IBM DB2 Everyplace

アプリケーション開発ガイド

バージョン 8.1.4

SC88-9479-01
(英文原典：SC18-7185-01)

IBM DB2 Everyplace

アプリケーション開発ガイド

バージョン 8.1.4

SC88-9479-01
(英文原典：SC18-7185-01)

ご注意

本書の情報およびそれによってサポートされる製品を使用する前に、 383 ページの『特記事項』に記載する一般情報をお読みください。

本書は、DB2 Everyplace のバージョン 8.1 (プロダクト番号 5724-D04) に適用されます。また、新しい版で特に断りがない限り、これ以降のすべてのリリースおよびモディフィケーションにも適用されます。

本書は、SC88-9479-00 の改訂版です。

本書には、IBM の専有情報が含まれています。その情報は、使用許諾条件に基づき提供され、著作権により保護されています。本書に記載される情報には、いかなる製品の保証も含まれていません。また、本書で提供されるいかなる記述も、製品保証として解釈すべきではありません。

IBM は、お客様が提供するいかなる情報も、お客様に対してなんら義務も負うことのない、自ら適切と信ずる方法で、使用もしくは配布することができるものとします。

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

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

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

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

(URL は、変更になる場合があります)

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

原　典： SC18–7185–01
Application Development Guide
Version 8 Release 1

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 2003.10

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

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

目次

第 1 部 概要

第 1 章 DB2 Everyplace 製品の概要

このセクションでは、DB2 Everyplace の概要、DB2 Everyplace ソリューションを構成するコンポーネントの説明、および一般的な DB2 Everyplace シナリオの例を説明します。本セクションには、以下のトピックが含まれています。

DB2 Everyplace の紹介

DB2 Everyplace は、パーベイシブ・コンピューティングにおける IBM ソリューションの一部です。DB2 Everyplace を使用することによって、モバイル環境を利用する職業に従事しているユーザー (営業担当者、調査官、監査員、フィールド・サービス技術者、医師、不動産業者、および保険査定員など) は、オフィスを離れている場合でも、必要な重要データにアクセスすることができます。

会社などの組織の DB2 エンタープライズ・データをモバイルおよび組み込みデバイスに配布することが可能になりました。DB2 Everyplace で、モバイル・デバイス上のデータベースにアクセスしたり更新することができます。DB2 Everyplace 同期サーバーを使用すれば、モバイル・デバイスのデータを自社内の他のデータ・ソースと同期させることができます。また、ファイル・アダプター機能を使用すると、ファイルやアプリケーションをモバイル・ユーザーに配布できます。

DB2 Everyplace データベースは、モバイル・デバイス上にあるリレーショナル・データベースです。モバイル・デバイスにあるデータにアクセスするために、ラピッド・アプリケーション開発ツール、サポートされている DB2 コール・レベル・インターフェース (CLI) 関数セット、Java Database Connectivity (JDBC) メソッド、または ADO.NET メソッドを使用してアプリケーションを作成することができます。

DB2 Everyplace ソリューションのコンポーネント

DB2 Everyplace ソリューションの主要なフィーチャーおよびコンポーネントは次のとおりです。

- DB2 Everyplace モバイル・データベース。
- DB2 Everyplace 同期サーバー。
- DB2 Everyplace 同期クライアント。

- DB2 Everyplace Mobile Application Builder。
- DB2 Everyplace サンプル・アプリケーション。

DB2 Everyplace モバイル・データベース

このデータベースは、モバイル・デバイスに配置されます。モバイル・データベースは、DB2 Everyplace Database Edition、 DB2 Everyplace Enterprise Edition、および DB2 Everyplace Software Development Kit に同梱されています。モバイル・データベースに関連したコンポーネントには、他に以下のものがあります。

- サンプル・アプリケーション (エンジン・サイド)

DB2 Everyplace モバイル・データベースは、以下のオペレーティング・システムで使用できます。

- Palm OS
- Symbian OS
- Windows CE/Pocket PC
- Win32 (Windows® 95、Windows® 98、Windows® NT®、Windows® 2000®、および Windows® XP®)
- QNX Neutrino、Linux、および組み込み Linux デバイス

DB2 Everyplace は MIDP データベースを使用する MIDP モバイル・デバイスもサポートしています。

DB2 Everyplace 同期サーバー

DB2 Everyplace 同期サーバーは、DB2 Everyplace Enterprise Edition に同梱されています。同期サーバーに関連したその他の重要なコンポーネントには以下のものがあります。

- DB2 Everyplace モバイル・デバイス管理センター
- サンプル・アプリケーション (サーバー・サイド)

DB2 Everyplace 同期サーバーと DB2 Everyplace 同期クライアントを使用して、DB2 Everyplace モバイル・デバイスとエンタープライズ・データ・ソースとの間でデータやアプリケーションを同期させることができます。

データ同期は、両方向か、または単一方向にすることができます。データは DB2 Everyplace モバイル・デバイスでも、エンタープライズ・データベースでも更新できます。例えば、DB2 for z/OS データベースのデータのサブセットをモバイル・デバイスの DB2 Everyplace データベースにダウンロードし、データを表示して、変更を加えてから、変更済みのデータを z/OS サーバーに同期させることができます。 DB2 Everyplace 同期サーバーは、競合を解決するためのメカニズムも備えています。

DB2 Everyplace 同期サーバーには、似たようなデータ同期のニーズを持つユーザーのグループを対象とする同期サービスを管理し提供するうえで役立つ管理ツールが用意されています。モバイル・デバイス管理センターの詳細については、「同期サーバー 管理ガイド」を参照してください。

DB2 Everyplace 同期サーバーは、リレーショナル・データと、 DB2 Universal Database などの JDBC インターフェースを持つ任意のデータ・ソースとの同期をサポートします。

DB2 Everyplace 同期サーバーは、リレーショナル・データと以下のデータ・ソースとの同期をサポートします。

- DB2 Universal Database for z/OS
- DB2 Universal Database for iSeries
- DB2 Universal Database for Linux、UNIX、および Windows
- JDBC インターフェースを持つすべてのデータ・ソース

DB2 Everyplace 同期クライアント

DB2 Everyplace 同期クライアントは、DB2 Everyplace Enterprise Edition に同梱されています。

モバイル・デバイスで実行される DB2 Everyplace 同期クライアントは、DB2 Everyplace 同期サーバーと共に稼働する複数のアプリケーションで構成されています。このクライアントは、エンタープライズ・リレーショナル・データと、モバイル・デバイス上の DB2 Everyplace モバイル・データベースとの双方向同期を処理します。このモバイル・デバイスは、デバイスへのモバイル・アプリケーションの配布を容易にするために、ファイル・サブスクリプション関連の操作も管理するほか、DB2 データベースに保管されているストアード・プロシージャーも実行することができます。

同期クライアントは、以下のオペレーティング・システムで使用できます。

- Palm OS
- Symbian OS
- Windows CE/Pocket PC
- Win32 (Windows® 95、Windows® 98、Windows® NT®、Windows® 2000®、および Windows® XP®)
- QNX Neutrino、Linux、および組み込み Linux デバイス

同期クライアントで用意されているアプリケーション・プログラミング・インターフェース (API) の詳細については、「*DB2 Everyplace アプリケーション開発ガイド*」を参照してください。

DB2 Everyplace Mobile Application Builder

DB2 Everyplace Mobile Application Builder は、Software Development Kit に同梱されています。また、IBM の Web サイトからダウンロードすることもできます。

DB2 Everyplace Mobile Application Builder を使用すると、Palm OS、WinCE、Symbian OS、およびユーザー・インターフェースと Java 仮想マシンをサポートするその他のプラットフォームで、DB2 Everyplace アプリケーションを開発することができます。 Mobile Application Builder を使用することで、1 行もコードを書くことなくアプリケーションを作成することができます。 Mobile Application Builder の入手方法については、DB2 Everyplace の Web サイトを参照してください。

その他の開発ツールとしては、WebSphere Studio Device Developer、Visual Age Micro Edition、Metrowerks CodeWarrior、および GNU Software Developer's Kit などがあります。

DB2 Everyplace のサンプル・アプリケーション

DB2 Everyplace を使用したアプリケーションの例を示す、サンプル・アプリケーションが用意されています。 Visiting Nurse (巡回医療サービス) サンプル・アプリケーションを使用することによって、モバイル・データベースと同期サーバーとの間の双方向同期を素早くテストすることができます。このサンプル・アプリケーションは、同期サーバー上で稼働する部分とモバイル・データベース上で稼働する部分の 2 つの部分から構成されます。このモバイル・データベースのサンプル・アプリケーションは、独立型環境におけるデータベース・エンジンの機能性を実際に示すためのものです。同期サーバーのサンプル・アプリケーションとデータベース・エンジンのサンプル・アプリケーションを共に用いると、 DB2 Everyplace の全コンポーネントを呼び出す完全なアプリケーションとして機能します。

IBM Sync も、MDAC に定義されているサブスクリプション表と同期をとるために DB2 Everyplace 同期クライアント API を使用する方法を示すサンプル・アプリケーションです。

コマンド行プロセッサーは、アプリケーション開発ツールです。このツールは、コマンド行インターフェースを持つプラットフォーム上の DB2 Everyplace を使用したサンプル・アプリケーションとして提供されます。コマンド行プロセッサーは、モバイル・デバイス上の DB2 Everyplace データベースで使用されます。これは同期サーバーでは使用しません。

DB2 Everyplace がサポートする SQL ステートメントを使用することで、ユーザーは表および索引の作成やドロップ、表の行の削除、挿入、および更新を行うことができます。

サポートされる SQL ステートメントの詳細については、「*DB2 Everyplace アプリケーション開発ガイド*」を参照してください。

DB2 Everyplace のサンプル・シナリオ

保険査定員は、保険金請求の申し立てをした顧客の所有物について損害査定する責任をもちます。多くの保険会社では、保険査定員が請求者の所有資産を実地検分し、所定の用紙に記入して請求が妥当であるか否かを判定し、請求者に支払われる損害額を査定します。保険査定員は、後でオフィスに戻ってから、用紙に記入した内容を会社のコンピューター・システムに手作業で入力しますが、これでは時間と費用がかかります。

DB2 Everyplace アプリケーションが使えるモバイル・デバイスを査定員に持たせることにより、この処理を大幅に合理化できます。モバイル・デバイスを使用することによって、査定員はどこにいても査定のスケジュールや手順、および請求者の保険契約情報にアクセスできます。また、モバイル・デバイス上で査定用紙に記入することも可能です。保険査定員はオフィスに戻った後、自分のモバイル・デバイスにある新しい査定用紙データを会社のエンタープライズ・データベースにアップロードして、会社のコンピューター・システムと同期させることができます。また、

社外にいるときに情報が必要になった場合には、モデムを介して直ちにモバイル・デバイス上のデータを会社のコンピューター・システムと同期させることができます。請求査定プロセスが紙をまったく使用せずに行えるようになるため、保険会社はコストを大幅に削減できます。保険査定員は会社のエンタープライズ・データベースに即時にアクセスできるため、請求も迅速に処理できます。

第 2 部 DB2 Everyplace アプリケーションの開発

第 2 章 DB2 Everyplace C/C++ アプリケーションの開発

この章では、DB2 Everyplace 用の C/C++ アプリケーションの開発について説明します。本章には、以下のセクションがあります。

- 『DB2 Everyplace C/C++ アプリケーションの開発』
- 13 ページの『C/C++ がサポートされているオペレーティング・システム』
- 12 ページの『サポートされる C/C++ 開発ツール』
- 13 ページの『C/C++ プロジェクトの準備、コンパイル、リンク』
- 16 ページの『C/C++ アプリケーションのテスト』
- 18 ページの『C/C++ を使用した DB2 Everyplace 同期クライアント・アプリケーションの開発』

DB2 Everyplace C/C++ アプリケーションの開発

C/C++ を使用して DB2 Everyplace アプリケーションを開発するには、DB2 Everyplace CLI/ODBC インターフェースを使用します。 このトピックでは、DB2 Everyplace を使用して C/C++ アプリケーションを開発するために実施する必要のある作業について概要を述べます。

C/C++ を使用して DB2 Everyplace アプリケーションを開発するには:

1. 開発ワークステーションに DB2 Everyplace をインストールします。指示の詳細については、「*DB2 Everyplace インストールおよびユーザーズ・ガイド*」を参照してください。
2. アプリケーションとそのデータ要件を定義します。

   エンド・ユーザーが表示または変更する必要があるデータを判別し、そのようなデータを DB2 Everyplace データベースで検索、保管、および更新する方法を決定します。
3. DB2 CLI インターフェースを理解し、どの DB2 CLI 関数をアプリケーションで使用するかを決定します。
4. DB2 Everyplace でサポートされている DB2 CLI 関数を使用して、C/C++ アプリケーション・プログラムを作成します。
5. DB2 Everyplace ヘッダー・ファイルとオペレーティング・システム・ライブラリーを使用して、アプリケーション・コードの準備、コンパイル、およびリンクを行います。
6. アプリケーションをテストします。
   a. DB2 Everyplace ライブラリーを、ご使用のオペレーティング・システムのエミュレーターまたはデバイスにコピーします。
   b. アプリケーションをデバイスでテストするか、またはエミュレーターを使用している場合はエミュレーターでテストします。

関連した概念:

- 101 ページの『第 13 章 C/C++ サンプル・アプリケーション』

関連した解説:

- 『サポートされる C/C++ 開発ツール』
- 13 ページの『C/C++ がサポートされているオペレーティング・システム』
- 202 ページの『DB2 CLI 関数の要約』

サポートされる C/C++ 開発ツール

DB2 Everyplace がサポートする DB2 CLI 関数を使用して、C/C++ アプリケーションを作成できます。

サポートされているオペレーティング・システムごとの、サポートされている標準の C/C++ 開発ツールは以下のとおりです。

Palm OS

以下のものを使用できます。

- DB2 Everyplace Mobile Application Builder. Mobile Application Builder については、DB2 Everyplace Web サイト http://www.ibm.com/software/data/db2/everyplace/ を参照してください。
- GNU Software Developer's Kit。
- Metrowerks CodeWarrior for Palm Computing Platform。この市販の開発環境により、Windows ワークステーションを使用して、Palm OS オペレーティング・システム用の C/C++ プログラムを作成することができます。

推奨事項: 他の Palm OS アプリケーションとの衝突を回避するために、アプリケーション作成者 ID を Palm, Inc. に登録してください。 DB2 Everyplace の表とアプリケーションは、IBDB または DB2*x* のような作成者 ID を持っています。*x* は、a から z までの英字です。作成者 ID についての詳細は、次の Web サイトをご覧ください。

http://www.palmos.com/dev/

Symbian OS バージョン 6.0

Microsoft Visual C++ バージョン 6 を Symbian バージョン 6.0 C++ Software Developer's Kit (SDK) と一緒に使用してアプリケーションを開発できます。

推奨事項: Symbian OS から UID を入手して、プロジェクト・ファイルに組み込んでください。これらの ID は SDK または次の Web サイトから入手することができます。

http://www.symbian.com/developer/techlib/papers/tn_uid/uidinfo.html

Symbian OS バージョン 7.0

Metrowerks CodeWarrior for Symbian を Symbian OS バージョン 7.0 SDK と一緒に使用してアプリケーションを開発できます。

Windows CE

アプリケーションを開発するには、Microsoft eMbedded Visual Tools 3.0 を使用することができます。

Windows NT および Windows 2000 オペレーティング・システム
アプリケーションを開発するには、Microsoft Visual C++ を使用することができます。

QNX Neutrino
アプリケーションを開発するには、Metrowerks CodeWarrior for QNX Neutrino または QNX Neutrino Software Developer's Kit (SDK) を使用することができます。

Linux アプリケーションを開発するには、組み込み Linux 配布版のクロス・プラットフォーム開発ツールを使用することができます。組み込み Linux カーネルは、ELF バイナリーのサポートを使用可能にする必要があります。

関連した概念:

- 101 ページの『第 13 章 C/C++ サンプル・アプリケーション』

関連したタスク:

- 11 ページの『DB2 Everyplace C/C++ アプリケーションの開発』

C/C++ がサポートされているオペレーティング・システム

C/C++ インターフェースは、以下のオペレーティング・システムで完全にサポートされています。

- Palm OS
- Symbian OS
- Windows CE® for Pocket PC
- Win32 (Windows 95、Windows 98、Windows NT、Windows 2000、および Windows XP)
- QNX Neutrino
- Linux および組み込み Linux

関連した概念:

- 101 ページの『第 13 章 C/C++ サンプル・アプリケーション』

関連したタスク:

- 11 ページの『DB2 Everyplace C/C++ アプリケーションの開発』

関連した解説:

- 12 ページの『サポートされる C/C++ 開発ツール』
- 202 ページの『DB2 CLI 関数の要約』

C/C++ プロジェクトの準備、コンパイル、リンク

このタスクは、C/C++ を使用して DB2 Everyplace アプリケーションを開発する、より大きなタスクの一部です。C/C++ プロジェクトの準備、コンパイル、およびリンクのステップが完了したら、11 ページの『DB2 Everyplace C/C++ アプリケーションの開発』に戻ります。

手順:

DB2 Everyplace には、アプリケーション開発のために、ヘッダー・ファイルおよびオペレーティング・システム・ライブラリー・ファイルが組み込まれます。

プロジェクト・ファイルを準備し、正しいコンパイラーを使用して DB2 Everyplace アプリケーションをコンパイルしリンクするには、次のようにします。

1. プロジェクト・ファイルを作成します。 この手順は、開発ツールと、開発対象のオペレーティング・システムによって異なります。
2. 下記の DB2 Everyplace ヘッダー・ファイルをプロジェクトに組み込みます。ヘッダー・ファイルには、DB2 Everyplace で提供される定数、データ・タイプおよび C/C++ 関数のプロトタイプが入っています。ヘッダー・ファイルには、次のものがあります。

   ```
   ¥db2everyplace¥Clients¥include¥sqlcli.h
   ¥db2everyplace¥Clients¥include¥sqlcli1.h
   ¥db2everyplace¥Clients¥include¥sqlext.h
   ¥db2everyplace¥Clients¥include¥sqlsystm.h
   ```

3. ユーザーのアプリケーションに固有のヘッダー・ファイルをすべて組み込みます。
4. 適切な DB2 Everyplace ライブラリーをプロジェクトに組み込みます。

   次の表で、それぞれのオペレーティング・システムごとに DB2 Everyplace ライブラリーを要約し、追加情報をリストします。

表 1. DB2 Everyplace ライブラリー

オペレーティング・システム	必要なライブラリー・ファイルと追加情報
Palm OS	¥db2everyplace¥clients¥palmos¥database¥DB2e.lib オプショナル: スタック・サイズを 8 KB に増やします。デフォルトは 4 KB です。 Palm OS アプリケーションのデフォルトのアプリケーション・スタック・サイズは制限されています。アプリケーションによっては、実行時にスタック・オーバーフローの問題が発生する可能性があります。この問題を避けるには、DB2 Everyplace に組み込まれている palm-pref.r ファイルに、より大きなスタック・サイズを指定してください。palm-pref.r ファイルにある説明に従い、それをプロジェクト・ファイルに組み込んでください。 PRC-Tools を使用してアプリケーションを開発している場合、アプリケーションの .def ファイルに stack=0x8000 を追加します。例: application {"MyApplicationName" APID stack=0x8000 }
Symbian OS v6	エミュレーター・アプリケーション: ¥db2everyplace¥clients¥symbian6¥database¥wins¥DB2e.lib デバイス・アプリケーション: ¥db2everyplace¥clients¥symbian6¥database¥armi¥DB2e.lib
Symbian OS v7	エミュレーター・アプリケーション: ¥db2everyplace¥clients¥Symbian7¥database¥wins¥DB2e.lib デバイス・アプリケーション: ¥db2everyplace¥clients¥Symbian7¥database¥armi¥DB2e.lib

表 1. DB2 Everyplace ライブラリー (続き)

オペレーティング・システム	必要なライブラリー・ファイルと追加情報
Windows CE	ARM プロセッサー: • V3.00 ¥db2everyplace¥clients¥wince¥database¥wce300¥armrel¥DB2e.lib • V4.00 ¥db2everyplace¥clients¥wince¥database¥wce400¥ARM4VRel¥DB2e.lib MIPS プロセッサー: • V3.00 ¥db2everyplace¥clients¥wince¥database¥wce300¥mipsrel¥DB2e.lib • V4.00 ¥db2everyplace¥clients¥wince¥database¥wce400¥MIPSIVRel¥DB2e.lib SH3 プロセッサー: • V3.00 ¥db2everyplace¥clients¥wince¥database¥wce300¥sh3rel¥DB2e.lib • V4.00 ¥db2everyplace¥clients¥wince¥database¥wce400¥SH3Rel¥DB2e.lib Windows CE エミュレーター: • V3.00 ¥db2everyplace¥clients¥wince¥database¥wce300¥x86emrel¥DB2e.lib (Pocket PC エミュレーター用) ¥db2everyplace¥clients¥wince¥database¥wce300¥x86rel¥DB2e.lib (Pocket PC 2002 エミュレーター用) • V4.00 ¥db2everyplace¥clients¥wince¥database¥wce400¥emulatorRel¥DB2e.lib (WinCE.NET エミュレーター用) UNICODE がプロジェクトで使用できることを確認します。UNICODE および _UNICODE を「プロジェクト設定 (Project Settings)」の 「**プリプロセッサー定義 (Preprocessor Definition)**」に追加します。 XScale プロセッサー: • v3.00 ¥db2everyplace¥clients¥wince¥database¥wce300¥xscale¥DB2e.lib
Win32	¥db2everyplace¥clients¥Win32¥database¥x86¥DB2e.lib
Neutrino	libdb2e.so Neutrino ライブラリー・ファイルは、DB2 Everyplace Neutrino の gzip された tar ファイル DB2EveryplaceNTO.tar.gz に入っています。このファイルは、¥db2everyplace¥clients¥neutrino¥database¥ ディレクトリーにあります。libdb2e.so は、/db2e/database/x86/ (x86 プロセッサー・タイプの場合) ディレクトリーにあります。
Linux	libdb2e.so Linux ライブラリー・ファイルは、DB2 Everyplace Linux の gzip された tar ファイル DB2EveryplaceLN.tar.gz に入っています。このファイルは、¥db2everyplace¥clients¥embeddedlinux¥database¥ ディレクトリーにあります。 libdb2e.so は /db2e/database/x86/ (x86 プロセッサー・タイプの場合) および /db2e/database/strongarm/ (strongarm プロセッサー・タイプの場合) ディレクトリーにあります。

5. オプショナル: UNICODE および _UNICODE マクロをプロジェクト・ファイルに定義して、ユニコード・サポートを利用できるようにします。

   ユニコードの詳細については、 377 ページの『DB2 Everyplace ユニコード・サポート』を参照してください。

6. プロジェクトをコンパイルして、オブジェクト・コードを適切な DB2 Everyplace ライブラリーとリンクします。

   アプリケーション開発ツールの多くでは、統合開発環境内からの自動コンパイルおよびリンクが提供されています。プロジェクトのコンパイルおよびリンクに関する追加情報については、ご使用のアプリケーション開発ソフトウェアの資料を参照してください。

関連した概念:

- 101 ページの『第 13 章 C/C++ サンプル・アプリケーション』

関連したタスク:

- 『C/C++ アプリケーションのテスト』

関連した解説:

- 12 ページの『サポートされる C/C++ 開発ツール』
- 13 ページの『C/C++ がサポートされているオペレーティング・システム』
- 202 ページの『DB2 CLI 関数の要約』
- 377 ページの『DB2 Everyplace ユニコード・サポート』

C/C++ アプリケーションのテスト

このタスクは、C/C++ を使用して DB2 Everyplace アプリケーションを開発する、より大きなタスクの一部です。C/C++ アプリケーションのテストするためのステップを完了したら、 11 ページの『DB2 Everyplace C/C++ アプリケーションの開発』に戻ってください。

手順:

アプリケーションをテストするには、次のようにします。

1. DB2 Everyplace ライブラリーを、ご使用のオペレーティング・システムのエミュレーターまたはデバイスにコピーします。 これらのファイルがないと、DB2 Everyplace アプリケーションはロードできません。次の表に、それぞれのオペレーティング・システムごとに必要な DB2 Everyplace ファイルを要約しています。

表 2. テストに必要な DB2 Everyplace ファイル

オペレーティング・システム	デバイスまたはエミュレーターで必要なファイル
Palm OS	¥db2everyplace¥clients¥palmos¥database¥DB2eCat.prc ¥db2everyplace¥clients¥palmos¥database¥DB2eCLI.prc ¥db2everyplace¥clients¥palmos¥database¥DB2eComp.prc ¥db2everyplace¥clients¥palmos¥database¥DB2eRunTime.prc ¥db2everyplace¥clients¥palmos¥database¥DB2eDMS.prc
Symbian OS バージョン 6.0	エミュレーターでのテストの場合、ファイル ¥db2everyplace¥clients¥symbian6¥database¥wins¥DB2e.dll を次の各エミュレーター・ディレクトリーにコピーしてください。 ¥EPOCROOT%EPOC32¥Release¥wins¥udeb¥ (デバッグ・エミュレーターの場合) ¥EPOCROOT%EPOC32¥Release¥wins¥urel¥ (リリース・エミュレーターの場合) デバイスでのテストの場合、 PsiWin 接続ソフトウェアを使用して次のファイルをインストールしてください。 ¥db2everyplace¥clients¥symbian6¥database¥armi¥DB2e.sis

表 2. テストに必要な *DB2 Everyplace* ファイル (続き)

オペレーティング・システム	デバイスまたはエミュレーターで必要なファイル
Windows CE	ご使用のオペレーティング・システムに適切なライブラリーをインストールします。 ARM プロセッサー: • V3.00 db2everyplace¥clients¥wince¥database¥wce300¥armrel¥DB2e.dll MIPS プロセッサー: • V3.00 ¥db2everyplace¥clients¥wince¥database¥wce300¥mipsrel¥DB2e.dll SH3 プロセッサー: • V3.00 ¥db2everyplace¥clients¥wince¥database¥wce300¥sh3rel¥DB2e.dll Windows CE エミュレーター: • V3.00 Pocket PC エミュレーター: db2everyplace¥clients¥wince¥database¥wce300¥x86emrel¥DB2e.dll Pocket PC 2002 エミュレーター: ¥db2everyplace¥clients¥wince¥database¥wce300¥x86rel¥DB2e.dll
Win32	¥db2everyplace¥clients¥win32¥database¥x86¥DB2e.dll を、アプリケーションの現行ディレクトリーか、システムの PATH 環境変数のいずれかにコピーします。
Neutrino	/db2e/database/x86/libdb2e.so (x86 プロセッサー・タイプ用) および /db2e/database/strongarm/libdb2e.so (strongarm プロセッサー・タイプ用)
Linux	/db2e/database/x86/libdb2e.so (x86 プロセッサー・タイプ用) および /db2e/database/strongarm/libdb2e.so (strongarm プロセッサー・タイプ用)

2. **Linux および Neutrino の場合**: 以下のいずれかの方法を使用してlibdb2e.so をライブラリー検索パスに追加します。
   - libdb2e.so をライブラリー検索パス内にあるディレクトリーへコピーします。これには、ルート権限が必要になる場合があります。
   - libdb2e.so を別のディレクトリーにコピーして、そのディレクトリーをライブラリー検索パスへ追加します。ディレクトリーをライブラリー検索パスへ追加する場合、永続的に /etc/ld.config にエントリーが必要になります。ライブラリー検索パスへ一時的にディレクトリーを追加するには、LD_LIBRARY_PATH 環境変数を適切に設定することで可能になります。

     たとえば、export LD_LIBRARY_PATH= というコマンドを入力します (現在のディレクトリー内の libdb2e.so で bash)。
3. テストするアプリケーション用のファイルをロードします。例えば、Visiting Nurse サンプル・アプリケーションを Palm OS でテストするには、NurseInit.prc と Nurse.prc のファイルをロードします。
4. アプリケーションをテストします。

関連した概念:

- 101 ページの『第 13 章 C/C++ サンプル・アプリケーション』

関連したタスク:

- 13 ページの『C/C++ プロジェクトの準備、コンパイル、リンク』

関連した解説:

- 12 ページの『サポートされる C/C++ 開発ツール』
- 13 ページの『C/C++ がサポートされているオペレーティング・システム』
- 202 ページの『DB2 CLI 関数の要約』

C/C++ を使用した DB2 Everyplace 同期クライアント・アプリケーションの開発

このトピックでは、IBM 同期クライアント C-API バージョン 8.1 に基づいて、C/C++ を使用して DB2 Everyplace 同期クライアント・アプリケーションを開発する方法の概要を説明します。 328 ページの『IBM 同期クライアント C-API の関数の要約』に、すべての C API 関数についての関数の仕様が記載されています。

前提条件:

開発に使用するワークステーションに DB2 Everyplace をインストールします。詳細については、「*DB2 Everyplace インストールおよびユーザーズ・ガイド*」を参照してください。

手順:

C/C++ を使用して DB2 Everyplace 同期クライアント・アプリケーションを開発するには、次のようにします。

1. 次のものが組み込まれた同期アプリケーションを定義します。
   - 同期化されるデータ
   - 許可される操作
   - ユーザーおよびユーザー・グループ
   - データ・セキュリティー (例えば、回線上でのデータ暗号化や、ローカルでのデータ暗号化など)

     同期化されるデータの定義およびユーザーの管理についての詳細は、「*DB2 Everyplace 同期サーバー 管理ガイド*」を参照してください。
2. DB2 Everyplace 同期クライアント・ヘッダー・ファイル (`isyncore.h`) を C アプリケーション・プログラムに組み込み、DB2 Everyplace 同期クライアント C API 関数を関数の仕様に従って使用します。
3. DB2 Everyplace 同期クライアント・オペレーティング・システム・ライブラリー `isyncconf` および `isyncore` を使用して、アプリケーション・コードを準備、コンパイル、およびリンクします。
4. アプリケーションをテストします。
   - DB2 Everyplace ライブラリーを、ご使用のオペレーティング・システムのエミュレーターまたはデバイスにインストールします。
   - エミュレーターでアプリケーションをテストします (エミュレーターを使用する場合)。

- デバイスでアプリケーションをテストします。

関連した概念:

- 129 ページの『同期クライアント C/C++ サンプル・アプリケーション』

関連した解説:

- 12 ページの『サポートされる C/C++ 開発ツール』

第 3 章 DB2 Everyplace Java アプリケーションの開発

この章では、DB2 Everyplace Java アプリケーションを開発する方法について説明します。説明されているトピックは、以下のとおりです。

- 『JDBC インターフェースがサポートされているオペレーティング・システム』
- 『DB2 Everyplace Java アプリケーションの開発』

JDBC インターフェースがサポートされているオペレーティング・システム

JDBC インターフェースは、以下のオペレーティング・システムでサポートされています。

- Palm OS
- Symbian OS
- Windows CE® for Pocket PC
- Win32 (Windows 95、Windows 98、Windows NT、Windows 2000、および Windows XP)
- QNX Neutrino
- Linux および組み込み Linux

DB2 Everyplace Java アプリケーションの開発

Java を使用して DB2 Everyplace アプリケーションを開発するには、Java Software Developer's Kit を DB2 Everyplace Java 用 JDBC (Java Database Connectivity) インターフェースと共に使用します。

このトピックでは、DB2 Everyplace と共に Java アプリケーションを開発するのに必要なタスクについて、高所から見た概要を説明します。

制約事項:

DB2 Everyplace は Symbian ではマルチタスキングをサポートしません。2 番目のスレッドからデータベースにアクセスするには、最初のスレッドの Connection オブジェクトを、 2 番目のスレッドで接続が確立できる前に閉じなければなりません。スレッド間で同じ Connection オブジェクトを共有することはできません。

前提条件:

DB2 Everyplace にアクセスする Java アプリケーションは、DB2 Everyplace JDBC ドライバーを使用します。 Java および JDBC をワークステーションにまだインストールしていない場合は、インストールしてください。

手順:

Java を使用して DB2 Everyplace アプリケーションを開発するには、次のようにします。

1. java.sql パッケージとその他の必要な Java クラスをインポートします。
2. DB2 Everyplace JDBC ドライバーをロードします。 クラス名は com.ibm.db2e.jdbc.DB2eDriver です。
3. jdbc:*subprotocol:subname* という形式の URL を使用してデータベースに接続します。DB2 Everyplace のサブプロトコル は db2e です。データベースが c:¥dir1¥dir2 にある場合は、URL jdbc:db2e:c:/dir1/dir2/ を使用してください。*subname* の相対パスを使用することもできます。
4. Statement オブジェクトを作成します。
5. データベースにアクセスします (ユーザーのアプリケーション・ロジックはここに移動します)。
   - Statement オブジェクトを使用して SQL ステートメントを実行します。
   - 戻された ResultSet オブジェクトからデータを検索します (実行した SQL ステートメントが照会である場合)。
6. ResultSet オブジェクト、Statement オブジェクト、Connection オブジェクトをクローズすることによって、データベースおよび JDBC リソースを解放します。

関連した概念:

- 103 ページの『Java サンプル・アプリケーションの概要』

関連したタスク:

- 105 ページの『ターゲットとなる Palm OS での Java サンプル・アプリケーションのコンパイルおよび実行』
- 111 ページの『ターゲットとなる Palm OS 以外の OS 上での Java サンプル・アプリケーションのコンパイルと実行』

関連した解説:

- 296 ページの『DB2 Everyplace JDBC サポートの概要』

第 4 章 Java 同期クライアント・アプリケーションの開発

この章では、DB2 Everyplace 同期クライアントの Java アプリケーションを開発する方法について説明します。説明されているトピックは、以下のとおりです。

- 『Java 同期 API がサポートされているオペレーティング・システム』
- 『IBM Java 同期 API』

Java 同期 API がサポートされているオペレーティング・システム

Java 同期 API は、以下のオペレーティング・システムで使用可能です。

- Win32
- Symbian OS
- Windows CE (MIPS および ARM プロセッサーを使用)
- Palm OS
- Linux
- QNX Neutrino

IBM Java 同期 API

Java Database Connectivity (JDBC) と Java インターフェースを使用して Java アプリケーションを作成して、 DB2 Everyplace Database と同期サーバーの機能を統合することができます。

DB2 Everyplace によってサポートされている IBM Java 同期 API で提供されるインターフェース、クラス、および例外に関する詳しい情報については、 `Clients¥javadoc` ディレクトリーにある Javadoc ドキュメンテーションを参照してください。

関連した概念:

- 103 ページの『Java サンプル・アプリケーションの概要』

関連したタスク:

- 21 ページの『DB2 Everyplace Java アプリケーションの開発』

DB2 Everyplace 同期プロバイダーの概要

このトピックでは DB2 Everyplace がサポートする同期クライアント Java-API について説明します。この API は、開発者が DB2 Everyplace とエンタープライズ・リレーショナル・データベースとの間で双方向でデータの同期化を行うアプリケーションを作成することができるようにするライブラリー・セットです。この同期クライアントは DB2 Everyplace 同期サーバーと共に作動して、リレーショナル・データおよびファイルの同期を単純化します。同期サーバーは、競合を解決し、モバイル PDA や組み込みデバイスまたは MIDP 1.0 対応のデバイスとの間のデータの移動を管理します。

同期クライアント Java API は、次の 2 つのタイプの同期プロバイダーで構成されています。

- 『DB2 Everyplace ネイティブ同期』
- 31 ページの『DB2 Everyplace Java 同期』

これらのプロバイダーに基づいたクライアント・デバイスでの Java アプリケーションの作成方法は、サンプル・ファイルに示されています。

関連したタスク:

- 25 ページの『JNI ベースのネイティブ同期プロバイダーのインストール』
- 28 ページの『トラップ・ベースのネイティブ同期プロバイダーのインストールおよび検証』

関連した概念:

- 131 ページの『Java ネイティブ同期アプリケーションのサンプル』
- 135 ページの『Java MIDP 同期アプリケーションのサンプル』

DB2 Everyplace ネイティブ同期

ネイティブ同期プロバイダーは、ネイティブ同期クライアント・ライブラリーを呼び出す Java インターフェースを提供します。

注: ネイティブ同期プロバイダーは、当リリースでは、スレッド・セーフティーのためのサポートはありません。アプリケーション側でスレッドの同期を調整してください。

DB2 Everyplace ネイティブ同期プロバイダーには、以下の 2 つのタイプがあります。

- Java Native Interface (JNI) ベースのネイティブ同期プロバイダー
- Palm OS トラップ・ベースのネイティブ同期プロバイダー

関連したタスク:

- 25 ページの『JNI ベースのネイティブ同期プロバイダーのインストール』
- 28 ページの『トラップ・ベースのネイティブ同期プロバイダーのインストールおよび検証』

関連した概念:

- 23 ページの『DB2 Everyplace 同期プロバイダーの概要』

DB2 Everyplace ネイティブ同期プロバイダーのインストール

この章では、DB2 Everyplace のネイティブ同期プロバイダーをインストールする方法について説明します。説明されているトピックは、以下のとおりです。

- 25 ページの『JNI ベースのネイティブ同期プロバイダーのインストール』
- 28 ページの『トラップ・ベースのネイティブ同期プロバイダーのインストールおよび検証』

JNI ベースのネイティブ同期プロバイダーのインストール

JNI ベース同期プロバイダーは、Java ネイティブ・インターフェースをサポートする Java VM と共に作動します。

このプロバイダーは、次のオペレーティング・システムでサポートされます。

- Win32
- Symbian Release 6 (Nokia 9210/9290 Communicator デバイスの場合)
- Symbian Release 7 (Sony Ericsson P800 デバイスの場合)
- Windows CE (Pocket PC デバイスの場合)
- Linux
- QNX Neutrino

前提条件:

JNI ベースの同期プロバイダーは、以下のファイルを必要とします。

- `isync4j.jar` ファイル
- 以下のネイティブ同期クライアントのバイナリー
  - `isyncore.dll`
  - `isyncconf.dll`
  - `imsadb2e.dll`
  - `imsafile.dll`
  - `imsaconfig.dll`
  - `wbxmllib.dll`
  - `isync4j.dll`
  - `isyncxpt.dll`

ご使用のアプリケーションが JNI ベースのネイティブ同期プロバイダーを使用している場合は、以下の isync4j Java パッケージをインポートする必要があります。

- `com.ibm.mobileservices.isync`
- `com.ibm.mobileservices.isync.event`
- `com.ibm.mobileservices.isync.sql`

以下のソフトウェアがご使用のシステムにインストール済みであることを確認してください。

- DB2 Everyplace 同期サーバー、バージョン 8
- DB2 Everyplace 同期クライアント・ライブラリー バージョン 8
  詳細については、「*DB2 Everyplace インストールおよびユーザーズ・ガイド*」を参照してください。
- Java ネイティブ・インターフェースをサポートする Java VM

サポートされている各オペレーティング・システムに JNI ベースの同期プロバイダーをインストールする方法の詳細については、以下のトピックを参照してください。

- 26 ページの『Win32 における JNI ベースの同期プロバイダーのインストール』

- 『Symbian V6 を使用した Nokia 9210/9290 Communicator デバイスにおける JNI ベースの同期プロバイダーのインストール』
- 27 ページの『Windows CE における JNI ベースの同期プロバイダーのインストール』

Win32 における JNI ベースの同期プロバイダーのインストール

Win32 オペレーティング・システムで JNI ベースの同期プロバイダーをインストールするには、ISyncSample プログラムをコンパイルおよび実行する必要があります。Win32 デバイスに対する JNI ベースのインプリメンテーションは、Sun MicroSystems Java™ VM および IBM Java™ 2 Standard Edition Developer Kit でテスト済みです。

手順:

1. ISyncSample プログラムをコンパイルします。
   a. 次のディレクトリーが組み込まれるように PATH システム変数を変更します。

      ```
      <DB2e_InstDir>¥Clients¥Win32¥database¥x86
      <DB2e_InstDir>¥Clients¥Win32¥sync
      ```

   b. isync4j.jar ファイルが組み込まれるように CLASSPATH 変数を変更します。

      ```
      <DB2e_InstDir>¥Clients¥Win32¥Sync¥isync4j.jar
      ```

   c. <DB2e_InstDir>¥Clients¥clientapisample¥Java_API ディレクトリーにあるサンプル・ファイルをコンパイルします。例えば、次のようにします。

      ```
      javac ISyncSample.java
      ```

2. isyncdb2e.properties ファイルを編集して、サーバー URL、ユーザー、およびパスワードを指定します。
3. ISyncSample プログラムを実行します。
   a. 以下のコマンドを入力します。

      ```
      java.exe ISyncSample <property file>
      ```

      ここで、*<property file>* はクライアント・データベースのプロパティー・ファイルです。例えば、次のようにします。

      ```
      java.exe -classpath .; isync4j.jar ISyncSample isyncdb2e.properties
      ```

関連したタスク:

- 25 ページの『JNI ベースのネイティブ同期プロバイダーのインストール』

Symbian V6 を使用した Nokia 9210/9290 Communicator デバイスにおける JNI ベースの同期プロバイダーのインストール

Symbian V6 を使用した Nokia 9210/9290 Communicator デバイスで JNI ベースの同期プロバイダーをインストールするには、ISyncSample プログラムをコンパイルし実行する必要があります。Symbian V6 を使用した Nokia 9210/9290 デバイスの JNI ベース・インプリメンテーションは、Symbian OS 6.0 PersonalJava JVM でテスト済みです。

手順:

1. ご使用のワークステーションで ISyncSample プログラムを編集し、コンパイルします。
   a. ISyncSample.java を編集して isyncdb2e.properties をパラメーターとします。
   b. 以下のコマンドを入力して、クラスパスに isync4j.jar を指定して ISyncSample.java をコンパイルします。

      ```
      javac -classpath isync4j.jar ISyncSample.java
      ```

   c. isyncdb2e.properties を編集してサーバー URL、ユーザー、およびパスワードを指定します。
2. ISyncSample プログラムを実行します。
   a. DB2 Everyplace データベースおよび同期クライアント・ライブラリーがデバイスにインストールされていることを確認します。
   b. ISyncSample.class ファイルと isyncdb2e.properties ファイルをデバイスの C:¥System¥Apps¥ISync ディレクトリーにコピーします。
   c. Windows のファイル・マネージャーを使用して、isync4j.jar ファイルを探して選択します。「**入力 (Enter)**」をクリックします。 Nokia デバイスにインストールされている Redirect プログラムを使用して、 Java プログラムから出力を転送し、この出力をコンソールに表示するか、またはファイルに書き込みます。

関連した解説:

- 49 ページの『サポートされる JSP バージョン 1.1 サブセット』
- 53 ページの『JSP アプリケーション・データベース・アクセスのための IBM カスタム・タグ』

Windows CE における JNI ベースの同期プロバイダーのインストール

Windows CE オペレーティング・システムで JNI ベースの同期プロバイダーをインストールするには、ISyncSample プログラムをコンパイルおよび実行する必要があります。 Windows CE デバイスに対する JNI ベースの同期プロバイダーは、OTI J9 JVM でテスト済みです。

手順:

1. ご使用のワークステーションで ISyncSample プログラムをコンパイルします。
   a. 以下のコマンドを入力して、クラスパスに isync4j.jar を指定して ISyncSample.java をコンパイルします。

      ```
      javac -classpath isync4j.jar ISyncSample.java
      ```

   b. isyncdb2e.properties を編集してサーバー URL、ユーザー、およびパスワードを指定します。
2. ISyncSample プログラムを実行します。
   a. J9 Java 仮想マシン (JVM) ランタイム環境がデバイス (例えば ¥wsdd) にインストールされていることを確認します。加えて、DB2 Everyplace および同期クライアント・ライブラリーがインストールされている必要があります。
   b. ISyncSample.class ファイルと isyncdb2e.properties ファイルをデバイス (例えば ¥) にコピーします。

c. 次のいずれかの方法で、クラスパスに isync4j.jar を指定して ISyncSample プログラムを呼び出します。

Java コンソール

以下のコマンドを入力します。

```
j9.exe -bp:¥wsdd¥classes.zip -cp:¥wsdd;¥Windows¥isync4j.jar ISyncSample
  <property file>
```

例えば、次のようにします。

```
j9.exe -bp:¥wsdd¥classes.zip -cp:¥wsdd;¥Windows¥isync4j.jar
 ISyncSample isyncdb2e.properties
```

Windows ショートカット

ご使用のワークステーションで ISyncSample.lnk という Windows ショートカットを作成して編集します。例えば、次のようにします。

```
255#"¥wsdd¥j9.exe" "-bp:¥wsdd;¥Windows¥isync4j.jar;¥wsdd¥classes.zip"
"ISyncSample" "isyncdb2e.properties"
```

ショートカットは 1 行で入力し、各フィールドは二重引用符で囲みます。最初に指定するフィールドは、実行可能ファイルの名前でなければなりません。指定するファイルおよびディレクトリーは、完全に修飾されていなければなりません。

d. サンプル・プログラムを実行し、同期化されたデータがプロパティー・ファイルで指定されたターゲット・ディレクトリーにあることを確認します。

関連したタスク:

- 25 ページの『JNI ベースのネイティブ同期プロバイダーのインストール』

トラップ・ベースのネイティブ同期プロバイダーのインストールおよび検証

トラップ・ベースのネイティブ同期プロバイダーは、WebSphere Studio Device Developer's (WSDD) J9 JVM と共に Palm OS プラットフォームでのみ使用されます。

このトピックでは、Palm OS 用の DB2 Everyplace isync4j を J9 の jclMidp (J2ME MIDP) 構成で使用する方法について説明します。この同期プロバイダーは、com.ibm.oti.palmos パッケージを参照しているため、 WSDD J9 JVM for PalmOS v1.5 以降でのみ稼働します。

以下の表は、API を Palm デバイスにインストールする際に使用するプログラムの場所を示しています (ここで、%DSYINSTDIR% は DB2 Everyplace のインストール・ディレクトリーを表します)。

ディレクトリー	説明
%DSYINSTDIR%/Clients/PalmOS/Sync/isync4j-palm/lib	Palm OS Java クラス用の isync4j が入っているフォルダー。これらのクラスは、インプリメンテーション時にインポートされます。
%DSYINSTDIR%/Clients/PalmOS/Sync/isync4j-palm/sample	サンプル isync4j アプリケーション用のソース・コードが入っているフォルダー。
%DSYINSTDIR%/Clients/PalmOS/Sync/isync4j-palm/bin/ISyncSample.prc	J9 Palm OS CLDC ライブラリーと共に使用されるサンプル isync4j アプリケーション。

前提条件:

トラップ・ベースのネイティブ同期プロバイダーでは、以下の同期クライアント・ネイティブ共有ライブラリーおよび DB2 Everyplace ライブラリーが必要です。

- isyncore.prc
- isyncconf.prc
- imsaconfig.prc
- imsafile.prc
- imsadb2e.prc
- wbxmllib.prc
- isyncxpt.prc

さらに、デバイスに J9 Palm OS JVM バイナリーがインストールされている必要があります。

アプリケーションでトラップ・ベースのネイティブ同期プロバイダーを使用している場合、以下の isync4j Java パッケージをインポートする必要があります。

- com.ibm.mobileservices.isync
- com.ibm.mobileservices.isync.db2e.sti
- com.ibm.mobileservices.isync.event
- com.ibm.mobileservices.isync.sql

システムに以下のソフトウェアがインストールされていることを確認します。

- Palm OS バージョン 3.5 以降 (少なくとも 8 MB のメモリーが必要)
- WebSphere Studio Device Developer (WSDD) バージョン 4.0
- DB2 Everyplace database for Palm OS バージョン 7.1 以降
- DB2 Everyplace 同期クライアント・ライブラリー バージョン 8.1 以降

WSDD のインストール後、Palm OS ターゲットをセットアップする必要があります。Palm OS ターゲットをセットアップする方法は、「WSDD Development Environment & Tools Product Documentation」の『Getting Started with Palm OS Targets』の章を参照してください。 WSDD 文書は、製品 CD-ROM の IBM¥wsdd¥wsdd4.0¥doc¥wsddCustomer.pdf にあります。最後に、WSDD サンプル・アプリケーションを作成および実行して、WSDD が正しくインストールされていることを確認します。

手順:

WSDD が正しくインストールされていることを検証するには、次のようにします。

1. isync4j サンプル・アプリケーション用の新規プロジェクトを作成します。
   a. WSDD で「Java パースペクティブ (Java Perspective)」を開きます。
   b. 「**ファイル**」->「**新規**」->「**その他**」を選択します。
   c. J2ME for J9 のウィザードを選択して、MIDlet Suite を作成します。
   d. 「MIDlet Suite Creation」ダイアログで、カスタム・プロジェクト、MIDlet 名、および MIDlet クラス名を指定します。
   e. 「**次へ**」をクリックします。
   f. 再度「**次へ**」をクリックして、「Java 設定」に進みます。

g. 「Java 設定」で「**ライブラリー**」タブをクリックして、「**フォルダーの作成 ...**」をクリックします。「新規クラス・フォルダー」ダイアログで **lib** と入力します。

h. 「**完了**」をクリックします。

2. DB2 Everyplace ISYNC4J Java クラスをインポートして、ビルド・パスをセットアップします。

a. 「**パッケージ**」ビューでプロジェクトをクリックして、メニュー項目「**ファイル**」->「**インポート ...**」をクリックします。

b. `%DSYINSTDIR%/Clients/PalmOS/Sync/isync4j-palm/lib` フォルダーをインポートして、ソース・ディレクトリーとして `%DSYINSTDIR%/Clients/PalmOS/Sync/isync4j-palm/lib` を選択します。

c. `lib` ディレクトリーを展開して、`/lib` の下にある com ディレクトリーのチェック・ボックスを選択します。「**インポートするリソースの宛先の選択:**」で「**フォルダー:**」フィールドに、プロジェクト名の後に /lib を付けて入力します。例えば、プロジェクト名が ISyncSample の場合には、フィールドは ISyncSample/lib となります。

d. 「**完了**」をクリックします。

e. lib フォルダーを展開すると、次の ISYNC4J Java パッケージがあります。

```
com.ibm.mobileservices.isync
com.ibm.mobileservices.isync.db2e.sti
com.ibm.mobileservices.isync.event
com.ibm.mobileservices.isync.sql
```

3. サンプル・アプリケーションを作成および実行して isync4j ライブラリーのセットアップを検証します。

a. サンプル・アプリケーションをインポートします。

- 「**パッケージ**」ビューでプロジェクトの src フォルダーをクリックして、メイン・メニューから「**ファイル**」->「**インポート**」とクリックします。
- ISyncSample.java をインポートします。ソース・ディレクトリーとして `%DSYINSTDIR%/Clients/PalmOS/Sync/isync4j-palm/samples/ISyncSample/` を選択して、ISyncSample.java のチェック・ボックスを選択します。インポートされたリソースの宛先が <project>/src であることを確認します.

