したり、ハイパーテキスト・リンクを利用して関連情報に移動したりすることができます。また、1 つの端末を超えてライブラリーを容易に共用することができます。

オンライン・ブックやサンプル・プログラムは、 HTML バージョン 3.2 仕様に準拠するすべてのブラウザーを使って表示できます。

オンライン・ブックまたはサンプル・プログラムは、次のようにして表示します。

- DB2 管理ツールを実行している場合、インフォメーション・センターを使用します。
- ブラウザーで、**ファイル (File) —>ページを開く (Open Page)** をクリックします。次のようなページを開いて、DB2 情報に関する説明とリンクを表示してください。
  - UNIX ベースのプラットフォームでは、以下のページを開きます。

    INSTHOME/sqllib/doc/*%L*/html/index.htm

    ここで *%L* はロケール名です。
  - その他のプラットフォームでは、以下のページを開きます。

    sqllib¥doc¥html¥index.htm

    パスは DB2 がインストールされているドライブです。

    インフォメーション・センターをインストールしていない場合、 **DB2 Information** アイコンをダブルクリックしてページを開くことができます。このアイコンは、ご使用のシステムに応じて、製品のメイン・フォルダー内または Windows 「スタート」メニューにあります。

Netscape ブラウザーのインストール: システムに Web ブラウザーがインストールされていない場合、製品の箱の中にある Netscape CD-ROM から Netscape をインストールすることができます。インストールに関する詳細な説明については、以下を参照してください。

1. Netscape CD-ROM を挿入します。
2. UNIX ベースのプラットフォームでは、CD-ROM をマウントします。マウントの手順については、*概説およびインストール* を参照してください。

3. インストールの手順については、 CDNAV*nn*.txt ファイルを参照します。ここで、*nn* は 2 桁の言語識別子を表します。ファイルは CD-ROM のルート・ディレクトリーにあります。

インフォメーション・センターを使用した情報へのアクセス: インフォメーション・センターを使用すると、DB2 製品情報にすばやくアクセスすることができます。インフォメーション・センターは、DB2 管理ツールを使用できるすべてのプラットフォームで利用できます。

インフォメーション・センターは「インフォメーション・センター (Information Center)」アイコンをダブルクリックすることによってオープンできます。このアイコンのある場所はシステムによって異なります。メイン・プロダクト・フォルダーか Windows の**「スタート」**メニューのどちらかです。

Windows プラットフォームの DB2 では、ツールバーおよび**ヘルプ**・メニューを使用して、インフォメーション・センターにアクセスすることもできます。

インフォメーション・センターは 6 種類の情報を提供します。適切なタブをクリックすると、種類ごとに提供されているトピックが表示されます。

タスク (Tasks)
DB2 を使用して実行できる主要なタスク。

参照 (Reference)
DB2 参照情報 (キーワード、コマンド、API など)。

ブック (Books)
DB2 ブック。

トラブルシューティング (Troubleshooting)
エラー・メッセージのカテゴリーと、メッセージに対する回復処置。

サンプル・プログラム (Sample Programs)
DB2 アプリケーション開発クライアントに付属のサンプル・プログラム。 DB2 アプリケーション開発クライアントをインストールしていない場合、このタブは表示されません。

Web
WWW 上にある DB2 情報。この情報にアクセスするには、ご使用のシステムから Web への接続が必要です。

リストから項目を 1 つ選択すると、インフォメーション・センターはビューアーを立ち上げて情報を表示します。選択した情報の種類に応じて、ビューアーはシステム・ヘルプ・ビューアー、エディター、または Web ブラウザーです。

インフォメーション・センターには検索機能が備わっており、リストを参照せずに特定のトピックを探すことができます。

テキストの全検索を行うには、インフォメーション・センター内のハイパーテキスト・リンク「**DB2 オンライン情報の検索 (Search DB2 Online Information)**」検索フォームに従います。

通常、HTML 検索サーバーは自動的に始動します。 HTML 情報の検索がうまくいかない場合は、以下の方法の 1 つを使用して、検索サーバーを始動しなければならない場合もあります。

Windows では

「スタート」をクリックし、**「プログラム」—> 「IBM DB2」—> 「Information」—> 「Start HTML Search Server」**を選択します。

OS/2 では

「DB2 (OS/2 版)」フォルダーをダブルクリックして、**「Start HTML Search Server」**アイコンをダブルクリックします。

HTML 情報の検索でこの他の問題が発生した場合は、リリース情報を参照してください。

注: 検索機能は、Linux、DYNIX/ptx、および Silicon Graphics IRIX 環境では利用できません。

DB2 ウィザードの使用

ウィザードを使用すると、各タスクをステップごとに進めることによって、さまざまな管理タスクを遂行することができます。ウィザードは、コントロール・センターおよびクライアント構成アシスタントを通して使用できます。以下の表では、ウィザードとその目的をリストしています。