b. サンプル・アプリケーションのビルド・ファイルを作成します。

- エディターで、「**in/exclusion**」タブをクリックしてから、「**新規**」をクリックします。
- メイン・クラスに **ISyncSample** と入力して、プラットフォームで「**J9 for Palm 68k**」を選択します。「**次へ**」をクリックします。
- クリエーター ID を入力し、アプリケーション名に ISyncSample を入力します。「**次へ**」を 2 回クリックします。
- 「**Prc Application on PalmOS emulator**」を選択します。「**完了**」をクリックします。

c. ISyncSample.jxeLinkOptions ファイルを変更します。

- 「**パッケージ**」ビューで、プロジェクトの **palm68k** フォルダーを展開します。

- 「ISyncSample.jxeLinkOptions」をダブルクリックします。
- エディターで「**in/exclusion**」タブをクリックして、「**新規**」をクリックします。
- 規則パターンに **com.ibm.mobileservices.isync.db2e.sti.DB2eISyncProvider** を入力して、「**OK**」をクリックします。
- エディターで、「**ソース**」タブをクリックします。
- **-vmOption -ms:15** と入力して、スタック・サイズを設定します。
- 変更を保管します。

d. サンプル・アプリケーションを実行します。
- メニューで「**実行**」アイコンをクリックして、ビルド・ファイルで「**実行**」->「**ビルド**」->「**起動**」を選択します。
- サンプル・アプリケーションのターゲットを選択して、「**完了**」をクリックします。
- エラーが無ければ、Palm OS エミュレーターが開始されてアプリケーションを実行します。

これで、独自のアプリケーションを作成することができます。新規アプリケーションを作成する際には、DB2 Everyplace isync4j の新規プロジェクト名をプロジェクトのビルド・パスに組み込みます。アプリケーションのビルド・ファイルを作成後、アプリケーションの要件を満たすように、 jxeLinkOptions ファイルを変更します。

関連した概念:

- 23 ページの『DB2 Everyplace 同期プロバイダーの概要』

DB2 Everyplace Java 同期プロバイダー

この章では DB2 Everyplace の Java 同期プロバイダーについて説明します。説明されているトピックは、以下のとおりです。

- 『DB2 Everyplace Java 同期』
- 32 ページの『DB2 Everyplace J2ME MIDP 同期』
- 33 ページの『Cloudscape 用の DB2 Everyplace Java 同期クライアント』

DB2 Everyplace Java 同期

Java 同期プロバイダーは、Java 同期クライアント・ライブラリーを呼び出す Java インターフェースを提供します。

DB2 Everyplace Java 同期プロバイダーには、以下の 2 つのタイプがあります。

- 32 ページの『DB2 Everyplace J2ME MIDP 同期』
- 33 ページの『Cloudscape 用の DB2 Everyplace Java 同期クライアント』

関連したタスク:

- 25 ページの『JNI ベースのネイティブ同期プロバイダーのインストール』
- 28 ページの『トラップ・ベースのネイティブ同期プロバイダーのインストールおよび検証』

関連した概念:

- 23 ページの『DB2 Everyplace 同期プロバイダーの概要』

DB2 Everyplace J2ME MIDP 同期

J2ME MIDP ISync Client を使用することで、MIDP Record Store Management System (RMS) へのサブスクリプションを同期化するアプリケーションを作成することができます。J2ME MIDP ISync Client は、DB2 Everyplace 同期サーバーと共に動作して、エンタープライズ・データベースと MIDP 1.0 対応デバイスとの間のリレーショナル・データの同期を単純化するライブラリー・セットです。同期サーバーは、MIDP デバイスとの間のデータの移動を管理します。

このトピックには J2ME MIDP ISync Client についての以下の情報が含まれています。

- J2ME MIDP ISync Client のインストールに必要な Web サーバー・ソフトウェア
- J2ME MIDP ISync Client を Motorola iDEN 電話で稼動させるのに必要なソフトウェアとハードウェア
- J2ME MIDP ISync Client インストール・ディレクトリーのレイアウト

J2ME MIDP ISync Client のインストールに必要な Web サーバー・ソフトウェア:

J2ME MIDP ISync Client をインストールするには、次のいずれかのソフトウェア製品が必要です。

- WebSphere Application Server アドバンスド・シングル・サーバー版 バージョン 4.x 以降。このソフトウェアのフリー・トライアル・バージョンは、 IBM Web サイト (http://www-3.ibm.com/software/webservers/appserv/advanced.html) からダウンロードできます。
- Apache Tomcat バージョン 4.0.x 以降。このソフトウェアのフリー・コピーは、http://jakarta.apache.org/tomcat/ でダウンロードできます。

J2ME MIDP ISync Client を Motorola iDEN 電話で稼動させるのに必要なソフトウェアとハードウェア:

Motorola iDEN 電話に MIDP 同期プロバイダーをインストールして実行するには、次のハードウェアおよびソフトウェアが必要です。

- Sun Microsystems JavaTM 2 Platform Micro Edition、Wireless Toolkit
- iDEN アップデートおよびデータ・ケーブル (電話へのアプリケーションのロード用)
- Apache ANT
- RetroGuard Ofuscator
- isync4j Java パッケージ (DB2 Everyplace に組み込まれています)
  - com.ibm.mobileservices.isync
  - com.ibm.mobileservices.isync.midp
  - com.ibm.mobileservices.isync.event

J2ME MIDP ISync Client インストール・ディレクトリーのレイアウト:

J2ME MIDP ISync Client のインストール・プロセスでは、次の 4 つの初期ディレクトリーを作成します。

- bin - %DSYINSTDIR%¥Clients¥Midp¥bin には、コマンド行から WTK エミュレーターを実行するスクリプトが含まれています。
- lib - %DSYINSTDIR%¥Clients¥Midp¥lib には、MIDP ISync API の jar、MIDP のサーブレット、FilterServlet.jar ファイル、および関連 JAD ファイルがあるサンプル MIDLet が含まれています。
- docs - %DSYINSTDIR%¥Clients¥Midp¥doc には、J2ME MIDP ISync Client Javadoc が含まれています。
- samples - %DSYINSTDIR%¥Clients¥Midp¥samples には、サンプル isync4j アプリケーションのソース・コードが含まれています。ここで、 %DSYINSTDIR% は、DB2 Everyplace のインストール・ディレクトリーです。

サンプルを再コンパイルする場合には、事前検査および難読化に、複数の build¥*Classes ディレクトリーが使用されます。

関連したタスク:

- 25 ページの『JNI ベースのネイティブ同期プロバイダーのインストール』
- 28 ページの『トラップ・ベースのネイティブ同期プロバイダーのインストールおよび検証』

関連した概念:

- 23 ページの『DB2 Everyplace 同期プロバイダーの概要』

Cloudscape 用の DB2 Everyplace Java 同期クライアント

Cloudscape 用の DB2 Everyplace Java 同期クライアントを使用することで、Cloudscape データベースへのサブスクリプションを同期化するアプリケーションを作成することができます。Cloudscape 用の Java 同期クライアントは、DB2 Everyplace Java 同期サーバーと共に動作して、エンタープライズ・データベースと Cloudscape クライアントとの間のリレーショナル・データの同期を単純化するライブラリー・セットです。同期サーバーは、デバイスとの間のデータの移動を管理します。

このトピックには Cloudscape 用の Java 同期クライアントについての以下の情報が含まれています。

- Cloudscape 用の Java 同期クライアントを稼動させるのに必要なソフトウェア
- Cloudscape 用 Java 同期クライアントのディレクトリー・レイアウト
- CLASSPATH 環境変数の設定

Cloudscape 用の Java 同期クライアントを稼動させるのに必要なソフトウェア:

Cloudscape 用の Java 同期クライアントを稼動させるには、以下のソフトウェア製品が必要です。

- DB2 Everyplace バージョン 8.1.4 またはそれ以降
- Cloudscape データベース

Cloudscape 用の Java 同期クライアントのインストール・ディレクトリー・レイアウト:

Cloudscape 用の Java 同期クライアントのファイルは以下のディレクトリーにあります。

- %DSYINSTDIR%¥Clients¥javaclient には Cloudscape ISync API jar ファイルが入っています。
- %DSYINSTDIR%¥doc¥javadoc¥javaclient には Cloudscape 用の Java 同期クライアントの Javadoc が入っています。

CLASSPATH 環境変数の設定:

Cloudscape 用の Java 同期クライアントを使用するには、CLASSPATH 環境変数を設定して以下のファイルを組み込む必要があります。

- インストールした Cloudscape からの Cloudscape jar ファイル (db2j.jar、db2jtools.jar)。
- Cloudscape ISync API jar ファイル (db2jisync.jar)。
- オプショナル: サンプル・アプリケーション (ISyncSample および GoISyncConsole)。

関連したタスク:

- 143 ページの『GoISyncConsole サンプル Java 同期アプリケーションのコンパイルおよび実行』

第 5 章 Visual Basic アプリケーションの開発

この章では、Visual Basic を使用した、DB2 Everyplace アプリケーションの開発についての情報を提供します。説明されているトピックは、以下のとおりです。

- 『DB2 Everyplace Visual Basic アプリケーションの開発』
- 36 ページの『Visual Basic インターフェースがサポートされているオペレーティング・システム』

DB2 Everyplace Visual Basic アプリケーションの開発

Visual Basic の DB2 Everyplace アプリケーションを開発する場合は、 DB2 Everyplace CLI/ODBC インターフェースを使用します。このトピックでは、DB2 Everyplace を使用して Visual Basic アプリケーションを開発するために完了すべきタスクについて、高所から見た概要を説明します。

制約事項:

Visual Basic を使用して DB2 Everyplace 用のアプリケーションを開発する場合は、以下の制限と要件を考慮する必要があります。

- SQLAllocHandleVer 関数をユーザーのアプリケーション・コードで直接使用しないでください。SQLAllocHandleVer は、DB2 Everyplace によって内部的に使用されます。これをユーザーのアプリケーション・コードで使用すると、プログラム障害が発生する恐れがあります。
- Visual Basic が DLL 内で関数呼び出しをロードしたり処理したりする方法が異なるため、デバッグが行えない場合があります。
- `db2e.dll` の DB2 Everyplace 関数を呼び出す Visual Basic 関数には、″On Error Resume Next″ ステートメントが含まれていなければなりません。そうでないと、プログラムが正しく機能しません。

手順:

DB2 Everyplace Visual Basic アプリケーションを開発するための基本ステップは、次のようになります。

1. 新規の Visual Basic プロジェクトを作成する。
2. `db2ecli.bas` ファイルを (DB2 Everyplace Visual Basic プロジェクト・ディレクトリーから) プロジェクト・フォルダーにコピーし、このファイルを Visual Basic プロジェクトに挿入する。
3. `DB2e.dll` をプロジェクト・フォルダーにコピーする。 `DB2e.dll` をプロジェクト・フォルダーに入れたくない場合は、 `db2ecli.bas` ファイルの関数宣言に含まれている `DB2e.dll` へのパスを変更します。
4. ユーザー独自のアプリケーション・コードを作成する。参考のため、DB2 Everyplace Visual Basic サンプル・プログラムを使用することもできます。
5. メニュー項目「**ファイル**」->「**Make** *project*」と選択して、アプリケーション用の実行可能プログラムを作成する。

関連した概念:

- 121 ページの『Visual Basic サンプル・アプリケーションの概要』

関連した解説:

- 202 ページの『DB2 CLI 関数の要約』
- 『Visual Basic インターフェースがサポートされているオペレーティング・システム』

Visual Basic インターフェースがサポートされているオペレーティング・システム

Visual Basic インターフェースは、以下のオペレーティング・システムで完全にサポートされています。

- Windows CE® for Pocket PC
- Win32 (Windows 95、Windows 98、Windows NT、Windows 2000、および Windows XP)

第 6 章 JSP アプリケーションの開発

この章では、JavaServer Pages を使用した、DB2 Everyplace アプリケーションの開発についての情報を提供します。説明されているトピックは、以下のとおりです。

- 『JSP がサポートされているオペレーティング・システム』
- 『DB2 Everyplace の JSP アプリケーションの開発』
- 38 ページの『DB2 Everyplace の JSP サポートの概要』

JSP がサポートされているオペレーティング・システム

JSP サポートは、以下のオペレーティング・システムで使用できます。

- Win32 (Windows® NT® および Windows® 2000®)
- Windows CE® for Pocket PC
- Symbian OS

関連したタスク:

- 39 ページの『Windows ワークステーションでの JSP サポートの検証』
- 40 ページの『Windows CE デバイスでの JSP 開発のためのセットアップ』

関連した概念:

- 38 ページの『DB2 Everyplace の JSP サポートの概要』
- 127 ページの『第 16 章 JSP サンプル・アプリケーション』

DB2 Everyplace の JSP アプリケーションの開発

DB2 Everyplace は JavaServer Pages (JSP) をサポートして、DB2 Everyplace Web ベース・アプリケーションを簡単に作成できるようにします。作成するアプリケーションはオペレーティング・システムから独立しており、モバイル装置上で、切断モードで、またはローカル・エリア・ネットワークからの切断中に、実行できます。

JSP テクノロジーは、動的な Web ページの開発および保守を行うためのより簡単かつ高速な方法を提供します。 JSP テクノロジーではユーザー・インターフェースをコンテンツ生成から分離するため、基本となる動的コンテンツを変更せずに、ページ・レイアウトの作成と更新を行うことができます。

JSP は JDBC テクノロジーを使用します。JSP を使用して作成した Web ベース・アプリケーションは、DB2 Everyplace JDBC ドライバーを通して DB2 Everyplace データベースにアクセスできます。

<DB2Everyplace>¥SDK¥JSP¥doc に入っている添付資料を参照して、WebSphere Studio を使用して DB2 Everyplace をアクセスする JSP ページの生成方法についての情報を入手してください。

47 ページの『Windows ワークステーションでの JSP アプリケーションの実行』のステップを実行して JSP アプリケーションをテストしてください。JSP アプリケー

ションは、デバイスに転送する前に必ずワークステーション上で実行するようにしてください。ワークステーションで JSP アプリケーションを実行することで、一部のアプリケーション・ファイルに必要なプリプロセスが行われます。

関連したタスク:

- 39 ページの『Windows ワークステーションでの JSP サポートの検証』
- 40 ページの『Windows CE デバイスでの JSP 開発のためのセットアップ』

関連した概念:

- 『DB2 Everyplace の JSP サポートの概要』
- 127 ページの『第 16 章 JSP サンプル・アプリケーション』

関連した解説:

- 49 ページの『サポートされる JSP バージョン 1.1 サブセット』
- 53 ページの『JSP アプリケーション・データベース・アクセスのための IBM カスタム・タグ』

DB2 Everyplace の JSP サポートの概要

DB2 Everyplace の JSP サポートは、以下の 2 つのコンポーネントから構成されます。

- ミニ HTTP Web サーバー
- JSP プロセッサー

ミニ HTTP Web サーバーは、Web ブラウザーから要求を受け取り、Web ブラウザーに応答を返します (要求および応答にはプロトコルとして HTTP 1.1 を使用します)。 JSP プロセッサーは、JSP ファイルを解析し、対応する Java ソース・コードを生成し、そのソース・コードをコンパイルします。 Java ソース・コードには、JSP ページが要求されるときに動的コンテンツを生成する JavaBeans が含まれる場合があります。JSP ページが要求されるときに、ミニ HTTP Web サーバーは対応する Java コードを実行し、要求に対する応答として、Web ブラウザーに出力を返します。

Web ブラウザーで `http://localhost/request.jsp` (**request.jsp** はユーザーが要求している JSP ページ) のような URL を入力すると、Web ブラウザーはミニ HTTP Web サーバーに要求を送信します。以下の条件のうちいずれかが当てはまる場合に、この要求は JSP プロセッサーに転送されます。

- JSP ページに対応する .class ファイルがない (例えば、JSP ページが新しく作成される場合)。
- JSP ページに対応する .class ファイルはあるが、JSP ファイルのタイム・スタンプが .class ファイルのタイム・スタンプより新しい (例えば、JSP ページが変更された場合)。

  注: request.jsp では、対応する .class ファイルは **_request_jsp_.class** です。

要求が JSP プロセッサーに転送されて JSP ファイルの構文が有効であれば、ミニ HTTP Web サーバーは Web ブラウザーに出力を送信して JSP ページが有効であることを示します。出力内の **request.jsp** リンクをクリックして JSP ページを見てください。

要求が JSP プロセッサーに転送されて JSP ファイル構文が無効な場合、ミニ HTTP Web サーバーは、Web ブラウザーに診断情報を送信します。

要求を JSP プロセッサーに転送する必要がない場合、ミニ HTTP Web サーバーは、JSP ページに対応する .class ファイルを実行し、Web ブラウザーに出力を送信します。

JSP アプリケーション開発は Windows ワークステーション上で行ってください。開発中は常に JSP ページをテストすることが重要です。JSP プロセッサーが JSP ページ内の構文エラーをキャッチします。 エラーが修正されてから、Web ブラウザーの「更新」ボタンをクリックして、JSP ページを再度テストします。JSP ページに加えた変更が反映される前に、Web ブラウザーのインターネット一時ファイルのフォルダーかキャッシュに存在するファイルを削除することが必要になる場合があります。アプリケーションが完成したなら、アプリケーションをデバイスに転送し、そのデバイス上で実行することができます。

注: JSP プロセッサーはワークステーション上で稼働し、デバイス上では必要ありません。

関連したタスク:

- 44 ページの『JSP アプリケーションの Windows CE デバイスへの転送』
- 『Windows ワークステーションでの JSP サポートの検証』
- 40 ページの『Windows CE デバイスでの JSP 開発のためのセットアップ』

関連した概念:

- 37 ページの『DB2 Everyplace の JSP アプリケーションの開発』
- 127 ページの『第 16 章 JSP サンプル・アプリケーション』

JSP 開発のためのセットアップ

このセクションでは、Java Server Pages を使用して DB2 Everyplace アプリケーションを開発するためのセットアップ方法についての情報を提供します。説明されているトピックは、以下のとおりです。

- 『Windows ワークステーションでの JSP サポートの検証』
- 40 ページの『Windows CE デバイスでの JSP 開発のためのセットアップ』
  - 41 ページの『Windows CE デバイスでの J9 JVM ランタイム環境のインストール』
  - 42 ページの『Windows CE デバイスでの JSP サポートのインストールおよび検証』
- 43 ページの『Symbian OS バージョン 6 デバイスでの JSP サポートのインストールおよび検証』

Windows ワークステーションでの JSP サポートの検証

前提条件:

Windows ワークステーションに以下のソフトウェアがインストールされていることを検証します。

- DB2 Everyplace Software Development Kit
- Java 2 Standard Development Kit、Standard Edition
  - Version 1.1.8 (ターゲットが Symbian OS バージョン 6 のデバイスの場合)
  - Version 1.2.2 以降 (その他のターゲットの場合)
- Web ブラウザー
  - Internet Explorer バージョン 5.50 以降、または
  - Netscape Navigator バージョン 6.2.1 以前

手順:

ご使用の Windows ワークステーションが DB2 Everyplace JSP サポートを使用するようにセットアップされているかどうかを検証するには、次のようにします。

- Visiting Nurse (巡回医療サービス) サンプル JSP アプリケーションを実行します。
  1. ミニ HTTP Web サーバーを開始します。
     a. MS-DOS ウィンドウを開きます。
     b. cd コマンドを使用して、<DB2Everyplace>¥SDK¥JSP¥Win32 ディレクトリーに移動します。
     c. runJspServer と入力します。
  2. Web ブラウザーを開いて URL: http://localhost/VisitingNurse/schedule.jsp にアクセスします。

     代わりに、URL http://localhost/ を入力して VisitingNurse/schedule.jsp ページをブラウズすることもできます。

DB2 Everyplace JSP サポートを使用できるようにワークステーションを正常にセットアップできたら、Visiting Nurse (巡回医療サービス) のスケジュール表が Web ブラウザーに表示されるはずです。

関連したタスク:

- 47 ページの『Windows ワークステーションでの JSP アプリケーションの実行』

関連した解説:

- 49 ページの『サポートされる JSP バージョン 1.1 サブセット』
- 53 ページの『JSP アプリケーション・データベース・アクセスのための IBM カスタム・タグ』

Windows CE デバイスでの JSP 開発のためのセットアップ

前提条件:

ご使用のワークステーションに、以下のソフトウェアがインストールされていることを確認します。

- DB2 Everyplace Software Development Kit

デバイスの ¥Windows ディレクトリーに以下のファイルが含まれていることを確認します。

```
<DB2Everyplace>¥Clients¥WinCE¥database¥ver¥proc¥CryptoPlugin.dll
<DB2Everyplace>¥Clients¥WinCE¥database¥ver¥proc¥DB2e.dll
<DB2Everyplace>¥Clients¥WinCE¥database¥ver¥proc¥DB2eJDBC.dll
<DB2Everyplace>¥Clients¥WinCE¥database¥jdbc¥db2ejdbc.jar
```

ここで、*ver* はご使用のデバイスの Windows CE オペレーティング・システムのバージョン番号で、*proc* はプロセッサー・タイプです。

ご使用のデバイスに、以下のソフトウェアがインストールされていることを確認します。

- Web ブラウザー

手順:

1. デバイスに J9 ランタイム環境をインストールします。
2. デバイスに DB2 Everyplace JSP サポートをインストールし、検証します。

関連したタスク:

- 48 ページの『Windows CE デバイスでの JSP アプリケーションの実行』

関連した解説:

- 49 ページの『サポートされる JSP バージョン 1.1 サブセット』
- 53 ページの『JSP アプリケーション・データベース・アクセスのための IBM カスタム・タグ』

Windows CE デバイスでの J9 JVM ランタイム環境のインストール

現在、StrongARM デバイスのみが Pocket PC でサポートされています。ご使用のデバイスが異なるプロセッサー・タイプの場合、JNI をサポートする他の JVM (例えば、Sun PersonalJava、Insignia Jeode、NSIcom CrEme 等) を試してみることもできます。J9 JVM 以外の JVM を使用する場合、それに合わせて ¥SDK¥JSP¥WinCE¥MiniHttpServer.lnk を変更する必要があります。

この J9 のインストールで、サンプル JSP アプリケーションを実行できます。ご使用の独自のアプリケーションに、追加の J9 ファイルのインストールが必要な場合があります。

このタスクは、Windows CE デバイスでの JSP 開発用のセットアップにおける、メインタスクの一部です。このステップが完了したら、40 ページの『Windows CE デバイスでの JSP 開発のためのセットアップ』に戻ります。

手順:

J9 JVM ランタイム環境をインストールするには、以下のようにします。

1. ご使用のワークステーションに WebSphere Studio Device Developer v5.5 (WSDD) をインストールします。 WSDD の評価版は、http://www.ibm.com/software/pervasive/products/wsdd/ からダウンロードできます。以降のステップの説明では、<WSDD> は WSDD をインストールしたディレクトリーのことを表します。
2. WSDD では、更新マネージャーを使用して WCE Tooling for WSDD をインストールします。

a. 「**ヘルプ**」 —> 「**ソフトウェア更新** 」 —> 「**更新マネージャー**」 とクリックして、「パースペクティブのインストール/更新」をオープンします。

b. 「フィーチャーの更新」ビューで、「**アクセス先サイト**」 —> 「**WebSphere Custom Environment**」 —> 「**WebSphere Custom Environment**」のノードを展開します。

c. WCE Tooling for WSDD 5.5.0 をクリックします。

d. 「プレビュー」ビューで、「**インストール**」ボタンをクリックし、以下のステップに従ってインストールします。

e. 同様のステップで以下の追加フィーチャーをインストールします。

- WCE jclMax Class Library
- WCE Database Enabler Library
- WCE Personal Configuration Class Library

3. デバイス上に以下のディレクトリー構造を作成します。

```
¥wsdd¥bin
¥wsdd¥lib
¥wsdd¥lib¥jclMax
```

4. 以下のファイルを ¥wsdd¥bin にコピーします。

```
<WSDD>¥wsdd5.0¥ive¥runtimes¥pocketpc¥arm¥ive¥bin¥j9.exe
<WSDD>¥wsdd5.0¥ive¥runtimes¥pocketpc¥arm¥ive¥bin¥j9dyn20.dll
<WSDD>¥wsdd5.0¥ive¥runtimes¥pocketpc¥arm¥ive¥bin¥j9int20.dll
<WSDD>¥wsdd5.0¥ive¥runtimes¥pocketpc¥arm¥ive¥bin¥j9max20.dll
<WSDD>¥wsdd5.0¥ive¥runtimes¥pocketpc¥arm¥ive¥bin¥j9prt20.dll
<WSDD>¥wsdd5.0¥ive¥runtimes¥pocketpc¥arm¥ive¥bin¥j9thr20.dll
<WSDD>¥wsdd5.0¥ive¥runtimes¥pocketpc¥arm¥ive¥bin¥j9vm20.dll
<WSDD>¥wsdd5.0¥ive¥runtimes¥pocketpc¥arm¥ive¥bin¥j9zlib20.dll
<WSDD>¥wsdd5.0¥ive¥runtimes¥pocketpc¥arm¥ive¥bin¥iverel20.dll
<WSDD>¥wsdd5.0¥ive¥runtimes¥pocketpc¥arm¥ive¥bin¥swt-win32-2104.dll
```

5. 以下のファイルを ¥wsdd¥lib にコピーします。

```
<WSDD>¥wsdd5.0¥ive¥runtimes¥common¥ive¥lib¥charconv.zip
```

6. 以下のファイルを ¥wsdd¥lib¥jclMax にコピーします。

```
<WSDD>¥wsdd5.0¥ive¥runtimes¥common¥ive¥lib¥jclMax¥classes.zip
<WSDD>¥wsdd5.0¥ive¥runtimes¥common¥ive¥lib¥jclMax¥database_enabler.jar
<WSDD>¥wsdd5.0¥ive¥runtimes¥common¥ive¥lib¥jclMax¥locale.zip
<WSDD>¥wsdd5.0¥ive¥runtimes¥pocketpc¥common¥ive¥lib¥jclMax¥prsnlwin.jar
```

40 ページの『Windows CE デバイスでの JSP 開発のためのセットアップ』に戻ります。

Windows CE デバイスでの JSP サポートのインストールおよび検証

このタスクは、Windows CE デバイスでの JSP 開発のためのセットアップにおける、メインタスクの一部です。このステップが完了したら、 40 ページの『Windows CE デバイスでの JSP 開発のためのセットアップ』に戻ります。

手順:

JSP サポートをインストールし検証するには:

1. 以下のファイルをデバイスにインストールします。

```
<DB2 Everyplace>¥SDK¥JSP¥WinCE¥DB2eJSP.CAB
```

2. Visiting Nurse (巡回医療サービス) サンプル JSP アプリケーションを実行して、ご使用のデバイスが DB2 Everyplace JSP サポートを使用するようにセットアップされているかどうかを検証します。
   a. ご使用の JVM によってはオプショナル: デバイスの MiniHttpServer.lnk を変更します。J9 JVM 以外の JVM を使用している場合は、ショートカットを変更する必要があります。ショートカットが始まる数は、「#」文字の後にある文字の数です。「#」文字の後にある文字の最大数は 256 です。
   b. ミニ HTTP Web サーバーを開始します。
      1) File Explorer を開きます。
      2) ルート・ディレクトリーへナビゲートします。
      3) MiniHttpServer ショートカットをクリックします。
   c. Web ブラウザーを開いて URL: http://localhost/VisitingNurse/schedule.jsp にアクセスします。

      代わりに、URL http://localhost/ を入力して VisitingNurse/schedule.jsp ページをブラウズすることもできます。

DB2 Everyplace JSP サポートを使用できるようにデバイスを正常にセットアップできたら、Visiting Nurse (巡回医療サービス) のスケジュール表が Web ブラウザーに表示されるはずです。

40 ページの『Windows CE デバイスでの JSP 開発のためのセットアップ』に戻ります。

Symbian OS バージョン 6 デバイスでの JSP サポートのインストールおよび検証

前提条件:

以下のソフトウェアが使用中のワークステーションにインストールされていることを確認します。

- DB2 Everyplace Software Development Kit

以下のソフトウェアがデバイスにインストールされていることを確認します。

- Web ブラウザー
- Java ランタイム環境 (Java.sis)

  Java.sis は、インターネットからダウンロード可能です。古いデバイスでは、デバイスの製品 CD-ROM のパッケージで Java.sis が提供されます。

デバイスの ¥system¥libs ディレクトリーに、以下のファイルが含まれていることを確認します。

- <DB2Everyplace>¥Clients¥Symbian6¥database¥armi¥CryptoPlugin.dll
- <DB2Everyplace>¥Clients¥Symbian6¥database¥armi¥DB2e.dll
- <DB2Everyplace>¥Clients¥Symbian6¥database¥armi¥db2ejdbc.dll
- <DB2Everyplace>¥Clients¥Symbian6¥database¥armi¥ECSPKCS11.DLL

デバイスの ¥system¥java¥ext ディレクトリーに、以下のファイルが含まれていることを確認します。

- <DB2Everyplace>¥Clients¥Symbian6¥database¥armi¥db2ejdbc.jar

手順:

Symbian OS バージョン 6 デバイスでの JSP サポートをインストールし検証するには、以下のようにします。

1. ファイル <DB2Everyplace>¥SDK¥JSP¥Symbian6¥DB2eJSP.sis をデバイスにインストールします。
2. Visiting Nurse (巡回医療サービス) サンプル JSP アプリケーションを実行して、デバイスが DB2 Everyplace JSP サポートを使用するようにセットアップされているかどうかを検証します。
   a. ミニ HTTP Web サーバーを開始します。
      1) デバイスの Extras 画面に移動します。
      2) DB2eJSP アプリケーションを開始します。
   b. Web ブラウザーを開いて URL: http://localhost/VisitingNurse/schedule.jsp にアクセスします。

      代わりに、URL http://localhost/ を入力して VisitingNurse/schedule.jsp ページをブラウズすることもできます。

DB2 Everyplace JSP サポートを使用できるようにデバイスを正常にセットアップできたら、Visiting Nurse (巡回医療サービス) のスケジュール表が Web ブラウザーに表示されるはずです。

関連したタスク:

- 49 ページの『Symbian OS バージョン 6 デバイスでの JSP アプリケーションの実行』

関連した解説:

- 49 ページの『サポートされる JSP バージョン 1.1 サブセット』
- 53 ページの『JSP アプリケーション・データベース・アクセスのための IBM カスタム・タグ』

JSP アプリケーションの Windows CE デバイスへの転送

前提条件:

JSP アプリケーションをデバイスに転送する前に、以下のタスクを完了してください。

- JSP サポートをデバイスにインストールします。
- ワークステーション上でアプリケーションを開発しテストします。

手順:

JSP アプリケーションを Windows CE デバイスに転送するには、以下のようにします。

1. アプリケーションが使用しているデータベースがデバイスにない場合は、これをデバイスにコピーします。アプリケーションが予期しているディレクトリーにデ

ータベースが書き込まれていることを確認してください。つまり、アプリケーションが接続する URL によって指定されたディレクトリーになります。

2. アプリケーション・ファイルを、デバイスの MiniHttpConfig.properties の中の JspPath プロパティーに対して指定されたディレクトリーにコピーします。アプリケーションがワークステーションの JspPath のサブディレクトリーにある場合、デバイスの JspPath の下に同じサブディレクトリー構成を作成してください。アプリケーション・ファイルをデバイスにコピーする際に、以下の規則に従います。

表 3.

ワークステーション上のファイル	デバイスにコピーするファイル
<page>.jsp	<page>_jsp_.class
<webapp>¥web.xml	<webapp>¥<webapp>_config_.class

注:

- ワークステーションでアプリケーションを実行する場合、デバイスにコピーする .class ファイルは JSP プロセッサーで生成されます。これらのファイルは、対応する .jsp および web.xml ファイルと同じディレクトリーにあります。
- アプリケーションには web.xml ファイルが含まれていない場合があります。WebSphere Studio Application Developer が生成したアプリケーションは web.xml ファイルを使用します。
- ショートカットの最大長は、ファイルの先頭で、「#」文字の後の 256 文字です。最大長のままにするには、以下の WebSphere Studio Application Developer サンプル・ファイルをサンプル・アプリケーションのディレクトリーではなくルート・ディレクトリーにコピーしてください。
  - dbbeans.jar
  - 任意の ViewBean クラス・ファイル

  これらのファイルを使用する WebSphere Studio によって生成されるアプリケーションを実行するには、MiniHttpServer.lnk 内のクラスパスに dbbeans.jar と、ViewBean クラス・ファイルが入ったディレクトリーがなければなりません。
- .jsp ファイルまたは web.xml ファイルでないアプリケーション・ファイルも、同様にデバイスにコピーしてください。

例えば、 Visiting Nurse (巡回医療サービス) サンプル JSP アプリケーションをデバイスにコピーするには、次のようにします。

a. Visiting Nurse (巡回医療サービス) サンプル・データベースをデバイスにコピーします。
   1) ディレクトリー構造 ¥sample¥data をデバイスに作成します。 これは、アプリケーションが接続する URL で指定されたディレクトリーです。
   2) ワークステーションの <DB2Everyplace>¥SDK¥JSP¥sample¥data ディレクトリーの内容をデバイスの ¥sample¥data にコピーします。
b. アプリケーション・ファイルをデバイスにコピーします。

1) ディレクトリー構造 ¥sample¥jsp をデバイスに作成します。これは、デフォルトの MiniHttpConfig.properties ファイルにある JspPath 用に指定されたディレクトリーです。
2) ¥sample¥jsp にサブディレクトリー ¥VisitingNurse を作成します。
3) 以下のファイルを ¥sample¥jsp¥VisitingNurse へコピーします。

```
<DB2Everyplace>¥SDK¥JSP¥sample¥jsp¥VisitingNurse¥_schedule_jsp_.class
<DB2Everyplace>¥SDK¥JSP¥sample¥jsp¥VisitingNurse¥_contact_jsp_.class
<DB2Everyplace>¥SDK¥JSP¥sample¥jsp¥VisitingNurse¥_medrecord_jsp_.class
<DB2Everyplace>¥SDK¥JSP¥sample¥jsp¥VisitingNurse¥_person_jsp_.class
```

関連したタスク:

- 40 ページの『Windows CE デバイスでの JSP 開発のためのセットアップ』

関連した解説:

- 49 ページの『サポートされる JSP バージョン 1.1 サブセット』
- 53 ページの『JSP アプリケーション・データベース・アクセスのための IBM カスタム・タグ』

JSP アプリケーションの実行

このセクションでは、JSP アプリケーションを実行する方法について説明します。説明されているトピックは、以下のとおりです。

- 『ミニ HTTP Web サーバーの構成』 (ご使用のワークステーションまたはアプリケーションの構成によってはオプショナルです)
- 47 ページの『Windows ワークステーションでの JSP アプリケーションの実行』
- 48 ページの『Windows CE デバイスでの JSP アプリケーションの実行』
- 49 ページの『Symbian OS バージョン 6 デバイスでの JSP アプリケーションの実行』

ミニ HTTP Web サーバーの構成

手順:

以下のファイルは、 Windows ワークステーションのデフォルトの MiniHttpConfig.properties ファイルです。このデフォルト・ファイルを使用するか、ユーザーのアプリケーションおよびシステムの要件に合うように変更することができます。 また、デバイスのデフォルトの MiniHttpConfig.properties ファイルも変更できます。

```
# Mini HTTP Web server properties - Win32

# JspPath

# Specifies: the path that contains the JSP pages (.jsp and .class files).
# Default: JspPath=<directory where the mini HTTP Web server is started>
#
# Note: use ¥¥ to denote the directory separator
#
JspPath=sample¥¥jsp

# Port
#
# Specifies: port that the server listens on
# Default: Port=80
#
# Note: If you are not using the default Port, start URL requests with:
```

```
#  http://localhost:Port/
# (instead of http://localhost/) where Port is the specified Port.

# Mime
#
# Specifies: Mime types
# Default: Mime=text/html wml htm html,text/plain txt,image/gif gif,image/jpeg jpg
#
# Note: Additional Mime types can be added using the following format:
# Mime=mime_type_A ext1 ext2 ext3 ...,mime_type_B ext4 ext5 ...,...
#
Mime=application/octet-stream exe class,image/jpeg jpeg jpg

# LogFile
#
# Specifies: log file for the server
# Values:
# "" - log entries are written to the console
# "no" - no log entries are kept
# "<log_file_name>" - log entries are written to <log_file_name>
# Default: LogFile=
#
LogFile=JspServer.log

# Index
#
# Specifies: the index page (displayed if http://localhost is requested)
# Values:
# "" - the JspPath directory will be loaded
# "no" - no page will be loaded
# "<index_file>" - <index_file> will be loaded
# Default: Index=
```

関連したタスク:

- 43 ページの『Symbian OS バージョン 6 デバイスでの JSP サポートのインストールおよび検証』

関連した解説:

- 49 ページの『サポートされる JSP バージョン 1.1 サブセット』
- 53 ページの『JSP アプリケーション・データベース・アクセスのための IBM カスタム・タグ』
- 37 ページの『JSP がサポートされているオペレーティング・システム』

Windows ワークステーションでの JSP アプリケーションの実行

手順:

Windows ワークステーションで JSP アプリケーションを実行するには、次のようにします。

1. 必要な場合、<DB2Everyplace>¥SDK¥JSP¥Win32 ディレクトリーに MiniHttpConfig.properties ファイルを構成します。
2. ミニ HTTP Web サーバーを開始します。
   a. MS-DOS ウィンドウを開きます。
   b. cd コマンドを使用して ディレクトリーを <DB2Everyplace>¥SDK¥JSP¥Win32 に移動します。
   c. runJspServer と入力します。
3. Web ブラウザーを開いて URL を入力し、JSP アプリケーションを開始します。 例えば、アプリケーションの開始ページが start.jsp である場合は、http://localhost/start.jsp と入力します。 MiniHttpConfig.properties ファ

イルでポート・プロパティーを構成した場合は、代わりに http://localhost:Port/start.jsp と入力します。この Port は指定されたポート番号です。

4. JSP アプリケーションの実行が終了した際にミニ HTTP Web サーバーを停止するには、次のようにします。
   a. サーバーを起動した MS-DOS ウィンドウに移動します。
   b. Ctrl+C とタイプした後、「y」をタイプしてバッチ・ジョブを終了します。

関連したタスク:

- 39 ページの『Windows ワークステーションでの JSP サポートの検証』

関連した解説:

- 49 ページの『サポートされる JSP バージョン 1.1 サブセット』
- 53 ページの『JSP アプリケーション・データベース・アクセスのための IBM カスタム・タグ』

Windows CE デバイスでの JSP アプリケーションの実行

手順:

Windows CE デバイスで JSP アプリケーションを実行するには、以下のようにします。

1. アプリケーションをご使用のデバイスに転送します。
2. 必要な場合、デバイスの MiniHttpServer.lnk を変更します。J9 JVM 以外の JVM を使用している場合は、ショートカットを変更する必要があります。ショートカットが始まる数は、「#」文字の後にある文字の数です。
3. 必要な場合、Windows CE デバイス用に MiniHttpConfig.properties ファイルを構成します。
4. ミニ HTTP Web サーバーを開始します。
   a. File Explorer を開きます。
   b. ルート・ディレクトリーへナビゲートします。
   c. MiniHttpServer ショートカットをクリックします。
5. Web ブラウザーを開いて URL を入力し、JSP アプリケーションを開始します。 例えば、アプリケーションの開始ページが start.jsp である場合は、http://localhost/start.jsp と入力します。 MiniHttpConfig.properties ファイルでポート・プロパティーを構成した場合は、代わりに http://localhost:Port/start.jsp と入力します。この Port は指定されたポート番号です。
6. JSP アプリケーションの実行が終了した際にミニ HTTP Web サーバーを停止するには、次のようにします。
   a. J9 コンソール・ウィンドウに移動します。
   b. 「**ファイル**」 —> 「**閉じる**」とクリックします。

関連したタスク:

- 40 ページの『Windows CE デバイスでの JSP 開発のためのセットアップ』

関連した解説:

- 『サポートされる JSP バージョン 1.1 サブセット』
- 53 ページの『JSP アプリケーション・データベース・アクセスのための IBM カスタム・タグ』

Symbian OS バージョン 6 デバイスでの JSP アプリケーションの実行

手順:

1. アプリケーションをデバイスに転送します。
2. 必要な場合、Symbian デバイス用に `MiniHttpConfig.properties` ファイルを構成します。
3. ミニ HTTP Web サーバーを開始します。
   a. デバイスの Extras 画面に移動します。
   b. DB2eJSP アプリケーションを起動します。
4. Web ブラウザーを開いて URL を入力し、JSP アプリケーションを開始します。 例えば、 アプリケーションの開始ページが `start.jsp` である場合は、http://localhost/start.jsp と入力します。`MiniHttpConfig.properties` ファイルでポート・プロパティーを構成したい場合は、代わりに http://localhost:Port/start.jsp と入力します (この Port は指定されたポート番号です)。
5. JSP アプリケーションの実行が終了した際にミニ HTTP Web サーバーを停止するには、ソフト・リセットを実行します。

関連したタスク:

- 43 ページの『Symbian OS バージョン 6 デバイスでの JSP サポートのインストールおよび検証』

関連した解説:

- 『サポートされる JSP バージョン 1.1 サブセット』
- 53 ページの『JSP アプリケーション・データベース・アクセスのための IBM カスタム・タグ』
- 37 ページの『JSP がサポートされているオペレーティング・システム』

サポートされる JSP バージョン 1.1 サブセット

このセクションでは、DB2 Everyplace の JSP サポートに組み込まれているディレクティブ、暗黙オブジェクト、スクリプト・エレメント、および標準アクションを一覧で示し説明します。

ディレクティブ:

ページ・ディレクティブ:

説明 ページ・ディレクティブは、ページに依存する属性を定義します。

構文

```
<%@ page page_directive_attr_list %>
page_directive_attr_list ::=
{language="scriptingLanguage"}
{extends="className" }
{import="importList" }
{contentType="ctinfo" }
```

属性　このディレクティブの 4 つの有効な属性は次のとおりです。

- **language** - 指定されている場合は「java」でなくてはなりません。
- **extends** - JSP ページがトランスフォームされたクラスのスーパークラスに付ける、完全修飾された Java プログラム言語クラス名。
- **import** - デフォルトのインポート・リストは、`com.ibm.db2e.jsp.server.*`、`java.io.*`、`java.sql.*`、および `java.util.*` です。
- **contentType** - 任意の値でかまいません (text/html、text/xml、application/x-octet など)。

例

```
<%@ page contentType="text/html" %>
```

組み込みディレクティブ:

説明　組み込みディレクティブを使用してデータを JSP ページに組み込みます。組み込まれるファイルには、処理もされるエレメントが含まれる場合があります。

構文

```
<%@ include file="relativeURLspec" %>
```

属性　このディレクティブの属性は **file** で、ページに相対するパスでなければなりません。パスが「/」で始まることはありません。パスは現行 JSP ページに相対的にインタープリットされます。

例

```
<%@ include file="copyright.html" %>
```

暗黙オブジェクト:

JSP ページを作成するときに、特定の暗黙オブジェクトにアクセスします。暗黙オブジェクトは、最初に宣言しなくても、スクリプトレットおよび式の中で使用できます。各暗黙オブジェクトには、Java コア・テクノロジーまたは表 4 に示される `com.ibm.db2e.jsp.server` パッケージで定義されるクラスがあります。

表 4. 暗黙オブジェクト

暗黙変数	タイプ	表記	メソッドのサマリー
request	com.ibm.db2e.jsp.server.MiniHttpRequest	JSP ページについての要求。	`java.lang.String getParameter(java.lang.String name)` `java.lang.String getQueryString()`

response	com.ibm.db2e.jsp.server.MiniHttpResponse	要求への応答。	`java.lang.String encodeURL(java.lang.String url)`
			`void setDateHeader(java.lang.String name, long date)`
			`void setHeader(java.lang.String name, java.lang.String value)`

in	java.io.BufferedReader	このオブジェクトは現在使用できない。	
out	java.io.PrintStream	Web ブラウザーに書き込むオブジェクト。	
config	com.ibm.db2e.jsp.server.DB2eJspConfig	この JSP ページの DB2eJspConfig。	`java.lang.String getInitParameter(java.lang.String name)`
exception	java.lang.Throwable	JSP ページの実行中にスローされる例外。	

注: 上記の暗黙オブジェクトのタイプの一部は、 DB2 Everyplace の JSP サポートの実装のために、JSP 1.1 のタイプと異なります。

スクリプト・エレメント:

宣言:

説明 宣言を使用して、JSP ページで使用する Java 変数およびメソッドを宣言します。宣言は、JSP ページ用の Java クラスのメンバー変数 (フィールドおよびメソッド) です。

構文

```
<%!declaration(s) %>
```

例

```
<%!
String name = "Joe Smith";

public String getName() {
 return name;
}
%>
```

標準のアクション:

<jsp:useBean>:

説明 jsp:useBean アクションは、使用可能な「ページ」有効範囲内に定義された Java プログラム言語オブジェクトのインスタンスと与えられた ID とを、同じ ID を持つ新しく宣言されたスクリプト変数を介して関連付けます。

構文

```
<jsp:useBean id="name" scope="page|request|session|application" typeSpec/>
typeSpec ::=
class="className"
```

属性 このタグの 3 つの属性は次のとおりです。

- **id** - この Bean の ID を指定します。この名前を JSP ページで再使用しないでください。この属性は必須です。
- **scope** - この属性は指定しても無視されます。デフォルト有効範囲の「ページ」が使用されます。
- **class** - この Bean が表すクラスを指定します。この属性は必須です。

例 <jsp:useBean id=″masterViewDBBean″
class=″Query1HTMLResultsMasterViewBean″ />

<jsp:setProperty>:

説明 jsp:setProperty アクションはプロパティーの値を Bean 内に設定します。

構文

```
<jsp:setProperty name="beanName" prop_expr />
prop_expr ::=
property="propertyName" value="propertyValue"
propertyValue ::= string
値 propertyValue は、要求時の属性値にすることもできます。
propertyValue ::= expr_scriptlet
```

属性 このタグの 3 つの属性は次のとおりです。

- **name** - このアクションが現れる前に <jsp:useBean> エレメントにより定義された Bean インスタンスの名前。Bean インスタンスには、設定したいプロパティーが入っていなくてはなりません。この属性は必須です。
- **property** - 設定したい値を持つ Bean プロパティーの名前。この属性は必須です。
- **value** - 指定されたプロパティーに割り当てる値。この属性は要求時の属性表現を値として受け入れます。この属性は必須です。

例

```
<jsp:setProperty name="masterViewDBBean" property="username"
value='<%=config.getInitParameter("username")%>' />
```

スクリプトレット:

説明 スクリプトレットを使用して、任意の有効な Java コード・フラグメントを保持します。これらのコード・フラグメントは JSP ページのソース・コード内に置かれ、JSP ページの他のエレメントと関連します。

構文

```
<% scriptlet %>
```

例

```
<%
try {
String name = Query1DBBean.getString(1);
out.println("Name = " + name);
}
catch (SQLException e) {

}
%>
```

式:

説明 式は、データ・タイプのストリング表記です。 式は照会および HTML コメントで使用できます。 アプリケーションは実行時に式を評価し、式をストリングに変換します。

構文

```
<%= expression %>
```

例

```
<%= new java.util.Date() %>
```

注: *__db2ejsp__* で始まる変数名はキーワードであり、内部で使用されます。 このような変数は JSP ページで使用しないでください。

関連した概念:

- 38 ページの『DB2 Everyplace の JSP サポートの概要』
- 37 ページの『DB2 Everyplace の JSP アプリケーションの開発』
- 127 ページの『第 16 章 JSP サンプル・アプリケーション』

関連したタスク:

- 39 ページの『Windows ワークステーションでの JSP サポートの検証』
- 40 ページの『Windows CE デバイスでの JSP 開発のためのセットアップ』

関連した解説:

- 『JSP アプリケーション・データベース・アクセスのための IBM カスタム・タグ』

JSP アプリケーション・データベース・アクセスのための IBM カスタム・タグ

以下のタグは、 DB2 Everyplace データベースにアクセスするために JSP アプリケーションで使用できる、IBM カスタム・タグです。

<tsx:dbconnect>:

説明 このタグは、DB2 Everyplace JDBC ドライバーを使用して、指定された DB2 Everyplace データベースへの接続を確立します。

構文

```
<tsx:dbconnect
id="connection_id"
driver="com.ibm.db2e.jdbc.DB2eDriver"
url="jdbc:db2e:database"
userid="db_user"
passwd="user_password">
</tsx:dbconnect>
```

属性 このタグの 5 つの属性は、次のとおりです。

- **id** - この接続の ID を指定します。 この名前を JSP ページで再使用しないでください。 この属性は必須です。
- **driver** - DB2 Everyplace JDBC ドライバーを指定します。この属性は必須です。
- **url** - DB2 Everyplace データベースを指定します。*database* は、DB2 Everyplace データベースのパスを表します。この属性は必須です。
- **userid** - アクセスするデータベースの有効なユーザー ID を指定します。この属性はオプショナルです。
- **passwd** - userid 属性のユーザー・パスワードを指定します。userid が指定された場合、この属性は必須です。

例

```
<tsx:dbconnect
id="conn"
driver="com.ibm.db2e.jdbc.DB2eDriver"
url="jdbc:db2e:sample/data/" >
</tsx:dbconnect>
```

<tsx:dbquery>:

説明 このタグは、 <tsx:dbconnect> タグを通じて指定された接続を使用してデータベースに照会をサブミットし、カーソルが結果セットの最初の行を指す java.sql.ResultSet オブジェクトを作成します。 この結果セットは、この照会の ID および java.sql.ResultSet についての DB2 Everyplace JDBC インターフェースを使用して参照できます。

構文

```
<tsx:dbquery id="query_id" connection="connection_id" limit="value">
select_SQL_statement
</tsx:dbquery>
```

属性 このタグの属性は次のとおりです。

- **id** - この照会の ID を指定します。 この照会 ID を JSP ページで再使用しないでください。 この属性は必須です。
- **connection** - この JSP ファイルでの <tsx:dbconnect> タグの ID を指定します。この属性は必須です。
- **limit** - 照会で戻すことができる行の最大数を指定します。この属性はオプショナルです。

パラメーター

このタグについて有効なパラメーターは次のとおりです。

- **select_SQL_statement** - データベースにサブミットしたい SQL 照会を指定します。 この SQL 照会ステートメントには動的データを含むことができます。