注: データベース作成、索引作成、複数サイト更新の構成、およびパフォーマンス構成ウィザードは、区分データベース環境で使用できます。

ウィザード	内容	利用方法
データベース追加 (Add Database)	クライアント・ワークステーション上にデータベースのカタログを作成します。	クライアント構成アシスタントから、「**追加 (Add)**」をクリックします。

ウィザード	内容	利用方法
データベース・バックアップ *(Back up Database)*	バックアップ計画を決定、作成、およびスケジュールします。	「コントロール・センター (Control Center)」からバックアップするデータベースを右クリックし、**「バックアップ (Backup)」—>「ウィザードを使用するデータベース (Database Using Wizard)」**を選択します。
複数サイト更新の構成 (Configure Multisite Update)	複数サイト更新、分散トランザクション、または 2 フェーズ・コミットを構成します。	「コントロール・センター (Control Center)」から、**「データベース (Databases)」**フォルダーを右クリックして、**「複数サイト更新 (Multisite Update)」**を選択します。
データベース作成 (Create Database)	データベースを作成し、いくつかの基本的な構成タスクを実行します。	「コントロール・センター (Control Center)」から、**「データベース (Databases)」**フォルダーを右クリックして、**「作成 (Create)」—> 「ウィザードを使用するデータベース (Database Using Wizard)」**を選択します。
表作成 (Create Table)	基本的なデータ・タイプを選択して、表の基本キーを作成します。	「コントロール・センター (Control Center)」から、**「表 (Tables)」**アイコンを右クリックして、**「作成 (Create)」—>「ウィザードを使用する表 (Table Using Wizard)」**を選択します。
表スペース作成 (Create Table Space)	新しい表スペースを作成します。	「コントロール・センター (Control Center)」から、**「表スペース (Table Spaces)」**アイコンを右クリックして、**「作成 (Create)」—> 「ウィザードを使用する表スペース (Table Space Using Wizard)」**を選択します。
索引作成 (Create Index)	すべての照会について、作成すべき索引および除去すべき索引を提案します。	「コントロール・センター (Control Center)」から、**「索引 (Index)」**アイコンを右クリックして、**「作成 (Create)」—>「ウィザードを使用する索引 (Index Using Wizard)」**を選択します。

ウィザード	内容	利用方法
パフォーマンス構成 (Performance Configuration)	ビジネス要件に適合するように構成パラメーターを更新して、データベースのパフォーマンスを調整します。	「コントロール・センター (Control Center)」から、調整したいデータベースを右クリックして、**「ウィザードを使用するパフォーマンスの構成 (Configure Performance Using Wizard)」**を選択します。 区分データベース環境では、「Database Partitions」視点から、調整したい最初のデータベース区画を右クリックして、**「ウィザードを使用するパフォーマンスの構成 (Configure Performance Using Wizard)」**を選択します。
データベース復元 (Restore Database)	障害の後、データベースを回復します。どのバックアップを使用し、どのログを再生するかの判別を支援します。	「コントロール・センター (Control Center)」から復元するデータベースを右クリックし、**「復元 (Restore)」—>「ウィザードを使用するデータベース (Database Using Wizard)」**を選択します。

文書サーバーのセットアップ

デフォルトでは、DB2 情報はローカル・システムにインストールされます。つまり、DB2 情報にアクセスする必要のある各担当者が同じファイルをインストールする必要があります。 DB2 情報を 1 か所に格納するには、次のようにします。

1. `¥sqllib¥doc¥html` のすべてのファイルとサブディレクトリーを、ローカル・システムから Web サーバーにコピーします。各ブックには独自のサブディレクトリーがあり、そのブックを構成する必要な HTML および GIF ファイルが入っています。ディレクトリー構造は常に同じ状態に保つ必要があります。
2. Web サーバーを構成して、ファイルを新しい場所で検索するようにします。さらに詳しい情報については、 *インストールおよび構成 補足* の NetQuestion 付録を参照してください。
3. インフォメーション・センターの Java バージョンをご使用の場合は、すべての HTML ファイルのベース URL を指定できます。この URL はブックのリストに使用してください。

4. 資料ファイルが表示されるようになったなら、よく使うトピックにはブックマークを付けておいてください。ブックマークを付けるページは、たとえば以下のものがあります。
   - ブックのリスト
   - 頻繁に使用されるブックの目次
   - 頻繁に参照する情報 (たとえば、ALTER TABLE トピックなど)
   - 検索フォーム

中央のマシンから DB2 ユニバーサル・データベース オンライン文書ファイルを提供する方法については、 *インストールおよび構成 補足* の NetQuestion 付録を参照してください。