例

```
<tsx:dbquery id="Query1DBBean" connection="conn">
select <%= request.getParameter("column") %> from vnperson
</tsx:dbquery>
```

<tsx:dbmodify>:

説明 このタグは、<tsx:dbconnect> タグを通じて指定された接続を使用してデータベース内のデータを変更するコマンドをサブミットします。このタグには結果がありません。

構文

```
<tsx:dbmodify connection="connection_id">
modify_command
</tsx:dbmodify>
```

属性 このタグの属性は次のとおりです。

- **connection** - この JSP ファイルでの <tsx:dbconnect> タグの ID を指定します。この属性は必須です。

パラメーター

このタグについて有効なパラメーターは次のとおりです。

- **modify_command** - データを変更するためにデータベースに対してサブミットしたい SQL コマンドを指定します。この modify コマンドには動的データが含まれます。

例

```
<tsx:dbmodify connection="conn">
update vnperson set Name = '<%=Name%>' where ID = '<%=id%>'
</tsx:dbmodify>
```

<tsx:repeat>:

説明 このタグを使用して、照会結果内の各行をループします。start および stop 属性によってループ処理を制御します。start および stop 属性を指定しない場合は、 <tsx:getProperty> タグによって参照される結果セットのカーソルが結果セットの最後に到達すると、ループが終了します。 このタグはネストできます。

構文

```
<tsx:repeat index="name" start="starting_index" stop="ending_index">
repeat_block
</tsx:repeat>
```

属性 このタグの属性は次のとおりです。

- **index** - このタグの索引の ID を指定します。この属性はオプショナルです。
- **start** - 反復ブロックを処理する前にスキップする行数を指定します。デフォルトは 0 です。この属性はオプショナルです。
- **stop** - この反復ブロックを終了する索引値を指定します。デフォルトは 2,147,483,647 KB です。この属性はオプショナルです。

パラメーター

このタグについて有効なパラメーターは次のとおりです。

- **repeat_block** - <tsx:getProperty> タグ構文、およびコンテンツのフォーマット設定に使用される HTML タグを含む、HTML タグ付けのブロックを指定します。反復ブロックに <tsx:getProperty> タグを配置すると、反復ブロックが処理されるたびに、カーソルが次の行に進みます。

例

```
<TABLE border="1">
<TR>
<TH>Name</TH>
</TR>

<tsx:repeat>
<TR>
<TD>
<tsx:getProperty name="Query1DBBean" property="Name" />
</TD>
</TR>
</tsx:repeat>
</TABLE>
```

<tsx:getProperty>:

説明 このタグは ResultSet Bean の値を検索して、 JSP ページ (結果的には HTML ページ) に表示します。 <tsx:repeat> ブロック・タグの内部にこのタグを配置すると、反復ブロックが処理されるたびに、ResultSet Bean のカーソルが次の行に進みます。

構文

```
<tsx:getProperty name="bean_name" property="property_name" />
```

属性 このタグの属性は次のとおりです。

- **name** - すでにこの JSP ファイル内で <tsx:dbquery> タグを使用して宣言した ResultSet Bean の名前を指定します。
- **property** - アクセスする ResultSet Bean の列を指定します。

例

```
<tsx:getProperty name="Query1DBBean" property="FIRSTNAME" />
```

関連した概念:

- 38 ページの『DB2 Everyplace の JSP サポートの概要』
- 37 ページの『DB2 Everyplace の JSP アプリケーションの開発』
- 127 ページの『第 16 章 JSP サンプル・アプリケーション』

関連したタスク:

- 39 ページの『Windows ワークステーションでの JSP サポートの検証』

- 40 ページの『Windows CE デバイスでの JSP 開発のためのセットアップ』

関連した解説:

- 49 ページの『サポートされる JSP バージョン 1.1 サブセット』

JSP アプリケーションのトラブルシューティング

トラブルシューティング:

ミニ HTTP Web サーバー・コンソール出力:

1. WebSphere Studio Application Developer 関連
   - java.lang.NoClassDefFoundError: javax/sql/DataSource

     WSAD v4.0.3+ には、このエラーの修正が入っています。WSAD v4.0.3 からの jar である、<DB2Everyplace>¥SDK¥JSP¥sample¥jsp¥VNSchedule_wsad40 内の dbbeans.jar が使用できます。
   - 「列タイプ名 <name> の検索値が判別できません。検索可能 = true と想定しました。(Cannot determine search value for column type name <name>. Assumed searchable = true.)」

     このメッセージは dbbeans.jar により生成されており、無視してかまいません。

Web ブラウザー:

- 「お探しのページが見つかりません。 (The page you are looking for cannot be found.)」

  「http://localhost/」の URL に接続しようとしたときに Pocket Internet Explorer がこのメッセージを表示した場合は、ご使用の Windows CE デバイスは恐らく Pocket PC v3.0.11171 より古いものです。Web ブラウザーの修正を入手する必要があります。(Pocket Internet Explorer を使用して localhost に接続する方法については、インターネット上で検索してください。) ミニ HTTP Web サーバーが稼動していることを確認してください。

アプリケーションの開発:

- JSP ページに加えた変更内容が Web ブラウザーに反映されていない

  これが発生した場合、ご使用の Web ブラウザーのインターネット一時ファイルのフォルダーかキャッシュに入っているファイルを削除する必要がある場合があります。

関連した概念:

- 38 ページの『DB2 Everyplace の JSP サポートの概要』
- 37 ページの『DB2 Everyplace の JSP アプリケーションの開発』
- 127 ページの『第 16 章 JSP サンプル・アプリケーション』

関連したタスク:

- 39 ページの『Windows ワークステーションでの JSP サポートの検証』
- 40 ページの『Windows CE デバイスでの JSP 開発のためのセットアップ』

関連した解説:

- 53 ページの『JSP アプリケーション・データベース・アクセスのための IBM カスタム・タグ』

第 7 章 .NET アプリケーションの開発

この章では、.NET を使用した DB2 Everyplace アプリケーションの開発に関する情報を提供します。説明されているトピックは、以下のとおりです。

- 『同期サポート』
- 63 ページの『.NET アプリケーションの作成のサポート』

同期サポート

このセクションでは、DB2 Everyplace .NET 同期サポートに関する情報を提供します。説明されているトピックは、以下のとおりです。

- 『ISync.Net API ファイルの場所』
- 60 ページの『ISync.NET API の使用』
- 61 ページの『ISyncComponent の使用』
- 62 ページの『ISync.NET API を使用した簡単なアプリケーション例』

ISync.Net API ファイルの場所

DB2 Everyplace 同期クライアントは、開発者が DB2 Everyplace 同期サーバー用の管理されたアプリケーションを作成できるようにする、2 つの API を提供しています。これらの API には、以下のものが含まれます。

- ISyncComponent
- ISync.Net

ISyncComponent は ISync.Net より小型ですが、ビジュアル・デザインの機能を使用したい開発者のために、ビジュアル・デザイン・サポートを提供しています。

データベース・エンジン用の .Net プロバイダーの詳細については、 63 ページの『クライアント・データベースでアプリケーションを作成する際の .NET サポートの概要』を参照してください。

表 5. ISync.NET 管理のプロバイダーの場所とネーム・スペース

使用可能なプロバイダー	ネーム・スペース	サポートされるプラットフォーム	場所、¥DB2Everyplace¥Clients
.NET Framework 用非ユニコード	IBM.Data.Sync IBM.Data.Sync.DB2e	Win 32	Win32¥Sync¥NMP¥IBM.Data.Sync.DB2e.dll
.NET Framework 用ユニコード	IBM.Data.Sync IBM.Data.Sync.DB2e	Win32 ユニコード	Win32¥Sync¥NMP¥unicode¥IBM.Data.Sync.DB2e.dll
.NET Compact Framework	IBM.Data.Sync IBM.Data.Sync.DB2e.CF	WinCE	WinCE¥Sync¥NMP¥IBM.Data.Sync.DB2e.CF.dll

サンプル・アプリケーション ISync.NET

¥DB2Everyplace¥Clients¥clientapisample¥NMP¥*lang* には、以下の 2 つのサンプル・アプリケーションがあります。

- GoISync

- sample1

ここで、*lang* は C# (cs) または Visual Basic (vb) です。

ISync.NET API 仕様

ISync.NET の API 仕様は、
¥DB2Everyplace¥Clients¥clientapisample¥NMP¥ISync.NET.chm にあります。

関連した概念:

- 62 ページの『ISync.NET API を使用した簡単なアプリケーション例』
- 63 ページの『クライアント・データベースでアプリケーションを作成する際の .NET サポートの概要』
- 68 ページの『WinCE および Win32 用のサンプル DB2 Everyplace .NET Data Provider アプリケーション・コード』

関連したタスク:

- 『ISync.NET API の使用』
- 61 ページの『ISyncComponent の使用』
- 64 ページの『DB2 Everyplace .NET Data Provider を使用した ADO.NET アプリケーションの開発の概要』

ISync.NET API の使用

ISync.NET 用の API 仕様は
¥DB2Everyplace¥Clients¥clientapisample¥NMP¥ISync.NET.chm にあります。

前提条件:

ソフトウェア要件

- DB2 Everyplace バージョン 8.1.4、 ベータ 1
- Microsoft. NET Standard Framework 1.0 (Visual Studio 2002 に同梱) — Win32 でのアプリケーション開発に必要
- Microsoft .NET Compact Framework (Visual Studio 2003 に同梱) — WinCE でのアプリケーション開発に必要

ISync.NET プロバイダーは、プラットフォームおよび言語からは独立していますが、依然として基礎となるネイティブ同期クライアント・ライブラリーに依存しています。プロバイダーと同期クライアント・ライブラリーの両方が、アプリケーション実行時にユーザー・パスに含まれていなければなりません。DB2 Everyplace のインストール時に、ユーザー・パスを更新してください。

クライアント・ライブラリーの詳細については、「*DB2 Everyplace インストールおよびユーザーズ・ガイド*」の第 3 章 『モバイル・デバイスでの DB2 Everyplace のインストール』を参照してください。

手順:

Isync.NET をアプリケーションで使用するには、すべての Visual Studio Frameworks およびアプリケーション・タイプで以下のステップを実行する必要があります。

1. Microsoft Visual Studio .NET で、選択した言語で新規プロジェクトを作成します。
2. アプリケーションに DB2 Everyplace ネーム・スペースをインポートします。以下は標準フレームワークの例です。

```
[Visual Basic]
     Imports IBM.Data.Sync
     Imports IBM.Data.Sync.DB2e
     [C#]
     using IBM.Data.Sync;
     using IBM.Data.Sync.DB2e;
```

   詳細については、¥DB2Everyplace¥Clients¥clientapisample¥NMP にあるサンプルの同期アプリケーションを参照できます。
3. リファレンスを追加します。
   a. Visual Studio でプロジェクト名を右クリックして「**リファレンスの追加 (Add Reference)**」を選択します。
   b. 「プロジェクト」タブで、**IBM.Data.Sync.DB2e.dll** の位置をブラウズします。
   c. コマンド行で、**csc /t:exe /r:IBM.Data.Sync.DB2e.dll ISyncSample.cs** と入力します。

関連した概念:

- 59 ページの『ISync.Net API ファイルの場所』
- 62 ページの『ISync.NET API を使用した簡単なアプリケーション例』
- 63 ページの『クライアント・データベースでアプリケーションを作成する際の .NET サポートの概要』
- 68 ページの『WinCE および Win32 用のサンプル DB2 Everyplace .NET Data Provider アプリケーション・コード』

関連したタスク:

- 『ISyncComponent の使用』
- 64 ページの『DB2 Everyplace .NET Data Provider を使用した ADO.NET アプリケーションの開発の概要』

ISyncComponent の使用

手順:

ISyncComponent は、標準フレームワークに関して最小限の設計サポートも提供します。この基本サポートにより、フォームへのドラッグ・アンド・ドロップと、ConnectionString (サーバー、ポート、およびユーザー名) および TargetPath (データのターゲット・ディレクトリー) のプロパティーの変更が可能になります。Visual Studio Windows アプリケーションを開発している場合は、必ず DB2 Everyplace のコンポーネント **IBM.Data.Sync.DB2e.dll** を自分の「**ツールボックス**」に追加するようにしてください。

注: このプロセスが正常に完了するためには、ネイティブ同期クライアント・ライブラリーがすでにユーザー・パスになければなりません。

標準フレームワークの場合、IBM.Data.Sync.DB2e.ISyncComponent を使用して、より簡単な API を使用するオプションがあります。

```
ISyncComponent comp1 = new ISyncComponent();
comp1.ConnectionString = SERVER=localhost;PORT=80;UID=username;PWD=password;
comp1.TargetPath = data;
comp1.Sync();
comp1.Close();
```

関連した概念:

- 59 ページの『ISync.Net API ファイルの場所』
- 『ISync.NET API を使用した簡単なアプリケーション例』
- 63 ページの『クライアント・データベースでアプリケーションを作成する際の .NET サポートの概要』
- 68 ページの『WinCE および Win32 用のサンプル DB2 Everyplace .NET Data Provider アプリケーション・コード』

関連したタスク:

- 60 ページの『ISync.NET API の使用』
- 64 ページの『DB2 Everyplace .NET Data Provider を使用した ADO.NET アプリケーションの開発の概要』

ISync.NET API を使用した簡単なアプリケーション例

このトピックには、ISync.NET API を使用する方法についてクイック・リファレンスとなる例が含まれています。

```
// Synchronization properties
private Hashtable userProps = new Hashtable();

// Get an instance DB2eISyncProvider
ISyncProvider provider = DB2eISyncProvider.GetInstance();

// Set up properties
userProps.Add("isync.user", username);
userProps.Add("isync.password", password);

// Get an instance of synchronization service from the provider
ISyncService service = provider.CreateSyncService(http://localhost:80, userProps);

// Get an instance of the configuration store
ISyncConfigStore config = service.GetConfigStore(data);

// Get an instance of the sync driver to perform synchronization
ISyncDriver syncer = config.GetSyncDriver();

// Perform synchronization
syncer.Sync();

// Close objects
syncer.Close();
config.Close();
service.Close();
```

関連した概念:

- 59 ページの『ISync.Net API ファイルの場所』
- 63 ページの『クライアント・データベースでアプリケーションを作成する際の .NET サポートの概要』
- 68 ページの『WinCE および Win32 用のサンプル DB2 Everyplace .NET Data Provider アプリケーション・コード』

関連したタスク:

- 60 ページの『ISync.NET API の使用』
- 61 ページの『ISyncComponent の使用』

- 64 ページの『DB2 Everyplace .NET Data Provider を使用した ADO.NET アプリケーションの開発の概要』

.NET アプリケーションの作成のサポート

このセクションでは、.NET アプリケーションの作成のサポートに関する情報を提供します。説明されているトピックは、以下のとおりです。

- 『クライアント・データベースでアプリケーションを作成する際の .NET サポートの概要』
- 64 ページの『DB2 Everyplace .NET Data Provider を使用した ADO.NET アプリケーションの開発の概要』
- 68 ページの『WinCE および Win32 用のサンプル DB2 Everyplace .NET Data Provider アプリケーション・コード』

クライアント・データベースでアプリケーションを作成する際の .NET サポートの概要

DB2 Everyplace では、開発者が ADO.NET API を使用するアプリケーションを作成して、DB2 Everyplace データベースによって管理されるデータを操作できるようにするツールを提供しています。DB2 Everyplace には、2 つの .NET データ・プロバイダーが含まれています。1 つのプロバイダーは .NET Framework 1.0 で稼働し、 もう 1 つのプロバイダーは .NET Compact Framework で稼動します。これらのプロバイダーまたは API は、以下の場所にあります。

- **Win32 の場合:**

  ```
  ¥DB2Everyplace¥Clients¥Win32¥database¥nmp¥IBM.Data.DB2.DB2e.dll
  ```

- **WinCE の場合:**

  ```
  ¥DB2Everyplace¥Clients¥WinCE¥database¥nmp¥IBM.Data.DB2.DB2e.CF.dll
  ```

API 仕様は、以下の場所にあります。

```
¥DB2Everyplace¥Clients¥Win32¥database¥nmp¥doc¥readme.html
```

同期クライアント用の .NET プロバイダーの詳細については、「*DB2 Everyplace 同期サーバー 管理ガイド* 」の、『Windows ソース・システムのソース・データベースの構成 』というセクションを参照してください。

今まで Microsoft ODBC .NET Data Provider を使用していたプログラマーがスムーズに移行できるように、新しい DB2 Everyplace .NET Data Provider のインターフェースは、Microsoft ODBC .NET Data Provider とほぼ同じです。例えば、Microsoft ODBC .NET Data Provider には OdbcConnection クラスがありますが、IBM DB2 Everyplace .NET Data Provider には 同等の関数クラスとして DB2eConnection があります。同じように、他のクラス名の「Odbc」を「DB2e」に置き換えることにより、対応する DB2 Everyplace .NET Data Provider クラスを得ることができます。

関連した概念:

- 59 ページの『ISync.Net API ファイルの場所』
- 62 ページの『ISync.NET API を使用した簡単なアプリケーション例』
- 68 ページの『WinCE および Win32 用のサンプル DB2 Everyplace .NET Data Provider アプリケーション・コード』

関連したタスク:

- 61 ページの『ISyncComponent の使用』
- 『DB2 Everyplace .NET Data Provider を使用した ADO.NET アプリケーションの開発の概要』

DB2 Everyplace .NET Data Provider を使用した ADO.NET アプリケーションの開発の概要

DB2 Everyplace .NET Data Provider 用のネーム・スペースは、以下の通りです。

- **.NET Compact Framework で実行する場合:** IBM.Data.DB2.DB2e.CF
- **.NET Framework で実行する場合:** IBM.Data.DB2.DB2e

DB2 Everyplace .NET Data Provider は、DB2 Everyplace データ・ソースへの接続、コマンドの実行、および結果の検索についての機能を提供します。これらの結果は、直接処理するか、または ADO.NET DataSet に入れて、後で切断状態の間に処理することができます。DataSet に入っている間は、データをユーザーに公開するか、複数のソースからの他のデータと組み合わせるか、または層の間でリモートでの受け渡しを行うことができます。したがって、DataSet に入っている間にデータに対して実行されるすべての処理は、データ・ソースと調整できます。

DB2 Everyplace .NET Data Provider は、軽量化するように設計されています。DB2 Everyplace とユーザー・コードの間は最小限の層で構成されており、パフォーマンスを犠牲にせずに機能を拡張できます。

DB2 Everyplace .NET Data Provider クラスは、他の .NET Framework クラスやインターフェースからのメンバーを継承またはインプリメントします。このプロバイダー・ドキュメンテーションには、これらのクラスのそれぞれに含まれるサポート・メンバーのサマリーが含まれています。特定の継承メンバーに関する詳細については、Microsoft® .NET Framework SDK の該当するトピックを参照してください。

前提条件:

表 6. DB2 Everyplace .NET Data Provider を使用する場合の前提条件

コンポーネント	最小必要要件
Microsoft.NET Framework	Microsoft.NET Framework 1.0 アプリケーション開発用の DB2 Everyplace .NET Data Provider をインストールする前に、インストールされている必要があります。
Microsoft Visual Studio.NET 2003	モバイル・アプリケーション開発用の Microsoft Visual Studio.NET 2003
Microsoft.NET Compact Framework	モバイル開発用の Microsoft .NET Compact Framework 1.0 モバイル開発用の DB2 Everyplace .NET Data Provider をインストールする前に、デバイスにインストールされている必要があります。

表 6. DB2 Everyplace .NET Data Provider を使用する場合の前提条件 (続き)

コンポーネント	最小必要要件
DB2 Everyplace 製品	• バージョン 8.1.4 以降の DB2e.dll • バージョン 8.1.4 以降の AgentProxy.dll (リモート・ストアード・プロシージャー呼び出し用に必要) • バージョン 8.1.4 以降の wbxmllib.dll (リモート・ストアード・プロシージャー呼び出し用に必要) • DB2 Everyplace 同期サーバー ・バージョン 8.1.4 以降 (リモート・ストアード・プロシージャー呼び出し用に必要) DB2e.dll、AgentProxy.dll、および wbxmllib.dll はネイティブ・ライブラリーであるため、プロセッサーに依存します。そのため、オペレーティング・システムでは、DB2 Everyplace .NET Data Provider を適切に機能させるために、これらのネイティブ・ライブラリーの位置を (例えば、環境変数 PATH を設定して) 指定する必要があります。

制約事項:

プロバイダーの制限:

- 主キー列の更新は、現在 DB2 Everyplace では許可されていません。
- リモート・ストアード・プロシージャー呼び出しを使用した結果セット検索では、結果セットのサイズに制限があります。
- ローカル・ストアード・プロシージャー呼び出しはサポートされていません。
- サポートしていないメソッドやプロパティーについては、System.NotSupportedException がスローされます。

スレッド・セーフティー:

このプロバイダーの共通非インスタンス・メンバーは、マルチスレッド操作に対してスレッド・セーフです。インスタンス・メンバーのスレッド・セーフは保証されていません。

手順:

DB2 Everyplace .NET データ・プロバイダーを構成するコア・オブジェクトが 4 つあります。以下の表に、これらのオブジェクトとその機能を示します。

表 7. DB2 Everyplace .NET Data Provider、コア・オブジェクト

オブジェクト	説明
DB2eConnection	DB2 Everyplace データ・ソースへの接続を確立し、トランザクション を開始できる。
DB2eCommand	DB2 Everyplace サーバーでコマンドを実行し、パラメーター を公開する。

表 7. *DB2 Everyplace .NET Data Provider*、コア・オブジェクト *(続き)*

オブジェクト	説明
DB2eDataAdapter	*DataSet* にデータを追加し、DB2 Everyplace データ・ソースで更新を解決する。
DB2eDataReader	DB2 Everyplace データ・ソースからのデータの順方向のみのストリームを公開し、読み取る。

DB2 .NET Data Provider には、前述の表にリストされたコア・クラスに加えて、以下の表にリストされているクラスも含まれています。

表 8. *DB2 Everyplace .NET Data Provider*、*追加クラス*

オブジェクト	説明
DB2eCommandBuilder	A*DB2eDataAdapter* のコマンド・プロパティーを自動的に生成したり、またはストアード・プロシージャーからパラメーターを取得して *DB2eCommand* オブジェクトの *DB2eParameters* コレクションのデータを追加するヘルパー・オブジェクト。注: *DB2eCommandBuilder* を使用することは、非効率な SQL ステートメントや、場合によっては無効な SQL ステートメントを生成する可能性があるため、お勧めできません。
DB2eError	DB2 Everyplace データ・ソースから戻された警告やエラーの情報を公開する。
DB2eException	DB2 Everyplace データ・ソースでエラーが見つかると戻される。クライアントで見つかったエラーの場合、.NET Data Provider が .NET Framework 例外をスローする。
DB2eParameter	コマンドおよびストアード・プロシージャー用の入力、出力、および戻り値のパラメーターを定義する。
DB2eTransaction	DB2 Everyplace データ・ソースにおけるトランザクションにコマンドを入れられるようにする。

DB2 Everyplace .NET Data Provider を使用するには、以下のコードが示すように、IBM.Data.DB2.DB2e、つまりネーム・スペースに対する imports ステートメントまたは using ステートメントをアプリケーション .DLL に追加する必要があります。

[Visual Basic]

```
Imports IBM.Data.DB2.DB2e
```

[C#]

```
using IBM.Data.DB2.DB2e;
```

また、コードをコンパイルする際に .DLL への参照を追加する必要があります。例えば、Microsoft® Visual C#™ プログラムをコンパイルしている場合、コマンド行には次の行を追加しなければなりません。

```
csc /r:IBM.Data.DB2.DB2e.dll
```

.NET Compact Framework の場合、ネーム・スペースは IBM.Data.DB2.DB2e.CF であり、アプリケーションは IBM.Data.DB2.DB2e.CF.dll アセンブリーを参照する必要があります。

このネーム・スペースをうまく使用する方法については、以下の DB2 Everyplace.NET Data Provider クラスの資料を参照してください。

- DB2eDataAdapter
- DB2eCommand
- DB2eConnection
- DB2eDataReader

.NET Framework 内で DB2 Everyplace .NET Data Provider がどのように機能するかについての詳細は、IBM.Data.DB2.DB2e 階層を参照してください。

表 9. クラス

オブジェクト	説明
DB2eCommand	データ・ソースに対して実行する SQL ステートメントまたはストアード・プロシージャーを表す。このクラスは継承できない。
DB2eCommandBuilder	関連するデータ・ソースを持つ *DataSet* に対して行われた変更を調整するのに使用する単一表コマンドを自動的に生成する。このクラスは継承できない。
DB2eConnection	データ・ソースへのオープン接続を表す。
DB2eDataAdapter	*DataSet* を充てんしデータ・ソースを更新するのに使用する、データ・コマンドのセットおよびデータ・ソースへの接続を表す。このクラスは継承できない。
DB2eDataReader	データ・ソースからのデータ行の順方向のみのストリームを読み取る方法を提供する。このクラスは継承できない。
DB2eError	データ・ソースから戻された警告またはエラーに関する情報を収集する。このクラスは継承できない。
DB2eException	DB2 Everyplace データ・ソースから警告またはエラーが戻される場合に生成される例外。このクラスは継承できない。
DB2eParameter	*DB2eCommand* へのパラメーターと、オプションで *DataColumn* へのマッピングを表す。このクラスは継承できない。
DB2eTransaction	データ・ソースで作成される SQL トランザクションを表す。このクラスは継承できない。

表 10. 代行

代行	説明
DB2eInfoMessageEventHandler	DB2eConnection の InfoMessage イベントを処理するメソッドを表す。
DB2eRowUpdatedEventHandler	DB2eDataAdapter の RowUpdated イベントを処理するメソッドを表す。
DB2eRowUpdatingEventHandler	DB2eDataAdapter の RowUpdating イベントを処理するメソッドを表す。

表 11. 列挙型

列挙型	説明
DB2eType	フィールド、プロパティー、または DB2eParameter のデータ・タイプを指定する。

表 12. DB2 Everyplace .NET Provider 接続ストリング・キーワード

キーワード	説明
Database	データベースの場所。例えば、C:¥data1¥
UID	ユーザー ID
PWD	パスワード

C# の例:

```
string connString = @"Database=C:¥data1¥; UID=user; PWD=userpwd";
```

関連した概念:

- 59 ページの『ISync.Net API ファイルの場所』
- 62 ページの『ISync.NET API を使用した簡単なアプリケーション例』
- 63 ページの『クライアント・データベースでアプリケーションを作成する際の .NET サポートの概要』
- 『WinCE および Win32 用のサンプル DB2 Everyplace .NET Data Provider アプリケーション・コード』

関連したタスク:

- 60 ページの『ISync.NET API の使用』
- 61 ページの『ISyncComponent の使用』

WinCE および Win32 用のサンプル DB2 Everyplace .NET Data Provider アプリケーション・コード

DB2 Everyplace .NET Data Provider を使用して WinCE および Win32 用のアプリケーションを開発する方法を示すサンプル・アプリケーションは、以下の 2 つです。

- DB2eSample1.cs
- DB2eSample2.sc

いずれのファイルも以下のディレクトリーにあります。

- **Win32 および WinCE の場合**

  ```
  ¥DB2Everyplace¥Clients¥Win32¥database¥nmp¥samples
  ```

サンプル・アプリケーションの例は、以下の通りです。

```
using System;
using System.Text;
using System.Data;

using IBM.Data.DB2.DB2e;

/*
```

```
 * Sample1
 *
 * The following example creates a table, insert some rows to it, fetches
 * all the rows from the table, and finally drops the table.
 *
 */
namespace IBM.Data.DB2.DB2e.Samples
{
        class DB2eSample1
        {
                /// <summary>



     /// The main entry point for the application.
    /// </summary>
    [STAThread]
    static void Main(string[] args)
      {
        DB2eConnection conn   = null;
        DB2eCommand cmd       = null;
        DB2eDataReader reader = null;
         String connString     = @"database=.¥; uid=user1; pwd=user1";
         int rowsAffected      = 0;

          try
               {
                conn = new DB2eConnection(connString);
                conn.Open();

                  Console.WriteLine("creating table t1...");
                  cmd = new DB2eCommand("create table t1 (c1 int primary key not null,
       c2 smallint, c3 char(10), c4 varchar(10), c5 decimal(8,2), c6 date,
       c7 time, c8 timestamp )", conn);
                      rowsAffected = cmd.ExecuteNonQuery();
                      Console.WriteLine("inserting a row into table t1...");
                      cmd.CommandText = "insert into t1 values (1, 10, 'John',
        'Yip', null, current date, current time, current timestamp)";
                 rowsAffected = cmd.ExecuteNonQuery();
                 Console.WriteLine("inserting a row into table t1...");
                 cmd.CommandText = "insert into t1 values (2, 20, 'Mary', 'Jann',
        2.2, current date, current time, current timestamp)";
                 rowsAffected = cmd.ExecuteNonQuery();
                 cmd.CommandText = "select * from t1";
                 Console.WriteLine("fetching resultset from table t1...");
                 reader = cmd.ExecuteReader();
                     while (reader.Read())
                       {
                         if (!reader.IsDBNull(0))
                            Console.Write(reader.GetInt32(0) + "¥t");
                         else
                             Console.Write("NULL " + "¥t");
                         if (!reader.IsDBNull(1))
                             Console.Write(reader.GetInt16(1) + "¥t");
                         else
                             Console.Write("NULL " + "¥t");
                         if (!reader.IsDBNull(2))
                             Console.Write(reader.GetString(2) + "¥t");
                         else
                             Console.Write("NULL " + "¥t");
```

```
            if (!reader.IsDBNull(3))
                Console.Write(reader.GetString(3) + "¥t");
            else
                Console.Write("NULL " + "¥t");
            if (!reader.IsDBNull(4))
                Console.Write(reader.GetDecimal(4) + "¥t");
            else
                Console.Write("NULL " + "¥t");
            if (!reader.IsDBNull(5))
                Console.Write(reader.GetDate(5) + "¥t");
            else
                Console.Write("NULL " + "¥t");
            if (!reader.IsDBNull(6))
                Console.Write(reader.GetTime(6) + "¥t");
            else
                Console.Write("NULL " + "¥t");
            if (!reader.IsDBNull(7))
                Console.Write(reader.GetDateTime(7) + "¥t");
            else
                Console.Write("NULL " + "¥t");
                Console.WriteLine();
            }
          reader.Close();
          reader = null;
          Console.WriteLine("dropping table t1...");
          cmd.CommandText = "drop table t1";
          cmd.ExecuteNonQuery();
        }
          catch (DB2eException e1)
         {
              int cnt = e1.Errors.Count;
              for (int i=0; i < cnt; i++)
                {
                  Console.WriteLine("Error #" + i + "¥n" +
                     "Message: " + e1.Errors[i].Message + "¥n" +
                     "Native: " + e1.Errors[i].NativeError.ToString() + "¥n" +
                     "SQL: " + e1.Errors[i].SQLState + "¥n");
                }
            }
          catch (Exception ex)
            {
                  Console.WriteLine(ex.Message);
            }
          finally
             {
                  if (reader != null)
                     {
                        reader.Close();
                        reader = null;
                     }
                  if (conn != null)
                     {
                        conn.Close();
                        conn = null;
                     }
             }
    }  // end of Main
```

```
        }  // end of class

} // end of namespace
```

関連した概念:

- 59 ページの『ISync.Net API ファイルの場所』
- 62 ページの『ISync.NET API を使用した簡単なアプリケーション例』
- 63 ページの『クライアント・データベースでアプリケーションを作成する際の .NET サポートの概要』

関連したタスク:

- 60 ページの『ISync.NET API の使用』
- 61 ページの『ISyncComponent の使用』
- 64 ページの『DB2 Everyplace .NET Data Provider を使用した ADO.NET アプリケーションの開発の概要』

第 8 章 DB2 Everyplace データベースへの接続

この章では、DB2 Everyplace データベースに接続する方法を説明します。説明されているトピックは、以下のとおりです。

- 『DB2 Everyplace データベース表の概要』
- 『命名による競合の取り扱い』
- 74 ページの『DB2 Everyplace データベースへの接続』
- 75 ページの『接続のシリアライゼーション』
- 76 ページの『読み取り専用メディアにおける DB2 Everyplace データベース』

DB2 Everyplace データベース表の概要

DB2 Everyplace データベースは、複数のシステム・カタログ表と多数のユーザー定義表からなります。表はそれぞれ 2 つのファイルに保管されます。1 つはデータそのものを保管するファイル、もう 1 つは索引として使用するファイルです。 索引はすべて同じ索引ファイルに保管されます。 DB2 Universal Database と異なり、DB2 Everyplace データベースは名前を持たず、カタログまたはアンカタログできません。つまり、データベース名は無視されます。 DB2 Everyplace では、データベースは単にコピーしたり別のロケーションに移動したりできる多数のファイルです。 システム・カタログ表には、ユーザー定義表におけるメタデータが含まれます。 例えば、ユーザー定義表でファイルを除去してもカタログ表の対応するエントリーを削除しなければ、このアクションによって不整合が生じます。 DB2 Everyplace データベースには、以下のシステム・カタログ表が含まれていなくてはなりません。

- DB2eSYSTABLES
- DB2eSYSCOLUMNS
- DB2eSYSRELS
- DB2eSYSUSERS (この表はローカル・データ暗号化を使用する場合に作成されます)

照会時にカタログ表をアクセスするには、区切り文字で区切った ID を使用する必要があります。例えば、次の照会では T という表が存在すればそれを返します。

```
SELECT 1 FROM "DB2eSYSTABLES" WHERE TNAME = 'T'
```

関連した解説:

- 367 ページの『第 19 章 DB2 Everyplace システム・カタログ基本表』

命名による競合の取り扱い

手順:

このトピックでは、ユーザー定義表について、ファイルの命名の競合を処理する方法の例をいくつか示します。アプリケーションが以下の CREATE TABLE ステートメントを実行するとします。

```
CREATE TABLE T (PK INT NOT NULL PRIMARY KEY, A INT)
```

このステートメントが実行されると、DB2 Everyplace は表 *T* について以下の 2 つのファイルを作成します。

- DSY_T (データ)
- DSY_iT (索引)

ユーザーが別の表を作成して *iT* という名前を使用すると、DB2 Everyplace は 2 つの追加ファイル DSY_iT (データ) と DSY_iiT (索引) を作成します。 表 *T* の索引ファイルと表 *iT* のデータ・ファイルは、同じ名前であるために競合が発生します。両方のファイルが DSY_iT という名前になります。 この問題を避けるために、 DB2 Everyplace はファイル名のマッピングをサポートしています。 つまり、ファイル名は DB2 Everyplace によって完全に作成および管理されます。このフィーチャーを使用するには、アプリケーションが接続属性を設定しなければならず、最初の表が作成される前に実行されなければなりません。 例えば CLI では次のようになります。

```
SQLSetConnectAttr(hdbc, SQL_ATTR_FILENAME_FORMAT,
                  (SQLPOINTER)SQL_FILENAME_FORMAT_83, 0)
```

あるいはコマンド行プロセッサーでは次のようになります。

```
DISABLE LONG FILENAME
```

このコマンドが実行されて最初の表が作成されると、生成されるファイルは表 *T* のものになります。

- 0001.DBd
- 0001.DBi

関連した概念:

- 73 ページの『DB2 Everyplace データベース表の概要』
- 75 ページの『接続のシリアライゼーション』

関連したタスク:

- 『DB2 Everyplace データベースへの接続』

DB2 Everyplace データベースへの接続

アプリケーションでは、Win32 プラットフォームの C:¥TEMP フォルダーのような特定のロケーションに、表を作成してアクセスする必要が生じることがよくあります。以下の CLI (コマンド行インターフェース) 呼び出しを使用し、データベース・パスを指定して、データベースに接続することができます。

```
rc = SQLConnect(hdbc, "C:¥¥TEMP¥¥db", SQL_NTS, uid, SQL_NTS, pwd, SQL_NTS);
```

ここで *db* はデータベース名です (DB2 Everyplace では無視されます)。

さらに、次の CLI 呼び出しを使用できます。

```
rc = SQLConnect(hdbc, "C:¥¥TEMP¥¥", SQL_NTS, uid, SQL_NTS, pwd, SQL_NTS);
```

デバイスまたは WinCE オブジェクト・ストアの拡張メモリーへの接続には、特殊なパスの指定が必要です。

- PalmOS での Sony のメモリー・スティックの場合

```
rc = SQLConnect(hdbc, "#0:¥¥", SQL_NTS, uid, SQL_NTS, pwd, SQL_NTS);
```

- WinCE オブジェクト・ストアの場合

```
rc = SQLConnect(hdbc, "@:¥¥", SQL_NTS, uid, SQL_NTS, pwd, SQL_NTS);
```

コマンド行プロセッサーを使用すると、「CONNECT TO」コマンドを使用して特定のロケーションに接続できます。 例えば、以下のコマンドは、Win32 プラットフォームの C:¥TEMP¥ フォルダーに保持されるデータベースに接続します。

```
CONNECT TO C:¥TEMP¥
```

関連した概念:

- 73 ページの『DB2 Everyplace データベース表の概要』

関連したタスク:

- 73 ページの『命名による競合の取り扱い』

接続のシリアライゼーション

DB2 Everyplace データベースに対する接続は一度に 1 つのみです。DB2 Everyplace は、Palm を除いた すべてのプラットフォームで接続のシリアライゼーションをサポートします。接続のシリアライゼーションによって、データ・ソースに対する接続要求がシリアライズされます。特定のデータ・ソースに対して同時に確立できるアクティブな接続は 1 つのみです。その他の接続要求はキューに入れられます。タイムアウト期間は、SQLSetConnectAttr() 関数の SQL_ATTR_LOGIN_TIMEOUT 属性を使用して設定することができます。一般的なシリアライゼーションのシナリオは、以下のとおりです。

- 単一のデータ・ソースへの接続をオープンしようとしている複数のプロセスが直列化される。指定されたデータ・ソースは、同時に 1 つのプロセスに対してのみオープンになります。
- 単一のデータ・ソースへの接続をオープンしようとしている単一プロセス内の複数のスレッドが直列化される。
- 複数の異なるプロセスの 1 つ 1 つが、複数の異なるデータ・ソースに対して並行して 1 つの接続をオープンする。

例として、以下の CLI /JDBC 呼び出しでは接続のタイムアウト期間を 10 に設定します。つまり、同じデータベースに対して別の接続があると、アプリケーションがエラーを受け取ります。

CLI の場合:

```
 int i = 10; // 10 seconds timeout
rc = SQLSetConnectAttr(hdbc, SQL_ATTR_LOGIN_TIMEOUT, (SQLPOINTER) i, 0);
```

JDBC の場合:

```
int waitTime = 10;
String url = "jdbc:db2e:mysample";

Properties prop = new Properties();
prop.setProperty("LOGIN_TIMEOUT", Integer.toString(waitTime));
```

```
Connection con = driver.connect(url,prop);
```

注:

1. デフォルトのタイムアウト期間は 0 秒です。
2. マルチスレッド・アプリケーションでは、あるスレッドを使用してデータベースに接続したら、別のスレッドを使用してデータベースから切断します。
3. アプリケーションがデータベースに正常に接続すると、'DSYLOCK' という名前のファイルが作成されます。アプリケーション・プロセスが異常終了すると、ファイル DSYLOCK は自動的に除去されます。
4. 制限: 接続のシリアライゼーションは、ネットワーク・ドライブに常駐しているデータベースについては動作しないことがあります。
5. JDBC プログラムでは、タイムアウト値はプロパティーとして DriverManager.getConnection() メソッドに渡されると、無視されてゼロに設定されます。

関連したタスク:

- 74 ページの『DB2 Everyplace データベースへの接続』

読み取り専用メディアにおける DB2 Everyplace データベース

DB2 Everyplace データベースおよびアプリケーションは、 CD-ROM や組み込みデバイスの ROM チップなどの読み取り専用メディアから直接実行できます。例えば、季刊の商品カタログのサンプル・アプリケーションが、セールス担当者に CD-ROM で配布されることがあります。セールス担当者は一季ごとに、企業の全商品のカタログを含む CD-ROM と、顧客の特定のニーズに合う商品の商品情報をブラウズ、表示、および照会するための DB2 Everyplace アプリケーションを受け取ります。 DB2 Everyplace アプリケーションは CD-ROM から直接実行され、最初にワークステーションにインストールする必要はありません。

DB2 Everyplace は、アプリケーションが読み取り専用メディアで実行されている (またはファイルが書き込み保護されている) ことを検出すると、読み取り専用モードに設定されます。このモードでは、update、insert、delete、create、および drop のステートメントは使用できません。使用するとエラーが戻されます。一部の選択照会では、DB2 Everyplace が一時表およびファイルを作成する場合があることに注意してください。これらはデフォルトの一時ディレクトリーに作成されます。Win32 プラットフォームでは、このディレクトリーは環境変数 *TEMP* によって指定されます。*TEMP* 環境変数が存在しない場合は、 *TMP* と指定されることもあります。Linux では、`/tmp/` ディレクトリーが使用されます。

このフィーチャーは Win32 プラットフォームおよび Linux プラットフォームでしかサポートされません。

第 9 章 CLI を介したデータの部分検索

ユーザーが CLI を介して DB2 Everyplace テーブルからデータを検索する方法には、以下の 2 つの方法があります。

- アプリケーションは、(*SQLDescribeCol()* を介した結果セットの中の列の知識または前の知識に基づいて) 列値が占有できる最大メモリーを割り振り、*SQLBindCol()* を介してそれをバインドすることを選択できます。
- アプリケーションは、*SQLGetData()* を呼び出して列データをアプリケーションが割り振ったバッファーに取り込むことができます。

バイナリー・データ (BLOB) または文字データ (CHAR や VARCHAR) の場合、列は非常に長くなる可能性があります。アプリケーション開発者は、列全体を保持するだけのバッファーを割り振りたくないか、またはこれだけのバッファーを割り振る余裕がないことがあります。加えて、アプリケーションでは、列の一部分だけしか必要としない場合もあります。このような場合に、データの断片検索が必要となります。

手順:

SQLGetData() の 1 つの機能により、アプリケーションは反復呼び出しを使用して、1 つの列の値を扱いやすい大きさに分けて順番に取得できます。実際、*SQLGetData()* を呼び出すと、SQL_SUCCESS_WITH_INFO (SQLSTATE 01004 付き) が戻され、この列にさらにデータがあることを示します。この列のデータ全体が検索されたことを示す SQL_SUCCESS が戻されるまで、*SQLGetData()* が繰り返し呼び出されます。

構文:

```
SQLRETURN   SQLGetData       (
                SQLHSTMT          StatementHandle,   /* hstmt */
                SQLUSMALLINT      ColumnNumber,      /* icol */
                SQLSMALLINT       TargetType,        /* fCType */
                SQLPOINTER        TargetValuePtr,    /* rgbValue */
                SQLINTEGER        BufferLength,      /* cbValueMax */
                SQLINTEGER   *FAR StrLen_or_IndPtr); /* pcbValue */
```

この場合、一度に *BufferLength* バイトが検索され、*StrLen_or_IndPtr* は残りの バイト数を示します。残りのバイトがある場合、関数の戻り値は SQL_SUCCESS_WITH_INFO (SQLSTATE 01004 付き) になります。それ以外の場合、戻り値が SQL_SUCCESS であれば、*StrLen_or_IndPtr* は、DB2 Everyplace CLI がデータを戻す際に使用できる *TargetValuePtr* バッファーのバイト数を示します。*SQLGetData()* をこのように使用して、C データ・タイプ (*TargetType*) が SQL_C_CHAR または SQL_C_BINARY の場合、あるいは *TargetType* が SQL_C_DEFAULT で列タイプがバイナリーまたは文字ストリングを示す場合に、長い列を検索することができます。

この *SQLGetData()* の機能を使用するには、まずステートメント属性 SQL_ATTR_GETDATA_MODE を SQL_PIECEMEAL_DATA に設定する必要があり

ます。この属性のデフォルト値は、SQL_CHUNK_DATAです。これら 2 つのモードの違いは、SQL_CHUNK_DATA モード (デフォルト・モード) では、切り捨てが発生した場合、SQLGetData() の戻り値 *StrLen_or_IndPtr* がこの列の *合計* バイト数を示し、2 番目の呼び出しでも列の先頭からデータを検索することです。

コード・フラグメントの例:

```
sqlrc = SQLSetStmtAttr(hstmt, SQL_ATTR_GETDATA_MODE, (SQLPOINTER) SQL_PIECEMEAL_DATA, 0);
    SQLCHAR * stmt = (SQLCHAR *) "SELECT blobColumn FROM t1 where c1 = ?";
    sqlrc = SQLPrepare( hstmt, stmt, SQL_NTS ) ;
    sqlrc = SQLBindParameter(hstmt, 1, SQL_PARAM_INPUT, SQL_C_CHAR, ...);
    sqlrc = SQLExecute( hstmt ) ;
    sqlrc = SQLFetch( hstmt );
    /* get BUFSIZ bytes at a time, bufInd indicates number of Bytes LEFT */
    sqlrc = SQLGetData (hstmt, 1, SQL_C_BINARY, (SQLPOINTER) buffer, BUFSIZ, &bufInd);
    while( sqlrc == SQL_SUCCESS_WITH_INFO ) {
        // handle BUFSIZ bytes of blob data in buffer
            :
        sqlrc = SQLGetData (hstmt, 1, SQL_C_BINARY, (SQLPOINTER) buffer, BUFSIZ, &bufInd);
    }
    if (sqlrc == SQL_SUCCESS) {  /* partial buffer on last GetData */
        // handle bufInd bytes of blob data in buffer
            :
    }
```

第 10 章 パラメーター・マーカー

この章では、DB2 Everyplace の照会でパラメーター・マーカーを使用する方法についての情報を提供します。説明されているトピックは、以下のとおりです。

- 『パラメーター・マーカーの概要』
- 『パラメーター・マーカーの使用例』
- 84 ページの『DB2 Everyplace でサポートされているパラメーター・マーカー』

パラメーター・マーカーの概要

何度も実行する必要のある SQL ステートメントについては、SQL ステートメントを 1 度作成し、実行時に入力値を置換するパラメーター・マーカーを使用して照会プランを再利用するのが有利です。

DB2 Everyplace では、パラメーター・マーカーは「?」文字で表されており、SQL ステートメント内の置換するアプリケーション変数の位置を示します。パラメーター・マーカーは番号で参照され、1 から始まり左から右に順に番号が付けられます。SQL ステートメントの実行前に、アプリケーションでは SQL ステートメント内に指定した各パラメーター・マーカーに変数ストレージ域をバインドする必要があります。また、バインドされた変数は有効なストレージ域であるとともに、準備されたステートメントがデータベースに対して実行される時点で入力データ値が入っている必要があります。

次の例は、2 つのパラメーター・マーカーを有する SQL ステートメントを示しています。

```
SELECT * FROM customers WHERE custid = ? AND lastname = ?
```

関連した概念:

- 『パラメーター・マーカーの使用例』

パラメーター・マーカーの使用例

DB2 Everyplace では、データを効率的にアクセスするインターフェースとして、CLI/ODBC、JDBC、および ADO.NET を含めた標準インターフェースを豊富に取りそろえています。以下のコード断片の例は、パラメーター・マーカーを有する準備済みステートメントを各データ・アクセス API で使用する方法を示しています。

表 t1 について、以下の表スキーマについて考えてみます。そこでは、列 c1 が表 t1 の主キーになっています。

表 13. 表スキーマの例

列名	DB2 Everyplace のデータ・タイプ	ヌル可能
c1	INTEGER	false
c2	SMALLINT	true

表 13. 表スキーマの例 (続き)

列名	DB2 Everyplace のデータ・タイプ	ヌル可能
c3	CHAR(20)	true
c4	VARCHAR(20)	true
c5	DECIMAL(8,2)	true
c6	DATE	true
c7	TIME	true
c8	TIMESTAMP	true
c9	BLOB(30)	true

以下に示す例は、準備済みステートメントを使用して行を表 t1 に挿入する方法を示しています。

CLI の例

```
void parameterExample1(void)
{
   SQLHENV henv;
   SQLHDBC hdbc;
   SQLHSTMT hstmt;
   SQLRETURN rc;
   TCHAR server[] = _T("C:¥¥mysample¥¥");
   TCHAR uid[] = _T("db2e");
   TCHAR pwd[] = _T("db2e");
   long p1 = 10;
   short p2 = 100;
   TCHAR p3[100];
   TCHAR p4[100];
   TCHAR p5[100];
   TCHAR p6[100];
   TCHAR p7[100];
   TCHAR p8[100];
   char  p9[100];
   long len = 0;

   _tcscpy(p3, _T("data1"));
   _tcscpy(p4, _T("data2"));
   _tcscpy(p5, _T("10.12"));
   _tcscpy(p6, _T("2003-06-30"));
   _tcscpy(p7, _T("12:12:12"));
   _tcscpy(p8, _T("2003-06-30-17.54.27.710000"));

   memset(p9, 0, sizeof(p9));
   p9[0] = 'X';
   p9[1] = 'Y';
   p9[2] = 'Z';

   rc = SQLAllocEnv(&henv);
   // check return code ...

   rc = SQLAllocConnect(henv, &hdbc);
   // check return code ...

   rc = SQLConnect(hdbc, (SQLTCHAR*)server, SQL_NTS,
    (SQLTCHAR*)uid, SQL_NTS, (SQLTCHAR*)pwd, SQL_NTS);
```

```
// check return code ...

rc = SQLAllocStmt(hdbc, &hstmt);
// check return code ...

   // prepare the statement
rc = SQLPrepare(hstmt, _T("INSERT INTO t1 VALUES (?,?,?,?,?,?,?,?,?)"), SQL_NTS);
// check return code ...

// bind input parameters
rc = SQLBindParameter(hstmt, (unsigned short)1, SQL_PARAM_INPUT,
    SQL_C_LONG, SQL_INTEGER, 4, 0, &p1, sizeof(p1), &len);
// check return code ...

rc = SQLBindParameter(hstmt, (unsigned short)2, SQL_PARAM_INPUT, SQL_C_LONG,
    SQL_SMALLINT, 2, 0, &p2, sizeof(p2), &len);
// check return code ...

len = SQL_NTS;
rc = SQLBindParameter(hstmt, (unsigned short)3, SQL_PARAM_INPUT, SQL_C_TCHAR,
    SQL_CHAR, 0, 0, &p3[0], 100, &len);
// check return code ...

rc = SQLBindParameter(hstmt, (unsigned short)4, SQL_PARAM_INPUT, SQL_C_TCHAR,
    SQL_VARCHAR, 0, 0, &p4[0], 100, &len);
// check return code ...

rc = SQLBindParameter(hstmt, (unsigned short)5, SQL_PARAM_INPUT, SQL_C_TCHAR,
    SQL_DECIMAL, 8, 2, &p5[0], 100, &len);
// check return code ...

rc = SQLBindParameter(hstmt, (unsigned short)6, SQL_PARAM_INPUT, SQL_C_TCHAR,
    SQL_TYPE_DATE, 0, 0, &p6[0], 100, &len);
// check return code ...

rc = SQLBindParameter(hstmt, (unsigned short)7, SQL_PARAM_INPUT, SQL_C_TCHAR,
    SQL_TYPE_TIME, 0, 0, &p7[0], 100, &len);
// check return code ...

rc = SQLBindParameter(hstmt, (unsigned short)8, SQL_PARAM_INPUT, SQL_C_TCHAR,
    SQL_TYPE_TIMESTAMP, 0, 0, &p8[0], 100, &len);
// check return code ...

len = 3;
rc = SQLBindParameter(hstmt, (unsigned short)9, SQL_PARAM_INPUT, SQL_C_BINARY,
    SQL_BINARY, 0, 0, &p9[0], 100, &len);
// check return code ...

// execute the prepared statement
rc = SQLExecute(hstmt);
// check return code ...

rc = SQLFreeStmt(hstmt, SQL_DROP);
// check return code ...

rc = SQLDisconnect(hdbc);
// check return code ...

rc = SQLFreeConnect(hdbc);
// check return code ...
```

```
   rc = SQLFreeEnv(henv);
   // check return code ...

}
```

JDBC の例

```
public static void parameterExample1() {

   String driver = "com.ibm.db2e.jdbc.DB2eDriver";
   String url    = "jdbc:db2e:mysample";
   Connection conn = null;
   PreparedStatement pstmt = null;

   try
   {
      Class.forName(driver);

      conn = DriverManager.getConnection(url);

      // prepare the statement
      pstmt = conn.prepareStatement("INSERT INTO t1 VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?)");

      // bind the input parameters
      pstmt.setInt(1, 1);
      pstmt.setShort(2, (short)2);
      pstmt.setString(3, "data1");
      pstmt.setString(4, "data2");
      pstmt.setBigDecimal(5, new java.math.BigDecimal("12.34"));
      pstmt.setDate(6, new java.sql.Date(System.currentTimeMillis() ) );
      pstmt.setTime(7,  new java.sql.Time(System.currentTimeMillis() ) );
      pstmt.setTimestamp (8,  new java.sql.Timestamp(System.currentTimeMillis() ) );
      pstmt.setBytes(9, new byte[] { (byte)'X', (byte)'Y', (byte)'Z' } );

      // execute the statement
      pstmt.execute();

      pstmt.close();

      conn.close();
   }
   catch (SQLException sqlEx)
   {
      while(sqlEx != null)
      {
          System.out.println("SQLERROR: ¥n" + sqlEx.getErrorCode() +
             ", SQLState: " + sqlEx.getSQLState() +
             ", Message: " + sqlEx.getMessage() +
             ", Vendor: " + sqlEx.getErrorCode() );
         sqlEx = sqlEx.getNextException();
      }
   }
   catch (Exception ex)
   {
      ex.printStackTrace();
   }
}
```

ADO.NET の例

[C#]

```
public static void ParameterExample1()
{
   DB2eConnection conn = null;
   DB2eCommand cmd  = null;
   String connString   = @"database=.¥; uid=db2e; pwd=db2e";
   int i = 1;

   try
   {
      conn = new DB2eConnection(connString);

      conn.Open();

      cmd = new DB2eCommand("INSERT INTO t1 VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?)", conn);

      // prepare the command
      cmd.Prepare();

      // bind the input parameters
      DB2eParameter p1 = new DB2eParameter("@p1", DB2eType.Integer);
      p1.Value = ++i;
      cmd.Parameters.Add(p1);

      DB2eParameter p2 = new DB2eParameter("@p2", DB2eType.SmallInt);
      p2.Value = 100;
      cmd.Parameters.Add(p2);

      DB2eParameter p3 = new DB2eParameter("@p3", DB2eType.Char);
      p3.Value = "data1";
      cmd.Parameters.Add(p3);

      DB2eParameter p4 = new DB2eParameter("@p4", DB2eType.VarChar);
      p4.Value = "data2";
      cmd.Parameters.Add(p4);

      DB2eParameter p5 = new DB2eParameter("@p5", DB2eType.Decimal);
      p5.Value = 20.25;
      cmd.Parameters.Add(p5);

      DB2eParameter p6 = new DB2eParameter("@p6", DB2eType.Date);
      p6.Value = DateTime.Now;
      cmd.Parameters.Add(p6);

      DB2eParameter p7 = new DB2eParameter("@p7", DB2eType.Time);
      p7.Value = new TimeSpan(23, 23, 23);
      cmd.Parameters.Add(p7);

      DB2eParameter p8 = new DB2eParameter("@p8", DB2eType.Timestamp);
      p8.Value = DateTime.Now;
      cmd.Parameters.Add(p8);

      byte []barr = new byte[3];
      barr[0] = (byte)'X';
      barr[1] = (byte)'Y';
      barr[2] = (byte)'Z';

      DB2eParameter p9 = new DB2eParameter("@p9", DB2eType.Blob);
```

```
      p9.Value = barr;
      cmd.Parameters.Add(p9);

      // execute the prepared command
      cmd.ExecuteNonQuery();
   }
   catch (DB2eException e1)
   {
      for (int i=0; i < e1.Errors.Count; i++)
      {
         Console.WriteLine("Error #" + i + "¥n" +
            "Message: " + e1.Errors[i].Message + "¥n" +
            "Native: " + e1.Errors[i].NativeError.ToString() + "¥n" +
            "SQL: " + e1.Errors[i].SQLState + "¥n");
      }
   }
   catch (Exception e2)
   {
       Console.WriteLine(e2.Message);
   }
   finally
   {
      if (conn != null && conn.State != ConnectionState.Closed)
      {
            conn.Close();
            conn = null;
      }
   }
}
```

関連した概念:

- 79 ページの『パラメーター・マーカーの概要』

DB2 Everyplace でサポートされているパラメーター・マーカー

疑問符 (?) によって表されるパラメーター・マーカーは、SQL ステートメント内のプレースホルダーであり、その値はステートメントの実行中に入手されます。アプリケーションでは、SQLBindParameter() を使用して、バインド・パラメーター・マーカーをアプリケーション変数に関連付けます。SQLExecute() と SQLExecDirect() DB2 CLI 関数の実行中に、これらの変数の値によって各パラメーター・マーカーが置き換えられます。このプロセス中にデータ変換が行われる場合があります。サポートされるデータ・タイプ変換の詳細については、 295 ページの表 95 を参照してください。

DB2 Everyplace は、SQL ステートメントの選択した場所で使用できる、タイプ無しパラメーター・マーカーのみをサポートします。表 14 では、パラメーター・マーカーの使用法に関する制約事項をリストしています。

表 14. パラメーター・マーカーの使用法の制約事項

タイプ無しパラメーター・マーカーの位置	データ・タイプ
式: 選択リスト内のみ	エラー
式: 算術演算子の両方のオペランド	エラー
述部: IN 述部の左側オペランド	エラー

表 14. パラメーター・マーカーの使用法の制約事項 (続き)

タイプ無しパラメーター・マーカーの位置	データ・タイプ
述部: 関係演算子の両方のオペランド	エラー
関数: 集約関数のオペランド	エラー

関連した解説:

- 147 ページの『DB2 Everyplace SQL ステートメント・サポートの概要』
- 185 ページの『割り当てと比較に関するデータ・タイプの互換性』
- 189 ページの『SQLState のリスト』
- 189 ページの『SQLState クラス・コードのサマリー』

第 11 章 接続のコンテキストにおけるカーソルの動作

他のステートメント・ハンドルとの書き込み競合時における一般的な読み取りカーソル:

アプリケーションには同時に同じ表に読み取りと書き込みの両操作を行うステートメント・ハンドルが複数存在する可能性があります。競合が発生するのは、あるハンドルが表に読み取りか書き込みの操作を行っている最中に、別のハンドルがその表に書き込み操作 (例えば、UPDATEとか DELETE とか INSERT) を行おうとするときです。 DB2 Everyplace の動作では、読み取りカーソルが継続的に作動し、いつも最新の現行データを読み取ります。索引を使用しているかどうかに関係なく、書き込みが競合しても、読み取りが継続されます。例えば、あるアプリケーションに 2 つのステートメント・ハンドルが存在するとします。ハンドル #1 は表 T から行を取り出すのに使用され、ハンドル #2 は同じ表から行をいくつか削除するのに使用されます。それぞれのハンドルは異なるスレッド (例えば、Java スレッド環境) で作成された可能性があります。

この場合、考えられるシナリオは次のとおりです。

```
// Fetch 2 rows from table T
   Statement handle 1: execute "SELECT A FROM T WHERE primary_key < 10"
   Statement handle 1: fetch one row; fetch another row
    // Delete some rows in table T
   Statement handle 2: prepare "DELETE FROM T WHERE primary_key = ?"
   Statement handle 2: execute
    // Continue to fetch one more row from T
   Statement handle 1: fetch one row
```

実行のこの時点において、ステートメント・ハンドル #1 は索引を使用しているかどうかにかかわらず、次の行 (次の行が存在する場合) のフェッチを継続することができます。上記のシナリオでは、主キーが存在することから、索引が使用されています。重要な点は、DB2 Everyplace はハンドル #1 のカーソル位置を、先に進める前にカーソルの現在位置を使用して変更しようとするということです。現在位置がもはや存在しない場合 (例えば、該当する行が別のステートメント・ハンドルにより削除されてしまった場合) には、カーソルはフェッチが終了すると即座に次の位置に進みます。同様に、次の位置が別のステートメント・ハンドルにより削除された場合には、カーソルは発生した「穴」をスキップして次の位置まで進むことができます。

他のステートメント・ハンドルとの書き込み競合時における両方向スクロール・カーソル:

前のセクションで取りあげた例と似通った例について考えてみます。読み取りカーソルが両方向スクロール・カーソルであるという点が前回と異なります。そのカーソルが「インセンシティブ (無反応)」な両方向スクロール・カーソルである場合は、問題にはなりません。結果セットが定義によって変化しないからです。カーソルが「インセンシティブ」でない場合は、その動作は上述した通常の読み取りカーソルと一致します。競合後の読み取りカーソルの動作は、基本的に、結果セットは最新の現行表データに従って再計算され、現在行セットの開始点は維持されます。現在行が削除されていれば、カーソルは次の行に進みます。

次の例では、CLP を使用した両方向スクロール・カーソルの場合を示しています。表 T には、6 つの行があるものとします。

```
create table T (a int, b int)
     create index idx1 on T(a)
     insert into T values (1, 1)
     insert into T values (2, 2)
     insert into T values (3, 1)
     insert into T values (3, 2)
     insert into T values (3, 3)
     insert into T values (4, 4)
```

さらに、1 つは読み取り用でもう 1 つは削除用の 2 つのステートメント・ハンドルを有するアプリケーションの例を考えてみます。

```
Statement handle 1: enable scrollable cursor;
Statement handle 1: execute "SELECT A FROM T WHERE a < 10"
Statement handle 2: prepare "DELETE FROM T WHERE a = ?"
Statement handle 1: fetchscroll with SQL_FETCH_FIRST
-- get (1, 1)
Statement handle 1: fetchscroll with SQL_FETCH_NEXT
-- get (2, 2)
Statement handle 1: fetchscroll with SQL_FETCH_NEXT
-- get (3, 1)
Statement handle 2: execute
--- suppose delete row (2, 2)
Statement handle 1: fetchscroll with SQL_FETCH_NEXT
-- re-compute previous rows, and return (3, 2)
Statement handle 1: fetchscroll with SQL_FETCH_PRIOR
-- get (3, 1)
Statement handle 1: fetchscroll with SQL_FETCH_PRIOR
-- get (1, 1)  note that (2, 2) is gone
Statement handle 1: fetchscroll with SQL_FETCH_ABSOLUTE, offset 2
-- get (3, 1)  note that (2, 2) is gone
Statement handle 1: fetchscroll with SQL_FETCH_ABSOLUTE, offset 5
-- get (4, 4)
```

自動コミット・モードを含めた、コミットとロールバックにおけるカーソル:

トランザクション・モードか自動コミット・モードであるかにかかわりなく、オープン・カーソルは、コミットが行われている間はオープン状態のままです。オープン・カーソルはロールバックでクローズされます。

オブジェクトの従属関係:

ステートメント・ハンドル H を介して SQL ステートメントを作成すると、特定のオブジェクトとの間に何らかの従属関係が生じることがあります。例えば、索引 Idx を介して表 T から行を選択するには、表 T と索引 Idx が存在しなくてはなりません。これらのオブジェクトが別のステートメント・ハンドルで削除された場合 (例えば、索引 Idx がドロップした場合)、H を使用して SQL ステートメントを再実行するには、そのステートメントの再コンパイルを強いられることになります。その結果、照会プランが以前と異なったり、エラーが戻されたりすることがあります。

第 12 章 ローカル・データの暗号化

この章では、DB2 Everyplace データベースのローカル・データを暗号化する方法について説明します。説明されているトピックは、以下のとおりです。

- 『ローカル・データ暗号化の概要』
- 91 ページの『DB2 Everyplace データベースへの接続の確立』
- 91 ページの『ユーザーへの暗号化特権の付与』
- 92 ページの『暗号化された表の作成』
- 92 ページの『暗号化特権の管理』
- 93 ページの『DB2eCLP を使用した暗号化』

ローカル・データ暗号化の概要

DB2 Everyplace における暗号化は、モバイル・デバイスまたは組み込みデバイス上のデータを保護する目的で設計されています。このトピックでは、入門編としてローカル・データの暗号化を概説します。以下の項目について説明します。

- ローカル・データ暗号化を使用する理由
- ローカル・データの暗号化の目的
- 最初に作成する暗号化された表
- 暗号化された表へのその後のアクセス
- ユーザー特権の管理

ローカル・データ暗号化を使用する理由:

顧客の連絡先情報が含まれた企業の営業アプリケーションを例に取ります。モバイル環境を利用する営業担当者は、顧客を訪問する際に PDA でこのデータを携行します。アプリケーションまたは PDA にセキュアなストレージ・システムがなければ、アプリケーションを使用したり、モバイル・デバイスのネイティブ・ファイル・システムを調べたりすることによって、簡単にこのデータへアクセスできてしまいます。機密データの暗号化は、企業情報の保護において極めて重要な意味を持ちます。

ローカル・データ暗号化の目標:

DB2 Everyplace には、アプリケーションに企業のセキュリティー・ポリシーをインプリメントするためのソリューションが用意されています。第 1 の目的は、DB2 Everyplace の表に保管された重要な情報および機密情報を暗号化することです。データは、暗号鍵をインプリメントする DES などの標準的な暗号化方式を使用して暗号化されます。第 2 の目的は、データの暗号化に使用する鍵を管理できるようなセキュア・フレームワークを提供することです。ユーザーがデータベースに接続する際には、ユーザー ID とパスワードが要求されます。詳細については、92 ページの『暗号化特権の管理』を参照してください。

データ暗号化の使用に関する詳細については、93 ページの『DB2eCLP を使用した暗号化』を参照してください。

前提条件:

本セクションでは、各プラットフォームで暗号化を使用可能にする方法を説明し、DB2 Everyplace データベースで必要なライブラリーのほかに必要となるライブラリーのリストを記載します。

Win32 の場合:

- プラグイン・ライブラリー: CryptoPlugin.dll (DB2 Everyplace に用意されています)。
- 暗号化ライブラリー: Crypt32.dll (暗号強度 128 ビットの暗号化パッケージ、IE5.5 以降に付属しています)。
  http://www.microsoft.com/windows/ie/downloads/critical /q313675/download.asp を参照してください。

Windows CE/Pocket PC の場合:

- プラグイン・ライブラリー: CryptoPlugin.dll (DB2 Everyplace に用意されています)。
- 暗号化ライブラリー: Microsoft High Encryption Pack for Pocket PC V1.0。
  http://www.microsoft.com/mobile/pocketpc/downloads/ssl128.asp を参照してください。

Palm OS の場合:

- プラグイン・ライブラリー: CryptoPlugin.PRC (DB2 Everyplace に用意されています)。
- 暗号化ライブラリー: PBSPKcs11.prc (DB2 Everyplace に用意されています)。

Linux/Neutrino の場合

- プラグイン・ライブラリー: libcryptoplugin.so (DB2 Everyplace により提供)
- 暗号化ライブラリー: libpvcpkcs11.so (DB2 Everyplace により提供)

Symbian の場合

- プラグイン・ライブラリー: CRYPTOPLUGIN.DLL (DB2 Everyplace により提供)
- 暗号化ライブラリー: ECSPKCS11.DLL (DB2 Everyplace により提供)

手順:

データ暗号化を使用するには、以下のようにします。

1. DB2 Everyplace データベースへの接続を確立する。
2. ユーザーに暗号化特権を付与する。
3. 最初に暗号化された表を作成する。

   後続の暗号化された表へのアクセス: データベースに DB2eSYSUSERS 表がある場合、後続のすべてのデータベース接続では、指定のユーザー ID とパスワードを使用したユーザー認証が行われます。認証が失敗した場合、アプリケーションは暗号化されていない表にのみアクセスできます。アプリケーションでは、暗号化された表を新規に作成したり、既存の暗号化された表をドロップしたり、あるいは暗号化されたデータへアクセスしてそれを更新したりすることはできません。
4. 暗号化特権を管理する。

DB2 Everyplace データベースへの接続の確立

このタスクは、ローカル・データ暗号化のメインタスクの一部です。このステップが完了したら、 89 ページの『ローカル・データ暗号化の概要』に戻ります。

手順:

DB2 Everyplace データベースと対話を行うには、接続を確立する必要があります。また、ユーザーが暗号化された表にアクセスしたり、それを作成したりするには、アプリケーションでは、以下の CLI 関数を使用して、空ではないユーザー名とパスワードを指定して DB2 Everyplace に接続しなければなりません。

```
rc = SQLConnect(hdbc, "C:¥temp¥", SQL_NTS, "user1", SQL_NTS, "pwd1", SQL_NTS)
```

ここで、*"C:¥temp¥"* には、アプリケーションがユーザー ID *"user1"* とパスワード *"pwd1"* を使用して接続するデータベースのディレクトリーが入ります。

JDBC インターフェースの場合も、同様にデータベース接続を確立することができます。

関連した概念:

- 75 ページの『接続のシリアライゼーション』

ユーザーへの暗号化特権の付与

このタスクは、ローカル・データ暗号化のメインタスクの一部です。このステップを完了したら、 89 ページの『ローカル・データ暗号化の概要』に戻ります。

手順:

表の暗号化を初めて行う前に、アプリケーションはユーザーに暗号化特権を付与しなければなりません。例えば、アプリケーションは以下の SQL ステートメントを発行することができます。

```
rc = SQLExecDirect(..., "GRANT ENCRYPT ON DATABSE TO ¥" user1¥"" +
                         " using ¥"pwd1¥" new ¥"pwd1¥"", SQL_NTS)
```

この SQL ステートメントを実行する際、DB2 Everyplace は DB2eSYSUSERS という名前のシステム・カタログ表を作成し、この表に行を挿入します。これは、ユーザー *"user1* が、対応するパスワードとともに登録され、暗号化された表の作成やその表へのアクセスなどのすべての暗号化特権が付与されたことを意味します。

この表はデータベースおよび暗号化されたデータに緊密にバインドされており、暗号化されたデータにアクセスするために別の DB2 Everyplace データベースへ移動することはできません。これは、異なるデータベースは、暗号化および暗号化解除用に別の鍵を持っているためです。結果として、あるデータベースの暗号化された表にアクセス可能なユーザーが、同じユーザー ID とパスワードを使用して別のデータベースにアクセスすることはできません。他のシステム・カタログ表と同様、アプリケーションは SQL の SELECT ステートメントを使用して行を取り出すことができますが、この表にあるデータを INSERT、DELETE、UPDATE、CREATE、または DROP の各ステートメントを使用して変更することはできません。

関連した概念:

- 93 ページの『DB2eCLP を使用した暗号化』

関連したタスク:

- 93 ページの『DB2eCLP を使用した暗号化』
- 『暗号化された表の作成』
- 『暗号化特権の管理』

暗号化された表の作成

このタスクは、ローカル・データ暗号化のメインタスクの一部です。このステップを完了したら、 89 ページの『ローカル・データ暗号化の概要』 に戻ります。

手順:

DB2 Everyplace データベースへの接続が確立され、ユーザーに暗号化特権が付与されると、アプリケーションで拡張 CREATE TABLE ステートメントを使用して、暗号化された表を作成することができます。例えば、以下のような従業員表を作成することができます。

```
SQLExecDirect(..., "CREATE TABLE EMPLOYEES (EMPNO INT PRIMARY KEY, NAME VARCHAR(30),
SALARY DECIMAL(10,2)) WITH ENCRYPTION", SQL_NTS)
```

関連した概念:

- 93 ページの『DB2eCLP を使用した暗号化』

関連したタスク:

- 93 ページの『DB2eCLP を使用した暗号化』
- 91 ページの『ユーザーへの暗号化特権の付与』
- 『暗号化特権の管理』

暗号化特権の管理

このタスクは、ローカル・データ暗号化のメインタスクの一部です。このステップを完了したら、 89 ページの『ローカル・データ暗号化の概要』に戻ります。

手順:

認証されたユーザー ID およびパスワードを使用してデータベースに接続すると、アプリケーションは、新規ユーザーを作成したり、パスワードを変更したり、あるいはシステムから登録済みユーザーを除去したりすることができます。新規ユーザーを作成する、あるいはパスワードを変更する際の構文は、以下のとおりです。

```
GRANT ENCRYPT ON DATABASE TO "newuser" USING "grantorpassword" NEW "newpassword"
```

登録済みユーザーを除去する際の構文は、以下のとおりです。

```
REVOKE ENCRYPT ON DATABASE FROM "user"
```

注: DB2eSYSUSERS 表から (REVOKE ステートメントを使用して) すべての登録済みユーザーが除去されると、既存の暗号化された表へのアクセスを含め、暗号化操作を実行することはできなくなります。リカバリー機構はありません。

関連した概念:

- 『DB2eCLP を使用した暗号化』

関連したタスク:

- 『DB2eCLP を使用した暗号化』
- 92 ページの『暗号化された表の作成』
- 91 ページの『ユーザーへの暗号化特権の付与』

DB2eCLP を使用した暗号化

このセクションには、アプリケーションでデータ暗号化の使用法を示す目的で設計されたサンプルの対話式セッションの例が含まれています。各操作を説明したコメントが追加されています。

```
-- DB2eCLP を使用した暗号化
--
-- これは提供されたサンプル・コマンド行インターフェース・プログラム
-- DB2eCLP を使用した暗号化セッションの例です。
--
-- 失敗した場合にはステートメントの戻りコードのみを、
-- 正常に完了した場合には選択の結果のみを
-- 示しています。
-- DB2 Everyplace に入力できるコマンドには、
-- 先頭に「CLP:>」というストリングが付いています。
--
-- -- (CLI:SQLConnect, SQL:CREATE TABLE, SQL:GRANT, SQL:REVOKE)
--
-- DB2eCLP を開始すると、自動的に現行ディレクトリーにある
-- デフォルト・データベースに接続されます。
-- これは以下のコマンドを実行するのと同じことです。
--
CLP:> CONNECT TO anything;

-- 「anything」という名前だけで特定のパスが指定されていないため、
-- 現行ディレクトリーに接続されます。
--
-- ここで、スウェーデン語の数を表す言葉へのマッピングが含まれた
-- 暗号化されていない表を作成することにします。

CLP:> CREATE TABLE swedish(nummer INT, ord VARCHAR(32));
CLP:> INSERT INTO swedish VALUES(1, 'ett');
CLP:> INSERT INTO swedish VALUES(3, 'tre');
CLP:> INSERT INTO swedish VALUES(4, 'fyra');
CLP:> INSERT INTO swedish VALUES(5, 'fem');
CLP:> INSERT INTO swedish VALUES(7, 'sju');
CLP:> INSERT INTO swedish VALUES(99, 'nittionio');

-- データを見ると、以下のようになっています。
CLP:> SELECT * FROM swedish;

NUMMER      ORD
----------- --------------------------------
          1 ett
          3 tre
          4 fyra
          5 fem
          7 sju
         99 nittionio
6 row(s) returned.

-- ここで、英語の対応する表の作成を試みます。
-- ただし、ここでは暗号化を使用します。
--
CLP:> CREATE TABLE english(number INT, word VARCHAR(32)) WITH ENCRYPTION;
Statement failed [sqlstate = 42501].

-- ここではまだ許可を受けていないため、エラー・コードで示されているとおり
-- 操作は失敗します。
-- そのため再度接続する必要があります。
--
CLP:> CONNECT TO something USER jsk USING hemligt;

-- これで同じデータベース (デフォルト/現行ディレクトリー) に接続されますが、
-- 特定のユーザー ID「jsk」とパスワード「hemligt」が使用されます。
-- CONNECT TO コマンドは SQL ステートメントではないため、
-- DB2eCLP アプリケーションによって解釈されます。このコマンドは、
-- 以下を使用して DB2 Everyplace データベースへの接続を切断し、
```

```
-- 再び接続します。
--    SQLConnect(hdbc, "something", SQL_NTS, "jsk", SQL_NTS, "hemligt", SQL_NTS);
--
-- ここで、最初に許可ユーザーを作成する必要があります。最初のユーザー
-- を作成する際は、ログインされたユーザーと同じ名前と同じパスワード
-- を持っていなければなりません。
--
CLP:> GRANT ENCRYPT ON DATABASE TO "jsk" USING "hemligt" NEW "hemligt";

-- GRANT の場合、名前とパスワードは二重引用符で囲まなければなりません。
-- これは名前とパスワードには大文字小文字の区別があり、
-- ステートメントが直接 DB2 Everyplace に渡されるからです。
--
-- これで許可された暗号化ユーザーが作成され、暗号化された表
-- を作成できるようになります。
--

CLP:> CREATE TABLE english(number INT, word VARCHAR(32)) WITH ENCRYPTION;
CLP:> INSERT INTO english VALUES(1, 'one');
CLP:> INSERT INTO english VALUES(3, 'three');
CLP:> INSERT INTO english VALUES(4, 'four');
CLP:> INSERT INTO english VALUES(5, 'five');
CLP:> INSERT INTO english VALUES(7, 'seven');
CLP:> INSERT INTO english VALUES(99, 'ninety nine');

-- データを見ると、以下のようになっています。
CLP:> SELECT * FROM english;

NUMBER      WORD
----------- --------------------------------
          1 one
          3 three
          4 four
          5 five
          7 seven
         99 ninety nine
6 row(s) returned.

-- スウェーデン語で大きな数をランダムに選びます。
--
CLP:> SELECT * FROM swedish WHERE nummer > 42;

NUMMER      ORD
----------- --------------------------------

         99 nittionio
1 row(s) returned.

-- 英語で大きな数をランダムに選びます。
--
CLP:> SELECT * FROM english WHERE number > 42;

NUMBER      WORD
----------- --------------------------------
         99 ninety nine
1 row(s) returned.

--「fyra」を英語に変換します。
--
CLP:> SELECT word FROM swedish, english WHERE number = nummer AND ord = 'fyra';

WORD
--------------------------------
four
1 row(s) returned.

-- 変換表を取得します。
--
CLP:> SELECT number, ord, word FROM swedish, english WHERE number = nummer;

NUMBER      ORD                              WORD
----------- -------------------------------- --------------------------------
          1 ett                              one
          3 tre                              three
          4 fyra                             four
          5 fem                              five
          7 sju                              seven
         99 nittionio                        ninety nine
6 row(s) returned.

-- 別のユーザーに、そのユーザー自身のパスワードを使用して暗号化されたデータ
-- にアクセスすることを許可することを試みます。
--
CLP:> GRANT ENCRYPT ON DATABASE TO "xin" USING "notKnown" NEW "notKnown";
```

```
Statement failed [sqlstate = 42506].

-- 新しいユーザーを追加するためには、ログインするユーザーが自分自身を妥当性検査
-- しなければならないため、この操作は失敗します。この検査は、USING 文節の後
-- に自分のパスワードを指定することによって行われます。
--
CLP:> GRANT ENCRYPT ON DATABASE TO "xin" USING "hemligt" NEW "notKnown";

-- 新規ユーザーに再接続します。
--
CLP:> CONNECT TO samething USER xin USING notknown;
Statement failed [sqlstate = 42505].

-- パスワードが同じでないため同じキーが生成されず、アクセスが拒否されるため、
-- この操作は失敗します。
--
CLP:> CONNECT TO samething USER ksin USING notKnown;

-- ユーザー ksin が存在せず、そのためユーザーの認証を試みないので、
-- この操作は失敗することはありません。
-- しかし、SQLGetInfo を使用すれば、このようなケースと、ユーザーが
-- 正常に認証されたケースとを区別することができます。
--
CLP:> SELECT * FROM swedish;

NUMMER      ORD
----------- --------------------------------
          1 ett
          3 tre
          4 fyra
          5 fem
          7 sju
         99 nittionio
6 row(s) returned.

-- 暗号化されていないデータの選択は正しく行われますが、許可されたユーザー
-- が接続されない限り、暗号化されたデータの読み取り/更新はできません。
--
CLP:> SELECT * FROM english;
Statement failed [sqlstate = 42501].

-- 最後に正しいユーザー名とパスワードを用いて、新規ユーザーとして接続します。
--
CLP:> CONNECT TO samething USER xin USING notKnown;

-- 認証が済み、データにアクセスできるようになっているか検査します。
--
CLP:> SELECT * FROM english;

NUMBER      WORD
----------- --------------------------------
          1 one
          3 three
          4 four
          5 five
          7 seven
         99 ninety nine
6 row(s) returned.

-- 別のユーザーを追加します。
--
CLP:> GRANT ENCRYPT ON DATABASE TO "thf" USING "notKnown" NEW "heimlich";

-- 現在存在するユーザーをリストします。
--
CLP:> SELECT username, grantorname FROM "DB2eSYSUSERS";

USERNAME            GRANTORNAME
------------------- -------------------
jsk                 jsk
xin                 jsk
thf                 xin
3 row(s) returned.

-- 再度「jsk」として接続します。
--
CLP:> CONNECT TO itagain USER jsk USING hemligt;
Statement completed successfully.

-- パスワードを「secret」に変更します。
--
CLP:> GRANT ENCRYPT ON DATABASE TO "jsk" USING "secret" NEW "secret";
Statement failed [sqlstate = 42506].
```

```
-- 最初に古いパスワードを指定してから新しいパスワードを指定する必要があるため、
-- この操作は失敗します。
--
CLP:> GRANT ENCRYPT ON DATABASE TO "jsk" USING "hemligt" NEW "secret";

-- 新規パスワードを試してみます。
--
CLP:> CONNECT TO itagain USER jsk USING secret;

-- 暗号化されたデータにアクセスできることを確認します。
--
CLP:> SELECT * FROM english;

NUMBER      WORD
----------- --------------------------------
          1 one
          3 three
          4 four
          5 five
          7 seven
         99 ninety nine
6 row(s) returned.

-- 「xin」の暗号化特権を取り消します。
--
CLP:> REVOKE ENCRYPT ON DATABASE FROM "xin";

-- ユーザーをリストします。
--
CLP:> SELECT username, grantorname FROM "DB2eSYSUSERS";

USERNAME            GRANTORNAME
------------------- -------------------
jsk                 jsk
thf                 xin
2 row(s) returned.

-- 存在しないユーザーに再度接続します。エラーは出ません。
--
CLP:> CONNECT TO thedatabase USER xin USING idontknow;

-- 許可を受けずに暗号化操作を試みます。
--
CLP:> SELECT * FROM english;
Statement failed [sqlstate = 42501].

CLP:> DROP TABLE english;
Statement failed [sqlstate = 42501].

CLP:> REVOKE ENCRYPT FROM "jsk";
Statement failed [sqlstate = 42601].

CLP:> GRANT ENCRYPT ON DATABASE TO "xin" USING "idontknow" NEW "idontknow";
Statement failed [sqlstate = 42502].

-- 「thf」として接続します。
--
CLP:> CONNECT TO thedatabase USER thf USING heimlich;

-- 暗号化されたデータを読み取れるかどうかチェックします。
--
CLP:> SELECT * FROM english;

NUMBER      WORD
----------- --------------------------------
          1 one
          3 three
          4 four
          5 five
          7 seven
         99 ninety nine
6 row(s) returned.

-- 接続されているユーザーの特権を取り消します。
--
CLP:> REVOKE ENCRYPT ON DATABASE FROM "thf";

-- そのユーザーがデータにアクセスできなくなったことを確認します。
--
CLP:> SELECT * FROM english;
Statement failed [sqlstate = 42501].

-- 以下のようにして、唯一残っているユーザー「jsk」としてデータベースに接続した場合に、
--
```

```
CLP:> CONNECT TO thedatabase USER jsk USING secret;

-- 接続されているユーザーを除去すると、そのユーザーはデータにアクセスできなくなります。
-- 実際に、暗号化されたデータに再びアクセスする方法はありません。
--

CLP:> REVOKE ENCRYPT ON DATABASE FROM "jsk";

-- ユーザーが残っていないことを確認します。
--
CLP:> SELECT username, grantorname FROM "DB2eSYSUSERS";

USERNAME            GRANTORNAME
------------------- -------------------
0 row(s) returned.

-- これで暗号化されたデータにアクセスすることはできなくなりました。
--
CLP:> SELECT * FROM english;
Statement failed [sqlstate = 42501].

-- これでセッションの例は終了です。
```

関連したタスク:

- 89 ページの『ローカル・データ暗号化の概要』
- 91 ページの『ユーザーへの暗号化特権の付与』
- 92 ページの『暗号化された表の作成』
- 92 ページの『暗号化特権の管理』

第 3 部 サンプル・アプリケーション

第 13 章 C/C++ サンプル・アプリケーション

それぞれのオペレーティング・システムごとに、C/C++ サンプル・アプリケーションが少なくとも 1 つ提供されています。完全なサンプル・アプリケーションをソース・コードとともに参照するには、適切なクライアント・ディレクトリーを調べてください。

関連したタスク:

- 11 ページの『DB2 Everyplace C/C++ アプリケーションの開発』

関連した解説:

- 12 ページの『サポートされる C/C++ 開発ツール』
- 13 ページの『C/C++ がサポートされているオペレーティング・システム』

第 14 章 サンプル Java アプリケーション

この章ではサンプル Java アプリケーションについての情報を提供します。説明されているトピックは、以下のとおりです。

- 『Java サンプル・アプリケーションの概要』
- 105 ページの『ターゲットとなる Palm OS での Java サンプル・アプリケーションのコンパイルおよび実行』
- 111 ページの『ターゲットとなる Palm OS 以外の OS 上での Java サンプル・アプリケーションのコンパイルと実行』

Java サンプル・アプリケーションの概要

このトピックでは、DB2eAppl.java および DB2eJavaCLP.java というサンプル・アプリケーションについて説明します。

サンプル 1: DB2eAppl.java:

DB2eAppl.java は DB2 Everyplace の JDBC アプリケーションをコーディングする方法を具体的に示します。

DB2eAppl.java アプリケーションの主なステップは次のとおりです。

ステップ 1:
java.sql パッケージをインポートする。

ステップ 2:
DB2 Everyplace JDBC ドライバー com.ibm.db2e.jdbc.DB2eDriver をロードする。

ステップ 3
現行ディレクトリー (DB2eAppl.java アプリケーションが実行されるディレクトリー) のデータベースに接続する。

ステップ 4
Statement オブジェクトを作成する。

ステップ 5
レコードが 2 つ入った EMPLOYEE 表からなる、非常に簡単なサンプル・データベースをセットアップする。これは、java.sql.Statement インターフェースの executeUpdate(String sql) メソッドを使用して行います。

ステップ 6
EMPLOYEE 表のレコードをすべて選択し、java.sql.ResultSet インターフェースの next() メソッドを使用してすべての行を検索する。

ステップ 7
データベースから EMPLOYEE 表を除去し、データベースおよび JDBC リソースを解放する。

以下に示す DB2eAppl.java ソース・コードには、上で説明したステップがどこで使用されているのかを示すコメントが入っています。

```
import java.sql.*; //ステップ 1

public class DB2eAppl
{
    public static void main(String[] args) {

        String driver = "com.ibm.db2e.jdbc.DB2eDriver";
        String url    = "jdbc:db2e:mysample";

        try {
            Class.forName(driver); //ステップ 2
            Connection con = DriverManager.getConnection(url);//ステップ 3
            Statement st = con.createStatement();//ステップ 4

            //表 employee の作成  //ステップ 5
            st.executeUpdate("CREATE TABLE employee (EMPNO CHAR(6), FIRSTNAME VARCHAR(12))");
            System.out.println("*** Created table: employee");

            //employee へのレコードの追加
            st.executeUpdate("INSERT INTO employee VALUES ('112233','John')");
            st.executeUpdate("INSERT INTO employee VALUES ('445566','Mary')");
            System.out.println("*** Inserted two records");

            //結果の照会と表示 //ステップ 6
            ResultSet rs = st.executeQuery("SELECT * FROM employee");
            System.out.println("*** Query results:");
            while (rs.next()) {
                System.out.print("EMPNO=" + rs.getString(1) + ", ");
                System.out.println("FIRSTNAME=" + rs.getString(2));
            }

            //表 employee の削除  //ステップ 7
            st.executeUpdate("Drop table employee");
            System.out.println("*** Deleted table: employee");

            rs.close();
            st.close();
            con.close();

        } catch (SQLException sqlEx) {
            while(sqlEx != null)
            {
                System.out.println("[SQLException] " +
                                   "SQLState: " + sqlEx.getSQLState() +
                                   ", Message: " + sqlEx.getMessage() +
                                   ", Vendor: " + sqlEx.getErrorCode() );
                sqlEx = sqlEx.getNextException();
            }
        } catch (Exception ex) {
            ex.printStackTrace();
        }
    }
}
```

サンプル 2: DB2eJavaCLP.java:

DB2eJavaCLP.java は DB2 Everyplace の Java コマンド行プロセッサーです。

制約事項: Palm OS では、DB2eJavaCLP.java サンプル・アプリケーションはサポートされていません。

関連したタスク:

- 21 ページの『DB2 Everyplace Java アプリケーションの開発』
- 105 ページの『ターゲットとなる Palm OS での Java サンプル・アプリケーションのコンパイルおよび実行』
- 106 ページの『Palm OS ターゲットの DB2eAppl.java 用の WSDD プロジェクトの作成』
- 112 ページの『非 Palm OS ターゲットでの WSDD プロジェクトの作成および DB2eAppl.java のビルド・パスへの jar ファイルの追加』

- 114 ページの『Win32 での DB2eAppl.java の実行』
- 115 ページの『Windows CE での DB2eAppl.java の実行』
- 108 ページの『Palm OS エミュレーターでの DB2eAppl.java の実行』
- 118 ページの『QNX Neutrino または組み込み Linux での DB2eAppl.java の実行』
- 119 ページの『Symbian での DB2eAppl.java の実行』
- 111 ページの『ターゲットとなる Palm OS 以外の OS 上での Java サンプル・アプリケーションのコンパイルと実行』

関連した概念:

- 131 ページの『Java ネイティブ同期アプリケーションのサンプル』

ターゲットとなる Palm OS での Java サンプル・アプリケーションのコンパイルおよび実行

以下の一連のトピックでは、ターゲットとなる Palm OS で DB2eAppl.java サンプル Java コードをコンパイルし実行する方法について説明します。

開発環境として、WebSphere Studio Device Developer (WSDD) を使用することをお勧めします。WSDD では J9 VM を使用しますが、これはユーザーのデバイスのプロセッサー・タイプをサポートしない可能性があります。 これとは異なる開発環境と JVM をご使用になる場合は、JVM が JNI をサポートしていることを確認してください (DB2 Everyplace の JDBC ドライバーが JNI を使用するため)。その他の互換性のある JVM としては、 Sun PersonalJava、Insignia Jeode、および NSIcom CrEme などがあります。 現在ご使用のターゲットが Palm OS であれば、WSDD についてくる J9 VM を使用する必要があります。WSDD の評価版は http://www.ibm.com/software/pervasive/products/wsdd/ からダウンロードできます。

前提条件:

1. 以下のソフトウェアがインストール済みであることを確認する。
   - J9 Java 仮想マシンを組み込んだ WSDD 5.5
2. WSDD 資料に従って、使用するターゲットおよび開発環境を準備する。 WSDD サンプル・アプリケーションを作成および実行して WSDD のインストールを検証してください。
3. 使用しているターゲット・デバイスに DB2 Everyplace をインストールする。指示の詳細については、「*DB2 Everyplace インストールおよびユーザーズ・ガイド*」を参照してください。

手順:

ターゲットとなる Palm OS で Java サンプル・コードをコンパイルし実行するには、次のようにします。

1. WSDD の WCE ツールをインストールする。
2. DB2eAppl.java 用の WSDD プロジェクトを作成する。
3. DB2 Everyplace の JDBC ドライバーと java.sql パッケージをビルド・パスに追加する。
4. DB2eAppl.java を WSDD にインポートする。

5. DB2eAppl.java を Palm OS エミュレーターで実行する。

関連したタスク:

- 111 ページの『ターゲットとなる Palm OS 以外の OS 上での Java サンプル・アプリケーションのコンパイルと実行』

Palm OS ターゲット用 WSDD の WCE ツールのインストール

このタスクは、Palm OS ターゲットのサンプル Java アプリケーションをコンパイルおよび実行する場合の、メインタスクの一部です。このステップを完了したら、105 ページの『ターゲットとなる Palm OS での Java サンプル・アプリケーションのコンパイルおよび実行』に戻ります。

手順:

1. WSDD で、「**ヘルプ**」 —> 「**ソフトウェア更新**」 —> 「**更新マネージャー**」をクリックして「パースペクティブのインストール/更新」をオープンします。
2. 「パースペクティブのインストール/更新」の「フィーチャーの更新」ビューで、「**アクセス先サイト**」 —> 「**WebSphere Custom Environment**」 —> 「**WebSphere Custom Environment**」と展開します。
3. 「WCE Tooling for WSDD 5.5.0」を選択します。
4. 「プレビュー」ビューで、「**インストール**」をクリックします。
5. インストール手順に従って、WCE Tooling for WSDD フィーチャーをインストールします。
6. jclXtr クラス・ライブラリーを使用することを選択したら、類似のステップに従って WCE jclXtr クラス・ライブラリーもインストールする必要があります。

105 ページの『ターゲットとなる Palm OS での Java サンプル・アプリケーションのコンパイルおよび実行』に戻ります。

Palm OS ターゲットの DB2eAppl.java 用の WSDD プロジェクトの作成

2 つのバージョンの Palm OS 用 DB2 Everyplace JDBC ドライバーが使用できます。 1 つのバージョンは、J2ME CLDC 構成と互換性があります。もう 1 つのバージョンは、WSDD が提供する JCL Extreme Palm カスタム構成と互換性があります。 次の手順に従って、必要なプロジェクトのタイプを作成してください。

このタスクは、Palm OS ターゲットでサンプル Java アプリケーションをコンパイルおよび実行するメインタスクの一部です。このステップを完了したら、105 ページの『ターゲットとなる Palm OS での Java サンプル・アプリケーションのコンパイルおよび実行』に戻ります。

手順:

jclCldc 構成を使用して WSDD Palm OS プロジェクトを作成するには:

1. WSDD で、**「ウィンドウ」 —>「パースペクティブを開く」 —>「Java」** とクリックして Java パースペクティブに切り替えます。
2. 「DB2 Everyplace Sample for Palm OS CLDC」プロジェクトを作成します。
   a. **「ファイル」 —> 「新規」 —>「その他」** とクリックします。

b. 「新規プロジェクト」ウィンドウの「選択」ページで、左ペインで「J2ME for J9」を選択し、右ペインで「J2ME プロジェクトの作成 (Create J2ME project)」を選択し、「**次へ**」をクリックします。

c. 「新規プロジェクト」ウィンドウの「J2ME プロジェクト (J2ME Project)」ページで、プロジェクト名フィールドに `DB2 Everyplace Sample for Palm OS CLDC` と入力し、「**次へ**」をクリックします。

d. 「新規プロジェクト」ウィンドウの「ライブラリー選択 (Library selection)」ページで、WME jclCldc (jclCldc) を選択し、「**完了**」をクリックします。

jclXtr 構成を使用して WSDD Palm OS プロジェクトを作成するには:

1. WSDD で、**「ウィンドウ」 —>「パースペクティブを開く」 —>「Java」** とクリックして Java パースペクティブに切り替えます。
2. 「DB2 Everyplace Sample for Palm OS XTR」プロジェクトを作成します。
   a. **「ファイル」 —>「新規」 —>「その他」**とクリックします。
   b. 「新規プロジェクト」ウィンドウの「選択」ページで、左ペインで「WCE for J9」を選択し、右ペインで「WCE プロジェクトの作成 (Create WCE project)」を選択し、「**次へ** 」をクリックします。
   c. 「新規プロジェクト」ウィンドウの「カスタム・プロジェクト (Custom Project)」ページで、「プロジェクト名」フィールドに `DB2 Everyplace Sample for Palm OS XTR` と入力して「**次へ**」をクリックします。
   d. 「新規プロジェクト」ウィンドウの「ライブラリー選択 (Library selection)」ページで WCE jclXtr (jclXtr) を選択して、「**完了**」をクリックします。

105 ページの『ターゲットとなる Palm OS での Java サンプル・アプリケーションのコンパイルおよび実行』に戻ります。

DB2 Everyplace JDBC ドライバーおよび java.sql パッケージのビルド・パスへの追加

以下のステップは 「DB2 Everyplace Sample for Palm OS CLDC」プロジェクトに適用されます。「DB2 Everyplace Sample for Palm OS XTR」プロジェクトにも類似のステップがあります。

このタスクは、Palm OS ターゲットのサンプル Java アプリケーションをコンパイルおよび実行する場合の、メインタスクの一部です。このステップを完了したら、105 ページの『ターゲットとなる Palm OS での Java サンプル・アプリケーションのコンパイルおよび実行』に戻ります。

手順:

DB2 Everyplace JDBC ドライバー (および `java.sql` パッケージ) をビルド・パスへ追加するには、次のようにします。

1. 「Java パースペクティブ」の「パッケージ・エクスプローラー」ビューにある「DB2 Everyplace Sample for Palm OS CLDC」プロジェクトを右クリックして、ポップアップ・メニューから「**プロパティー**」をクリックします。
2. オープンしたプロパティー・ウィンドウで、左ペインの「Java ビルド・パス」をクリックした後で、右ペインの「ライブラリー」タブをクリックします。

3. 「**外部 JAR の追加**」をクリックします。「JAR 選択」ウィンドウで、<DB2Everyplace>¥¥Clients¥PalmOS¥database¥JDBC¥cldc¥db2ejdbc.jar をブラウズし、「**開く**」をクリックします。
4. 前のステップを繰り返して database_enabler_cldc.jar および DB2eJDBC_Cldc_maps.jar をビルド・パスへ追加します。
5. 「DB2 Everyplace Sample for Palm OS CLDC のプロパティー (Properties for DB2 Everyplace Sample for Palm OS CLDC)」ウィンドウに戻り、「**OK**」をクリックします。

105 ページの『ターゲットとなる Palm OS での Java サンプル・アプリケーションのコンパイルおよび実行』に戻ります。

Palm OS 用 WSDD への DB2eAppl.java のインポート

以下のステップは 「DB2 Everyplace Sample for Palm OS CLDC」プロジェクトに適用されます。「DB2 Everyplace Sample for Palm OS XTR」プロジェクトにも類似のステップがあります。

このタスクは、Palm OS ターゲットのサンプル Java アプリケーションをコンパイルおよび実行する場合の、メインタスクの一部です。このステップを完了したら、105 ページの『ターゲットとなる Palm OS での Java サンプル・アプリケーションのコンパイルおよび実行』に戻ります。

手順:

WSDD へ DB2eAppl.java をインポートするには、以下のようにします。

1. 「Java パースペクティブ」の「パッケージ・エクスプローラー」ビューで、「DB2 Everyplace Sample for Palm OS CLDC」プロジェクト内の src フォルダーを右クリックして、 ポップアップ・メニューから「**インポート**」をクリックします。
2. 「インポート」ウィンドウの「選択」ページで、インポート元として「ファイル・システム」を選択した後、「**次へ**」をクリックします。
3. 「インポート」ウィンドウの「ファイル・システム」ページで、「**ブラウズ**」をクリックします。
4. <DB2Everyplace>¥Clients¥PalmOS¥database¥JDBC¥cldc¥sample フォルダーをブラウズし、「**OK**」をクリックします。
5. 右ペインの「**DB2eAppl.java**」チェック・ボックスを選択したあとで、「**完了**」をクリックします。

105 ページの『ターゲットとなる Palm OS での Java サンプル・アプリケーションのコンパイルおよび実行』に戻ります。

Palm OS エミュレーターでの DB2eAppl.java の実行

以下のステップは、「DB2 Everyplace Sample for Palm OS CLDC」プロジェクトに適用されます。「DB2 Everyplace Sample for Palm OS XTR」プロジェクトにも類似のステップがあります。

このタスクは、Palm OS ターゲットでサンプル Java アプリケーションをコンパイルおよび実行する場合のメインタスクの一部です。このステップを完了したら、105 ページの『ターゲットとなる Palm OS での Java サンプル・アプリケーションのコンパイルおよび実行』に戻ります。

前提条件:

まだ DB2 Everyplace JDBC ドライバーを使用するようにシステムをセットアップしていない場合は、以下の JDBC ドライバー用のファイルをデバイスにインストールしてください。

```
<DB2 Everyplace>¥Clients¥PalmOS¥database¥JDBC¥cldc¥DB2eJDBC.prc
<DB2 Everyplace>¥Clients¥PalmOS¥database¥JDBC¥cldc¥DB2eJDBC_Cldc.prc
```

「DB2 Everyplace Sample for Palm OS XTR」プロジェクトの作業を行っている場合は、代わりに、次の JDBC ドライバー用のファイルをデバイスにインストールしてください。

```
<DB2 Everyplace>¥Clients¥PalmOS¥database¥JDBC¥xtr¥DB2eJDBC.prc
<DB2 Everyplace>¥Clients¥PalmOS¥database¥JDBC¥xtr¥DB2eJDBC_Xtr.prc
```

手順:

Palm OS エミュレーターで DB2eAppl.java を実行するには、次のようにします。

1. 次のようにして、Palm OS エミュレーターを構成します。
   a. 「**デバイス**」 —> 「**構成**」とクリックします。
   b. 「デバイス構成 (Device Configurations)」ウィンドウで、左ペインの Palm Emulator を選択して、「**新規**」をクリックします。
   c. 右ペインに表示される構成で、次の情報を入力します。
      - 「**デバイス名**」フィールドに DB2 Everyplace Palm Emulator と入力します。
      - 「**Palm エミュレーター実行可能プログラム (Palm emulator executable)**」フィールドで、 <*PalmEmulator*>¥Emulator.exe をブラウズします (ここで、<*PalmEmulator*> は Palm エミュレーターをインストールしたディレクトリーです)。
      - 「**エミュレーター実行引き数 (Emulator run arguments)**」フィールドで、-psf <*file*>.psf と入力します (ここで、<*file*>.psf は DB2 Everyplace および J9 VM がインストールされた .psf ファイルです)。
   d. 「適用」をクリックした後、「OK」をクリックします。
2. DB2eAppl.java をビルドします。
   a. 「Java パースペクティブ」の「パッケージ・エクスプローラー」ビューで、「DB2 Everyplace Sample for Palm OS CLDC」プロジェクト用の wsddbuild.xml ファイルをダブルクリックします。
   b. wsddbuild.xml のエディターで、「**ビルドの追加 (Add Build)**」をクリックします。
   c. 「**プラットフォーム**」リストから J9 for Palm 68k を選択し、「**メイン・クラス**」および「**ビルド名 (Buildname)**」フィールドをデフォルト値のままにして、「**次へ**」をクリックします。

d. PalmOS 設定ページで、「**作成者 ID (Creator id)**」フィールドに DB2e と入力し、「**アプリケーション名 (App Name)**」フィールドに DB2eAppl と入力した後、「**次へ**」をクリックします。

e. 「Jxelink オプション (Jxelink Option)」ページはデフォルトのままにして、「**完了** 」をクリックします。

3. 「DB2eAppl.jxeLinkOptions」ファイルを変更します。

a. 「Java パースペクティブ」の「パッケージ・エクスプローラー」で、「DB2 Everyplace Sample for Palm OS CLDC」プロジェクト用の palm68k フォルダーを展開します。 DB2eAppl.jxeLinkOptions をダブルクリックして、DB2eAppl.jxeLinkOptions 用のエディターをオープンします。

b. エディターで、「入力」タブをクリックします。「マップ・ファイルからのクラスの読み取り (前提条件) (Read classes from map files (prereq))」セクションで「**新規**」をクリックします。表示される「前提条件の追加 (Add prereq)」ウィンドウで、前提条件として「DB2eJDBC_Cldc」を入力し、「**OK**」をクリックします。**注:** 「DB2 Everyplace Sample for Palm OS XTR」プロジェクトの作業を行っている場合は、代わりに、前提条件として DB2eJDBC_Xtr と入力し、次の 2 つのステップを省略してください。

c. エディターで、「Jxe」タブをクリックします。「Jxe プラットフォーム情報 (Jxe Platform Information)」で、「jxe セクションを実行するときに VM オプションを使用する (Use VM options when running jxe section)」に対して「**新規**」をクリックします。

d. 表示される「VM オプションの追加 (Add VM options)」ウィンドウで、VM オプション用に -jcl:cldc:loadlibrary=db2ejdbc と入力して「**OK**」をクリックします。

e. Ctrl+S を押して、変更内容を保管します。

f. wsddbuild.xml のエディターで、jxe2prc palm68k/DB2eAppl を選択し、「**ビルドの実行 (Perform Build)**」をクリックします。

4. DB2eAppl.java を実行します。

a. メイン・メニューから「**実行**」 —>「**実行**」とクリックします。「起動構成」ウィンドウがオープンします。

b. 「起動構成」ウィンドウで、左ペインの Device Java Application を選択し、「**新規**」をクリックします。

c. 右ペインに表示される構成で、「**名前**」フィールドに DB2eAppl Palm CLDC と入力します。

d. 「Java アプリケーション」パネルで、次の情報を入力します。

1) 「プロジェクト」フィールドで、「DB2 Everyplace Sample for Palm OS CLDC」をブラウズします。

2) 「Java アプリケーション」フィールド内の「**検索**」をクリックします。

3) 「ターゲットの選択」ウィンドウで、「DB2eAppl.prc (Target "jxe2prc palm68k/DB2eAppl" in wsddbuild.xml)」を選択した後、「**完了**」をクリックします。

4) 「**デバイスまたは JRE (Device or JRE)**」リストから DB2 Everyplace Palm Emulator を選択します。

5) 「起動構成」ウィンドウに戻り、「**適用**」をクリックした後、「**実行**」をクリックします。Palm エミュレーターが始動し、DB2eAppl を実行します。Palm エミュレーター画面、または .psf ファイルが入っているディレクトリーの `j9stdout.txt` ファイルで、サンプル・アプリケーションの出力を見ることができます。J9 Java VM の「標準出力を画面に表示 (Display Stdout on Screen)」プリファレンスを変更していない場合、出力は `j9stdout.txt` ファイルにあります。また、`j9stderr.txt` でエラーがないかチェックしてください。

105 ページの『ターゲットとなる Palm OS での Java サンプル・アプリケーションのコンパイルおよび実行』に戻ります。

ターゲットとなる Palm OS 以外の OS 上での Java サンプル・アプリケーションのコンパイルと実行

以下の一連のトピックでは、WebSphere Studio Device Developer (WSDD) 5.5 と J9 Java 仮想マシンを使用して Java のサンプル・コードをコンパイルし実行する方法について説明します。

開発環境として、WSDD を使用することをお勧めします。WSDD では J9 VM を使用しますが、これはユーザーのデバイスのプロセッサー・タイプをサポートしない可能性があります。 これとは異なる開発環境と JVM をご使用になる場合は、JVM が JNI をサポートしていることを確認してください (DB2 Everyplace の JDBC ドライバーが JNI を使用するため)。その他の互換性のある JVM としては、Sun PersonalJava、Insignia Jeode、および NSIcom CrEme などがあります。 WSDD の評価版は http://www.ibm.com/software/pervasive/products/wsdd/ からダウンロードできます。

前提条件:

1. 以下のソフトウェアがインストール済みであることを確認する。
   - J9 Java 仮想マシンまたはその他の互換 JVM を組み込んだ WSDD 5.5
2. WSDD 資料に従って、使用するターゲットおよび開発環境を準備する。 WSDD サンプル・アプリケーションを作成および実行して WSDD のインストールを検証してください。
3. 使用しているターゲット・デバイスに DB2 Everyplace をインストールする。指示の詳細については、「*DB2 Everyplace インストールおよびユーザーズ・ガイド*」を参照してください。

手順:

ターゲットとなる Palm OS 以外の OS 上で Java サンプル・コードをコンパイルし実行するには、次のようにします。

1. WSDD の WCE ツールをインストールする。
2. WSDD プロジェクトを作成し、jar ファイルを `DB2eAppl.java` のビルド・パスに追加する。
3. `DB2eAppl.java` を WSDD にインポートする。
4. `DB2eAppl.java` を実行する。使用するステップはご使用のオペレーティング・システムによって変わります。

- 114 ページの『Win32 での DB2eAppl.java の実行』
- 115 ページの『Windows CE での DB2eAppl.java の実行』
- 118 ページの『QNX Neutrino または組み込み Linux での DB2eAppl.java の実行』
- 119 ページの『Symbian での DB2eAppl.java の実行』

関連したタスク:

- 105 ページの『ターゲットとなる Palm OS での Java サンプル・アプリケーションのコンパイルおよび実行』

非 Palm OS ターゲット用 WSDD の WCE ツールのインストール

このタスクは、非 Palm OS ターゲットでサンプル Java アプリケーションをコンパイルおよび実行する場合の、メインタスクの一部です。このステップを完了したら、 111 ページの『ターゲットとなる Palm OS 以外の OS 上での Java サンプル・アプリケーションのコンパイルと実行』に戻ります。

手順:

1. WSDD で、「**ヘルプ**」 —> 「**ソフトウェア更新**」 —> 「**更新マネージャー**」をクリックして「パースペクティブのインストール/更新」をオープンします。
2. 「パースペクティブのインストール/更新」の「フィーチャーの更新」ビューで、「**アクセス先サイト**」 —> 「**WebSphere Custom Environment**」 —> 「**WebSphere Custom Environment**」と展開します。
3. 「WCE Tooling for WSDD 5.5.0」を選択します。
4. 「プレビュー」ビューで、「**インストール**」をクリックします。
5. インストール手順に従って、WCE Tooling for WSDD フィーチャーをインストールします。
6. WCE データベース・イネーブラー・ライブラリーおよび WCE jclMax クラス・ライブラリーを、同様のステップでインストールします。

111 ページの『ターゲットとなる Palm OS 以外の OS 上での Java サンプル・アプリケーションのコンパイルと実行』に戻ります。

非 Palm OS ターゲットでの WSDD プロジェクトの作成および DB2eAppl.java のビルド・パスへの jar ファイルの追加

このタスクは、非 Palm OS ターゲットでサンプル Java アプリケーションをコンパイルおよび実行する場合の、メインタスクの一部です。このステップが完了したら、 111 ページの『ターゲットとなる Palm OS 以外の OS 上での Java サンプル・アプリケーションのコンパイルと実行』に戻ります。

手順:

1. WSDD で、「**ウィンドウ**」 —>「**パースペクティブを開く**」 —>「**Java**」 とクリックして Java パースペクティブを切り替えます。
2. DB2 Everyplace Sample プロジェクトを作成します。
   a. 「**ファイル**」 —>「**新規**」 —>「**その他**」とクリックします。

b. 「新規プロジェクト」ウィンドウの「選択」ページで、左ペインの「WCE for J9」を選択し、右ペインの「WCE プロジェクトの作成 (Create WCE project)」を選択して、「**次へ**」をクリックします。

c. 「新規プロジェクト」ウィンドウの「カスタム・プロジェクト(Custom Project)」ページで、プロジェクト名に「DB2 Everyplace Sample」を入力し、「**次へ**」をクリックします。

d. 「新規プロジェクト」ウィンドウの「ライブラリー選択」ページで、WCE jclMax (jclMax) を選択した後で、「**次へ**」をクリックします。

e. 「新規プロジェクト」ウィンドウの「Java 設定」ページで、「**ライブラリー**」タブをクリックします。

f. db2ejdbc.jar をビルド・パスへ追加するには、以下のようにします。

   1) 「**外部 JAR の追加**」をクリックします。

   2) 「JAR 選択」ウィンドウで、***<DB2Everyplace>***¥Clients¥Win32¥database¥jdbc¥db2ejdbc.jar をブラウズし、「**開く**」をクリックします。

g. 「Java 設定」ページに戻り、database_enabler.jar をビルド・パスへ追加します。

   1) 「**外部 JAR の追加**」をクリックします。

   2) 「JAR 選択」ウィンドウで、*<WSDD>*¥wsdd5.0¥ive¥lib¥jclMax¥database_enabler.jar をブラウズし、「**開く**」をクリックします。

h. 「新規プロジェクト」ウィンドウの「Java 設定」ページに戻り、「**完了**」をクリックします。

111 ページの『ターゲットとなる Palm OS 以外の OS 上での Java サンプル・アプリケーションのコンパイルと実行』に戻ります。

非 Palm OS ターゲット用 WSDD への DB2eAppl.java のインポート

このタスクは、非 Palm OS ターゲットでサンプル Java アプリケーションをコンパイルおよび実行する場合の、メインタスクの一部です。このステップが完了したら、111 ページの『ターゲットとなる Palm OS 以外の OS 上での Java サンプル・アプリケーションのコンパイルと実行』に戻ります。

手順:

非 Palm OS ターゲット用 WSDD へ DB2eAppl.java をインポートするには、以下のようにします。

1. 「Java パースペクティブ」の「パッケージ・エクスプローラー」で、DB2 Everyplace Sample プロジェクトを展開し、src フォルダーを選択して「**ファイル**」 —>「**インポート**」とクリックします。
2. 「インポート」ウィンドウの「**選択**」ページで、 インポート元として「ファイル・システム」を選択した後に、「**次へ**」をクリックします。

3. 「インポート」ウィンドウの「**ファイル・システム**」ページで、「**ディレクトリー**」フィールド の「**ブラウズ**」をクリックして、*<DB2Everyplace>*¥Clients¥Win32¥database¥jdbc フォルダーをブラウズし、「**OK**」をクリックします。
4. 「インポート」ウィンドウの「**ファイル・システム**」ページに戻り、右ペインの「**DB2eAppl.java**」チェック・ボックスを選択した後、「**完了**」をクリックします。

111 ページの『ターゲットとなる Palm OS 以外の OS 上での Java サンプル・アプリケーションのコンパイルと実行』に戻ります。

サンプル Java アプリケーションの実行

この章では、サンプル Java アプリケーションを実行する方法について説明します。説明されているトピックは、以下のとおりです。

Win32 での DB2eAppl.java の実行

このタスクは、非 Palm OS ターゲットでサンプル Java アプリケーションをコンパイルおよび実行する場合のメインタスクの一部です。このステップを完了したら、111 ページの『ターゲットとなる Palm OS 以外の OS 上での Java サンプル・アプリケーションのコンパイルと実行』に戻ります。

前提条件:

まだ DB2 Everyplace JDBC ドライバーを使用するようにシステムをセットアップしていない場合は、次のようにします。

1. set コマンドを使用して、ディレクトリー <DB2Everyplace>¥Clients¥Win32¥database¥x86 を PATH システム変数に組み込みます。
2. set コマンドを使用して、ファイル <DB2Everyplace>¥Clients¥Win32¥database¥jdbc¥db2ejdbc.jar を CLASSPATH システム変数に組み込みます。

注: WSDD がオープンしている場合、この変更を WSDD に反映させるために WSDD を再始動する必要があります。

手順:

Windows ワークステーションで DB2eAppl.java を実行するには、次のようにします。

1. DB2eAppl.java をビルドします。

a. 「Java パースペクティブ」の「パッケージ・エクスプローラー」ビューで、DB2 Everyplace Sample プロジェクト用の wsddbuild.xml ファイルをダブルクリックします。

b. wsddbuild.xml のエディターで、「**ビルドの追加 (Add Build)**」ボタンをクリックします。

c. 「新規 Ant ビルド・ターゲットの作成 (Create New Ant Build Target)」ウィンドウで、「**メイン・クラス (Main class)**」フィールドの「**ブラウズ**」をクリックします。

d. 開いているウィンドウで、DB2eAppl - (default package) - DB2 Everyplace Sample/src を選択し、「**完了**」をクリックします。

e. 「新規 Ant ビルド・ターゲットの作成 (Create New Ant Build Target)」ウィンドウに戻り、プラットフォーム・リストから J9 for Windows X86 を選択し、「ビルド名 (Buildname)」フィールドはデフォルトのままにして「**次へ**」をクリックします。

f. 「Jxelink オプション (Jxelink Options)」ページはデフォルトのままにして、「**完了**」をクリックします。

g. wsddbuild.xml のエディターに戻り、smartlink winx86/DB2eAppl を選択して「**ビルドの実行 (Perform Build)**」をクリックします。

2. DB2eAppl.java を実行します。

a. **「実行」 —> 「実行」** とクリックします。「起動構成」ウィンドウがオープンします。

b. 「起動構成」ウィンドウで、左ペインの Java アプリケーションを選択して「**新規**」をクリックします。

c. 右ペインに表示される構成で、「**新規**」フィールドに DB2eAppl Win32 と入力します。

d. メイン・ページで、 以下のステップを実行します。

- 「**プロジェクト**」フィールドで 「**ブラウズ**」をクリックします。「プロジェクト選択 (Project Selection)」ウィンドウで、 DB2 Everyplace Sample を選択し、 「**OK**」をクリックします。
- 「**メイン・クラス (Main class)**」フィールドで、「**検索**」をクリックします。「メイン・タイプの選択 (Choose Main Type)」ウィンドウで DB2eAppl を選択し、「**OK**」をクリックします。

e. 「起動構成」ウィンドウに戻り、「**適用**」をクリックした後に、「**実行**」をクリックします。WSDD コンソールにサンプル・アプリケーションの出力が表示されます。

111 ページの『ターゲットとなる Palm OS 以外の OS 上での Java サンプル・アプリケーションのコンパイルと実行』に戻ります。

Windows CE での DB2eAppl.java の実行

このタスクは、非 Palm OS ターゲットでサンプル Java アプリケーションをコンパイルおよび実行する場合のメインタスクの一部です。このステップを完了したら、111 ページの『ターゲットとなる Palm OS 以外の OS 上での Java サンプル・アプリケーションのコンパイルと実行』に戻ります。

前提条件:

まだ DB2 Everyplace JDBC ドライバーを使用するようにシステムをセットアップしていない場合は、次のステップを実行します。

1. ファイル <DB2Everyplace>¥Clients¥WinCE¥database¥*proc*¥*ver*¥db2ejdbc.dll および <DB2Everyplace>¥Clients¥WinCE¥database¥jdbc¥db2ejdbc.jar を使用中のデバイスの ¥Windows ディレクトリーにコピーします (ここで、*proc* はプロセッサー・タイプで、*ver* はデバイス上の Windows CE オペレーティング・システムのバージョン番号です)。
2. Windows CE Remote Registry Editor を使用して、デバイスのクラスパスに以下のファイルを組み込むように、デバイスのレジストリーを変更します。

   ```
   ¥Windows¥db2ejdbc.jar
   ¥wsdd¥lib¥jclMax¥database_enabler.jar
   ```

   ただし、J9 を使用中のデバイスのルート・ディレクトリーにインストールしていると想定しています。

   代わりに、WSDD が生成した DB2eAppl ショートカットを更新して、上記のファイルを以下のクラスパスに組み込めます。

```
256#"¥wsdd¥bin¥j9.exe" "-Xbootclasspath:¥Windows¥db2ejdbc.jar;
¥wsdd¥lib¥jclMax¥database_enabler.jar;¥wsdd¥lib¥jclMax¥classes.zip;
¥wsdd¥lib¥jclMax¥locale.zip;¥wsdd¥lib¥charconv.zip" "-jcl:max" "-jxe:¥Temp¥DB2eAppl.jxe"
```

手順:

Windows CE デバイスで DB2eAppl.java を実行するには、次のようにします。

1. Windows CE デバイスを構成します。
   a. **「デバイス」—> 「構成」** とクリックします。
   b. 「デバイス構成 (Device Configurations)」ウィンドウで左ペインの PocketPC Handheld を選択して、「**新規**」をクリックします。
   c. 右ペインに表示される構成で、次のステップを実行します。
      1) 「**デバイス名**」フィールドで、DB2 Everyplace PocketPC Handheld と入力します。
      2) 「**J9 ランタイム・ロケーション (J9 runtime location)**」フィールドで「**ブラウズ**」をクリックします。 「デバイス」ウィンドウの「フォルダーのブラウズ (Browse for Folder)」で、wsdd (デバイスのルート・ディレクトリーに J9 をインストールしたと想定) を選択し、「**OK**」をクリックします。
      3) 「**アプリケーションのインストール・ロケーション (Application install location)**」フィールドで、「**ブラウズ**」をクリックします。 「デバイス」ウィンドウの「フォルダーのブラウズ (Browse for Folder)」で Temp を選択し、「**OK**」をクリックします。
      4) 「**ショートカットのインストール・ロケーション (Shortcut install location)**」フィールドで「**ブラウズ**」をクリックします。「デバイス」ウィンドウの「フォルダーのブラウズ (Browse for Folder)」で Temp を選択し、「**OK**」をクリックします。
   d. 「デバイス構成 (Device Configurations)」ウィンドウに戻り、「**適用**」をクリックした後、「**OK**」をクリックします。

2. DB2eAppl.java をビルドします。
   a. 「Java パースペクティブ」の「パッケージ・エクスプローラー」ペインで、DB2 Everyplace Sample プロジェクト用の wsddbuild.xml ファイルをダブルクリックします。
   b. wsddbuild.xml のエディターで、「**ビルドの追加 (Add Build)**」ボタンをクリックします。
   c. 「新規 Ant ビルド・ターゲットの作成 (Create New Ant Build Target)」ウィンドウで、「**メイン・クラス (Main class)**」フィールドの「**ブラウズ**」をクリックします。
   d. 開いているウィンドウで、DB2eAppl - (default package) - DB2 Everyplace Sample/src を選択し、「**完了**」をクリックします。
   e. 「新規 Ant ビルド・ターゲットの作成 (Create New Ant Build Target)」ウィンドウに戻り、「**プラットフォーム**」リストから J9 for PocketPC ARM を選択し、「**ビルド名 (Buildname)**」フィールドはデフォルトのままにして「**次へ**」をクリックします。
   f. 「**Jxelink オプション (Jxelink Options)**」ページはデフォルトのままにして、「**完了**」をクリックします。
   g. wsddbuild.xml のエディターで、smartlink ppcarm/DB2eAppl を選択し、「**ビルドの実行 (Perform Build)**」をクリックします。
3. DB2eAppl.java を実行します。
   a. **「実行」 —> 「実行」**とクリックします。「起動構成」ウィンドウがオープンします。
   b. 「起動構成」ウィンドウで、左ペインの Device Java Application を選択し、「**新規**」をクリックします。
   c. 右ペインに表示される構成で、「**名前**」フィールドに DB2eAppl WinCE と入力します。
   d. メイン・ページで、以下のステップを実行します。
      1) 「**プロジェクト**」フィールドで 「**ブラウズ**」をクリックします。「プロジェクト選択 (Project Selection)」ウィンドウで、 DB2 Everyplace Sample を選択し、 「**OK**」をクリックします。
      2) 「Java アプリケーション」フィールドで「**検索**」をクリックします。「ターゲットの選択」ウィンドウで、DB2eAppl.jxe (Target "smartlink ppcarm/DB2eAppl" in wsddbuild.xml) をクリックし、「**完了**」をクリックします。
      3) **デバイスまたは JRE** のリストから DB2 Everyplace PocketPC Handheld を選択します。
   e. 「起動構成」ウィンドウに戻り、「**適用**」をクリックした後に、「**実行**」をクリックします。デバイスの J9 Console にサンプル・アプリケーションの出力が表示されます。

111 ページの『ターゲットとなる Palm OS 以外の OS 上での Java サンプル・アプリケーションのコンパイルと実行』に戻ります。

QNX Neutrino または組み込み Linux での DB2eAppl.java の実行

このタスクは、非 Palm OS ターゲットでサンプル Java アプリケーションをコンパイルおよび実行する場合の、メインタスクの一部です。このステップを完了したら、111 ページの『ターゲットとなる Palm OS 以外の OS 上での Java サンプル・アプリケーションのコンパイルと実行』に戻ります。

前提条件:

まだ DB2 Everyplace JDBC ドライバーを使用するようにシステムをセットアップしていない場合は、次のようにします。

1. export コマンドを使用して、ユーザーのデバイスに適切な libdb2e.so および libdb2ejdbc.so ネイティブ・ライブラリーを含むディレクトリー (1 つ以上) を LD_LIBRARY_PATH システム変数に組み込みます。

手順:

QNX Neutrino または組み込み Linux デバイスで `DB2eAppl.java` を実行するには、次のようにします。

1. `DB2eAppl.java` をビルドします。
   a. Java パースペクティブの「パッケージ・エクスプローラー」ペインで、DB2 Everyplace サンプル・プロジェクト用の `wsddbuild.xml` ファイルをダブルクリックします。
   b. `wsddbuild.xml` のエディターで、「**ビルドの追加 (Add Build)**」をクリックします。
   c. 「新規 Ant ビルド・ターゲットの作成 (Create New Ant Build Target)」ウィンドウで、「**ブラウズ**」をクリックしてメイン・クラスをブラウズします。オープンしている「ターゲットの選択」ウィンドウで、「DB2eAppl - (default package) - DB2 Everyplace Sample/src」を選択し、「**完了**」をクリックします。
   d. 「新規 Ant ビルド・ターゲットの作成 (Create New Ant Build Target)」の「ビルドのセットアップ (Set up build)」ページに戻り、「**プラットフォーム**」リストから適切なプラットフォームを選択し、「ビルド名 (Buildname)」フィールドはデフォルトのままにして「**次へ**」をクリックします。
   e. 「Jxelink オプション (Jxelink Option)」ページはデフォルトのままにして、「**完了** 」をクリックします。
   f. wsddbuild.xml のエディターに戻り、適切なビルドを選択して「**ビルドの実行（Perform Build)**」をクリックします。
2. `DB2eAppl.java` を実行します。
   a. 以下から適切な DB2eAppl.jxe ファイルをデバイスにコピーします。

      ```
      <WSDD>¥workspace¥DB2 Everyplace Sample¥<target>
      ```

      ここで、<target> はターゲット・デバイスおよびプロセッサー・タイプを表します。
   b. 以下のコマンドを使用してアプリケーションを開始します。

      ```
      j9 -Xbootclasspath:/wsdd/lib/jclMax/classes.zip:/wsdd/lib/jclMax/database_enabler.jar -cp:
      /DB2e/db2ejdbc.jar:. -jxe:DB2eAppl.jxe
      ```

111 ページの『ターゲットとなる Palm OS 以外の OS 上での Java サンプル・アプリケーションのコンパイルと実行』に戻ります。

Symbian での DB2eAppl.java の実行

このタスクは、非 Palm OS ターゲットでサンプル Java アプリケーションをコンパイルおよび実行する場合の、メインタスクの一部です。このステップが完了したら、 111 ページの『ターゲットとなる Palm OS 以外の OS 上での Java サンプル・アプリケーションのコンパイルと実行』に戻ります。

手順:

Symbian デバイスの一部は JVM に付属しています。 テキスト・ベースの Java アプリケーション (例えば、Java サンプル・プログラム) を実行したい場合は、Redirect (Symbian SDK for Java で `Redirect.sis` として提供される) をインストールして、テキスト・ベースのアプリケーションを開始する前に Redirect アプリケーションを開始する必要があります。テキスト出力は Redirect によってキャプチャーされます。

111 ページの『ターゲットとなる Palm OS 以外の OS 上での Java サンプル・アプリケーションのコンパイルと実行』に戻ります。

第 15 章 Visual Basic サンプル・アプリケーション

この章では Visual Basic サンプル・アプリケーションについての情報を提供します。説明されているトピックは、以下のとおりです。

- 『Visual Basic サンプル・アプリケーションの概要』
- 125 ページの『サンプル Visual Basic プログラムのコンパイルおよびテスト』

Visual Basic サンプル・アプリケーションの概要

この Visual Basic サンプル・アプリケーションは、Visual Basic を使用して DB2 Everyplace データをアクセスする方法を示します。Pocket PC (WinCE) と Win32 の両方のオペレーティング・システムで、同じアプリケーション・ロジックとユーザー・インターフェースを持つアプリケーションを開発することができます。 DB2 Everyplace には、2 つの Visual Basic サンプル・アプリケーションが提供されています。1 つは Pocket PC (WinCE) オペレーティング・システム用であり、他の 1 つは Win32 オペレーティング・システム用です。 これら 2 つのサンプル・アプリケーションのアプリケーション・ロジックとユーザー・インターフェースは同じです。アプリケーション・ロジックが含まれる db2evb.bas ファイルは、両方のオペレーティング・システムで共通に使用できます。詳細については、122 ページを参照してください。

サンプル・アプリケーションに含まれているファイル:

サンプル・アプリケーションの入った Visual Basic プロジェクト・ディレクトリーは、DB2 Everyplace をインストールしたディレクトリーの下にあります。Pocket PC (WinCE) の場合、これらのファイルは、¥db2everyplace¥clients¥**wince**¥database¥visualbasic にあります。Win32 オペレーティング・システムの場合、これらのファイルは、¥db2everyplace¥clients¥**win32**¥database¥visualbasic にあります。

この Visual Basic サンプル・アプリケーションには、以下のファイルが含まれています。

db2evb.bas
db2evb.bas ファイルには Visual Basic サンプル・アプリケーションが入っています。このサンプル・アプリケーションを使用すると、独自の Visual Basic アプリケーションを作成する際に役立ちます。

db2ecli.bas
db2ecli.bas ファイルは、DB2 Everyplace データベースに接続する Visual Basic インターフェースです。このファイルでは、 sqlcli.h、sqlcli1.h、sqlext.h、および sqlsystm.h に含まれている各種の DB2 Everyplace 制約を定義しています。このファイルで定義しているのは、最も一般的な制約のみです。必要であれば、sqlcli.h、sqlcli1.h、sqlext.h、および sqlsystm.h の他の制約も追加することができます。

DB2eForms (拡張子はオペレーティング・システムによって異なる)
アプリケーション・ユーザー・インターフェース・ファイル。

DB2eSample.exe (WinCE の場合は、DB2eSample.vb)
アプリケーション実行可能ファイル。

DB2eSample.vbw
アプリケーション・プロジェクト・ファイル。

DB2eSample.vbp (WinCE の場合は、DB2eSample.ebp)
アプリケーション・プロジェクト・ファイル。

Visual Basic 例: db2evb.bas:

サンプル・アプリケーション (db2evb.bas) で使用されている主なステップは、以下のとおりです。

DB2 Everyplace データベースに接続する。
- ステップ 1: 環境ハンドルを割り振る。
- ステップ 2: データベース・ハンドルを割り振る。
- ステップ 3: データベースに接続する。
- ステップ 4: ステートメント・ハンドルを割り振る。

DB2 Everyplace データにアクセスする。
- ステップ 5: 表を作成する。
- ステップ 6: データを表に挿入する。
- ステップ 7: 表からデータを検索する。

アプリケーションを終了する。

注: 終了する前に、アプリケーションが DB2 Everyplace データベースとの接続をクローズするようにしてください。

サンプル・アプリケーションのステップを説明するために、この例にコメントが追加されています。

```
Option Explicit

Public henv    As Long          ' Environment handle
Public hdbc    As Long          ' Database handle
Public hstmt   As Long          ' Statement handle
Public rc      As Integer       ' Return code

Public dbpath As String        ' filesystem path where DB2e will create tables.
Public userid As String        ' Userid: not used by DB2 Everyplace.
Public pass   As String        ' Password: not used by DB2 Everyplace

'------------------------------------------------------------------------------
' Function: DB2eTest
'
' Description: Function illustrating how calls to DB2 Everyplace can be made.
'
'------------------------------------------------------------------------------

Public Function DB2eTest() As Integer
    Dim errmsg As String
    Dim numCols As Integer
    Dim i As Integer
    Dim retLen As Long
    Dim data As String
    Dim crtStmt As String
    Dim insStmt1 As String
    Dim insStmt2 As String
    Dim selStmt As String
    On Error Resume Next 'Important: don't ask me why, but this line is needed
                         'in every function that calls functions from db2e.dll
                         'otherwise visual basic does strange mysterious things.
    '
    dbpath = ""
```

```
userid = ""
pass = ""
'
crtStmt = "CREATE TABLE x(a INT, b TIMESTAMP)"
insStmt1 = "INSERT INTO x VALUES(1, CURRENT TIMESTAMP)"
insStmt2 = "INSERT INTO x VALUES(2, CURRENT TIMESTAMP)"
selStmt = "SELECT * FROM x"
'
data = String(80, " ")
' ステップ 1: 環境ハンドルを割り振る。
'

DB2eForm.DB2eText.Text = vbCrLf & vbCrLf & " Allocating an environment handle"

rc = SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HENV, henv)
If (rc <> 0) Then
    rc = DB2eError()
    rc = DB2eTerminate()
    Exit Function
End If

'
' ステップ 2: データベース・ハンドルを割り振る
'

DB2eForm.DB2eText.Text = DB2eForm.DB2eText.Text & vbCrLf &
                         " Allocating a database handle"

rc = SQLAllocHandle(SQL_HANDLE_DBC, henv, hdbc)
If (rc <> 0) Then
    rc = DB2eError()
    rc = DB2eTerminate()
    Exit Function
End If

'
' ステップ 3: データベースに接続する
'
DB2eForm.DB2eText.Text = DB2eForm.DB2eText.Text & vbCrLf &
                         "  Connecting to the database"

rc = SQLConnect(hdbc, dbpath, SQL_NTS, userid, SQL_NTS, pass, SQL_NTS)
If (rc <> 0) Then
    rc = DB2eError()
    rc = DB2eTerminate()
    Exit Function
End If

'
' ステップ 4: ステートメント・ハンドルを割り振る。
'

DB2eForm.DB2eText.Text = DB2eForm.DB2eText.Text & vbCrLf &
                         "  Allocating a statement handle"

rc = SQLAllocHandle(SQL_HANDLE_STMT, hdbc, hstmt)
If (rc <> 0) Then
    rc = DB2eError()
    rc = DB2eTerminate()
    Exit Function
End If

DB2eForm.DB2eText.Text = DB2eForm.DB2eText.Text & vbCrLf

'
' Now we can use CLI function calls to execute SQL statements.
'
' ステップ 5: 表を作成する
'

DB2eForm.DB2eText.Text = DB2eForm.DB2eText.Text & vbCrLf & "  " & crtStmt
rc = SQLExecDirect(hstmt, crtStmt, SQL_NTS)
If (rc <> 0) Then
    rc = DB2eError()
    rc = DB2eTerminate()
    Exit Function
End If

'
' Create the same table again to force an error message and
' see if DB2eError works.
'
'rc = SQLExecDirect(hstmt, "create table p(a int)", SQL_NTS)
'If (rc <> 0) Then
```

```
    '   testmsg = MsgBox("BLA1", 1, "DB2 Everyplace Visual Basic")
    '   rc = DB2eError()
    '   testmsg = MsgBox("BLA2", 1, "DB2 Everyplace Visual Basic")
    '   rc = DB2eTerminate()
    '   testmsg = MsgBox("BLA3", 1, "DB2 Everyplace Visual Basic")
    '   Exit Function
    'End If
    '

    '
    ' ステップ 6: 表にデータを挿入する。
    '

    DB2eForm.DB2eText.Text = DB2eForm.DB2eText.Text & vbCrLf & "  " & insStmt1
    rc = SQLExecDirect(hstmt, insStmt1, SQL_NTS)
    If (rc <> 0) Then
        rc = DB2eError()
        rc = DB2eTerminate()
        Exit Function

    End If

    DB2eForm.DB2eText.Text = DB2eForm.DB2eText.Text & vbCrLf & "  " & insStmt2
    rc = SQLExecDirect(hstmt, insStmt2, SQL_NTS)
    If (rc <> 0) Then
        rc = DB2eError()
        rc = DB2eTerminate()
        Exit Function

    End If

 DB2eForm.DB2eText.Text = DB2eForm.DB2eText.Text & vbCrLf

    '
    ' ステップ 7: 表からデータを検索する。
    '

    DB2eForm.DB2eText.Text = DB2eForm.DB2eText.Text & vbCrLf & "  " & selStmt
                           & vbCrLf

    rc = SQLExecDirect(hstmt, selStmt, SQL_NTS)
    If (rc <> 0) Then
        rc = DB2eError()
        rc = DB2eTerminate()
        Exit Function
    End If


    rc = SQLNumResultCols(hstmt, numCols)
    If (rc <> 0) Then
        rc = DB2eError()
        rc = DB2eTerminate()
        Exit Function
    End If

    Do While (SQLFetch(hstmt) = SQL_SUCCESS)
        For i = 1 To numCols
            rc = SQLGetData(hstmt, i, SQL_C_CHAR, data, 80, retLen)
            DB2eForm.DB2eText.Text = DB2eForm.DB2eText.Text & "  " & data & vbCrLf
            If (rc <> 0) Then
                rc = DB2eError()
                rc = DB2eTerminate()
                Exit Function
            End If
        Next
        data = String(80, " ")
        DB2eForm.DB2eText.Text = DB2eForm.DB2eText.Text & vbCrLf
    Loop

    '
    ' ステップ 8: アプリケーションの終了前に DB2e データベースとの接続をクローズする。
    '
    rc = DB2eTerminate()

    DB2eTest = 0
End Function
```

関連したタスク:

- 35 ページの『DB2 Everyplace Visual Basic アプリケーションの開発』

サンプル Visual Basic プログラムのコンパイルおよびテスト

手順:

DB2 Everyplace サンプル・プログラムをコンパイルしてテストするには、以下のようにします。

1. Visual Basic プロジェクト・ファイル DB2eSample.vbp (WinCE の場合は、DB2eSample.ebp) を開く。.
2. サンプル・プログラムをビルドする。
   - Win32 の場合: 「**ファイル**」->「**DB2eSample.exe**」と選択する。DB2eSample.exe がビルドされます。
   - WinCE の場合: 「**ファイル**」->「**DB2eSample.vb**」と選択する。DB2eSample.vb がビルドされます。
3. 以下のファイルをコピーする。
   - Win32 の場合: DB2e.dll (Win32 オペレーティング・システムの場合) を現行プロジェクト・ディレクトリー、または PATH 環境変数の DB2e.dll パスにコピーする。
   - WinCE の場合: DB2eSample.vb、 DB2e.dll (Pocket PC オペレーティング・システムの場合)、および Visual Basic Runtime を任意のディレクトリーにコピーする。
4. DB2Sample.exe (WinCE の場合は、DB2Sample.vb) を実行する。

関連した概念:

- 121 ページの『Visual Basic サンプル・アプリケーションの概要』

関連した解説:

- 202 ページの『DB2 CLI 関数の要約』
- 36 ページの『Visual Basic インターフェースがサポートされているオペレーティング・システム』

第 16 章 JSP サンプル・アプリケーション

以下にリストしたファイルは <DB2Everyplace>¥SDK¥JSP¥sample¥jsp ディレクトリーに関連しています。サンプル JSP アプリケーションはすべて、<DB2Everyplace>¥SDK¥JSP¥sample¥data ディレクトリーにある Visiting Nurse (巡回医療サービス) サンプル・データベースを使用します。チュートリアルは <DB2Everyplace>¥SDK¥JSP¥doc ディレクトリーに入っています。

WebSphere Studio Professional/Entry Edition v4.0 を使用して開発されたアプリケーション

Visiting Nurse Schedule (巡回医療サービス・スケジュール)

説明 このサンプルは動的に Visiting Nurse データベースに照会し、その結果を表にして表示します。

開始ページ

VNSchedule_ws40¥scheduleHTMLResults.jsp

チュートリアル

ws40.pdf

WebSphere Studio Application Developer v4.0 を使用して開発されたアプリケーション

Visiting Nurse Schedule (巡回医療サービス・スケジュール)

説明 このサンプルは動的に Visiting Nurse データベースに照会し、その結果を表にして表示します。このサンプル・アプリケーションは JDBC 2.0 を必要とし、Symbian OS バージョン 6 では実行できません。

開始ページ

VNSchedule_wsad40¥scheduleMasterView.jsp

その他のファイル

VNSchedule_wsad40¥web.xml
VNSchedule_wsad40¥dbbeans.jar

チュートリアル

wsad40.pdf

WebSphere Studio Application Developer v5.0 を使用して開発されたアプリケーション

Visiting Nurse Schedule (巡回医療サービス・スケジュール)

説明 このサンプルは動的に Visiting Nurse データベースに照会し、その結果を表にして表示します。

開始ページ

VNSchedule_wsad50¥scheduleMasterView.jsp

その他のファイル

VNSchedule_wsad50¥web.xml
VNSchedule_wsad50¥dbbeans.jar
VNSchedule_wsad50¥scheduleMasterViewBean.class

チュートリアル

wsad50.pdf

WebSphere Studio の外部で開発されたアプリケーション

Visiting Nurse (巡回医療サービス)

説明

Visiting Nurse サンプル・アプリケーションの説明については、「*DB2 Everyplace インストールおよびユーザーズ・ガイド*」を参照してください。

開始ページ

VisitingNurse¥schedule.jsp

その他のファイル

VisitingNurse¥contact.jsp
VisitingNurse¥medrecord.jsp
VisitingNurse¥person.jsp

関連したタスク:

- 39 ページの『Windows ワークステーションでの JSP サポートの検証』
- 40 ページの『Windows CE デバイスでの JSP 開発のためのセットアップ』

関連した概念:

- 37 ページの『DB2 Everyplace の JSP アプリケーションの開発』
- 38 ページの『DB2 Everyplace の JSP サポートの概要』

第 17 章 サンプル同期アプリケーション

この章ではサンプル同期アプリケーションについての情報を提供します。説明されているトピックは、以下のとおりです。

- 『同期クライアント C/C++ サンプル・アプリケーション』
- 131 ページの『Java ネイティブ同期アプリケーションのサンプル』
- 135 ページの『Java MIDP 同期アプリケーションのサンプル』
- 139 ページの『Sun Wireless Toolkit を使用した MIDP アプリケーション用の isync4j の開発』
- 141 ページの『ANT および Sun Wireless Toolkit コマンド行を使用した MIDP アプリケーション用の isync4j の開発』
- 143 ページの『GoISyncConsole サンプル Java 同期アプリケーションのコンパイルおよび実行』

同期クライアント C/C++ サンプル・アプリケーション

次の例は、アプリケーションを作成するための、いくつかの DB2 Everyplace 同期クライアント API 関数の使用方法を示しています。
¥DB2e¥Clients¥clientapisample¥C_API にもソース・コードの例があります。

```
/*****************************************************************/
/**
 * This function defines the sync listener. See isyncore.h for more
 * information.
 * param: listenerData, your personal data.
 * param: event, event object
 * param: pExtraInfo (reserved)
 * return: integer, when event type is ISCEVTTYPE_Retry:
 *   . ISCRTNCB_ReplyYes    : retry less than 3 times
 *   . ISCRTNCB_ReplyNo     : retry more than or equal to 3 times
 *     when event type is ISCEVTTYPE_Info:
 *   . ISCRTNCB_Done
 *     when event type is ISCEVTTYPE_Query and its event code is ISCEVT_QueLogin:
 *   . ISCRTNCB_Done        : username and password are entered correctly
 *   . ISCRTNCB_Default     : username and password are not entered
 *     others (ISCEVTTYPE_Fatal, ISCEVTTYPE_Error, ISCEVTTYPE_Query,
 *             and ISCEVTTYPE_Conflict)
 *   . ISCRTNCB_Default     : take default action
 **/
static isy_INT32 syncListener(
       isy_UINT32 listenerData,
       ISCEVT    *event,
       isy_VOID  *pExtraInfo)
{
    // appEventCodeToMessage is some user function to map an event code to
    // some descriptive event message
    char *statusMsg = appEventCodeToMessage(event);
    int   timesRetried;

    switch (event->type) {
        case ISCEVTTYPE_Fatal:
        case ISCEVTTYPE_Error:
            printf("Error: %s¥n", statusMsg);
            return ISCRTNCB_Default ;

        case ISCEVTTYPE_Retry:
            timesRetried = event->retry;
            if (timesRetried >= 3)
```

```
                    return ISCRTNCB_ReplyNo;
            else {
                char ans;
                printf("%s [Y/N] ", statusMsg);
                ans = getchar();
                getchar();
                if(tolower(ans) == 'y')
                        return ISCRTNCB_ReplyYes;
                else
                        return ISCRTNCB_ReplyNo;
            }

        case ISCEVTTYPE_Info:
            switch (event->code) {
                case ISCEVT_InfSucceeded:
                case ISCEVT_InfFailed:
                case ISCEVT_InfCanceled:
                    printf("Conclusion: %s¥n", statusMsg);
                    break;
                case ISCEVT_InfGeneral:
                case ISCEVT_InfCancelingSync:
                case ISCEVT_InfPrepMsg:
                case ISCEVT_InfSendMsg:
                case ISCEVT_InfWaitMsg:
                case ISCEVT_InfApplyMsg:
                    printf("Status: %s¥n", statusMsg);
                    break;
                default: // ignore other event code
                    break;
            } // switch (event->code)
            return ISCRTNCB_Done;

        case ISCEVTTYPE_Query:
            if (event->code == ISCEVT_QueLogin) {
                ISCLISTENARG *args = event->info;
                isy_TCHAR *target = args->argv[0];
                // Just an example, not intended to be free of memory leaks.
                  isy_TCHAR *username = (isy_TCHAR *) calloc(18, sizeof(isy_TCHAR));
                  isy_TCHAR *password = (isy_TCHAR *) calloc(254, sizeof(isy_TCHAR));
       char c;
                  int  i;

                printf("Query on target data(%s): %s ...¥n", target, statusMsg);
                // Ask for the username
                printf("Username: ");
                for(i = 0; (c = getchar()) != '¥n'; i++) username[i] = c;
                username[i] = '¥0';
                if (i == 0) return ISCRTNCB_Default; // username not entered
                // Ask for the password
                printf("Password: ");
                for(i = 0; (c = getchar()) != '¥n'; i++) password[i] = c;
                password[i] = '¥0';
      args->argv[1] = username;
                  args->argv[2] = password;
               return ISCRTNCB_Done;
            }
            return ISCRTNCB_Default;

          // all other event types, don't care
          default:
            return ISCRTNCB_Default;
    } // switch (event->type)
}

// Sample SyncClient
main()
{
    isy_TCHAR user[]     = isy_T("user1");
    isy_TCHAR password[] = isy_T("password");
    HISCSERV  hServ;
    HISCCONF  hConf;
    HISCENG   hEngine;
    isy_INT32 rc;
```

```
    rc = iscConfigOpen(hServ, isy_T(".¥isyncPath"), &hConf;);
    rc = iscEngineOpen(hConf, &hEngine;);
    iscEngineSetListener(hEngine, syncListener, NULL);

    iscEngineSyncConfig(hEngine); // get the configuration first
    iscConfigEnableSubsSet(hConf, NULL); // enable all subscription sets
    rc = iscEngineSync(hEngine); // sync config + subscription sets

    if (rc == ISCRTN_Failed) {
        HISCCSR   hCursor;
        isy_TCHAR  id[ISCLEN_SubsSetID];
        isy_TCHAR  name[ISCLEN_SubsSetName];
        isy_INT32  enabled;

        iscConfigOpenCursor(hConf, &hCursor;);
        while (iscConfigGetNextSubsSet(hConf, hCursor, id, name)
               == ISCRTN_Succeeded) {
            enabled = iscConfigSubsSetIsEnable(hConf, id);
            if (enabled != ISCRTN_True) continue; // forget about those which have
                                                  // been disabled
            rc = iscConfigGetSubsSetStatus(hConf, id);
            if (rc != ISCRTN_Succeeded)
// Then, the application can have some code
// processing the failing subscription sets here.
// To disable the subscription set, call:

                iscConfigDisableSubsSet(hConf, id);
        }
        iscConfigCloseCursor(hConf, hCursor);
        rc = iscEngineSync(hEngine); // sync config + subscription sets
    }
    // close all handles
    iscEngineClose(hEngine);
    iscConfigClose(hConf);
    iscServiceClose(hServ);
} // main
```

関連したタスク:

- 18 ページの『C/C++ を使用した DB2 Everyplace 同期クライアント・アプリケーションの開発』

関連した解説:

- 12 ページの『サポートされる C/C++ 開発ツール』

Java ネイティブ同期アプリケーションのサンプル

DB2 Everyplace の Java 同期アプリケーションを作成するうえで役立ついくつかのサンプル Java プログラムがあります。

これらサンプルが入っている場所については、「*DB2 Everyplace インストールおよびユーザーズ・ガイド*」の『サンプル・アプリケーションの概要』というセクションを参照してください。

サンプル・プログラム ISyncSample.java では、DB2 Everyplace ネイティブ同期プロバイダー用の同期クライアント・アプリケーションのコーディング方法を示します。

ISyncSample.java サンプル・アプリケーションの主なステップは、次のとおりです。

ステップ 1:

DB2 Everyplace 同期パッケージをインポートする。

```
import com.ibm.mobileservices.isync.*;
```

```
import com.ibm.mobileservices.isync.event.*;
```

JNI ベース同期プロバイダーの場合は、
com.ibm.mobileservices.isync.db2e.jni.*; をインポートする

トラップ・ベース同期プロバイダーの場合は、
com.ibm.mobileservices.isync.db2e.sti.*; をインポートする

ステップ 2:
同期時のイベント通知用に ISyncListener インターフェースの eventIssued メソッドをインプリメントする。

ステップ 3
インスタンス DB2eISyncProvider を取得する。

ステップ 4
プロバイダー・オブジェクトから同期サービスのインスタンスを取得する。

ステップ 5
サービス・オブジェクトから構成保管のインスタンスを取得する。

ステップ 6
構成保管オブジェクトから同期ドライバーのインスタンスを取得する。

ステップ 7
同期時における同期ドライバー・オブジェクトからのイベント通知用の ISyncListener インターフェースをインプリメントするアプリケーション listener オブジェクトの登録を行う。