オンライン情報の検索

HTML ファイルの情報を検索するには、以下の方法のどれか 1 つを使用してください。

- 最上部にある**「検索 (Search)」**をクリックします。検索フォームを使用して特定のトピックを見つけます。この機能は、 Linux、DYNIX/ptx、または Silicon Graphics IRIX 環境ではご利用になれません。
- 最上部にある**「索引 (Index)」**をクリックします。索引を使用して、ブック内の特定のトピックを見つけます。
- HTML 資料またはヘルプの目次あるいは索引を表示してから、 Web ブラウザーの検索機能を利用して資料内の特定のトピックを見つけます。
- Web ブラウザーのブックマーク機能を使用して、特定のトピックにすばやく戻ります。
- インフォメーション・センターの検索機能を使用して、特定のトピックを検索します。詳しくは、738ページの『インフォメーション・センターを使用した情報へのアクセス』を参照してください。

特記事項

本書において、日本では発表されていない IBM 製品 (機械およびプログラム)、プログラミングまたはサービスについて言及または説明する場合があります。しかし、このことは、弊社がこのような IBM 製品、プログラミングまたはサービスを、日本で発表する意図があることを必ずしも示すものではありません。本書で、 IBM ライセンス・プログラムまたは他の IBM 製品に言及している部分があっても、このことは当該プログラムまたは製品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。これらのプログラムまたは製品に代えて、 IBM の知的所有権を侵害することのない機能的に同等な他社のプログラム、製品またはサービスを使用することができます。ただし、IBM によって明示的に指定されたものを除き、これらのプログラムまたは製品に関連する稼働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 および他社は、本書で説明する主題に関する特許権 (特許出願を含む)、商標権、または著作権を所有している場合があります。本書は、これらの特許権、商標権、および著作権について、本書で明示されている場合を除き、実施権、使用権等を許諾することを意味するものではありません。実施権、使用権等の許諾については、下記の宛先に、書面にてご照会ください。

〒106-0032 東京都港区六本木 3 丁目 2-31
AP 事業所
IBM World Trade Asia Corporation
Intellectual Property Law & Licensing

以下の保証は、国または地域の法律に沿わない場合は、適用されません。 IBM およびその直接または間接の子会社は、本書を特定物として現存するままの状態で提供し、商品性の保証、特定目的適合性の保証および法律上の瑕疵担保責任を含むすべての明示もしくは黙示の保証責任を負わないものとします。国または地域によっては、法律の強行規定により、保証責任の制限が禁じられる場合、強行規定の制限を受けるものとします。

本書に含まれる情報には、技術的に不正確なもの、または誤植が含まれる場合があります。これらに対する変更は、定期的に行われます。これらの変更は、資料の改訂版に含まれます。 IBM は、本書で説明している製品、プログラムに対して、予告なく改良、変更を加える場合があります。

本書において IBM 以外の Web サイトに言及している場合がありますが、便宜のため記載しただけであり、決してそれらの Web サイトを推奨するものではありません。それらの Web サイトにある資料は、この IBM 製品の資料の一部ではありません。それらの Web サイトは、お客様の責任でご使用ください。

IBM は、お客様が提供するいかなる情報も、お客様になんら義務も負わせない適切な方法で、使用もしくは配布することがあります。

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
1150 Eglinton Ave. East
North York, Ontario
M3C 1H7
CANADA

本プログラムに関する上記の情報は、適切な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。

本書に含まれるパフォーマンス・データは、制御された環境下で決定されています。したがいまして、その他の稼働環境で得られる結果とは、かなり異なる可能性もあります。一部の測定値は、開発中のシステムを使用している場合があり、これらの測定値が一般的に提供可能なシステムで同様の数値になることを保証するものではありません。さらに、一部の測定値が推定されたものもあります。実測値と異なる場合があります。本書のユーザーは、使用される特定の環境での該当データを確認してください。

IBM 以外の製品については、当該製品の提供者から直接、出版されている資料または一般公開されている情報から入手しました。 IBM は、これらの製品についてはテストを行っておらず、これらの IBM 以外の製品に関する性能、互換性またはその他の主張について確認することはできません。 IBM 以外の製品の機能に対する質問は、それぞれの製品提供者にお問い合わせください。

IBM の将来の方向性または意図については、予告なしに変更または中止する場合があり、 IBM の目的および目標のみ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が、これは説明に具体性を与えるために記載されたものであり、それらの例には、個人、企業、ブランドの、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。それらの名前はすべて架空のものであり、また名称や住所が類似する企業が実在しても、それは偶然に過ぎません。

著作権：

本書に含まれる情報には、サンプル・アプリケーション・プログラムがソース言語の形式で含まれており、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技法を示しています。お客様は、これらのサンプル・プログラムが書かれているオペレーティング・プラットフォームでアプリケーション・プログラミング・インターフェースが実行可能となるためのアプリケーション・プログラムを開発、使用、販売または配布もしくは転送する目的のためにのみ、サンプル・プログラムを、IBM に対する別途料金を支払うことなく、複製、変更、配布または転送することができます。これらのサンプルは、すべての条件下で十分にテストを行っていません。したがいまして、IBM は、これらのプログラムの信頼性、実用性または機能について、いかなる保証も負いません。

サンプル・プログラムまたはその改変版の複製物には、全部複製か部分複製かを問わず、次の著作権表示を必ず行うものとします。

© (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_. All rights reserved.