ステップ 8
使用可能なすべてのサブスクリプション・セットで同期化を実行する。同期の状況の戻りコードと例外を調べる。

ステップ 9
同期プロバイダーによって割り振られたすべてのリソースをクローズして解放する。

```
// 例 1:  ISync Java - 単純な API の使用法
//

 // ステップ 1:  同期クライアント Java パッケージのインポート
      //
      import com.ibm.mobileservices.isync.*;
      import com.ibm.mobileservices.isync.event.*;
      import com.ibm.mobileservices.isync.db2e.jni.*;


      // ステップ 2:  ISyncListener インターフェースの eventIssued() メソッドのインプリメント
      //    (オプショナル : イベント通知に興味がある場合)
      //
      public class ISyncSample implements ISyncListener {

         public ISyncSample () {}

         public int eventIssued(ISyncEvent evt) {

            int evtType = evt.getEventType();

            switch(evtType) {

            // display event status
            case ISync.EVTTYPE_INFO:
            case ISync.EVTTYPE_ERROR:

               System.out.println ("**********************");
               System.out.println ("SubsSet:    " + evt.getSubscriptionSetName() );
               System.out.println ("Subs:       " + evt.getSubscriptionName() );
               System.out.println ("SubsType:   " + evt.getSubscriptionType() );
```

```
            System.out.println ("Event Type: " + evtType );
            System.out.println ("Event Code: " + evt.getEventCode() );
            System.out.println ("Progress:   " + evt.getSyncProgress());
            System.out.println ("**********************¥n");

            return ISync.RTNCB_DONE;

         case ISync.EVTTYPE_RETRY:
            return ISync.RTNCB_REPLY_YES;

         case ISync.EVTTYPE_CONFLICT:
            return ISync.RTNCB_DONE;

         // ignore other event types
         default:
            break;
      }

      // let sync engine take default action
      return ISync.RTNCB_DEFAULT ;

   }

   public void runSample(String host, String port,
                         String userID, String passwrd) {

      ISyncProvider provider = null;
      ISyncService  service  = null;
      ISyncConfigStore config = null;
      ISyncDriver syncer = null;
      String      path = "data";         // a data directory under current dir
      ISyncSubscriptionSet ssArr[] = null;
      int rc = 0;

      try {

         // ステップ 3: インスタンス DB2eISyncProvider の取得
         //
         provider = DB2eISyncProvider.getInstance();

      // ステップ 4: プロバイダーからの同期サービスのインスタンスの取得
         //
/*
 For the DB2j sync client, the JDBC driver and url are required

 String driver = "com.ibm.db2j.jdbc.DB2jDriver";
 String jdbcUrl = jdbc:db2j:crtlDb;create=true;
*/

if (driver != null)
    userProps.put("target.db.driver", driver);
if (jdbcUrl != null)
    userProps.put("target.db.url", jdbcUrl);

            Properties userProps = new Properties();

    userProps.put("isync.user", user);
    userProps.put("isync.password", password);
    userProps.put("isync.trace", "detailed");

    service = provider.createSyncService(uri, userProps);

         // ステップ 5: 構成ストアのインスタンスの取得
            //
            config = service.getConfigStore(path);

            // ステップ 6: 同期を実行するための同期ドライバー
            //                                  のインスタンスの取得
           syncer = config.getSyncDriver();

            // ステップ 7: 同期時における同期ドライバー・オブジェクトからの
                  イベント通知のリスナー・オブジェクト
            //          の設定 (オプショナル)
            syncer.setSyncListener(this);

   // ステップ 8: 使用可能になったすべてのサブスクリプション・セットでの同期の実行
            //
            rc = syncer.sync();

            switch (rc) {
               case ISync.RTN_SUCCEEDED:
                  System.out.println("Synchronization succeeded");
                  break;
```

```
            case ISync.RTN_CANCELED:
               System.out.println ("Synchronization canceled");
               break;

            default:
               System.out.println ("Synchronization failed");
               break;
         }

         ssArr = config.getSubscriptionSets();

         for (int i=0; i < ssArr.length; i++ ) {

            System.out.print ("Subscription Set: " +
                              ssArr[i].getName() + " Status: ");

            switch(ssArr[i].getStatus()) {

               case ISync.STATUS_READY:
                  System.out.println("READY");
                  break;

               case ISync.STATUS_COMPLETED:
                  System.out.println ("COMPLETED");
                  break;

               case ISync.STATUS_CANCELED:
                  System.out.println ("CANCELED");
                  break;

               default:
                  System.out.println ("FAILED");
                  break;
            }
         }
      }
      catch (ISyncException ie) {
         System.out.println("Exception code: " + ie.getCode());
         ie.printStackTrace();
      }
      catch (Exception e) {
         e.printStackTrace();
      }
      finally {
         // ステップ 9: 割り振られたすべてのリソースのクローズと解放
         //

         try {

            if (syncer != null) {
               syncer.close();
               syncer = null;
            }

            if (config != null) {
               config.close();
               config = null;
            }

            if (service != null) {
               service.close();
               service = null;
            }
         }
         catch(ISyncException ie2) {

            System.out.println("Exception code: " + ie2.getCode());
            ie2.printStackTrace();
         }
      }
   }  // end runSample()


   public static void main(String args[]) {

      String host     = "localhost";
      String port     = "8080";
      String userID   = "nurse1";
      String passwrd  = "nurse1";

      ISyncSample isa = new ISyncSample();

      if (args.length > 0) {
      if (args.length == 4)
```

```
        {
          host     = args[0];
          port     = args[1];
          userID   = args[2];
          passwrd  = args[3];
        }
else
          System.out.println("Usage: java ISyncSample [host] [port] " +
                             "[userid] [password]");
    }
       isa.runSample(host, port, userID, passwrd);

    }  // end main()

 }  // end ISyncSample class
```

関連したタスク:

- 28 ページの『トラップ・ベースのネイティブ同期プロバイダーのインストールおよび検証』

関連した概念:

- 『Java MIDP 同期アプリケーションのサンプル』

関連した解説:

- 23 ページの『IBM Java 同期 API』

Java MIDP 同期アプリケーションのサンプル

DB2 Everyplace の Java 同期アプリケーションを作成するうえで役立つ、いくつかの Java サンプル・アプリケーションが用意されています。MIDP 同期プロバイダーの場合、サンプルは次の場所にあります。

```
%DSYINSTDIR%/Clients/Midp/samples
```

基本サンプルは、`com/ibm/mobileservices/demo` にある Visiting Nurse アプリケーション、`VNurse.java` および `NursesAid.jar` です。同じサンプル・ディレクトリーに、単純なアプリケーションを構成する 2 つのファイルがあります。このアプリケーションでは、Record Store Management (RMS) コードや固有のユーザー・インターフェースは用意されていません。ファイルは、次のとおりです。

- ISyncSample.java : MIDlet を駆動する
- ISyncWorker.java : データの同期化を行う worker

ISyncWorker.java ファイルの詳細:

サンプル・プログラム `SyncWorker.java` では、DB2 Everyplace MIDP 同期プロバイダー用の同期クライアント・アプリケーションのコーディング方法を示しています。

Java サンプル・アプリケーションは、次のステップを実行します。

1. DB2 Everyplace 同期パッケージをインポートする。

   ```
   import com.ibm.mobileservices.isync.*;
   import com.ibm.mobileservices.isync.event.*;
   import com.ibm.mobileservices.isync.midp.*;
   ```

2. 同期時のイベント通知用に ISyncListener インターフェースの eventIssued メソッドをインプリメントする。
3. インスタンス MIDPISyncProvider を取得する。
4. プロバイダー・オブジェクトから同期サービスのインスタンスを取得する。

5. サービス・オブジェクトから構成保管のインスタンスを取得する。
6. 構成保管オブジェクトから同期ドライバーのインスタンスを取得する。
7. 同期時における同期ドライバー・オブジェクトからのイベント通知用の ISyncListener インターフェースをインプリメントするアプリケーション listener オブジェクトの登録を行う。
8. 使用可能なすべてのサブスクリプション・セットで同期化を実行する。同期の状況の戻りコードと例外を調べる。
9. 同期プロバイダーによって割り振られたすべてのリソースをクローズして解放する。

ISyncSample.java の例:

次の例には、前のセクションのステップを参照するコメントが記載されています。

```
// 例 1:  ISync Java - 単純な API の使用法
//

 // ステップ 1:  同期クライアント Java パッケージのインポート
      //
      import com.ibm.mobileservices.isync.*;
      import com.ibm.mobileservices.isync.event.*;
      import com.ibm.mobileservices.isync.midp.*;



/**
 Supporting class which handles all of the synchronization tasks.
 Called by ISyncSample.
*/

public class SyncWorker extends Thread implements ISyncListener
{
 private ISyncSample midlet;
 private boolean mCancel;
 private ISyncProvider provider;
 private ISyncService service;
 private ISyncConfigStore config;
 private ISyncDriver syncer;
 private String eventString;

 public SyncWorker(ISyncSample midlet)
 {
  this.midlet = midlet;
  mCancel = false;
 }

   // ステップ 2:  ISyncListener インターフェースの eventIssued() メソッドのインプリメント
   //          (オプショナル：イベント通知に興味がある場合)
   //

 public int eventIssued(ISyncEvent evt)
 {
  int evtType = evt.getEventType();
  int evtCode = evt.getEventCode();
  int evtProg = evt.getSyncProgress();
         String ssName = evt.getSubscriptionSetName();
  Object listenerInfo = evt.getEventInfo();

  Exception e = null;
  ConflictReader cr = null;

  if (listenerInfo instanceof Exception)
   e = (Exception) listenerInfo;
  else if (listenerInfo instanceof ConflictReader)
   cr = (ConflictReader) listenerInfo;

  eventString += evtCode + ":";

  switch(evtType)
  {
   // display event status
    case ISync.EVTTYPE_INFO:

              switch (evtCode)
      {
```

```
                case ISync.EVT_INF_SYNCING_SUBS:
                    midlet.updateSyncStat1("Synchronizing " + ssName);
                    midlet.updateSyncStat2(" ");
                    break;
                case ISync.EVT_INF_SYNC_STARTED:
                    midlet.updateSyncStat1("Synchronization started");
                    midlet.updateSyncStat2(" ");
                    break;
                case ISync.EVT_INF_PREP_MSG:
                    midlet.updateSyncStat2("Preparing message...");
                    break;
                case ISync.EVT_INF_SEND_MSG:
                    midlet.updateSyncStat2("Sending message...");
                    break;
                case ISync.EVT_INF_WAIT_MSG:
                   midlet.updateSyncStat2("Awaiting server reply...");
                   break;
                case ISync.EVT_INF_APPLY_MSG:
                    midlet.updateSyncStat2("Applying server message...");
                    break;
                case ISync.EVT_INF_SYNC_CANCELED:
            midlet.updateSyncStat1("Synchronization canceled");
            midlet.updateSyncStat2(" ");
            break;
             case ISync.EVT_INF_SYNC_SUCCEEDED:
            midlet.updateSyncStat1("Synchronization succeeded");
            midlet.updateSyncStat2(" ");
            break;
                    case ISync.EVT_INF_SYNC_FAILED:
            midlet.updateSyncStat1("Synchronization failed");
            midlet.updateSyncStat2(" ");
            break;
                    default:
                        break;
                }

  return ISync.RTNCB_DONE;

  case ISync.EVTTYPE_ERROR:
   midlet.updateSyncStat2("Error: " + evtCode);
   return ISync.RTNCB_DONE;

  case ISync.EVTTYPE_RETRY:
   midlet.updateSyncStat2("Retry: " + evtCode);
   return ISync.RTNCB_REPLY_YES;

  case ISync.EVTTYPE_CONFLICT:
  if (evtCode == ISync.EVT_CFT_REJECT)
  {
   String tabName = evt.getSubscriptionName();
                    midlet.updateSyncStat2("Conflict: " + tabName);

   /*
    Application needs to do the right thing with conflictRow.
   */

   // System.out.println("Conflict table " + tabName
    // + " row: " + conflictRow);
  }
  return ISync.RTNCB_DONE;

  // ignore other event types
  default:
   break;
  }

  // let sync engine take default action
 return ISync.RTNCB_DEFAULT ;

}  // end of eventIssued()
/*
 Synchronization is implemented in a thread to allow the
 user to cancel the request which single-threaded, might
 be hung on in IO request
*/

public void run()
{
 sync();
}

public void cancel()
{
 try
```

```
 {
  if (syncer != null)
   syncer.cancelSync();
 }
 catch (ISyncException iex)
 {}

 mCancel = true;
}

private void sync()
{
 try
 {
  eventString = " ";

  String user = "nurse1";
  String password = "nurse1";
  String host = "localhost";
  String port = "9080";

  /*
   If jad file has values, use them, see
   DeployManifest.java in tools.

   In the Sun WirelessToolkit, under Settings, you can enter
   values in the User Defined tab.
  */

  String x = midlet.getAppProperty("Db2eSyncUserName");
   if (x != null)
   user = x;
  x = midlet.getAppProperty("Db2eSyncPassword");
  if (x != null)
   password = x;
  x = midlet.getAppProperty("Db2eSyncHost");
  if (x != null)
   host = x;
  x = midlet.getAppProperty("Db2eSyncPort");
  if (x != null)
   port = x;
  midlet.appendForm(host + ":" + port + " " + user + "/" +
          password);


        // ステップ 3: インスタンス MIDPISyncProvider の取得
        //
  provider = MIDPISyncProvider.getInstance();

        // ステップ 4: プロバイダーからの同期サービスのインスタンスの取得
        //
   Hashtable ht = new Hashtable();
   ht.put("isync.user", userName);
   ht.put("isync.password", password);
   ht.put("isync.trace", "detailed");

   service = provider.createSyncService(URI, ht);

        // ステップ 5: 構成ストアのインスタンスの取得
        //
  config = service.getConfigStore(null);

        // ステップ 6: 同期を実行するための同期ドライバーのインスタンスの取得
        //
  syncer = config.getSyncDriver();

        // ステップ 7: 同期時における同期ドライバー・オブジェクトからの
        //     イベント通知用のリスナー・オブジェクトの設定
        //
  syncer.setSyncListener(this);

  // ステップ 8: 使用可能になったすべてのサブスクリプション・セットでの同期の実行
          //

  int rc = syncer.sync();

  switch (rc)
  {
    case ISync.RTN_SUCCEEDED:
   midlet.reportSyncStatus("Synchronization succeeded "
    + eventString);
   break;

    case ISync.RTN_CANCELED:
```

```
    midlet.reportSyncStatus("Synchronization canceled "
      + eventString);
    break;

     default:
    midlet.reportSyncStatus("Synchronization failed "
      + eventString);
    break;
   }
  //  ステップ 9: すべてのリソースのクローズ
  //

   close();
  }
  catch (ISyncException iex)
  {
   midlet.reportSyncStatus("Exception Code: "
    + iex.getCode() + ", Event codes: " + eventString);
  }
  catch (Exception e)
  {
   midlet.reportSyncStatus(e.toString());
  }
  finally
  {
   mCancel = false;
  }
 }

 private void close() throws ISyncException
 {
  if (syncer != null)
  {
   syncer.close();
   syncer = null;
  }
  if (config != null)
  {
   config.close();
   config = null;
  }

  if (service != null)
  {
   service.close();
   service = null;
  }

  provider = null;
 }
}
```

関連したタスク:

- 『Sun Wireless Toolkit を使用した MIDP アプリケーション用の isync4j の開発』
- 141 ページの『ANT および Sun Wireless Toolkit コマンド行を使用した MIDP アプリケーション用の isync4j の開発』

関連した概念:

- 135 ページの『Java MIDP 同期アプリケーションのサンプル』

Sun Wireless Toolkit を使用した MIDP アプリケーション用の isync4j の開発

このトピックは、Sun Wireless Toolkit アプリケーションで MIDP 用の DB2 Everyplace ISYNC4J を開発する方法について説明します。このセクションで使用する例は、VNurse サンプル・アプリケーションに基づいています。

前提条件:

DB2 Everyplace J2ME MIDP 同期プロバイダーを使用した場合のハードウェアおよびソフトウェアの前提条件についての詳細は、23 ページの『DB2 Everyplace 同期プロバイダーの概要』を参照してください。

手順:

1. Wireless Toolkit を立ち上げます。コマンド行プロンプトで、Sun Wireless Toolkit がインストールされている bin ディレクトリーに移動します。**ktoolbar.bat** と入力します。

   注: Windows の「**スタート**」メニューではなく、コマンド・プロンプトを使用することをお勧めします。

2. isync4j サンプル・アプリケーション用の新規プロジェクトを作成します。
   a. J2ME Wireless Toolkit をオープンします。
   b. 「**新規プロジェクト**」を選択します。
   c. プロジェクト名を入力します (例えば、VNurse など)
   d. MIDlet クラス名を入力します (例えば、com.ibm.mobileservices.demo.VNurse など)
   e. 「**プロジェクトの作成**」をクリックします。
   f. ISyncMidp.jar ファイルを J2ME プロジェクト・ライブラリーにコピーします。例えば、次のようにします。

      ```
      c:¥>copy %DSYINSTDIR%¥Clients¥Midp¥lib¥ISyncMidp.jar ¥
        j2me_install_dir¥apps¥VNurse¥lib.
      ```

   g. オプショナル: MIDlet の実行中にトレース出力を表示したい場合は、ISyncMidpDebug.jar を *j2me_install_dir* ¥apps ¥VNurse ¥lib にコピーします。

      注: 電話にインストールする JAR ファイルのビルド時にトレースを使用しないでください。ビルドされた JAR ファイルが大きすぎてインストールできなくなります。

   h. オプショナル: 難読化を使用する場合は (コード・サイズを削減するため)、retroguard.jar ファイルを J2ME がインストールされている bin ディレクトリーにコピーします。
   i. 「**設定 (Settings)**」をクリックします。「プロジェクトの設定 (Settings for project)」ウィンドウがオープンします。「**ユーザー定義 (User Defined)**」タブをクリックし、「**追加**」をクリックして、以下の「**Key**」と「**Value**」項目を入力します。
      - Db2eSyncPassword、nurse1 [デフォルト]
      - Db2eSyncUserName、nurse1 [デフォルト]
      - PacketDownSize、2800 [デフォルト 30000]
      - PacketUpSize、1400 [デフォルト 30000]
      - Db2eSyncHost、localhost [デフォルト]
      - Db2eSyncPort、9080 [デフォルト]

      これらの値は、Sun Wireless Toolkit によって .jad ファイルに置かれ、MIDlet はその値を実行時に読み取ります。

j. DB2 Everyplace サンプル・アプリケーション (VNurse) が PNG イメージ・ファイルを表示します。「**MIDlet**」タブをクリックして、「**MIDlet-1**」を選択します。「**編集**」をクリックして、 VNurse.png を ibm.png に変更します。ibm.png を Midp¥samples¥images ディレクトリーから *J2ME_ install_ dir*¥apps¥VNurse¥res ディレクトリーにコピーする必要があります。

3. DB2 Everyplace サンプル Java ファイルをプロジェクトにインポートします。例えば、ディレクトリー構造を %DSYINSTDIR%¥Clients ¥Midp ¥samples ¥com から *J2ME_ install_ dir* ¥apps ¥VNurse ¥src ディレクトリーにコピーします。
4. VNurse サンプル・アプリケーションをビルドして実行します。Sun Wireless Toolkit のウィンドウで、「**ビルド**」をクリックして「**実行**」をクリックします。

関連したタスク:

- 『ANT および Sun Wireless Toolkit コマンド行を使用した MIDP アプリケーション用の isync4j の開発』

関連した概念:

- 23 ページの『DB2 Everyplace 同期プロバイダーの概要』

ANT および Sun Wireless Toolkit コマンド行を使用した MIDP アプリケーション用の isync4j の開発

このトピックは、ANT および Sun Wireless Toolkit コマンド行による MIDP 用 DB2 Everyplace ISYNC4J の開発方法について説明します。

前提条件:

提供されている例と共に動作させるために、以下のソフトウェアをダウンロードしてインストールしてください。

- Sun Microsystems JavaTM 2 Platform Micro Edition、Wireless Toolkit
- Apache ANT
- RetroGuard Ofuscator

手順:

1. オプショナル: 変更したい場合はデモを再コンパイルします。

   lib ディレクトリーには、プリコンパイルされた JAD ファイルおよび JAR ファイルが含まれています。Apache ANT、DeployManifest ツール、および RetroGuard Obfuscator の使用について説明するために、build.bat スクリプトおよび build.xml スクリプトが用意されています。

   a. retroInstallDir¥lib¥retroguard.jar を CLASSPATH 変数に追加します。ご使用の環境で、以下の変数を設定します。
      - ANT_HOME - ANT インストールのルートに設定
      - DB2m_HOME - %DSYINSTDIR%¥Clients¥Midp ディレクトリーに設定
      - J2MEWTK_HOME - Sun Wireless Toolkit インストールのルートに設定
      - JAVA_HOME - jdk13 または jdk131 (のみ) インストールのルートに設定
      - JAVA14_HOME - jdk14 ディレクトリーのルートに設定

b. MIDP クライアント・ディレクトリーのルートにある build.bat ファイルを実行して、新規 JAR ファイルおよび新規 JAD ファイルを MIDP の下の lib ディレクトリーに再移植します。それぞれのユーザーおよびデバイス ID 構成ごとに、 1 つの JAR ファイルと複数の JAD ファイルがあります。

事前検査および難読化には、複数の新規 build¥*Classes ディレクトリーが使用されます。それぞれのユーザーおよびデバイス ID 構成ごとに、 1 つの JAR ファイルと複数の JAD ファイルがあります。 JAD ファイルを表示して、ユーザー ID、パスワード、およびデバイス ID がどのように設定されているか、および MIDLet アプリケーションに渡される方法を調べます。

c. DeployManifest クラスは、lib¥FilterServlet*.jar に組み込まれており、build.xml ファイルから呼び出されます。このクラスを使用して、 JAR マニフェスト・ファイルと JAD ファイルの両方を生成します。ファイルを生成する際には、次の構文を使用します。

マニフェスト・ファイルを生成するには、次のようにします。

```
java DeployManifest -m <midletName> <className> <imageFileName> <outputfileName>
```

JAD ファイルを生成するには、次のようにします。

```
java DeployManifest -j <midletJarName> -U <uploadMaxPacket> -D ¥
  <downloadMaxPacket> -n <numClients> <JadBaseName> <outputFileName>
```

d. DeployManifest クラスは、Apache ANT により build.xml から内部的に呼び出されます。build.xml ファイルの setJad エントリーを編集して、ユーザー ID、パスワード、またはその他の属性を永続的に変更します。デフォルトは *nurse1* および *nurse1* です。

2. insync4j アプリケーションを実行します。

DB2 Everyplace のインストールでは、Vnurse データベースを、サブスクリプション・セット、ユーザー、およびグループと共に作成します。

a. 「**スタート**」->「**DB2 Everyplace**」->「**MDAC 開始 (Start MDAC)**」を選択して、 *nurse1* というユーザーが存在することを確認します。このユーザーのパスワードは、 *nurse1* に設定されています。このユーザー名を使用するか、または実行スクリプトに渡す lib¥<midlet>.jad ファイルを編集することができます。変更をコンパイルするたびに、JAD ファイルが上書きされる点に注意してください。ユーザーおよびパスワードを永続的に変更する方法は、samples¥DeployManifest.java を参照してください。

b. Tomcat または Websphere バージョン 4.0 以降のいずれかを使用して、同期サーバーを開始する必要があります。MIDP 電話からの HTTP 接続は HTTP 転送エンコードを使用します。これには、HTTP Servlet 2.3 仕様および HTTP 1.1 をサポートするサーブレット・エンジンが必要です。

c. Midp¥lib ディレクトリーにある JAD ファイルの名前を指定して、 Midp¥bin ディレクトリー内の BATCH ファイルを実行します。

- デモの未デバッグ・バージョンを実行するには、次のように入力します。

  ```
  run VNurse
  ```

- デバイス # 213 のユーザー ID とパスワードに「nurse3」を使用してデバッグ・バージョンを実行するには、次のように入力します。

  ```
  run VNurseDebug3
  ```

J2ME MIDP ISync Client は、com.ibm.mobileservices.isync.midp および com.ibm.mobileservices.isync と com.ibm.mobileservices.isync.event パッケージに定義されているインターフェースおよびクラスを使用します。

関連したタスク:

- 139 ページの『Sun Wireless Toolkit を使用した MIDP アプリケーション用の isync4j の開発』

関連した解説:

- 23 ページの『IBM Java 同期 API』
- 23 ページの『Java 同期 API がサポートされているオペレーティング・システム』

GoISyncConsole サンプル Java 同期アプリケーションのコンパイルおよび実行

GoISyncConsole は、DB2 Everyplace 同期クライアント Java API の使用法を実演するサンプル・アプリケーションです。

GoISyncConsole のファイル内容:

- `GoISyncConsole.java`
- `GoISyncConstants.java`
- `GoISyncListener.java`
- `isyncdb2.properties`
- `isyncdb2e.properties`
- `isyncdb2j.properties`

前提条件:

- DB2 Everyplace 同期サーバーのインストールおよび構成。
- 同期クライアント・バイナリーのデバイスへのインストール。これらのバイナリーは `Clients¥`*`platform`*`¥sync` ディレクトリーにあります。
- Cloudscape クライアントを使用している場合、デバイスに Cloudscape をインストールします。

手順:

1. GoISyncConsole アプリケーションをコンパイルします。

   これには、同期クライアント・バイナリーの 1 つである `isync4j.jar` ファイルが必要です。

   a. コマンド・プロンプトを開きます。

   b. 以下のコマンドを入力します。

   ```
   javac -classpath isync4j.jar *.java
   ```

2. 環境をセットアップします。

   パス環境は、同期クライアント・バイナリーを配置できるようにセットアップされていなければなりません。

   - Win32 の場合: PATH 変数を設定して、同期クライアント・バイナリーが配置されるフォルダーを組み込みます。

- Linux または Neutrino の場合: LD_LIBRARY_PATH をエクスポートして、同期クライアント・バイナリーが配置されるフォルダーを組み込みます。

3. サンプルを実行します。

   GoISyncConsole は、C クライアントまたは Java DB2j クライアントとのいずれかと共に使用できます。プロパティー・ファイルは、使用するクライアントを判別するのに使用します。DB2e と DB2j の両方のサンプル・プロパティー・ファイルが提供されています。

   - C クライアントを使用するには、以下のコマンドを入力して `isyncdb2e.properties` ファイルへ渡します。

     ```
     java -classpath isync4j.jar;. GoISyncConsole isyncdb2e.properties
     ```

   - Java DB2j クライアントを使用するには、以下のコマンドを入力して DB2j 同期クライアント jar および Cloudscape jar ファイルを組み込み、`isyncdb2j.properties` ファイルに渡します (Cloudscape のインストール・ディレクトリーが異なっている場合 italics 内のテキストを変更します)。

```
java -classpath c:¥cloudscape_5.1¥lib¥db2j.jar;
db2jisync.jar GoISyncConsole isyncdb2j.properties
```

   アプリケーションが、以下のオプションを含んだテキスト・メニューを表示して開始します。

   - (1) 同期の実行
   - (2) サブスクリプション・セットの使用可能、使用不可、またはリセット
   - (3) サーバー設定の変更
   - (4) ログのビュー
   - (5) 同期クライアントについて
   - (6) 終了

4. オプション (3) を指定してサーバー設定を構成します。このオプションにより、同期サーバーの IP アドレス、同期ユーザー名とパスワード、および他のオプションを指定できます。
5. オプション (1) を指定して同期を実行します。

   GoISyncConsole アプリケーションは、`ISync.properties` という別のプロパティー・ファイルを作成してユーザーの設定を保存します。`isyncdb2e.properties` または `isyncdb2j.properties` 内のプロパティーを変更するには、GoISyncConsole を再実行する前に `ISync.properties` を削除して新しい変更が確実に有効になるようにします。

関連した概念:

- 23 ページの『DB2 Everyplace 同期プロバイダーの概要』

第 4 部 解説

第 18 章 アプリケーション・プログラミング・インターフェース (API)

DB2 Everyplace SQL ステートメント・サポート

この章では、DB2 Everyplace によってサポートされる SQL ステートメントの構文図、意味の説明、規則、および使用例を記載しています。説明されているトピックは、以下のとおりです。

- 『DB2 Everyplace SQL ステートメント・サポートの概要』
- 185 ページの『割り当てと比較に関するデータ・タイプの互換性』
- 84 ページの『DB2 Everyplace でサポートされているパラメーター・マーカー』
- 189 ページの『SQLState のリスト』
- 189 ページの『SQLState クラス・コードのサマリー』
- 190 ページの『SQL が報告する SQLState メッセージ』
- 193 ページの『CLI が報告する SQLState メッセージ』

DB2 Everyplace SQL ステートメント・サポートの概要

サポートされている実行可能な SQL ステートメントは、コマンド行プロセッサー (CLP) を使用してモバイル・デバイスから対話式に実行するか、またはアプリケーション・プログラム内で使用して DB2 Everyplace データベース内のデータにアクセスすることができます。表 15 は、DB2 Everyplace によってサポートされる SQL ステートメントをリストアップしたものです。

表 15. サポートされている SQL ステートメント

SQL ステートメント	機能
CALL	DB2 Everyplace 同期サーバーのリモート照会およびストアード・プロシージャー・アダプター (AgentAdapter) を使用して、リモート・ストアード・プロシージャーを呼び出します。
CREATE INDEX	索引を作成します。
CREATE TABLE	表を定義します。
DELETE	表から 1 行以上を削除します。
DROP	データベースから表または索引を削除します。
EXPLAIN	SELECT ステートメントのアクセス・パス選択に関する情報を取得します。
GRANT	ユーザーに暗号化特権を付与します。
INSERT	表に 1 行以上を挿入します。
REORG TABLE	指定された表に関連する無駄なストレージを除去または削減します。
REVOKE	ユーザーの暗号化特権を取り消します。
SELECT	1 つ 以上の表から照会された結果表を指定します。
UPDATE	表の 1 行以上の中の 1 列以上の値を更新します。

190 ページの『SQL が報告する SQLState メッセージ』は、DB2 Everyplace SQL エンジンによって報告されるすべての SQLSTATE のリストです。

SQL ステートメントの長さは 64,000 文字を超えてはなりません。

カタログには、 DB2 Everyplace によって管理される DB2 Everyplace システム・テーブル DB2eSYSTABLES、DB2eSYSRELS、および DB2eSYSCOLUMNS が含まれています。

関連した解説:

- 185 ページの『割り当てと比較に関するデータ・タイプの互換性』
- 84 ページの『DB2 Everyplace でサポートされているパラメーター・マーカー』
- 189 ページの『SQLState のリスト』
- 189 ページの『SQLState クラス・コードのサマリー』

CALL

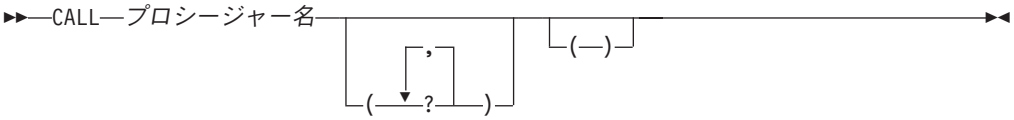
DB2 Everyplace 同期サーバーのリモート照会およびストアード・プロシージャー・アダプターで定義されたストアード・プロシージャーを呼び出します。ストアード・プロシージャーは、例えば、リモート・データベースのロケーションで実行を行い、DB2 Everyplace クライアント・アプリケーションにデータを戻します。

SQL CALL ステートメントを使用するプログラムは、2 つの部分 (クライアントとサーバー) で実行するように設計します。

呼び出し:

リモート・ストアード・プロシージャーを DB2 Everyplace アプリケーションから呼び出すには、`SQLPrepare()` に次の CALL ステートメント構文を渡し、その後に `SQLExecute()` を続けます。

構文:



説明:

プロシージャー名
リモート・サーバーで呼び出すプロシージャーを指定します。ここに指定するプロシージャーは、現行の同期サーバーで AgentAdapter サブスクリプションに定義しておく必要があります。

? CALL ステートメント構文図の *?* は、ストアード・プロシージャーの引き数に対応するパラメーター・マーカーを示します。引き数はすべて、パラメーター・マーカーを使用して渡さなければなりません。

規則:

なし

注意事項:

CALL ステートメントは、DB2 Everyplace 同期サーバーに組み込まれているリモート照会とストアード・プロシージャー・アダプターを使用します。 DB2 Everyplace 同期サーバーでは、DB2 Everyplace アプリケーションの中で CALL ステートメントを使用する必要があります。 DB2 Everyplace は、ローカルのストアード・プロシージャーをサポートしません。

追加情報については、「*DB2 Everyplace 同期サーバー 管理ガイド*」のデータ・ソースに関するセクションを参照してください。

例:

CALL ステートメントとリモート照会の使用法、およびストアード・プロシージャー・アダプターの完全な例は、「*DB2 Everyplace 同期サーバー 管理ガイド*」にあります。次の例は、サンプル・アプリケーション内の CALL ステートメントのコーディングのみを示したものです。

ストアード・プロシージャー MYPROC() は、ソース・サーバーでデータベース mysample に対して定義されています。 AgentAdapter サブスクリプションは、DB2 Everyplace 同期サーバーにおいて次の属性で定義されています。

```
User ID:  db2admin
Password: db2admin
Other:    dbname=mysample;procname= db2e.MYPROC
```

CALL ステートメントを使用するサンプル・プログラムは、次のとおりです。

```
int main(int argc, char * argv[])
{
  SQLHENV  henv;
  SQLHDBC  hdbc;
  SQLHSTMT hstmt;
  SQLRETURN rc;
  SQLCHAR  strSQL[] = "CALL db2e.MYPROC(?,?,?,?,?)";
  int   nInd4, nInd5;
  int   nSaving = 0, nChecking =0 ;
  int   nCmd =0, nAmount=0;
  SQLCHAR  strConnect[254];

 //*********************************************************************
 //* Check input parameters
 //*********************************************************************
 if ( argc < 4 ){
   printf("¥nUsage : myClient AccountName Cmd Amount");
   printf("¥n     cmd 1 : query balance");
   printf("¥n     cmd 2 : Transfer from Saving to Checking");
   printf("¥n     cmd 3 : Trnasfer from Checking to Saving");
   return (99);
 }
 nCmd = atoi(argv[2]);
 nAmount = atoi(argv[3]);

 //*********************************************************************
 //* Allocate handles
 //*********************************************************************
 rc = SQLAllocHandle( SQL_HANDLE_ENV,
      SQL_NULL_HANDLE,
      &henv; //checkerror
 rc = SQLAllocHandle( SQL_HANDLE_DBC,
      henv,
      &hdbc); //checkerror
 if (argc == 5){
 strcpy(strConnect,"http://");
 strcat(strConnect,argv[4]);
   strcat(strConnect,"/servlet/com.ibm.mobileservices.adapter.agent.AgentServlet?DB=mysample");
 }else{
  strcpy(strConnect,
             "http://127.0.0.1:8080/db2e/servlet/
```

```
            com.ibm.mobileservices.adapter.agent.AgentServlet?DB=mysample");
}

//*************************************************************
//* Connect to remote database
//*************************************************************
rc = SQLConnect(hdbc,
 strConnect,
 SQL_NTS,
 "userex", SQL_NTS,
 "userex", SQL_NTS  ); //checkerror
rc = SQLAllocHandle( SQL_HANDLE_STMT,
     hdbc,
     &hstmt); //checkerror
//*************************************************************
//* Prepare, Bind , and Execute the statement
//*************************************************************
rc = SQLPrepare(hstmt,strSQL, SQL_NTS); //checkerror
rc = SQLBindParameter(hstmt,
  1,
  SQL_PARAM_INPUT,
  SQL_C_CHAR,
  SQL_CHAR,
  0,
  0,
  (SQLPOINTER)argv[1],
  0,
  NULL ); //checkerror
rc = SQLBindParameter(hstmt,
  2,
  SQL_PARAM_INPUT,
  SQL_C_LONG,
  SQL_INTEGER,
  0,
  0,
  (SQLPOINTER)&nCmd,
  sizeof(int),
  NULL); //checkerror
rc = SQLBindParameter(hstmt,
  3,
  SQL_PARAM_INPUT,
  SQL_C_LONG,
  SQL_INTEGER,
  0,
  0,
  (SQLPOINTER)&nAmount,
  sizeof(int),
  NULL ); //checkerror
rc = SQLBindParameter(hstmt,
  4,
  SQL_PARAM_OUTPUT,
  SQL_C_LONG,
  SQL_INTEGER,
  0,
  0,
  (SQLPOINTER)&nSaving,
  sizeof(int),
  &nInd4 ); //checkerror
rc = SQLBindParameter(hstmt,
  5,
  SQL_PARAM_OUTPUT,
  SQL_C_LONG,
  SQL_INTEGER,
  0,
  0,
  (SQLPOINTER)&nChecking,
  sizeof(int),
  &nInd5 ); //checkerror
rc = SQLExecute(hstmt); //checkerror
//*************************************************************
//* Print the balance
//*************************************************************
printf("¥nSaving = %d",nSaving);
printf("¥nChecking = %d",nChecking);

SQLFreeHandle(SQL_HANDLE_STMT, hstmt);
SQLDisconnect(hdbc);
SQLFreeHandle(SQL_HANDLE_DBC, hdbc);
SQLFreeHandle(SQL_HANDLE_ENV, henv);
return 0;
```

関連した解説:

- 147 ページの『DB2 Everyplace SQL ステートメント・サポートの概要』
- 185 ページの『割り当てと比較に関するデータ・タイプの互換性』
- 84 ページの『DB2 Everyplace でサポートされているパラメーター・マーカー』
- 189 ページの『SQLState のリスト』
- 189 ページの『SQLState クラス・コードのサマリー』

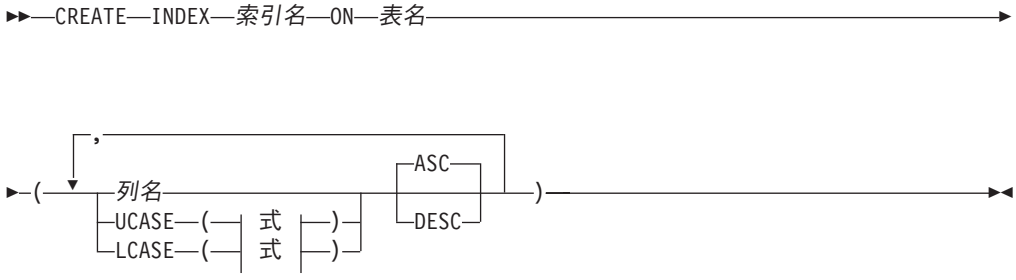
CREATE INDEX

CREATE INDEX ステートメントは、DB2 Everyplace 表に索引を作成するために使用します。

呼び出し:

このステートメントは、アプリケーション・プログラム内で DB2 CLI 関数を用いて使用することができます。また、CLP を介して出すこともできます。

構文:



説明:

INDEX *索引名*
索引に名前を付けます。

ON *表名*
表名 は、索引を作成する表の名前を指定します。

列名
索引の場合、列名は索引キーの一部となる列を示します。

各列名は、表の列を示す、修飾されていない名前でなければなりません。使用する列は 8 列以下としてください。列名を繰り返すことはできません (SQLSTATE 42711)。

指定された各列の長さは 1024 バイトを超えてはなりません。

ASC 索引項目を列の昇順で並べます。これはデフォルトです。

DESC 索引項目を列の降順で並べます。

LCASE / UCASE

LCASE 関数または LOWER 関数は、SBCS 文字がすべて小文字に変換されたストリングを戻します。つまり、文字 A から Z が文字 a から z に変換され、発音区別符号が付いた文字があれば、それらは同等の小文字に変換されます。

引き数は、値のデータ・タイプが CHAR か VARCHAR である式でなくてはなりません。

上記の関数の結果は、引き数と同じデータ・タイプと長さの属性を持つことになります。引き数がヌル値もとる場合は、結果がヌルになることがあります。つまり、引き数がヌルであれば、結果はヌル値になります。

EMPLOYEE 表の列 JOB の値に含まれる文字は小文字で戻されることを確認してください。例えば、次のようにします。

```
SELECT LCASE(JOB)
     FROM EMPLOYEE
     WHERE EMPNO = '000020';
```

規則:

- 主キーがない場合は、1 つの表に最高 15 個の索引を作成することができます。主キーがある場合は、1 つの表に最高 14 個の索引を作成することができます。
- 既存の索引と一致する索引を作成しようとすると、CREATE INDEX ステートメントは失敗します。以下の場合は、2 つの索引記述は重複していると見なされます。
  - 列セットと索引内のそれらの順序が、既存の索引の順序と同じである場合。
  - 順序付け属性が同じ場合。
- CREATE INDEX ステートメントでは、BLOB データ・タイプの列を使用することはできません。

注意事項:

- CREATE INDEX ステートメントには、最高 8 つの列を含めることができます。
- DB2 Everyplace は、索引の両方向スキャンをサポートします。次の 2 つの索引は定義が異なりますが、同じ機能を持ちます。

  ```
  CREATE INDEX IDX1 ON EMPLOYEE (JOB ASC)
  CREATE INDEX IDX1 ON EMPLOYEE (JOB DESC)
  ```

  通常は、索引は順序付けの方向を指定しないで作成してください。一般的に、索引の数が少ないほど索引の保守の手間が省けます。
- DB2 Everyplace は、索引の接頭部スキャンをサポートします。次の例について考えてみます。次の索引が作成されます。

  ```
  CREATE INDEX J1 ON T (A, B, C, D, E, F, G, K)
  ```

  この場合、T (A,B,C,D) に別の索引を作成する必要はありません。
- その表にデータが入っていない場合は、CREATE INDEX は索引の説明だけを作成し、その表にデータが挿入されたときに索引項目が作成されます。
- ダーティー・ビット索引用の索引を作成する場合は、以下の例を使用してください。

  ```
  CREATE INDEX <索引名>
    ON <表名>
    ($dirty)
  ```

  ダーティー・ビットの詳細については、287 ページを参照してください。

例:

EMPLOYEE 表に JOB_BY_DPT という名前の索引を作成します。索引項目を各部門 (WORKDEPT) 内でジョブ名称 (JOB) の昇順で並べます。

```
CREATE INDEX JOB_BY_DPT
ON EMPLOYEE (WORKDEPT, JOB)
```

関連した解説:

- 147 ページの『DB2 Everyplace SQL ステートメント・サポートの概要』
- 185 ページの『割り当てと比較に関するデータ・タイプの互換性』
- 84 ページの『DB2 Everyplace でサポートされているパラメーター・マーカー』
- 189 ページの『SQLState のリスト』
- 189 ページの『SQLState クラス・コードのサマリー』

CREATE TABLE

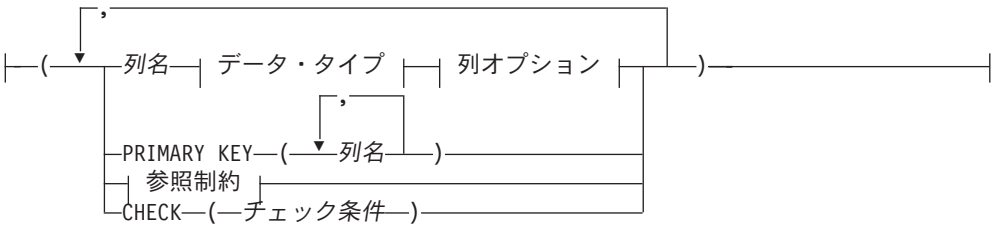
CREATE TABLE ステートメントにより表を定義します。定義の中には、表の名前およびその列の名前と属性を含めなければなりません。定義には、表の主キーなど、表に関するその他の属性も含めることができます。

呼び出し:

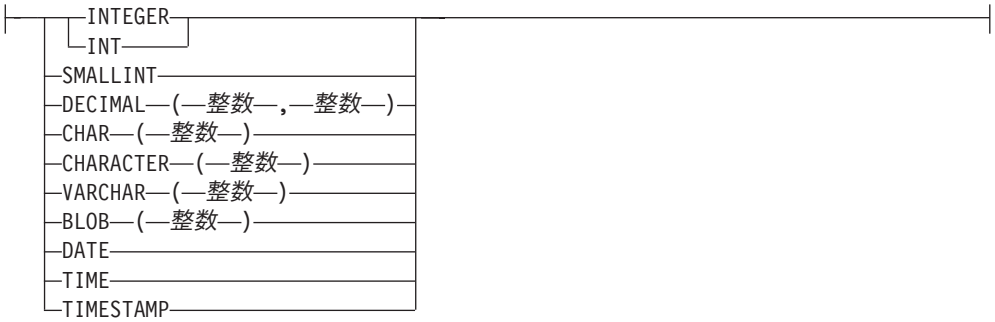
このステートメントは、アプリケーション・プログラム内で DB2 CLI 関数を用いて使用することができます。また、CLP を介して出すこともできます。

構文:

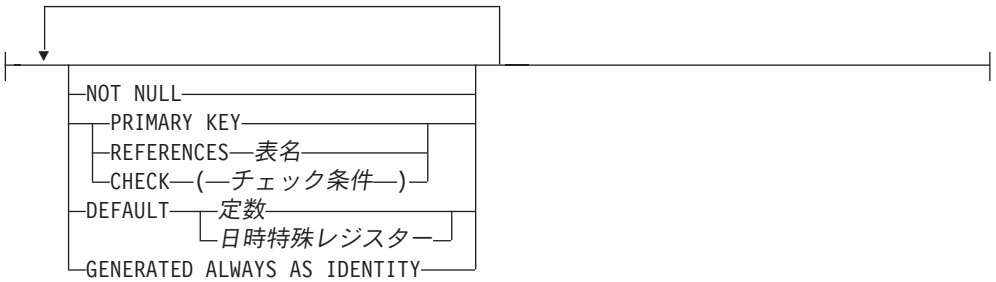
エレメント・リスト:



データ・タイプ:



列オプション:



参照制約:

説明:

表名

表の名前を指定します。この名前は 18 バイト以内でなければなりません。この名前は、カタログ内の表を示すものであってはなりません。名前はモバイル・デバイスとして固有なものでなければなりません。

表名は、カタログに保管される前に大文字に変換されます。区切り ID (二重引用符付き) を使用すると、このような変換が行われないようにすることができます。表名にブランクまたは特殊文字が含まれる場合は、区切り ID を使用する必要があります。

表名には、2 バイト文字セットの文字も使用できます。

制約事項: ユーザー名によって作成および命名された表に対応するシステム作成のデータ・ファイルでは、大文字と小文字は区別されません。例えば、TB と命名された表のデータ・ファイルは、DSY_TB と命名されます。「tb」と命名された表のデータ・ファイルもまた DSY_TB となります。したがって、データ保全性を維持するために、文字ケースを除き、既存の表名と同じ一連の文字を使用して表を命名しないようにすることを強くお勧めします。

WITH ENCRYPTION

暗号化されたユーザー表を作成します。表を暗号化するには、ユーザーが認証済み、かつ接続状態になければなりません。ユーザーは明示的に暗号化を認可されていなければなりません。 (詳細については、 166 ページの『GRANT』を参照してください。)

ユーザー表を暗号化できるのは、ユーザー表の作成時のみです。表が作成された後では、暗号化を追加したり、除去したり (表を削除する場合は除く) することはできません。

列名

表の列の名前を指定します。この名前は 18 バイト以内でなければなりません。この名前は修飾できません。また、表の複数の列に同じ名前を使ってはなりません。

列名は、カタログに保管される前に大文字に変換されます。区切り ID (二重引用符付き) を使用すると、このような変換が行われないようにすることができます。また、列名にブランクまたは特殊文字が含まれている場合も、区切り ID を使用する必要があります。

列名には、DBCS 文字も使用できます。

データ・タイプ
以下のリスト内のいずれかのタイプです。いずれかのタイプを指定してください。

INTEGER または **INT**
4 バイトの符号付き整数の場合。範囲は 2147483647 から -2147483648 までです。

SMALLINT
2 バイトの符号付き整数の場合。範囲は -32768 から 32767 までです。

DECIMAL*(精度整数, 位取り整数)*
10 進数の場合。最初の整数は数値の精度、すなわち、総桁数です。この範囲は 1 から 31 までです。 2 番目の整数は数値の位取り、すなわち、小数点の右側の桁数です。この範囲は 0 からその数値の精度までです。

CHAR*(整数)*
長さが整数 の固定長文字ストリングの場合。長さの範囲は 1 から 32767 までです。

CHARACTER*(整数)*
長さが整数 の固定長文字ストリングの場合。長さの範囲は 1 から 32767 までです。

VARCHAR*(整数)*
最大長が整数 の可変長文字ストリングの場合。最大長の範囲は 1 から 32767 までです。

BLOB*(整数)*
バイト単位で最大長を指定した 2 進ラージ・オブジェクト・ストリングの場合。

長さの範囲は 1 バイトから 32767 バイトまでです。

整数 は最大長です。

DATE
日付。入力値の形式は次のいずれかです。MM/DD/YYYY、YYYY-MM-DD、または DD.MM.YYYY。日付値の印刷は ISO 形式 YYYY-MM-DD のみです。

特殊レジスター CURRENT DATE を使用しても、ISO 形式の現在日付を生成できます。

TIME
時刻。入力値の形式は次のいずれかです。HH:MM AM (または PM)、HH:MM:SS、HH.MM AM (または PM)、あるいは HH.MM.SS。HH:MM:SS または HH.MM.SS 形式において、SS は秒を示すもので、指定はオプショナルです。時刻値の印刷は ISO 形式 HH:MM:SS のみです。

特殊レジスター CURRENT TIME を使用しても、ISO 形式の現在時刻を生成できます。

TIMESTAMP
タイム・スタンプ。入力値の形式は YYYY-MM-DD-HH.MM.SS.ZZZZZZ で

す。タイム・スタンプ値の印刷の形式は YYYY-MM-DD-HH.MM.SS.ZZZZZZ です。

特殊レジスター CURRENT TIMESTAMP を使用しても、現在のタイム・スタンプを生成できます。

列オプション
表の列に関する追加のオプションを定義します。

NOT NULL
列にヌル値を含めることができないようにします。

NOT NULL が指定されていない場合、列にヌル値を入れることができます。また、デフォルト値はヌル値か、DEFAULT 文節で提供された値になります。

PRIMARY KEY
これは、単一の列から構成される主キーをす早く定義するための方法を提供します。つまり、PRIMARY KEY を列 C の定義に指定すると、別個の文節として PRIMARY KEY(C) 文節を指定したのと同じことになります。

PRIMARY KEY の説明は、157ページにあります。

REFERENCES *表名*
REFERENCES の説明は 158ページにあります。

CHECK (*チェック条件***)**
CHECK の説明は 158ページにあります。

DEFAULT
INSERT ステートメントに値が指定されなかった場合に、デフォルト値を提供します。

列定義に DEFAULT が入っていないと、その列のデフォルトとしてヌル値が使用されることになります。そのような列が NOT NULL で定義されていると、その列には有効なデフォルトはないことになります。

定数
定数を列のデフォルト値として指定します。指定する定数は、以下のようなものでなければなりません。

- その列に割り当て可能な値である。
- この定数が 10 進定数である場合、列データ・タイプの位取りを超えて非ゼロの桁を指定していない。(例えば、1.234 は DECIMAL(5,2) の列のデフォルトにはならない)

日時特殊レジスター
INSERT の時にこの列のデフォルトとして、日時特殊レジスターの値 (CURRENT DATE、CURRENT TIME、または CURRENT TIMESTAMP) を指定します。この列のデータ・タイプは、指定する特殊レジスターに対応したデータ・タイプでなければなりません。(例えば、CURRENT DATE を指定する場合は、データ・タイプは DATE である必要があります。)

GENERATED ALWAYS AS IDENTITY
表を作成するときに、ユーザーは列を「GENERATED ALWAYS AS IDENTITY」と指定することができます。その後、この列の値は、ユーザー

が INSERT または副選択付きの INSERT を行うたびに、DB2 Everyplace によって生成されます。この列は数値タイプ (INTEGER、SMALLINT、または DECIMAL タイプ) である必要があり、DB2 Everyplace は、1 から始まり、毎回 1 ずつ増分される固有のシリアル番号を自動的に生成します。

IDENTITY 列に生成される値は 1 から始まり、表に行が挿入されるたびに 1 ずつ増えます。そのため、DB2 Everyplace が IDENTITY 列上に索引を自動的に作成しなくても、一意性は保証されます。IDENTITY 列上に索引が必要な場合は、明示的に索引を作成するか、またはその列を PRIMARY KEY として指定する必要があります。IDENTITY 列の値の範囲が限界 (最大値) に達した場合は、さらに INSERT ステートメントが使用されると、エラー (SQLSTATE 23522) となります。INT タイプと SMALLINT タイプの IDENTITY 列の最大値は、それら 2 つのタイプで許可されている最大値となります。DECIMAL タイプの IDENTITY 列の最大値は、(1) そのデータ・タイプの定義 (精度、位取り) および (2) IDENTITY 列に許可されている最大値 (2.15* (10∧18) (小数桁数 19)) によって決まります。(1) と (2) のいずれか小さい方が範囲の限界となります。DECIMAL タイプの IDENTITY 列の場合、値の小数部分は必ず 0 となり、整数部分は毎回 1 ずつ増分されます。