商標

次のものは、IBM Corporation の米国およびその他の国における商標です。

ACF/VTAM
AISPO
AIX
AIX/6000
AIXwindows
AnyNet
APPN
AS/400
BookManager
CICS
C Set++
C/370
DATABASE 2
DataHub
DataJoiner
DataPropagator
DataRefresher
DB2
DB2 Connect
DB2 Extenders
DB2 OLAP Server
DB2 Universal Database
Distributed Relational Database Architecture
DRDA
eNetwork
Extended Services
FFST
First Failure Support Technology
IBM
IMS
IMS/ESA
LAN DistanceMVS
MVS/ESA
MVS/XA
Net.Data
OS/2
OS/390
OS/400
PowerPC
QBIC
QMF
RACF
RISC System/6000
RS/6000
S/370
SP
SQL/DS
SQL/400
System/370
System/390
SystemView
VisualAge
VM/ESA
VSE/ESA
VTAM
WebExplorer
WIN-OS/2

次のものは、他社の商標または登録商標です。

Tivoli および NetView は、米国およびその他の国における Tivoli Systems Inc. の商標です。

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。

Java およびすべての Java 関連の商標およびロゴは、 Sun Microsystems, Inc. の米国およびその他の国における商標または登録商標です。

UNIX は、The Open Group がライセンスしている米国およびその他の国における登録商標です。

他の会社名、製品名およびサービス名等はそれぞれ各社の商標または登録商標です。

索引

日本語, 数字, 英字, 特殊文字の順に配列されています。なお, 濁音と半濁音は清音と同等に扱われています。

［ア行］

［カ行］

© Copyright IBM Corp. 1993, 2000

[サ行]

[タ行]

［ナ行］

［ハ行］

［マ行］

［ヤ行］

［ラ行］

A

B

C

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Z

IBM と連絡をとる

技術上の問題がある場合は、時間をとって「問題判別の手引き」に定義されている処置を検討し、それらの提案を実行した後で、DB2 顧客サービスに連絡をとってください。この資料には、DB2 顧客サービスがお客さまを支援するために必要とする情報が説明されています。

製品情報

以下の情報は英語で提供されます。内容は英語版製品に関する情報です。

http://www.ibm.com/software/data/

DB2 World Wide Web ページには、ニュース、製品説明、研修スケジュールなどの DB2 に関する最新情報が提供されています。ただし提供されている情報は英語です。

http://www.ibm.com/software/data/db2/library/

「DB2 Product and Service Technical Library」では、よくされる質問 (FAQ)、修正内容、資料、および最新の DB2 技術情報などの情報へのアクセスが提供されています。

注: この情報のご提供は英語のみとなりますのでご注意ください。

http://www.elink.ibmlink.ibm.com/pbl/pbl/

「International Publications」注文用 Web サイトでは、マニュアルの注文方法についての情報を提供しています。ただし、提供されている情報は英語です。

http://www.ibm.com/education/certify/

IBM の「Professional Certification Program」Web サイトでは、 DB2 を含むさまざまな IBM 製品の認証テストの情報を提供しています。ただし、提供されている情報は英語です。

ftp.software.ibm.com

匿名でログオンしてください。ディレクトリー `/ps/products/db2` には、 DB2 および多数の他製品に関連したデモ、修正プログラム、情報、およびツールがあります。ただし、提供されている情報は英語です。

© Copyright IBM Corp. 1993, 2000

comp.databases.ibm-db2, bit.listserv.db2-l

これらのインターネット・ニュースグループは、ユーザーが DB2 製品に関する自分の経験について話し合うために利用できます。ただし、提供されている情報は英語です。

Compuserve: GO IBMDB2

このコマンドを入力すると、IBM DB2 Family forum にアクセスできます。すべての DB2 製品が、このフォーラムでサポートされています。ただし、提供されている情報は英語です。

米国以外の国で IBM に連絡する方法については、 *IBM Software Support Handbook* の Appendix A を参照してください。この資料にアクセスするには、 Web ページ: http://www.ibm.com/support/ にアクセスし、ページの最下部にある「IBM Software Support Handbook」リンク・ボタンを選択します。

注: 国によっては、 IBM が承認している販売業者が、 IBM サポート・センターの代わりにそれら販売業者のサポート・センターに連絡する場合があります。

Printed in Japan

SC88-8514-00

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