IDENTITY 指定は、データ・タイプが 3 つの数値タイプ (INT、SMAIINT、DECIMAL) のいずれかである列でのみ定義できます。それ以外の場合はエラーとなります (SQLSTATE 42815)。表ごとに最大 1 つの IDENTITY 列が存在できます (その以外の場合はエラー SQLSTATE 428C1)。ユーザーは INSERT ステートメントに IDENTITY 列の値を指定することはできません (デフォルトにして、DB2 Everyplace システムが生成した値を使用するようにしなければなりません)。またユーザーは、IDENTITY 列を UPDATE することもできません。

PRIMARY KEY (*列名*, ...)
示された列で構成される主キーを定義します。この文節は 1 回しか指定してはならず、示されている列は NOT NULL として定義する必要があります。各列名は表の列を示すものでなければならず、同じ列を 2 回以上指定してはいけません。

指定できる列は 8 つまでです。

指定された列上に、固有の索引が自動的に作成されます。

1 つの表に定義できる主キーは、1 つだけです。

指定された各列の長さ属性は、1024 バイトを超えてはなりません。

参照制約
参照制約を定義します。

FOREIGN KEY (*列名*, ...)
指定した制約名で参照制約を定義します。

ステートメントの対象である表を T1 で表してみましょう。参照制約の外部キーは、示された列で構成されます。列名のリストの中のそれぞれの名前は T1 の列を示すものでなければならず、また、同じ列を 2 回以上指定してはいけません。指定できる列は 8 つまでです。外部キーは DB2 Everyplace では必須ではありません。

REFERENCES *表名*

REFERENCES 文節に指定する表は、カタログにある基本表でなければなりませんが、カタログ表を示すものであってはなりません。

参照制約の外部キーが、直前に指定した参照制約の外部キー表と同じである場合は、その参照制約は重複しています。

次のケースでは、指定する親表を T2 で表し、作成されている表を T1 で表します。

指定された外部キーは T2 の親キーと同じ数の列を持っていなければならず、外部キーの n 番目の列の記述は、その親キーの n 番目の列の記述に相当するものでなければなりません。日時列は、この規則上、ストリング列に相当するとは見なされません。 DB2 Everyplace では、外部キーは必須ではありません。

CHECK (*チェック条件***)**

チェック制約を定義します。*チェック条件* は、検索条件です。列参照は、作成している表の列でなければなりません。表に挿入する、または更新する値は、すべてのチェック制約を満たすものである必要があります。

チェック制約が列定義の一部として指定されている場合は、列の参照は同じ列に対してのみ行うことができます。表定義の一部として指定されたチェック制約には、事前に CREATE TABLE ステートメントで定義された列を示す列参照を持たせることができます。不整合か、重複しているか、同じ条件かどうかのチェックは、チェック制約に対しては行われません。したがって、矛盾した、または重複したチェック制約が定義される場合があります。

チェック条件として「IS NOT NULL」を指定することはできますが、列の NOT NULL 属性を直接指定することにより、ヌルになるようにすることをお勧めします。例えば、CHECK (salary + bonus > 30000) は salary が NULL にセットされていれば受け入れられます。それは、CHECK の制約は、満足するか、不明かのいずれかでなければならず、このケースでは salary が不明であるためです。しかし、CHECK (salary IS NOT NULL) とすると、salary が NULL にセットされている場合、誤りであり、制約に違反していると見なされます。

表に行を挿入したり、更新を行う場合は、チェック制約が適用されます。

CREATE TABLE ステートメントに定義されたすべてのチェック制約は、一体化してシステム・カタログに保管されます。DB2 Everyplace では、この一体化したチェック制約に 512 バイトまでという制限があります。

規則:

- 行のバイト・カウントの実合計が 65536 を超えてはなりません。

  詳細については、158 ページを参照してください。
- データ・タイプが BLOB の列の場合は、チェック、デフォルト、参照、あるいは外部キー制約を含むことができません (SQLSTATE 42962)。
- BLOB データ・タイプの列を、CREATE TABLE ステートメントの主キーの中で使用することはできません。

注意事項:

- 表および列は、大文字の名前で作成する必要があります。大文字小文字混合や小文字の名前は、言語によってはエラーが発生する場合があります。

- モバイル・デバイス上で新しい表を作成して、モバイル・デバイスをサーバーと同期させても、その表がエンタープライズ・データベース上に自動的に作成されることはありません。同期をとる前に、エンタープライズ・データベース上にその表を作成する必要があります。
- データのバイト・カウント: 以下のリストは、データ・タイプごとの列のバイト・カウントを示したものです。このカウントは、リリースによって変わる場合があります。各レコードには、NULL に関する情報も含まれています。NULL 情報は、32 列の各グループごとに 4 バイトを必要とします。 NULL 値は、依然として固定サイズの列サイズを使用します。

データ・タイプ	列バイト・カウント
INTEGER	4
SMALLINT	4
DECIMAL(n, m)	4 ～ 20
CHAR(n)	n+1
VARCHAR(n)	i+5、i は実際の長さ
BLOB	i+4、i は実際の長さ
DATE	4
TIME	4
TIMESTAMP	12

例:

列名 EMPNO、FIRSTNAME、LASTNAME、DEPT、PHONENO、SALARY、および HIREDATE を使用して表 EMPLOYEE を作成します。CHAR は、列に文字データが入ることを意味します。 NOT NULL は列にヌル値を入れることができないことを意味します。VARCHAR は、列に可変長文字データが入ることを意味します。主キーは列 EMPNO からなります。

```
CREATE TABLE EMPLOYEE
  (EMPNO       CHAR(3)    PRIMARY KEY,
   FIRSTNAME   VARCHAR(12)   NOT NULL,
   LASTNAME    VARCHAR(15)   NOT NULL,
   DEPT        CHAR(3),
   PHONENO     CHAR(4),
   SALARY      INT,
   HIREDATE    DATE)
```

関連した解説:

- 147 ページの『DB2 Everyplace SQL ステートメント・サポートの概要』
- 185 ページの『割り当てと比較に関するデータ・タイプの互換性』
- 84 ページの『DB2 Everyplace でサポートされているパラメーター・マーカー』
- 189 ページの『SQLState のリスト』
- 189 ページの『SQLState クラス・コードのサマリー』

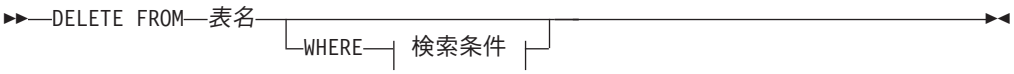
DELETE

DELETE ステートメントは、表から 1 行または複数の行を削除します。

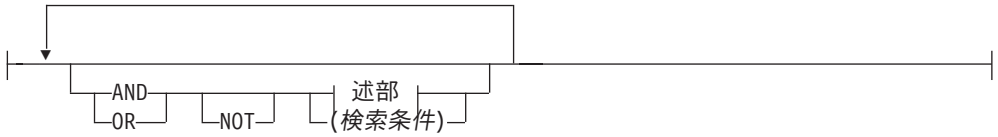
呼び出し:

このステートメントは、アプリケーション・プログラム内で DB2 CLI 関数を用いて使用することができます。また、CLP を介して出すこともできます。

構文:



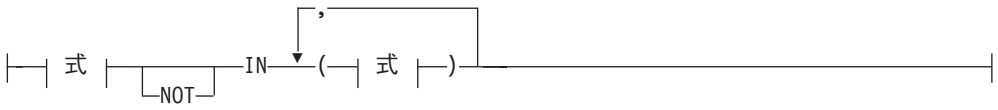
検索条件:



述部:

基本述部:

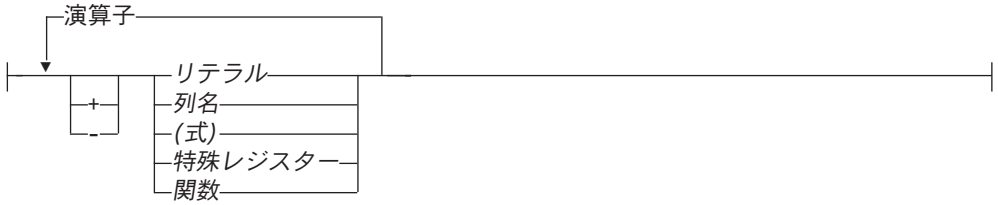
IN 述部:



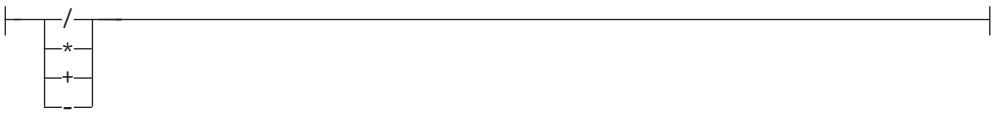
LIKE 述部:

NULL 述部:

式:



演算子:



注:

1 BLOB 式は、NULL 述部でのみ使用することができます。

説明:

FROM *表名*
行を削除すべき表を指定します。表の名前は、カタログに存在する表を指すものでなければなりませんが、カタログ表を指定してはなりません。

WHERE
削除する行を選択する条件を指定します。この文節は省略できます。省略しない場合には、検索条件を指定してください。この文節を省略すると、表のすべての行が削除されます。

検索条件
検索条件 は、指定された行に関して、真、偽、または不明となる条件を指定します。

検索条件 の結果は、指定した各述部の結果に対して、指定した*論理演算子* (AND、OR、NOT) を適用することによって得られます。述部は 2 つの値を比較します。論理演算子を指定しなければ、検索条件の結果は、指定した述部の結果となります。

括弧内の検索条件が先に評価されます。評価の順序を括弧で指定していない場合、NOT が AND の前に適用され、AND が OR の前に適用されます。同じ順序レベルの演算子が評価される順序は、検索条件の最適化ができるように未定義となっています。

表の各行に*検索条件* が適用され、*検索条件* の結果が真となった行が削除されます。

検索条件の中の各*列名* は、表の列を示すものでなければなりません。

NOT
NOT を指定すると、述部の結果は逆になります。

式 述部のオペランドを指定します。式 は、リテラル、列名、特殊レジスター、関数のいずれでもかまいません。

BLOB(n)、DATE、TIME、および TIMESTAMP データ・タイプに対する算術演算はサポートされません。

リテラル
リテラル は、データ・タイプ INTEGER、SMALLINT、DECIMAL、CHAR(n)、VARCHAR(n)、BLOB(n)、DATE、TIME、または TIMESTAMP の値にすることができます。

列名
述部のオペランドである列を指定します。

特殊レジスター
述部のオペランドである特殊レジスターを指定します。現在の日付、時刻およびタイム・スタンプを生成するために、特殊レジスター CURRENT DATE、CURRENT TIME および CURRENT TIMESTAMP を使用することができます。

関数
組み込める関数は MOD、LENGTH、および RTRIM のみです。

関係演算子
以下のいずれかの演算子を指定することができます。

= 等しい。

<> 等しくない。

< より小さい。

> より大きい。

<= より小さいか等しい。

>= より大きいか等しい。

LIKE 1 つの文字ストリングを突き合わせます。1 つの 1 バイト文字セット文字 (SBCS) を表すには、SBCS の下線を使用します。 1 つの 2 バイト文字セット文字 (DBCS) を表すには、DBCS の下線を使用します。例えば、条件が `WHERE PART_NUMBER LIKE '_0'` の場合、0 で終わるすべての 2 桁のパーツ番号 (例えば、20、30、40 など) が戻ります。ゼロ以上の SBCS または DBCS 文字のストリングを表す場合は、% (SBCS あるいは DBCS のいずれでも) を使用してください。例えば、条件が `WHERE DEPT_NUMBER LIKE '2%'` の場合、2 で始まるすべての部門番号 (20、27、234 など) が戻ります。

NOT LIKE
同じ文字が 1 つもありません。

IS NULL
ヌル値を含んでいます。

IS NOT NULL
ヌル値を含んでいません。

AND
指定した場合、論理演算子 AND が、指定した各述部の結果に適用されます。

OR
指定した場合、論理演算子 OR が、指定した各述部の結果に適用されます。

規則:

なし。

注意事項:

- 論理 DELETE は、論理的に削除されたレコードには適用されません。

例:

従業員番号 (EMPNO) 003002 を EMPLOYEE 表から削除します。

```
DELETE FROM EMPLOYEE
  WHERE EMPNO = '003002'
```

関連した解説:

- 147 ページの『DB2 Everyplace SQL ステートメント・サポートの概要』
- 185 ページの『割り当てと比較に関するデータ・タイプの互換性』
- 84 ページの『DB2 Everyplace でサポートされているパラメーター・マーカー』
- 189 ページの『SQLState のリスト』
- 189 ページの『SQLState クラス・コードのサマリー』

DROP

DROP ステートメントは表あるいは索引を削除します。

呼び出し:

このステートメントは、アプリケーション・プログラム内で DB2 CLI 関数を用いて使用することができます。また、CLP を介して出すこともできます。

構文:

説明:

TABLE *表名*

除去すべき基本表を指定します。*表名* は、カタログ内に記述されている表を示すものでなければなりません (SQLSTATE 42704)。

INDEX *索引名*

除去すべき索引を指定します。*索引名* は、カタログ内に記述されている索引を示すものでなければなりません (SQLSTATE 42704)。システムが主キー用に必要とする索引であってはなりません (SQLSTATE 42704)。

規則:

なし。

注意事項:

- 表が使用中であるとき (その表または索引を使用する照会でステートメント・ハンドルがアクティブになっている間) に表および索引をドロップしてはなりません。使用中の表および索引をドロップすると、その表および索引に関係するステートメント・ハンドルが無効になります。

例:

表 EMPLOYEE を除去します。

```
DROP TABLE EMPLOYEE
```

関連した解説:

- 147 ページの『DB2 Everyplace SQL ステートメント・サポートの概要』
- 185 ページの『割り当てと比較に関するデータ・タイプの互換性』
- 84 ページの『DB2 Everyplace でサポートされているパラメーター・マーカー』
- 189 ページの『SQLState のリスト』
- 189 ページの『SQLState クラス・コードのサマリー』

EXPLAIN

EXPLAIN ステートメントは、SELECT ステートメントのアクセス・パス選択に関する情報を取得します。取得した情報は、DB2ePLANTABLE という名前のユーザー表に入れられます。

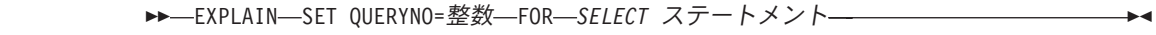
EXPLAIN ステートメントは、以下のプラットフォームでサポートされます。

- Win32 (Windows 95、Windows 98、Windows NT、Windows 2000、および Windows XP)
- Linux

呼び出し:

このステートメントは、アプリケーション・プログラム内で DB2 CLI 関数を用いて使用することができます。また、CLP を介して出すこともできます。

構文:



説明:

SET QUERYNO = 整数
整数 と SELECT ステートメントを関連付けます。EXPLAIN ステートメントによって、プラン表に挿入されたすべての行の QUERYNO 列には整数 値が入ります。

SELECT ステートメント
SELECT ステートメントの結果表の形式で、新規行のセットを指定します。

規則:

整数 値は正でなければなりません。

注意事項:

- EXPLAIN ステートメントを使用した場合、DB2ePLANTABLE がなければ、デフォルトによりそれが自動的に作成されます。
- DB2ePLANTABLE を明示的に作成する場合は、以下の例を使用してください。

```
create table "DB2ePLANTABLE"
 (query_no int, plan_no int, table_name char(18), index_name char(18), sort_temp char(1),
   expl_timestamp timestamp, remarks varchar(300))
```

表 16 は DB2ePLANTABLE 列を説明したものです。

表 16. DB2ePLANTABLE 列情報

列名	説明
query_no	EXPLAIN ステートメントを DB2ePLANTABLE 内の出力に接続する整数。
plan_no	ステートメントの実行ステップを示す整数 (昇順)。
table_name	表の名前、または表を一意的に識別する関連名 (該当しない場合はヌル)。
index_name	表アクセス上の索引の名前 (使用している場合)。索引を使用していない場合は、ヌルを戻します。
sort_temp	「Y」は、GROUP BY または ORDER BY を処理するために一時表のソートが必要であることを意味しています。ヌルが戻された場合は、一時表のソートが必要でないことを示しています。
expl_timestamp	EXPLAIN ステートメント実行時のタイム・スタンプ値。
注釈	注釈列にはヌル値が入っています。記帳のために、この列に注釈を追加することができます。

- DB2ePLANTABLE は、任意のアプリケーションによる変更や除去が可能なユーザー表です。

例:

新規のアプリケーションを開発する場合は、SELECT ステートメントに対してどのアクセス・パスを選択するかを決定することをお勧めします。この例では、新規のアプリケーションが SALES 表と EMPLOYEES 表を照会しています。EXPLAIN ステートメントは、SELECT ステートメントに対して適切な索引が選択されているかどうかを示しています。

```
 EXPLAIN SET QUERYNO = 100 FOR
  SELECT E.EMPNAME, S.SALES_AMOUNT
      FROM SALES S, EMPLOYEES E
            WHERE S.EMPNO = E.EMPNO
                  AND S.MONTH = ?

Index XSALES on SALES(MONTH)
Index XEMP on EMPLOYEES(EMPNO)

SELECT QUERY_NO, PLAN_NO, TABLE_NAME, INDEX_NAME,  SORT_TEMP
   FROM "DB2ePLANTABLE"

QUERY_NO  PLAN_NO  TABLE_NAME  INDEX_NAME  SORT_TEMP
--------------------------------------------------------------------------------
   100     1        SALES        XSALES        -
   100     2        EMPLOYEE     XEMP          -
```

関連した解説:

- 147 ページの『DB2 Everyplace SQL ステートメント・サポートの概要』
- 185 ページの『割り当てと比較に関するデータ・タイプの互換性』
- 84 ページの『DB2 Everyplace でサポートされているパラメーター・マーカー』
- 189 ページの『SQLState のリスト』
- 189 ページの『SQLState クラス・コードのサマリー』

GRANT

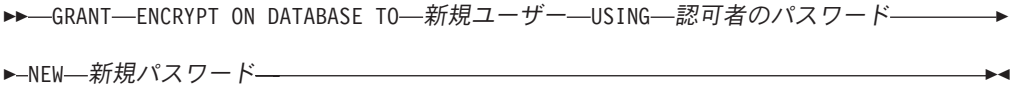
GRANT ステートメントは、データベース内での暗号化された表の作成、照会、および操作の許可を与えます。 GRANT 操作を実行するには、現在接続状態にあり、かつ認証済みでなければなりません。データベースが暗号化されていない場合、ユーザーは (最初のユーザーとして) GRANT 操作を実行するために必要な認証を自分自身に対して実行することができます。 (これの実行方法については、以下の例 1 を参照してください。)

自分のパスワードを変更するには、自分自身のユーザー ID に対して GRANT 操作を実行する必要があります。

呼び出し:

このステートメントは、アプリケーション・プログラム内で DB2 CLI 関数を用いて使用することができます。また、CLP を介して出すこともできます。

構文:



説明:

新規ユーザー
暗号化特権を付与されるユーザーを指定します。

認可者のパスワード
新規ユーザーに暗号化特権を付与する認証済みユーザーのパスワード。

新規パスワード
暗号化特権を付与されるユーザーのパスワード。

規則:

- ユーザー名とパスワードのパラメーターはどちらも、長さが 254 バイトまでに制限されています。
- マルチバイト文字の場合、保管時には UTF-8 エンコード方式が内部的に使用されます。したがって、国際文字セットを使用して書き込まれたユーザー名は、長さが制限されます。
- DB2 Everyplace では、認可者 (つまり、現在接続されているユーザー) は新規ユーザーへ特権を付与するために、認可者のパスワードを再入力する必要があります。この制限があることによって、装置のある場所に認可者が物理的に存在していることが確認されます。
- パスワードとユーザー ID は二重引用符で区切る必要があります。

注意事項:

- 既存のユーザーが自分のパスワードを変更するためには、接続状態にあり、かつ認証済みでなければなりません。ユーザーは自分のパスワードのみを変更することができます。
- GRANT ステートメントは、パラメーター・マーカーまたは SQLPrepare() 関数と共に使用することはできません。

- 無許可のユーザーと接続されている状態で特権を付与しようとすると、SQLSTATE 42502 が戻されます。GRANT ステートメントで間違ったパスワードを指定すると、SQLSTATE 42506 が発生します。

例:

例 1: 最初のユーザーが、まだ暗号化されていないデータベースで、GRANT 操作の実行に必要な認証を自分自身に対して認可します。

```
GRANT ENCRYPT ON DATABASE TO "jsk" USING "foo" NEW "foo"
```

例 2: ユーザー「jsk」(上記の*例 1*) が作成されて認証を受け、接続を所有します。「jsk」が別のユーザーを追加するには、以下のようにします。

```
GRANT ENCRYPT ON DATABASE TO "xin" USING "foo" NEW "bar"
```

例 3: 現在接続中のユーザー「jsk」が、自分のパスワードを変更します。

```
GRANT ENCRYPT ON DATABASE TO "jsk" USING "foo" NEW "fie"
```

例 4: 引き続き接続中のユーザー「jsk」が、自分の新規のパスワードを使用して別のユーザーを追加します。

```
GRANT ENCRYPT ON DATABASE TO "thf" USING "fie" NEW "fum"
```

関連した解説:

- 147 ページの『DB2 Everyplace SQL ステートメント・サポートの概要』
- 185 ページの『割り当てと比較に関するデータ・タイプの互換性』
- 84 ページの『DB2 Everyplace でサポートされているパラメーター・マーカー』
- 189 ページの『SQLState のリスト』
- 189 ページの『SQLState クラス・コードのサマリー』

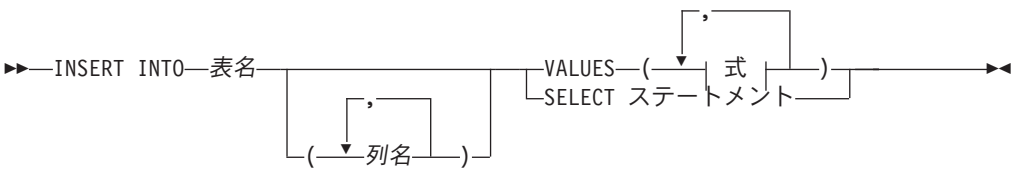
INSERT

INSERT ステートメントは、提供された値を使用して、1 つの表に 1 つ以上の行を挿入します。

呼び出し:

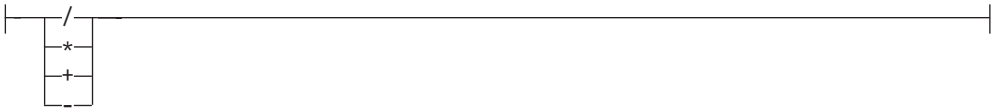
このステートメントは、アプリケーション・プログラム内で DB2 CLI 関数を用いて使用することができます。また、CLP を介して出すこともできます。

構文:



式:

演算子:



説明:

INTO *表名*
挿入操作に関する表を指定します。名前は、既存の表を示すものでなければなりませんが、カタログ表を指定してはなりません。

(*列名***,...)**
挿入値が用意される列を指定します。この名前はそれぞれ、表の列を示す非修飾名でなければなりません。同じ列を 2 回以上指定してはなりません。

列リストを省略すると、表の各列を左から右の順にリストしたものを暗黙指定したことになります。

VALUES
挿入すべき 1 行の値を指定します。

各行に関する値の数は、列リスト内の名前の数と同じでなければなりません。最初の値がリストの 1 列目に挿入され、2 番目の値が 2 列目に挿入されるといった形で、以下同様に続いていきます。

式 式はリテラル、特殊レジスター、関数、または複合式のいずれでもかまいません。

CHAR、VARCHAR、BLOB(n)、DATE、TIME および TIMESTAMP データ・タイプに対する算術演算はサポートされません。

リテラル
リテラルは、サポートされるデータ・タイプ INTEGER、SMALLINT、DECIMAL、CHAR(n)、VARCHAR(n)、BLOB(n)、DATE、TIME、または TIMESTAMP の値にすることができます。

特殊レジスター
現在の日付、時刻およびタイム・スタンプを生成するために、特殊レジスター CURRENT DATE、CURRENT TIME および CURRENT TIMESTAMP を使用することができます。

SELECT ステートメント
SELECT ステートメントの結果表の形式で、新規行のセットを指定します。これは、1 行以上あることもありますし、まったくないこともあります。結果表が空の場合は、SQLCODE が +100 にセットされ、SQLSTATE が 02000 にセットされます。SELECT ステートメントの基本オブジェクトは、INSERT の基本オブジェクトにすることはできません。

規則:

デフォルト値
列リストに存在しない列には、デフォルト値または NULL 値が挿入されます。デフォルト値または NULL 値が許されない列は、列リストに入れておく必要があります。

長さ 列の挿入値が数値である場合、その列は、数値の整数部分を表せるだけの容量を持つ数値列でなければなりません。列の挿入値がストリングである場合、その列は、ストリングの長さ以上の長さ属性を持つストリング列でなければなりません。

割り当て
挿入値は、「*DB2 Universal Database SQL リファレンス*」に記載されている割り当て規則に従って列に割り当てられます。

例:

例 1: 以下の指定に合致した従業員を EMPLOYEE 表に挿入します。

- 従業員番号 (EMPNO) は 002001
- ファーストネーム (FIRSTNAME) は John
- ラストネーム (LASTNAME) は Harrison
- 部門番号 (DEPT) は 600
- 電話番号 (PHONENO) は 4900
- 給与 (SALARY) は 50000
- 雇用開始日付 (HIREDATE) は 01/12/1989

```
INSERT INTO EMPLOYEE
 VALUES ('002001', 'John', 'Harrison', '600', '4900', 50000, '01/12/1989')
```

例 2: 以下の指定に合致した新しい従業員を EMPLOYEE 表に挿入します。

- 従業員番号 (EMPNO) は 003002
- ファーストネーム (FIRSTNAME) は Jim
- ラストネーム (LASTNAME) は Gray

```
    INSERT INTO EMPLOYEE (EMPNO, FIRSTNAME, LASTNAME)
      VALUES ('003002', 'Jim', 'Gray')
```

例 3: EMP_ACT_COUNT 表を作成します。従業員番号 (EMPNO) および関連プロジェクトの数を使用して、EMP_ACT_COUNT に、EMP_ACT 表から行をロードします。

```
CREATE TABLE EMP_ACT_COUNT
   ( EMPNO CHAR(6)  NOT NULL,
     COUNT          INTEGER)

INSERT INTO EMP_ACT_COUNT
    SELECT EMPNO, COUNT(*)
   FROM EMP_ACT
   GROUP BY EMPNO
```

制約事項:

1. SELECT ステートメントの列データ・タイプは、ターゲット表の列定義と等しくなければなりません (ヌルの場合を除く)。

2. ORDER BY および LIMIT 文節は許可されていません。

関連した解説:

- 147 ページの『DB2 Everyplace SQL ステートメント・サポートの概要』
- 185 ページの『割り当てと比較に関するデータ・タイプの互換性』
- 84 ページの『DB2 Everyplace でサポートされているパラメーター・マーカー』
- 189 ページの『SQLState のリスト』
- 189 ページの『SQLState クラス・コードのサマリー』

REORG TABLE

REORG TABLE ステートメントは、指定された表に関連するデータを圧縮します。

呼び出し:

このステートメントは、アプリケーション・プログラム内で DB2 CLI 関数を用いて使用することができます。また、CLP を介して出すこともできます。

構文:

説明:

REORG TABLE *表名*
再編成操作の表を識別します。名前は既存の表を識別しなければなりません。

int1
リカバリーが必要なバイトの最小パーセンテージ (オプショナル)。

int2
表圧縮を実行するためにリカバリーが必要なバイトの最少数。

規則:

- オプショナルである値 *int1* および *int2* は、一緒に使用するか、またはまったく使用しないかの、いずれかでなければなりません。
- オプショナルである値 *int1* は負の数であってはなりません。
- オプショナルである値 *int1* は 0 から 100 の範囲でなければなりません。

注意事項:

- 表の再編成は、DB2 Everyplace によって内部的に起動することができます。
- 最初のオプショナル・パラメーターは、表に含まれていなければならない使用不可バイトの % です (つまり、10 % は「少なくとも 10 % のスペースが使用不可である」ことになります)。 2 番目のオプショナル・パラメーターは、表に含まれていなければならない使用不可バイトのバイト数です (つまり、1000 は「少なくとも 1000 バイトが使用不可のスペースでなければならない」ことになります)。表の再編成を実際に実行する前に、両方の基準が満たされていなければなりません。

- パラメーターが指定されていない場合、DB2 Everyplace はこれらのオプションについてデフォルト値を使用します。デフォルトの % は 30、デフォルトのバイト数は 6144 です。したがって、「reorg table t1」は「reorg table t1 30 6144」と同じことになります。
- 再編成モードが使用可能に設定されている場合、DB2 Everyplace は自動的に表を再編成します。再編成が使用可能になっていると、DELETE または UPDATE 操作において、ステートメントの実行後にターゲット表に対して「reorg table table_name 50 30270」が実行されます。再編成が使用可能になっていると、DROP TABLE 操作において、表の除去処理の最後に「reorg table DB2eSYSTABLES 30 10240」(DB2eSYSCOLUMNS および DB2eSYSRELS も同様) が実行されます。
- C/C++ プログラムの場合、再編成モードは CLI/ODBC 関数 SQLSetStmtAttr に属性 SQL_ATTR_REORG_MODE を指定することによって設定します。 Java プログラムの場合、再編成モードは DB2eStatement インターフェースの enableReorg メソッドによって設定します。デフォルト値では、再編成は使用可能になっています。
- 表の再編成処理では、削除または更新によって作られた使用不可スペースを物理的に再利用することによって、表の含まれているデータ・ファイルを圧縮します。次に、表の索引が、行の新しい物理位置を指すように更新されます。
- DB2 Everyplace システム・カタログ基本表は再編成することができます。
- REORG TABLE ステートメントの実行中は、データベースで他のアクティビティーを行わないようにしてください。

例:

VNNURSE 表は、デフォルト値を用いて圧縮されます。

```
REORG TABLE VNNURSE
```

関連した解説:

- 147 ページの『DB2 Everyplace SQL ステートメント・サポートの概要』
- 185 ページの『割り当てと比較に関するデータ・タイプの互換性』
- 84 ページの『DB2 Everyplace でサポートされているパラメーター・マーカー』
- 189 ページの『SQLState のリスト』
- 189 ページの『SQLState クラス・コードのサマリー』

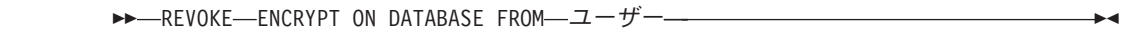
REVOKE

REVOKE ステートメントは、接続された認証済みユーザーによる既存ユーザーの暗号化特権の取り消しを許可します。

呼び出し:

このステートメントは、アプリケーション・プログラム内で DB2 CLI 関数を用いて使用することができます。また、CLP を介して出すこともできます。

構文:



説明:

ユーザー
暗号化特権が取り消されるユーザーを指定します。

規則:

- ユーザー・パラメーターは、区切り ID でなければなりません。このパラメーターの長さは 254 バイトまでに制限されています。
- マルチバイト文字の場合、保管時には UTF-8 エンコード方式が内部的に使用されます。したがって、国際文字セットを使用して書き込まれたユーザー名は、長さが制限されます。
- 暗号化特権を持つユーザーがすべて除去された場合でも、現行セッション中は引き続き暗号化された表にアクセスすることができます。現行セッションが終了すると、暗号化された表にはアクセスできなくなります。

注意事項:

- 既存のユーザーの特権を取り消す場合、取り消し操作を実行するユーザーは接続状態にあり、かつ認証済みでなければなりません。接続され、認証済みのユーザーであれば、自分自身を含む任意のユーザーの特権を取り消すことができます。
- REVOKE ステートメントをパラメーター・マーカーまたは SQLPrepare() 関数と共に使用することはできません。
- 無許可のユーザーとして接続されているのに特権を REVOKE しようとすると、SQLSTATE 42502 が戻されます。また、存在しないユーザーの特権を REVOKE しようとすると、SQLSTATE 42501 となります。

例:

現在接続状態にある、認証済みのユーザーが、ユーザー「jsk」の暗号化特権を除去します。

```
REVOKE ENCRYPT ON DATABASE FROM "jsk"
```

関連した解説:

- 147 ページの『DB2 Everyplace SQL ステートメント・サポートの概要』
- 185 ページの『割り当てと比較に関するデータ・タイプの互換性』
- 84 ページの『DB2 Everyplace でサポートされているパラメーター・マーカー』
- 189 ページの『SQLState のリスト』
- 189 ページの『SQLState クラス・コードのサマリー』

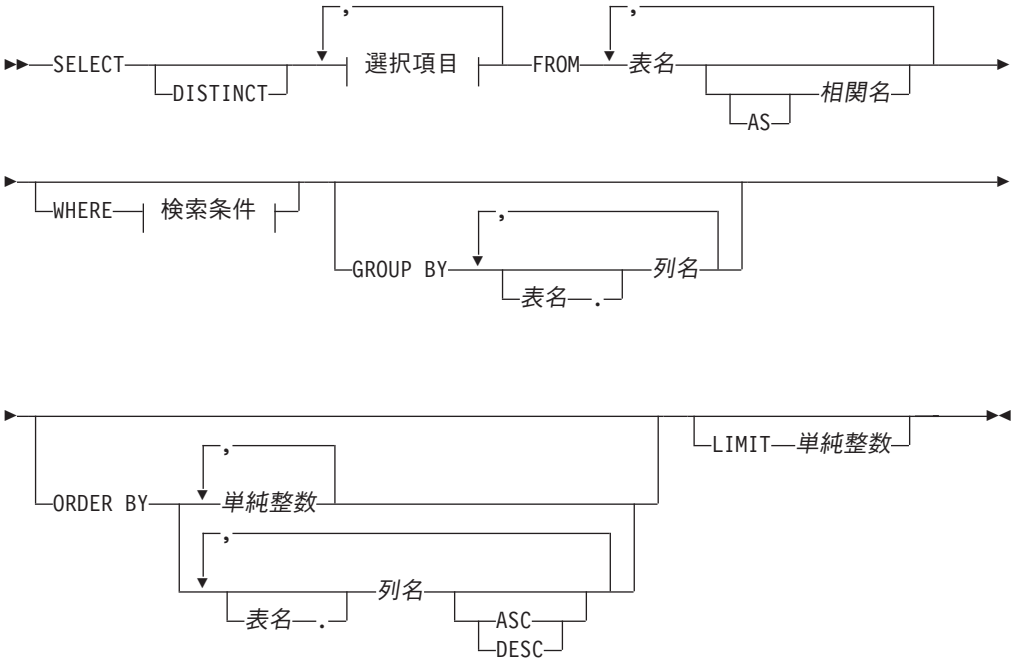
SELECT

SELECT ステートメントは照会の 1 形式です。

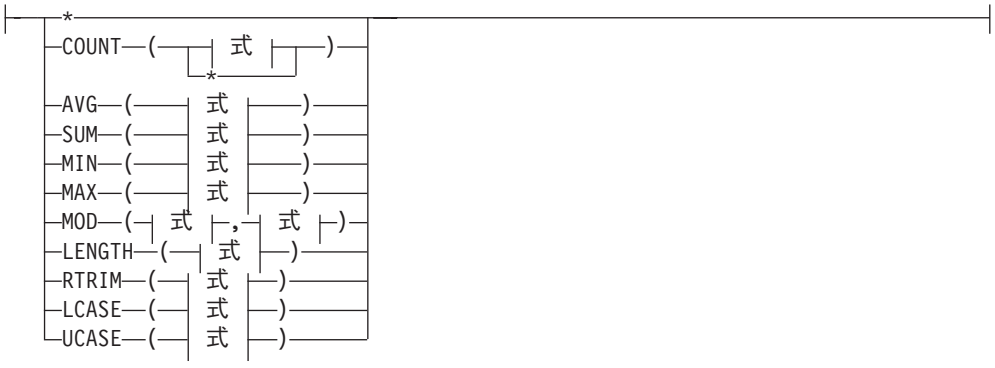
呼び出し:

このステートメントは、アプリケーション・プログラム内で DB2 CLI 関数を用いて使用することができます。また、CLP を介して出すこともできます。

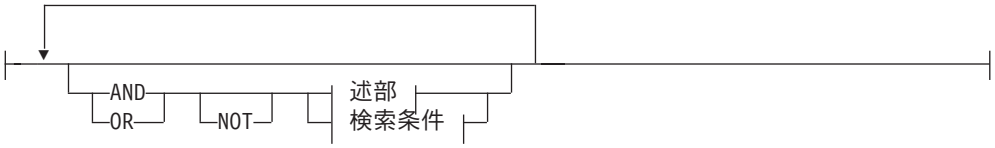
構文:



選択項目:



検索条件:



述部:

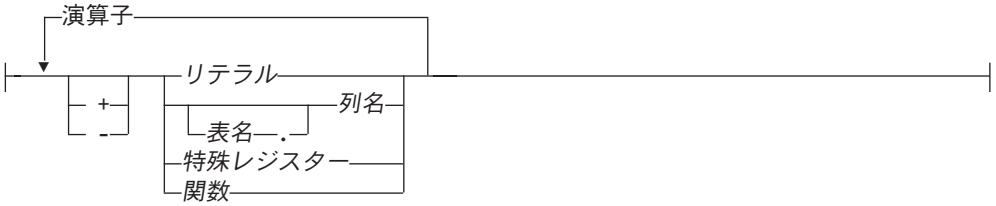
基本述部:

IN 述部:

LIKE 述部:

NULL 述部:

式:



演算子:

注:

1 BLOB 式は、NULL 述部でのみ使用することができます。

説明:

選択項目

***** すべての列を指定します。* を指定する場合、これが唯一の選択項目でなければなりません。

COUNT(*)
COUNT 関数は、1 組の行または値の中の行数または値の数を戻します。COUNT(*) の引き数は 1 組の行です。結果は、1 組の中の行数です。NULL 値しか入っていない行もカウントされます。

式 式 は、リテラル、列名、関数、または特殊レジスターのいずれでもかまいません。妥当な関数は、COUNT、AVG、SUM、MIN、MAX、MOD、LENGTH、および RTRIM です。

CHAR、VARCHAR、BLOB(n)、DATE、TIME および TIMESTAMP データ・タイプに対する算術演算はサポートされません。

リテラル
リテラル は、データ・タイプ INTEGER、SMALLINT、DECIMAL、CHAR(n)、VARCHAR(n)、BLOB(n)、DATE、TIME または TIMESTAMP の値にすることができます。

表名
照会する列を含んでいる表を指定します。

. 2 つの部分からなる列 ID の区切り文字。*表名.列名*

列名
照会する列を指定します。

COUNT(*式***)**
COUNT(式) の引き数は、1 組の行です。ヌル値の除去により、引き数値から引き出された 1 組の行にこの関数が適用されます。結果は、重複を含む、1 組の中の異なる非ヌル値の数です。

AVG(*式***)**
AVG(式) 関数は、式 の値の平均を戻します。引き数値は数値でなければなりません。また、合計は、結果のデータ・タイプの範囲内でなければなりません。ヌル値の除去により、引き数値から引き出された 1 組の値にこの関数が適用されます。結果はヌル値になる場合があります。

SUM(*式***)**
SUM(式) 関数は、式 の値の合計を戻します。引き数値は数値でなければなりません。また、合計は、結果のデータ・タイプの範囲内でなければなりません。ヌル値の除去により、引き数値から引き出された 1 組の値にこの関数が適用されます。

MIN(*式***)**
MIN(式) 関数は、式 の結果の最小値を戻します。引き数値は、BLOB 以外の任意の組み込みタイプの値を指定することができます。ヌル値の除去により、引き数値から引き出された 1 組の値にこの関数が適用されます。

MAX(*式***)**
MAX(式) 関数は、式 の結果の最大値を戻します。引き数値は、BLOB 以外の任意の組み込みタイプの値を指定することができます。ヌル値の除去により、引き数値から引き出された 1 組の値にこの関数が適用されます。

MOD(*式, 式***)**
MOD(式, 式) 関数は、最初の引き数を 2 番目の引き数で除算した余りを戻します。最初の引き数が負の値である場合にのみ、結果は負となります。

1 番目、2 番目の引き数は SMALLINT または INTEGER のいずれかです。

両方の引き数が SMALLINT であれば、結果は SMALLINT であり、それ以外の場合は、 INTEGER となります。結果がヌルになる場合があります。つまり、引き数がヌルであれば、結果はヌル値になります。

(式 || 式)

(式 || 式) は、2 つのストリング引き数を連結した結果を戻します。この 2 つの引き数は、互換タイプでなければなりません。

この関数の結果はストリングです。その長さは、2 つの引き数の長さの合計です。引き数がヌル値もとる場合は、結果がヌルになることがあります。つまり、引き数がヌルであれば、結果はヌル値になります。

LENGTH(式)

LENGTH(式) 関数は、値の長さを戻します。

引き数は式にもなり、以下の組み込まれたデータ・タイプの値を戻します。

- VARCHAR
- CHAR
- BLOB

この関数の結果は整数です。引き数がヌル値もとる場合は、結果がヌルになることがあります。つまり、引き数がヌルであれば、結果はヌル値になります。

結果は、引き数の長さです。可変長ストリングの長さは実際の長さで、最大長ではありません。

BLOB の長さは、値を表すために使用したバイトの数です。

ADDRESS という列名の VARCHAR(50) の列に、「895 Don Mills Road」という値が入っているとします。`LENGTH(ADDRESS)` は 18 という値を戻します。

RTRIM(式)

RTRIM(式) 関数は、ストリングの終わりからブランクを除去します。

引き数は、CHAR または VARCHAR のデータ・タイプにすることができます。

この機能の結果データ・タイプは常に VARCHAR となります。

戻されたタイプの長さパラメーターは、引き数データ・タイプの長さパラメーターと同じになります。

文字ストリングの結果の実際の長さは、ストリング式から、ブランク文字として除去したバイト数を引いた長さです。漢字ストリングの結果の実際の長さは、ストリング式から除去したブランク文字 (2 バイト) を引いた長さ (2 バイト文字) です。すべての文字が除去されると、結果は空になり、可変長ストリング (長さはゼロ) となります。

引き数がヌル値もとる場合は、結果がヌルになることがあります。つまり、引き数がヌルであれば、結果はヌル値になります。

NAME という列名の CHAR(50) の列に、「Cliff」という値が入っているとします。`RTRIM(NAME)` は「Cliff」を戻します。`LENGTH(RTRIM(NAME))` は 5 を戻します。

LCASE / UCASE

LCASE 関数または LOWER 関数は、SBCS 文字がすべて小文字に変換されたストリングを戻します。つまり、文字 A から Z が文字 a から z に変換され、発音区別符号が付いた文字があれば、それらは同等の小文字に変換されます。

引き数は、値のデータ・タイプが CHAR か VARCHAR である式でなくてはなりません。

上記の関数の結果は、引き数と同じデータ・タイプと長さの属性を持つことになります。引き数がヌル値もとる場合は、結果がヌルになることがあります。つまり、引き数がヌルであれば、結果はヌル値になります。

EMPLOYEE 表の列 JOB の値に含まれる文字は小文字で戻されることを確認してください。例えば、次のようにします。

```
SELECT LCASE(JOB)
     FROM EMPLOYEE
     WHERE EMPNO = '000020';
```

特殊レジスター

現在の日付、時刻およびタイム・スタンプを生成するために、特殊レジスター CURRENT DATE、CURRENT TIME および CURRENT TIMESTAMP を使用することができます。

FROM

FROM 文節は、中間結果表を指定します。

1 つの表参照を指定した場合、中間結果表は単純にその表参照の結果です。 2 つ以上の表参照を指定した場合、中間結果表は、指定した表参照の行の可能な組み合わせすべて (カルテシアン積) で構成されます。結果の各行は、最初の表参照からの行を 2 番目の表参照からの行と連結したもの、さらに、2 番目の行は 3 番目の表参照からの行と連結した行になります。以下同様です。結果における行数は、各表参照における行数の積となります。 FROM 文節には、最大 20 まで表を指定できます。

表名

表参照として指定する各*表名* は、既存の表を示す必要があります。

AS

表定義を指定します。

相関名

各*相関名* は、直前の*表名* を指定機能として定義します。表に関して相関名を指定した場合、表の列へのすべての修飾参照は、表名ではなく相関名を使用する必要があります。同じ*表名* を 2 回指定する場合、少なくとも 1 つの表名指定の後に*相関名* を付ける必要があります。*相関名* は、表の列への参照を修飾するために使用します。修飾子として相関名を使うことによって、あいまいになることを防いだり、相関参照を設定したりすることができます。また、相関名は、単に表名を短くした名前として使うこともできます。

WHERE

行を選択する条件を指定します。この文節は省略できます。省略しない場合には、検索条件を指定してください。この文節を省略すると、表のすべての行が選択されます。

検索条件

検索条件 は、指定された行に関して、真、偽、または不明となる条件を指定します。

検索条件 の結果は、指定した各述部の結果に対して、指定した*論理演算子* (AND、OR、NOT) を適用することによって得られます。述部は 2 つの値を比較します。論理演算子を指定しなければ、検索条件の結果は、指定した述部の結果となります。

括弧内の検索条件が先に評価されます。評価の順序を括弧で指定していない場合、NOT が AND の前に適用され、AND が OR の前に適用されます。同じ順序レベルの演算子が評価される順序は、検索条件の最適化ができるように未定義となっています。

表の各行に*検索条件* が適用され、*検索条件* の結果が真となった行が選択されます。

検索条件の中の各*列名* は、表の列を示すものでなければなりません。

NOT
NOT を指定すると、述部の結果は逆になります。

式 式 は、リテラル、列名、特殊レジスター、関数のいずれでもかまいません。

CHAR、VARCHAR、BLOB(n)、DATE、TIME および TIMESTAMP データ・タイプに対する算術演算はサポートされません。

リテラル
リテラル は、データ・タイプ INTEGER、SMALLINT、DECIMAL、CHAR(n)、VARCHAR(n)、BLOB(n)、DATE、TIME、または TIMESTAMP の値にすることができます。

表名
述部のオペランドである列を含んでいる表を指定します。

. 2 つの部分からなる列 ID の区切り文字。*表名.列名*

列名
述部のオペランドである列を指定します。

特殊レジスター
述部のオペランドである特殊レジスターを指定します。現在の日付、時刻およびタイム・スタンプを生成するために、特殊レジスター CURRENT DATE、CURRENT TIME および CURRENT TIMESTAMP を使用することができます。

関数
組み込める関数は LCASE、UCASE、MOD、LENGTH、および RTRIM です。

演算子
以下のいずれかの演算子を指定することができます。

= 等しい。

<> 等しくない。

< より小さい。

> より大きい。

<= より小さいか等しい。

>= より大きいか等しい。

|| 2 つのストリング引き数を連結した結果を戻します。

LIKE 1 つの文字ストリングを突き合わせます。1 つの 1 バイト文字セット文字 (SBCS) を表すには、SBCS の下線を使用します。 1 つの 2 バイト文字セット文字 (DBCS) を表すには、DBCS の下線を使用します。例えば、条件が `WHERE PART_NUMBER LIKE '_0'` の場合、0 で終わるすべての 2 桁のパーツ番号 (例えば、20、30、40 など) が戻ります。ゼロ以上の SBCS または DBCS 文字のストリングを表す場合は、% (SBCS あるいは DBCS のいずれでも) を使用してください。例えば、条件が `WHERE DEPT_NUMBER LIKE '2%'` の場合、2 で始まるすべての部門番号 (20、27、234 など) が戻ります。

NOT LIKE
同じ文字が 1 つもありません。

IN 値の集合と一致するもの。述部 IN は、値を値の集合と比較します。

例:

```
SELECT lname, fname FROM emp WHERE state IN ('CA', 'AZ',
'OR');

SELECT c1 FROM t1 WHERE c1*5-6 IN (mod(c2,2)+5,c3+4/2);
```

NOT IN
値の集合と一致しないもの。述部 NOT IN は、値を値の集合と比較します。

例:

```
SELECT empid FROM emp WHERE city NOT IN ('San Jose',
'Morgan Hill', 'Santa Clara');
```

IS NULL
ヌル値を含んでいます。

IS NOT NULL
ヌル値を含んでいません。

AND
指定した場合、論理演算子 AND が、指定した各述部の結果に適用されます。

OR
指定した場合、論理演算子 OR が、指定した各述部の結果に適用されます。

GROUP BY
R の行のグループ化により構成される中間結果表を作成します。R は副選択に関する前の文節の結果です。

ORDER BY
結果表の行の配列を指定します。

列名
通常は、結果表の列を指定します。この場合、*列名* は選択リスト内の指定された列の列名でなければなりません。

単純整数
1 以上で、結果表の列の数以下でなければなりません。整数 *n* は結果表の *n* 列目を示します。

ASC
列の値を昇順で使用します。

DESC
列の値を降順で使用します。

LIMIT *単純整数*
アプリケーションに戻す行の数を、応答セット内の最初の *n* 行に制限します。この *n* は整数です。 0 より大である必要があります。

関連演算子
以下のいずれかの演算子を指定することができます。

+ 加算

- 減算

***** 乗算

/ 除算

規則:

GROUP BY、ORDER BY、および DISTINCT 文節で BLOB データ・タイプ列を使用することはできません。

注意事項:

- SELECT DISTINCT ステートメントには、最高 8 つの列を含めることができます。
- GROUP BY 文節には、最高 8 つの列を含めることができます。
- ORDER BY 文節には、最高 8 つの列を含めることができます。
- ORDER BY 文節に指定する列はすべて、選択リストに指定しなければなりません。例えば、次の照会は無効です。

  ```
  SELECT EMPNO, FIRSTNAME FROM EMPLOYEE ORDER BY LASTNAME
  ```

  次の照会は有効です。

  ```
  SELECT LASTNAME, EMPNO, FIRSTNAME FROM EMPLOYEE ORDER BY LASTNAME
  ```

例:

例 1: EMPLOYEE 表から 01/01/1980 以降に雇用された従業員 (EMPNO および LASTNAME) を選択し、ラストネーム (LASTNAME) の順に並べます。

```
SELECT EMPNO, LASTNAME FROM EMPLOYEE
 WHERE HIREDATE > '01/01/1980'
 ORDER BY LASTNAME
```

例 2: EMPLOYEE 表の部門ごとに平均給与を計算します。

```
SELECT DEPT, AVG(SALARY) FROM EMPLOYEE
 GROUP BY DEPT
```

例 3: 各販売地域ごとに最大販売量を計算し、販売量の多い方から少ない方へと順に、地域別に結果を表示します。

```
SELECT REGION, MAX(SALES_VOL) FROM SALES
 GROUP BY REGION ORDER BY 2 DESC
```

関連した解説:

- 147 ページの『DB2 Everyplace SQL ステートメント・サポートの概要』
- 185 ページの『割り当てと比較に関するデータ・タイプの互換性』
- 84 ページの『DB2 Everyplace でサポートされているパラメーター・マーカー』
- 189 ページの『SQLState のリスト』
- 189 ページの『SQLState クラス・コードのサマリー』

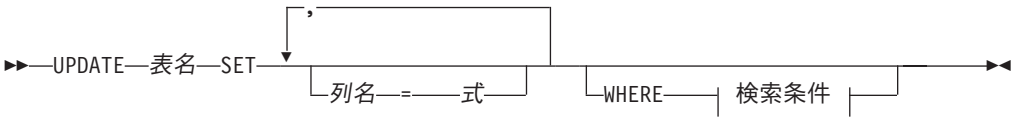
UPDATE

UPDATE ステートメントは、表の行の中の指定した列の値を更新します。

呼び出し:

このステートメントは、アプリケーション・プログラム内で DB2 CLI 関数を用いて使用することができます。また、CLP を介して出すこともできます。

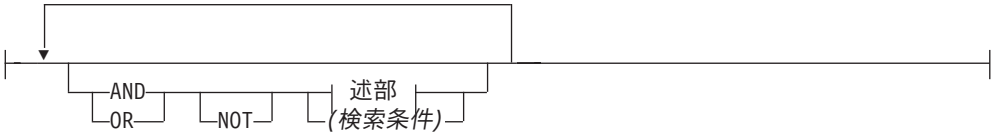
構文:



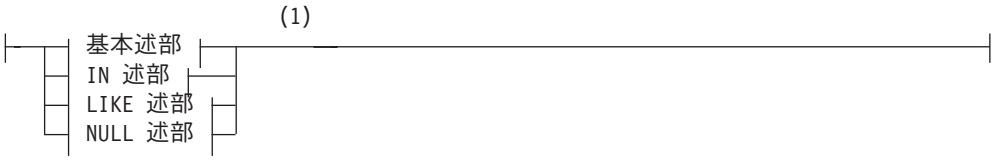
式:

演算子:

検索条件:

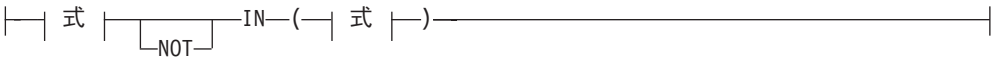


述部:

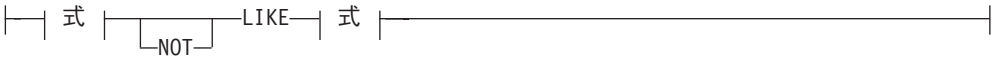


基本述部:

IN 述部:



LIKE 述部:



NULL 述部:

関係演算子:

注:

1 BLOB 式は、NULL 述部でのみ使用することができます。

説明:

表名

更新したい表の名前です。名前は、カタログ内で記述されている表を示すものでなければなりませんが、カタログ表を指定してはなりません。

SET

列名への値の割り当てを指定します。

列名
更新したい列を指定します。*列名* は指定した表の列を示すものでなければなりません。同じ列を 2 回以上指定してはなりません (SQLSTATE 42701)。

式 式 はリテラル、列名、または特殊レジスターのいずれでもかまいません。

BLOB(n)、DATE、TIME、および TIMESTAMP データ・タイプに対する算術演算はサポートされません。

リテラル
リテラル は、データ・タイプ INTEGER、SMALLINT、DECIMAL、CHAR(n)、VARCHAR(n)、BLOB(n)、DATE、TIME、または TIMESTAMP の値にすることができます。

特殊レジスター
現在の日付、時刻およびタイム・スタンプを生成するために、特殊レジスター CURRENT DATE、CURRENT TIME および CURRENT TIMESTAMP を使用することができます。

WHERE
更新する行を示す条件を指定します。この文節は省略しても構いません。また、指定する場合は、検索条件を指定することができます。この文節を省略すると、表のすべての行が更新されます。

検索条件
検索条件 は、指定された行に関して、真、偽、または不明となる条件を指定します。

検索条件 の結果は、指定した各述部の結果に対して、指定した*論理演算子* (AND、OR、NOT) を適用することによって得られます。述部は 2 つの値を比較します。論理演算子を指定しなければ、検索条件の結果は、指定した述部の結果となります。

括弧内の検索条件が先に評価されます。評価の順序を括弧で指定していない場合、NOT が AND の前に適用され、AND が OR の前に適用されます。同じ順序レベルの演算子が評価される順序は、検索条件の最適化ができるように未定義となっています。

表の各行に*検索条件* が適用され、*検索条件* の結果が真となった行が更新されます。

検索条件の中の各*列名* は、表の列を示すものでなければなりません。

検索条件の述部式では、CONCAT、MOD、LENGTH、および RTRIM 関数を使用することができます。MOD 関数の詳細については、175ページを参照してください。

NOT
NOT を指定すると、述部の結果は逆になります。

関係演算子
以下のいずれかの演算子を指定することができます。

= 等しい。

<> 等しくない。

< より小さい。

> より大きい。

<= より小さいか等しい。

>= より大きいか等しい。

LIKE 1 つの文字ストリングを突き合わせます。1 つの 1 バイト文字セット文字 (SBCS) を表すには、SBCS の下線を使用します。 1 つの 2 バイト文字セット文字 (DBCS) を表すには、DBCS の下線を使用します。例えば、条件が `WHERE PART_NUMBER LIKE '_0'` の場合、0 で終わるすべての 2 桁のパーツ番号 (例えば、20、30、40 など) が戻ります。ゼロ以上の SBCS または DBCS 文字のストリングを表す場合は、% (SBCS あるいは DBCS のいずれでも) を使用してください。例えば、条件が `WHERE DEPT_NUMBER LIKE '2%'` の場合、2 で始まるすべての部門番号 (20、27、234 など) が戻ります。

NOT LIKE
同じ文字が 1 つもありません。

IS NULL
ヌル値を含んでいます。

IS NOT NULL
ヌル値を含んでいません。

AND
指定した場合、論理演算子 AND が、指定した各述部の結果に適用されます。

OR
指定した場合、論理演算子 OR が、指定した各述部の結果に適用されます。

規則:

- ***割り当て***: 更新値は、「*DB2 Universal Database SQL リファレンス*」に記載されている割り当て規則に従って列に割り当てられます。
- UPDATE は、論理的に削除されたレコードには適用されません。

注意事項:

- システム・モードでは、デフォルトによりダーティー・ビットが設定されます。アプリケーションをシステム・モード (SQL_DIRTYBIT_SET_BY_SYSTEM) で実行している場合は、手動でダーティー・ビットを設定することはできません。ダーティー・ビットを設定しようとすると、エラーになります。

  詳細については、287ページを参照してください。

例:

EMPLOYEE 表内の従業員番号 (EMPNO) '003002' の電話番号 (PHONENO) を '1234' に変更します。

```
UPDATE EMPLOYEE
 SET PHONENO = '1234'
 WHERE EMPNO = '003002'
```

関連した解説:

- 147 ページの『DB2 Everyplace SQL ステートメント・サポートの概要』
- 『割り当てと比較に関するデータ・タイプの互換性』
- 84 ページの『DB2 Everyplace でサポートされているパラメーター・マーカー』
- 189 ページの『SQLState のリスト』
- 189 ページの『SQLState クラス・コードのサマリー』

割り当てと比較に関するデータ・タイプの互換性

割り当て演算は、INSERT および UPDATE ステートメントの実行時に行われます。比較演算は、述部を含むステートメントの実行時に行われます。関与するオペランドのデータ・タイプは、表 17から 186 ページの表 19に示してあるように、互換性がなくてはなりません。

データ・タイプの列の意味は、次のとおりです。

X 各オペランドのデータ・タイプに互換性があります。

ブランク
各オペランドのデータ・タイプに互換性がありません。

表 17. データ・タイプの互換性、表 1

SQL データ・タイプ	INT	SMALLINT	DECIMAL	BLOB
INT	X	X	X	
VARCHAR				
BLOB				X
DECIMAL	X	X	X	
CHAR				
SMALLINT	X	X	X	
DATE				
TIME				
TIMESTAMP				

表 18. データ・タイプの互換性、表 2

SQL データ・タイプ	CHAR	VARCHAR
INT		
VARCHAR	X	X
BLOB		
DECIMAL		
CHAR	X	X
SMALLINT		
DATE	X	X
TIME	X	X
TIMESTAMP	X	X

表 19. データ・タイプの互換性、表 3

SQL データ・タイプ	DATE	TIME	TIMESTAMP
INT			
VARCHAR	X	X	X
BLOB			
DECIMAL			
CHAR	X	X	X
SMALLINT			
DATE	X		
TIME		X	
TIMESTAMP			X

関連した解説:

- 147 ページの『DB2 Everyplace SQL ステートメント・サポートの概要』
- 84 ページの『DB2 Everyplace でサポートされているパラメーター・マーカー』
- 189 ページの『SQLState のリスト』

SQL のシンボリックおよびデフォルトのデータ・タイプ

表 20. SQL のシンボリックおよびデフォルトのデータ・タイプ

SQL データ・タイプ	シンボリック SQL データ・タイプ	デフォルト・シンボリック C データ・タイプ
BLOB	SQL_BLOB	SQL_C_BINARY
CHAR	SQL_CHAR	SQL_C_CHAR
DATE	SQL_TYPE_DATE	SQL_C_TYPE_DATE
DECIMAL	SQL_DECIMAL	SQL_C_CHAR
INTEGER	SQL_INTEGER	SQL_C_LONG
SMALLINT	SQL_SMALLINT	SQL_C_SHORT
TIME	SQL_TYPE_TIME	SQL_C_TYPE_TIME
TIMESTAMP	SQL_TYPE_TIMESTAMP	SQL_C_TYPE_TIMESTAMP
VARCHAR	SQL_VARCHAR	SQL_C_CHAR

データ・タイプ属性

以下のデータ・タイプ属性についての情報を示します。

- 精度
- 位取り
- 長さ
- ディスプレイ・サイズ

精度:

数値列または数値パラメーターの精度とは、その列またはパラメーターのデータ・タイプで使用される最大桁数のことを指します。非数値列または非数値パラメーターの精度とは一般的に、その列またはパラメーターの最大長か定義された長さのこ

とを指します。以下の表では、各 SQL データ・タイプの精度について定義します。

表 21. 精度

fSqlType	精度
SQL_CHARSQL_VARCHAR	定義された、列またはパラメーターの長さ。例えば、CHAR(10) と定義された列の精度は 10 です。
SQL_DECIMAL	定義された最大桁数。例えば、DECIMAL(10,3) と定義された列の精度は 10 です。
SQL_SMALLINT a	5
SQL_INTEGER a	10
SQL_BLOB	定義された、列またはパラメーターの長さ。例えば、BLOB(10) と定義された列の精度は 10 です。
SQL_DATE a	10 (yyyy-mm-dd という形式での文字の数)。
SQL_TIME a	8 (hh:mm:ss という形式での文字の数)。
SQL_TIMESTAMP	26 (TIMESTAMP データ・タイプで使用される、yyyy-mm-dd-hh.mm.ss.ffffff という形式での文字数。)

a: `SQLBindParameter()` の *cbParamDef* 引き数は、このデータ・タイプでは無視されます。

位取り:

数値列または数値パラメーターの位取りとは、小数点の右側に存在する桁の最大数のことを指します。以下の表で、各 SQL データ・タイプの位取りについて定義します。

表 22. 位取り

fSqlType	位取り
SQL_CHARSQL_VARCHAR	適用外。
SQL_DECIMAL	小数点の右側にある、定義された桁数。例えば、DECIMAL(10, 3) と定義された列の位取りは 3 です。
SQL_SMALLINTSQL_INTEGER	0
SQL_BLOB	適用外。
SQL_DATE SQL_TIME	適用外。
SQL_TIMESTAMP	6 (yyyy-mm-dd-hh.mm.ss.ffffff という形式での小数点の右側の桁数。)

長さ:

列の長さとは、データがデフォルトの C データ・タイプに転送されたときにアプリケーションに戻される最大バイト数のことです。文字データの場合、この長さにヌ

ル終了バイトは含まれません。列の長さは、データ・ソース上のデータを格納するのに必要なバイト数とは異なることに注意してください。

以下の表では、各 SQL データ・タイプの長さについて定義します。

表 23. 長さ

fSqlType	長さ
SQL_CHARSQL_VARCHAR	列の定義された長さ。例えば、CHAR(10) と定義された列の長さは 10 です。
SQL_DECIMAL	最大桁数に 2 を加算した数。これらのデータ・タイプは文字ストリングとして戻されるため、各桁の数字、符号、および小数点のための文字が必要となります。例えば、DECIMAL(10,3) と定義された列の長さは 12 です。
SQL_SMALLINT	2 (2 バイト)。
SQL_INTEGER	4 (4 バイト)。
SQL_BLOB	列の定義された長さ。例えば、BLOB(10) と定義された列の長さは 10 です。
SQL_DATE SQL_TIME	6 (DATE_STRUCT または TIME_STRUCT 構造体のサイズ)。
SQL_TIMESTAMP	16 (TIMESTAMP_STRUCT 構造体のサイズ)。

ディスプレイ・サイズ:

列のディスプレイ・サイズとは、データを文字書式で表示するのに必要な最大バイト数のことです。以下の表では、各 SQL データ・タイプのディスプレイ・サイズについて定義します。

表 24. ディスプレイ・サイズ

fSqlType	ディスプレイ・サイズ
SQL_CHARSQL_VARCHAR	列の定義された長さ。例えば、CHAR(10) と定義された列のディスプレイ・サイズは 10 です。
SQL_DECIMAL	列の精度に 2 を加算した値 (符号、数字の最大桁数、および小数点)。例えば、DECIMAL(10,3) と定義された列のディスプレイ・サイズは 12 です。
SQL_SMALLINT	6 (1 つの符号と 5 桁の数字)。
SQL_INTEGER	11 (1 つの符号と 10 桁の数字)。
SQL_BLOB	定義された列の長さに 2 を掛けた値 (各バイナリー・バイトは 2 桁の 16 進数で表現されます)。例えば、BLOB(10) と定義された列のディスプレイ・サイズは 20 です。
SQL_DATE	10 (yyyy-mm-dd という形式での日付)。
SQL_TIME	8 (hh:mm:ss という形式での時刻)。

表 24. ディスプレイ・サイズ (続き)

fSqlType	ディスプレイ・サイズ
SQL_TIMESTAMP	26 (yyyy-mm-dd-hh.mm.ss.ffffff という形式でのタイム・スタンプ)。

SQLState のリスト

このセクションでは、SQL または CLI から生成されたエラー・メッセージの解釈方法について説明します。

- 『SQLState クラス・コードのサマリー』に、エラーの一般的なカテゴリーのリストがあります。
- 190 ページの『SQL が報告する SQLState メッセージ』、 193 ページの『CLI が報告する SQLState メッセージ』、および 202 ページの『JDBC により報告される SQLState メッセージ』には、各エラーの説明と、SQL に関しては、そのエラーを発生させた関数の名前も載っています。

DB2 UDB がインストールされている場合は、以下のように DB2 コマンド行プロセッサーを使用して SQLSTATE の説明を見つけることもできます。

1. コマンド行プロセッサーを開くために、「**スタート**」—>「**プログラム**」—>「**DB2**」—>「**コマンド行プロセッサー (Command Line Processor)**」と選択します。
2. コマンド行に、? [*SQLSTATE*] と入力します。

関連した解説:

- 147 ページの『DB2 Everyplace SQL ステートメント・サポートの概要』
- 185 ページの『割り当てと比較に関するデータ・タイプの互換性』
- 『SQLState クラス・コードのサマリー』

SQLState クラス・コードのサマリー

193 ページの表 27 にある SQLState メッセージの**太字**になっている最初の 2 文字は、クラス・コードを表しています。これらのクラス・コードは、表 25 に要約 されています。

表 25. SQLState クラス・コード

コード	クラス
00	無条件正常終了
01	警告
02	データなし
07	動的 SQL エラー
08	接続例外
09	トリガー・アクション例外
0A	サポートされない機能
0F	トークンが無効
21	カーディナリティー違反
22	データ例外

表 25. SQLState クラス・コード (続き)

コード	クラス
23	制約違反
24	カーソル状態が無効
25	トランザクション状態が無効
26	SQL ステートメント ID が無効
28	許可指定が無効
2D	トランザクション終了が無効
2E	接続名が無効
34	カーソル名が無効
38	外部関数例外
39	外部関数呼び出し例外
40	トランザクションのロールバック
42	構文エラーまたはアクセス規則違反
44	チェック・オプション違反
46	Java DDL
51	アプリケーション状態が無効
54	SQL または製品の制限を超過
55	オブジェクトが前提条件状態ではない
56	各種 SQL エラーまたは製品のエラー
57	リソースが利用不能か、またはオペレーターの介入が必要
58	システム・エラー・リソース
59	DB2 Everyplace アドミニストレーター・エラー
HY	DB2 CLI または ODBC ドライバーが生成
IM	ODBC ドライバー・マネージャーが生成

関連した解説:

- 147 ページの『DB2 Everyplace SQL ステートメント・サポートの概要』
- 185 ページの『割り当てと比較に関するデータ・タイプの互換性』
- 189 ページの『SQLState のリスト』
- 84 ページの『DB2 Everyplace でサポートされているパラメーター・マーカー』

SQL が報告する SQLState メッセージ

表 26 は、DB2 Everyplace SQL エンジンによって報告される、SQL ステートメントに関するすべての SQLSTATE をリストアップしたものです。DB2 CLI により報告される SQLSTATE は、 202 ページの『DB2 CLI 関数の要約』に記載されている各 DB2 CLI 関数の説明のところにリストされています。

表 26. SQL が報告する SQLSTATE メッセージ

SQLSTATE	説明	解説
01000	警告。	通知メッセージ。(関数は、SQL_SUCCESS_WITH_INFO を戻す。)
01004	値が切り捨てられた。	値が、システムのキャスト関数または調整関数によって切り捨てられた。

表 26. SQL が報告する SQLSTATE メッセージ (続き)

SQLSTATE	説明	解説
01550	索引は作成されなかった。	指定した記述をもつ索引がすでに存在するため、この索引は作成されなかった。
02000	行が検出されなかった。	FETCH、DELETE、または UPDATE ステートメントの実行で行が検出されなかった。
07001	パラメーター数の間違い。	パラメーター・マーカーがバインドされていない。
07005	パラメーターが無効。	カーソルのステートメント名が、カーソルに関連付けることができない準備済みステートメントを示している。
07006	変数が無効。	データ・タイプが原因で、入力ホスト変数が使用できない。
08002	接続がすでに存在している。	接続がすでに存在している。
22001	値に切り捨てが必要。	システムのキャスト関数または調整関数による値の切り捨てが必要。
22002	ヌル標識が提供されない。	ストレージが提供されていないため、NULL 値を割り当てることができない。
22003	数値が範囲外。	数値がターゲット列の範囲内にない。
22007	無効な日付時刻形式。	日付時刻値に関するストリング表現の構文が正しくない。
22008	日付時刻値が範囲外。	日付時刻値に関するストリング表現が範囲外である。
22012	0 による除算。	0 による除算が試みられた。
22504	MBCS 文字がフラグメント化されている。	データに形式が適切ではないマルチバイト文字が含まれている。
23502	ヌル値が許されない。	NOT NULL 列に NULL 値を割り当てることはできない。
23505	値が固有ではない。	重複キーを生成することになるため、この操作は無効。
23513	値が無効。	INSERT または UPDATE ステートメントの結果、行がチェック制約の定義に合わない。
23515	複数の主キー文節が指定された。	複数の主キー文節が指定された。
24000	カーソル状態が無効。	*StatementHandle* は実行された状態であったが、 *StatementHandle* に関連付けられた結果セットが無かった。
24501	カーソルがオープンされていない。	結果セットが生成されていないため、FETCH は無効。
24505	カーソルが位置付けされていない。	カーソルが行上に位置付けされていないため、FETCH は無効。
34000	カーソル名が無効。	カーソル名が無効。
42501*	識別されたオブジェクトで指定のオペレーションを実行することが許可されていない許可 ID。	現行ユーザーは、存在しないユーザーから特権を除去しようとしている。
42502*	指定されているオペレーションを実行することが許可されていない許可 ID。	現行ユーザーは、認証された接続を所有していない。アプリケーション (暗号化ライブラリーや CryptoPlugin.dll がないアプリケーション) が暗号化関連の SQL コマンド (GRANT、REVOKE および CREATE TABLE) を実行すると、エラー「42502」が戻される。これにより、アプリケーションの破損が回避される。
42505*	接続許可が失敗した。	登録されたユーザーが接続しようとしたが、認証されなかった。
42506*	所有者許可が失敗した。	接続されたユーザーが認証されなかった。(パスワードの誤り。)
42601	構文エラー。	SQL ステートメントの構文エラーが検出された。

表 26. SQL が報告する SQLSTATE メッセージ (続き)

SQLSTATE	説明	解説
42603	ストリング定数に終了区切り文字がない。	ストリング定数または区切り ID に終了区切り文字がない。
42610	パラメーター・マーカーの無効な使用。	ステートメントに無効なパラメーター・マーカーが含まれている。パラメーター・マーカーの有効な使用法については、 84 ページの表 14 を参照してください。
42611	無効な長さ指定。	長さ指定が限界を超えている。
42614	重複キーワードが無効。	重複キーワードが無効。
42621	チェック制約が無効。	チェック制約が無効。
42622	名前が長すぎる。	ID の名前が長すぎる。
42702	未確定列名参照。	複数の列が参照されている可能性がある。
42703	未定義の列名。	列名が参照されている表にない。
42704	未定義のオブジェクト。	表が存在しない。
42710	指定されたオブジェクトがすでに存在している。	同じ名前の表がすでに存在している。
42711	列名が重複している。	同じ列名が複数回指定されている。
42802	値の数が列の数に一致しない。	割り当てられる値の数が、指定または暗黙指定された列の数と等しくない。
42803	SELECT リスト内の列参照が GROUP BY 文節に指定されていない。	列名および集約関数が選択リスト内に含まれているが、GROUP BY 文節がない。
42818	オペランドの非互換データ・タイプ。	演算のオペランドのデータ・タイプに互換性がない。
42820	リテラル値が範囲外。	指定された数値が許容範囲外である。
42821	非互換データ・タイプ。	値に、ターゲット列のデータ・タイプとの互換性がない。
42822	無効な ORDER BY 項目。	ORDER BY 項目が選択リストにない。
42824	LIKE オペランドが無効。	LIKE のオペランドがストリングではないか、第 1 オペランドが列ではない。
42829	FOR UPDATE OF が無効。	カーソルが指定する結果表を変更できないため、FOR UPDATE OF は無効。
42830	外部キーが親キーの記述に合っていない。	外部キーが親キーの記述に合っていない。
42831	主キーにヌル可能列がある。	主キー文節に指定する列は、ヌル可能にはできない。
42832*	システム・オブジェクトへの無許可アクセス。	システム・オブジェクトに関する操作が許可されていない。
42884	不明な関数名。	指定された名前と互換引き数を持つ、関数もプロシージャーも見つからなかった。
42887	サポートされないフィーチャー	このフィーチャーは現行リリースではサポートされていない。
42894	デフォルト値が無効。	デフォルト値が無効。
42902	オブジェクト表参照が重複している。	INSERT ステートメントのオブジェクト表が、FROM 文節でも指定されている。
42903	WHERE 文節または SET 文節に無効な参照が含まれている。	WHERE 文節または SET 文節に、列関数のような無効な参照が含まれている。

表 26. SQL が報告する SQLSTATE メッセージ (続き)

SQLSTATE	説明	解説
42962	LOB 列はキーとして使用できない。	LOB 列は主キーとして使用できない。
54001	ステートメントが長すぎる。	照会ステートメントが長すぎる。
54008	キーが長すぎる。	主キー、外部キー、または索引内の列数が多すぎる。
54010	表レコード長が長すぎる。	表のレコード長が長すぎる。
55002	DB2ePLANTABLE の定義が正しくない。	DB2ePLANTABLE の宣言が正しくないため、EXPLAIN を実行できない。
55009	ファイルが読み取り専用。	ファイルが読み取り専用。読み取り専用の環境では、SELECT 照会しか実行できない。
57001	表が使用可能になっていない。	トランザクション有効範囲内の表で REORG を実行できない。
57011	メモリー不足。	システムが動的メモリーを割り当てることができない。
57014	割り込みのために処理が取り消された。	照会の実行が、ユーザーの割り込みのために取り消された。
58004	内部システム・エラー (継続)。	重大ではないシステム・エラーが発生した。
58005	内部システム・エラー (停止)。	重大なシステム・エラーが発生した。

関連した解説:

- 147 ページの『DB2 Everyplace SQL ステートメント・サポートの概要』
- 185 ページの『割り当てと比較に関するデータ・タイプの互換性』
- 189 ページの『SQLState のリスト』
- 84 ページの『DB2 Everyplace でサポートされているパラメーター・マーカー』
- 189 ページの『SQLState クラス・コードのサマリー』

CLI が報告する SQLState メッセージ

表 27. CLI が報告する SQLState メッセージ

SQLSTATE	CLI 関数名	説明	解説
01000	SQLAllocHandle	警告。	通知メッセージ。(関数は、SQL_SUCCESS_WITH_INFO を戻す。)
01000	SQLFreeHandle	警告。	通知メッセージ。(関数は、SQL_SUCCESS_WITH_INFO を戻す。)
01002	SQLDisconnect	切断エラー。	切断中にエラーが発生した。しかし、切断は成功した。(関数は、SQL_SUCCESS_WITH_INFO を戻す。)
01004	SQLDescribeCol	データが切り捨てられた。	引き数 *ColumnName* に戻された列名が、引き数 *BufferLength* に指定した値より長かった。引き数 *NameLengthPtr* に、列名全体の長さが入っている。(関数は、SQL_SUCCESS_WITH_INFO を戻す。)
01004	SQLFetch	データが切り捨てられた。	1 つ以上の列に関して戻されたデータが切り捨てられた。ストリング値または数値は右側が切り捨てられる。(エラーがない場合には、SQL_SUCCESS_WITH_INFO が戻される。)

表 27. CLI が報告する SQLState メッセージ (続き)

SQLSTATE	CLI 関数名	説明	解説
01004	SQLGetData	データが切り捨てられた。	指定した列 (*ColumnNumber*) に対して戻されたデータが切り捨てられた。ストリング値または数値は右側が切り捨てられる。 (SQL_SUCCESS_WITH_INFO が戻される。)
01S06*	SQLFetchScroll	結果セットが最初の行セットを戻す前に、取り出そうとした。	現在位置が最初の行より後であり、かつ、*FetchOrientation* が SQL_PRIOR であるか、または *FetchOrientation* が SQL_RELATIVE であって、絶対値が現行の SQL_ATTR_ROW_ARRAY_SIZE 以下である負の *FetchOffset* が指定されている際に、要求された行セットが結果セットの先頭とオーバーラップした。(関数は、SQL_SUCCESS_WITH_INFO を戻す。)
07005	SQLDescribeCol	ステートメントが結果セットを戻さなかった。	*StatementHandle* に関連するステートメントが結果セットを戻さなかった。記述する列がなかった。(最初に `SQLNumResultCols()` を呼び出して、結果セット内に行があるかどうかを判別する。)
07006	SQLBindParameter	無効な変換。	*ValueType* 引き数に指定したデータ・タイプから *ParameterType* 引き数に指定したデータ・タイプへの変換が無意味な変換である。(例えば、SQL_C_DATE から SQL_DOUBLE への変換。)
07006	SQLFetch	無効な変換。	`SQLBindCol()` の *fCType* に指定したデータ・タイプへの、データ・タイプ変換が無意味である。
07006	SQLGetData	無効な変換。	データ・タイプは、引き数 *TargetType* に指定した C データ・タイプに変換できない。この関数は以前、同じ *ColumnNumber* 値に関して、異なる *TargetType* 値を使用して呼び出されていた。
07009	SQLBindCol	無効な記述子索引。	引き数 *ColumnNumber* に指定した値が、結果セット内の最大列数を超えている。
07009	SQLDescribeCol	無効な記述子索引。	*ColumnNumber* に指定した値が 0 以下である。引き数 *ColumnNumber* に指定した値が、結果セット内の列数より大きかった。
08001	SQLConnect	データ・ソースに接続できない。	DB2 CLI がデータ・ソース (サーバー) との接続を確立できなかった。
08002	SQLConnect	接続が使用中。	指定した *ConnectionHandle* が、データ・ソースとの接続を確立するためにすでに使用されており、その接続がまだオープンされている。
08003	SQLAllocHandle	接続がクローズしている。	*HandleType* 引き数が SQL_HANDLE_STMT であったが、 *InputHandle* 引き数により指定された接続がオープンされていなかった。 DB2 CLI がステートメント・ハンドルを割り当てるには、接続処理を正常に完了する (そしてその接続をオープンする) 必要がある。
08003	SQLDisconnect	接続がクローズしている。	引き数 *ConnectionHandle* に指定した接続がオープンしていない。

表 27. CLI が報告する SQLState メッセージ (続き)

SQLSTATE	CLI 関数名	説明	解説
08004	SQLConnect	アプリケーション・サーバーが接続の確立をリジェクトした。	データ・ソース (サーバー) が接続の確立をリジェクトした。
08S01	SQLFreeHandle	通信リンク障害。	*HandleType* 引き数が SQL_HANDLE_DBC であり、関数の処理が完了する前に、 DB2 CLI と接続を試みた先のデータ・ソースとの間の通信リンクに、障害が起きた。
22002	SQLFetch	無効な出力バッファーまたは標識バッファーの指定。	SQLBindCol() 内の引き数 *pcbValue* に指定したポインター値がヌル・ポインターで、対応する列の値がヌルである。 SQL_NULL_DATA を報告する方法がない。
22002	SQLGetData	無効な出力バッファーまたは標識バッファーの指定。	引き数 *StrLen_or_IndPtr* に指定したポインター値がヌル・ポインターで、列の値がヌルである。 SQL_NULL_DATA を報告する方法がない。
22003	SQLExecDirect	数値が範囲外。	数値タイプ列に対する数値割り当てによって、割り当て時または中間結果の計算中に、数値の整数部分の切り捨てが行われた。
22005	SQLGetData	割り当てのエラー。	戻り値が、引き数 *TargetType* に指定されるデータ・タイプと互換性がなかった。
39001 *	SQLExecute	ユーザー定義関数が無効な SQLSTATE を戻した。	ユーザー定義関数が無効な SQLSTATE を戻した。
40003 **08**S01	SQLBindCol	通信リンク・エラー。	関数の完了前に、アプリケーションとデータ・ソース間の通信リンクに障害が生じた。
40003 **08**S01	SQLBindParameter	通信リンク障害。	関数の完了前に、アプリケーションとデータ・ソース間の通信リンクに障害が生じた。
40003 **08**S01	SQLDescribeCol	通信リンク障害。	関数の完了前に、アプリケーションとデータ・ソース間の通信リンクに障害が生じた。
40003 **08**S01	SQLFreeStmt	通信リンク障害。	関数の完了前に、アプリケーションとデータ・ソース間の通信リンクに障害が生じた。
40003 **08**S01	SQLGetData	通信リンク障害。	関数の完了前に、アプリケーションとデータ・ソース間の通信リンクに障害が生じた。
40003 **08**S01	SQLNumResultCols	通信リンク障害。	関数の完了前に、アプリケーションとデータ・ソース間の通信リンクに障害が生じた。
40003 **08**S01	SQLRowCount	通信リンク障害。	関数の完了前に、アプリケーションとデータ・ソース間の通信リンクに障害が生じた。
42nnn*	SQLPrepare	構文エラー。	**42**nnn SQLSTATES は、ステートメントに関するさまざまな構文またはアクセスの問題を示す。文字 nnn は、そのクラス・コードを持ついずれかの SQLSTATE を示す。例えば、**42**nnn は、**42** クラスのいずれかの SQLSTATE を示す。

表 27. CLI が報告する SQLState メッセージ (続き)

SQLSTATE	CLI 関数名	説明	解説
42xxx	SQLExecDirect	構文エラーまたはアクセス規則違反。	**42**xxx SQLSTATES は、ステートメントに関するさまざまな構文またはアクセスの問題を示す。xxx は該当するクラス・コードを持ついずれかの SQLSTATE を示す。例えば、**42**xxx は、**42** クラスのいずれかの SQLSTATE を示す。
42xxx	SQLNumResultCols	構文エラー。	**42**xxx SQLSTATES は、ステートメントに関するさまざまな構文またはアクセスの問題を示す。xxx は該当するクラス・コードを持ついずれかの SQLSTATE を示す。例えば、**42**xxx は、**42** クラスのいずれかの SQLSTATE を示す。
58004	SQLBindCol	予期しないシステム障害。	回復不能なシステム・エラー。
58004	SQLBindParameter	予期しないシステム障害。	回復不能なシステム・エラー。
58004	SQLConnect	予期しないシステム障害。	回復不能なシステム・エラー。
58004	SQLDescribeCol	予期しないシステム障害。	回復不能なシステム・エラー。
58004	SQLDisconnect	予期しないシステム障害。	回復不能なシステム・エラー。
58004	SQLExecDirect	予期しないシステム障害。	回復不能なシステム・エラー。
58004	SQLFetch	予期しないシステム障害。	回復不能なシステム・エラー。
58004	SQLFreeStmt	予期しないシステム障害。	回復不能なシステム・エラー。
58004	SQLGetData	予期しないシステム障害。	回復不能なシステム・エラー。
58004	SQLPrepare	予期しないシステム障害。	回復不能なシステム・エラー。
58004	SQLNumResultCols	予期しないシステム障害。	回復不能なシステム・エラー。
58004	SQLRowCount	予期しないシステム障害。	回復不能なシステム・エラー。
59101*	SQLExecute	ユーザーが定義されていない。	ユーザーがモバイル・デバイス管理センター制御データベースに定義されていない。
59102*	SQLExecute	パスワードが正しくない。	ユーザー・パスワードがモバイル・デバイス管理センターで定義されているパスワードと一致しない。
59103*	SQLExecute	グループが定義されていない。	グループがモバイル・デバイス管理センターに定義されていない。
59104*	SQLExecute	アプリケーションが定義されていない。	アプリケーションがモバイル・デバイス管理センターに定義されていない。
59105*	SQLExecute	サブスクリプションが定義されていない。	AgentAdapter を使用するサブスクリプションがモバイル・デバイス管理センターに定義されていない。

表 27. CLI が報告する SQLState メッセージ (続き)

SQLSTATE	CLI 関数名	説明	解説
59106*	SQLExecute	サブスクリプションが完了していない。	サブスクリプションが、リモート・ストアード・プロシージャーを起動するために必要なすべての情報を保有していない。
59120*	SQLExecute	XML 変換エラー。	ユーザー入力データを XML 文書に変換するときに、AgentAdapter が失敗した。
59121*	SQLExecute	AgentAdapter の一般エラー。	AgentAdapter の一般エラー。
59122*	SQLExecute	ライブラリーのロードが失敗した。	必須ライブラリーのいくつかが、システムで見つからない。
HY000	SQLAllocHandle	一般エラー。	エラーは発生したが、特定の SQLSTATE は無い。`SQLGetDiagRec()` が **MessageText* バッファーに戻したエラー・メッセージが、エラーとその原因を示しています。
HY000	SQLFreeHandle	一般エラー。	エラーは発生したが、特定の SQLSTATE は無い。`SQLGetDiagRec()` が **MessageText* バッファーに戻したエラー・メッセージが、エラーとその原因を示しています。
HY001	SQLAllocHandle	メモリー割り振りエラー。	DB2 CLI が、指定されたハンドル用のメモリーを割り当てることができない。
HY001	SQLBindCol	メモリーの割り振りの失敗。	DB2 CLI が、関数を実行または完了させるのに必要なメモリーを割り当てることができない。
HY001	SQLBindParameter	メモリーの割り振りの失敗。	DB2 CLI が、関数を実行または完了させるのに必要なメモリーを割り当てることができない。
HY001	SQLConnect	メモリーの割り振りの失敗。	DB2 CLI が、関数を実行または完了させるのに必要なメモリーを割り当てることができない。
HY001	SQLDescribeCol	メモリーの割り振りの失敗。	DB2 CLI が、関数を実行または完了させるのに必要なメモリーを割り当てることができない。
HY001	SQLDisconnect	メモリーの割り振りの失敗。	DB2 CLI が、関数を実行または完了させるのに必要なメモリーを割り当てることができない。
HY001	SQLExecDirect	メモリーの割り振りの失敗。	DB2 CLI が、関数を実行または完了させるのに必要なメモリーを割り当てることができない。
HY001	SQLFetch	メモリーの割り振りの失敗。	DB2 CLI が、関数を実行または完了させるのに必要なメモリーを割り当てることができない。
HY001	SQLFreeHandle	メモリーの割り振りの失敗。	DB2 CLI が、関数を実行または完了させるのに必要なメモリーを割り当てることができない。
HY001	SQLFreeStmt	メモリーの割り振りの失敗。	DB2 CLI が、関数を実行または完了させるのに必要なメモリーを割り当てることができない。
HY001	SQLGetData	メモリーの割り振りの失敗。	DB2 CLI が、関数を実行または完了させるのに必要なメモリーを割り当てることができない。
HY001	SQLPrepare	メモリーの割り振りの失敗。	DB2 CLI が、関数を実行または完了させるのに必要なメモリーを割り当てることができない。
HY001	SQLNumResultCols	メモリーの割り振りの失敗。	DB2 CLI が、関数を実行または完了させるのに必要なメモリーを割り当てることができない。

表 27. CLI が報告する SQLState メッセージ (続き)

SQLSTATE	CLI 関数名	説明	解説
HY001	SQLRowCount	メモリーの割り振りの失敗。	DB2 CLI が、関数を実行または完了させるのに必要なメモリーを割り当てることができない。
HY002	SQLBindCol	無効な列番号。	引き数 *ColumnNumber* に指定した値が 0 より小さい。引き数 *ColumnNumber* に指定した値が、データ・ソースのサポートしている最大列数を超えていた。
HY002	SQLDescribeCol	無効な列番号。	引き数 *ColumnNumber* に指定した値が 1 より小さい。引き数 *ColumnNumber* に指定した値が、結果セット内の列数より大きい。
HY002	SQLGetData	無効な列番号。	指定した列が 0 より小さいか、結果の列数より大きい。
HY003	SQLBindCol	プログラム・タイプが範囲外。	*TargetType* が有効なデータ・タイプまたは SQL_C_DEFAULT ではない。
HY003	SQLBindParameter	プログラム・タイプが範囲外。	引き数 *ParameterNumber* によって指定した値が、有効なデータ・タイプまたは SQL_C_DEFAULT ではない。
HY003	SQLGetData	プログラム・タイプが範囲外。	*TargetType* が有効なデータ・タイプまたは SQL_C_DEFAULT ではない。
HY004	SQLBindParameter	SQL データ・タイプが範囲外。	引き数 *ParameterType* に指定した値が、無効な SQL データ・タイプである。
HY009	SQLBindParameter	無効な引き数値。	引き数 *ParameterValuePtr* と *StrLen_or_IndPtr* がヌル・ポインターであり、*InputOutputType* が SQL_PARAM_OUTPUT ではない。
HY009	SQLExecDirect	無効な引き数値。	*StatementText* がヌル・ポインターである。
HY009	SQLNumResultCols	無効な引き数値。	*StatementText* がヌル・ポインターである。
HY010	SQLDescribeCol	関数のシーケンス・エラー。	この関数は、*StatementHandle* に対する `SQLPrepare()` または `SQLExecDirect()` の呼び出しの前に呼び出される。
HY010	SQLExecute	関数のシーケンス・エラー。	指定した *StatementHandle* が準備済み状態ではない。最初に `SQLPrepare()` を呼び出さないで、`SQLExecute()` を呼び出した。
HY010	SQLFetch	関数のシーケンス・エラー。	この関数は、*StatementHandle* に対する `SQLPrepare()` または `SQLExecDirect()` の呼び出しの前に呼び出される。

表 27. CLI が報告する SQLState メッセージ (続き)

SQLSTATE	CLI 関数名	説明	解説
HY010	SQLFreeHandle	関数のシーケンス・エラー。	*HandleType* 引き数が SQL_HANDLE_ENV であり、少なくとも 1 つの接続が割り当て状態または接続状態である。*HandleType* に SQL_HANDLE_ENV を指定して SQLFreeHandle() を呼び出す前に、接続ごとに、*HandleType* に SQL_HANDLE_DBC を指定して SQLDisconnect() と SQLFreeHandle() を呼び出す必要がある。 *HandleType* 引き数が SQL_HANDLE_DBC であり、接続に対して SQLDisconnect() を呼び出す前に、この関数が呼び出される。 *HandleType* 引き数が SQL_HANDLE_STMT であり、SQLExecute() または SQLExecDirect() がステートメント・ハンドルを使用して呼び出され、SQL_NEED_DATA が戻された。(DM) SQLFreeHandle() の呼び出し前に、付随するすべてのハンドルおよび他のリソースが解放されなかった。
HY010	SQLGetData	関数のシーケンス・エラー。	最初に SQLFetch() を呼び出さないで、この関数を呼び出した。
HY010	SQLNumResultCols	関数のシーケンス・エラー。	この関数は、*StatementHandle* に対する SQLPrepare() または SQLExecDirect() の呼び出しの前に呼び出される。
HY010	SQLRowCount	関数のシーケンス・エラー。	この関数は、*StatementHandle* に対する SQLExecute() または SQLExecDirect() の呼び出しの前に呼び出される。
HY013	SQLAllocHandle	予期しないメモリー処理エラー。	*HandleType* 引き数が SQL_HANDLE_DBC または SQL_HANDLE_STMT であったが、この関数呼び出しを処理できない。この理由としては、低メモリー状態のために、基礎となるメモリー・オブジェクトにアクセスできないものと思われる。
HY013	SQLBindCol	予期しないメモリー処理エラー。	DB2 CLI が、関数を実行または完了させるのに必要なメモリーにアクセスできない。
HY013	SQLBindParameter	予期しないメモリー処理エラー。	DB2 CLI が、関数を実行または完了させるのに必要なメモリーにアクセスできない。
HY013	SQLConnect	予期しないメモリー処理エラー。	DB2 CLI が、関数を実行または完了させるのに必要なメモリーにアクセスできない。
HY013	SQLDescribeCol	予期しないメモリー処理エラー。	DB2 CLI が、関数を実行または完了させるのに必要なメモリーにアクセスできない。
HY013	SQLDisconnect	予期しないメモリー処理エラー。	DB2 CLI が、関数を実行または完了させるのに必要なメモリーにアクセスできない。
HY013	SQLExecDirect	予期しないメモリー処理エラー。	DB2 CLI が、関数を実行または完了させるのに必要なメモリーにアクセスできない。
HY013	SQLFetch	予期しないメモリー処理エラー。	DB2 CLI が、関数を実行または完了させるのに必要なメモリーにアクセスできない。

表 27. CLI が報告する SQLState メッセージ (続き)

SQLSTATE	CLI 関数名	説明	解説
HY013	SQLFreeHandle	予期しないメモリー処理エラー。	*HandleType* 引き数が SQL_HANDLE_STMT であり、関数呼び出しを処理できない。低メモリー状態のために、基礎となるメモリー・オブジェクトにアクセスできないためと思われる。
HY013	SQLGetData	予期しないメモリー処理エラー。	DB2 CLI が、関数を実行または完了させるのに必要なメモリーにアクセスできない。
HY013	SQLNumResultCols	予期しないメモリー処理エラー。	DB2 CLI が、関数を実行または完了させるのに必要なメモリーにアクセスできない。
HY013	SQLRowCount	予期しないメモリー処理エラー。	DB2 CLI が、関数を実行または完了させるのに必要なメモリーにアクセスできない。
HY014	SQLAllocHandle	ハンドル数の限界に到達済み。	*HandleType* 引き数によって示されたハンドルのタイプに割り当てることができるハンドル数の限界に到達した。
HY014	SQLExecDirect	ハンドル数の限界に到達済み。	内部リソースが原因で、DB2 CLI がハンドルを割り当てることができない。
HY014	SQLNumResultCols	ハンドル数の限界に到達済み。	内部リソースが原因で、DB2 CLI がハンドルを割り当てることができない。
HY017	SQLFreeHandle	自動的に割り当てられた記述子ハンドルの無効な使用。	*Handle* 引き数が、自動的に割り振られた記述子またはインプリメンテーション記述子用のハンドルに設定されている。
HY024	SQLSetStmtAttr	無効な属性値。	指定された属性 値では、*ValuePtr* に無効値が指定されている。
HY090	SQLBindCol	無効なストリング長またはバッファー長。	引き数 *BufferLength* に指定した値が 1 より小さく、引き数 *TargetType* が SQL_C_CHAR、SQL_C_BINARY または SQL_C_DEFAULT のいずれかである。
HY090	SQLBindParameter	無効なストリング長またはバッファー長。	引き数 *BufferLength* に指定した値が 0 より小さい。
HY090	SQLDescribeCol	無効なストリング長またはバッファー長。	引き数 *BufferLength* に指定した長さが 1 より小さい。
HY090	SQLExecDirect	無効なストリング長またはバッファー長。	引き数 *TextLength* が 1 より小さいが、SQL_NTS と等しくない。
HY090	SQLGetData	無効なストリング長またはバッファー長。	引き数 *BufferLength* の値が 0 より小さく、引き数 *TargetType* が SQL_C_CHAR または SQL_C_BINARY である。あるいは *TargetType* が SQL_C_DEFAULT で、デフォルト・タイプが SQL_C_CHAR、SQL_C_BINARY、または SQL_C_DBCHAR のいずれかである。
HY090	SQLNumResultCols	無効なストリング長またはバッファー長。	引き数 *TextLength* が 1 より小さいが、SQL_NTS と等しくない。
HY092	SQLAllocHandle	オプション・タイプが範囲外。	*HandleType* 引き数が以下のいずれでもない。 SQL_HANDLE_ENV SQL_HANDLE_DBC SQL_HANDLE_STMT

表 27. CLI が報告する SQLState メッセージ (続き)

SQLSTATE	CLI 関数名	説明	解説
HY092	SQLFreeStmt	オプション・タイプが範囲外。	引き数 *Option* に指定した値が SQL_DROP または SQL_RESET_PARAMS ではない。
HY093	SQLBindParameter	無効なパラメーター番号。	引き数 *ValueType* に指定した値が 1 より小さいか、またはサーバーがサポートするパラメーターの最大数より大きい。
HY094	SQLBindParameter	無効な位取り値。	*ParameterType* に指定した値が SQL_DECIMAL または SQL_NUMERIC であり、 *DecimalDigits* に指定した値が 0 より小さいか、引き数 *ParamDef* (精度) の値より大きい。
HY104	SQLBindParameter	無効な精度値。	*ParameterType* に指定した値が SQL_DECIMAL または SQL_NUMERIC のいずれかであり、*ParamDef* に指定した値が 1 より小さい。
HY105	SQLBindParameter	無効なパラメーター・タイプ。	*InputOutputType* が SQL_PARAM_INPUT ではない。
HY106	SQLFetchScroll	取り出しタイプが範囲外。	引き数 *FetchOrientation* に指定した値が、無効である。 SQL_CURSOR_TYPE ステートメント属性の値が SQL_CURSOR_FORWARD_ONLY であるが、*FetchOrientation* 引き数の値が SQL_FETCH_NEXT ではない。
HY107	SQLFetchScroll	行の値が範囲外。	SQL_ATTR_CURSOR_TYPE ステートメント属性で指定した値は SQL_CURSOR_KEYSET_DRIVEN であるが、SQL_ATTR_KEYSET_SIZE ステートメント属性で指定した値がゼロより大きく、SQL_ATTR_ROW_ARRAY_SIZE ステートメント属性で指定した値より小さい。
HY501	SQLConnect	*DataSource* 名が無効。	指定した *DataSource* 名が無効である。
HYC00	SQLBindCol	ドライバーが使用不可である。	引き数 *TargetType* に指定したデータ・タイプは、DB2 CLI が認識するが、サポートはされない。
HYC00	SQLBindParameter	ドライバーが使用不可である。	DB2 CLI またはデータ・ソースが、引き数 *ValueType* に指定した値と引き数 *ParameterType* に指定した値との組み合わせによって指定された変換をサポートしていない。引き数 *ParameterType* に指定した値が、DB2 CLI またはデータ・ソースではサポートされていない。
HYC00	SQLDescribeCol	ドライバーが使用不可である。	列 *ColumnNumber* の SQL データ・タイプを DB2 CLI が認識しない。
HYC00	SQLGetData	ドライバーが使用不可である。	指定したデータ・タイプの SQL データ・タイプが認識されたが、DB2 CLI によってサポートされていない。要求された、SQL データ・タイプからアプリケーション・データ *TargetType* への変換は、DB2 CLI またはデータ・ソースで実行できない。

関連した解説:

- 147 ページの『DB2 Everyplace SQL ステートメント・サポートの概要』
- 185 ページの『割り当てと比較に関するデータ・タイプの互換性』

- 189 ページの『SQLState のリスト』
- 84 ページの『DB2 Everyplace でサポートされているパラメーター・マーカー』
- 189 ページの『SQLState クラス・コードのサマリー』

JDBC により報告される SQLState メッセージ

表 28. JDBC により報告される SQLState メッセージ

SQLSTATE	説明	解説
0100C	1 つ以上の adhoc 結果セットが戻された。	DB2 Everyplace は、*ResultSet* オブジェクトの並行性モードのための ResultSet.CONCUR_UPDATABLE をサポートしていない。代わりに、ResultSet.CONCUR_READ_ONLY が使用される。
0641E	バッチ内に SELECT ステートメントがある。	SELECT ステートメントはバッチ内では使用できない。
0643E	バッチ内にステートメントがまったくない。	このバッチにはステートメントがまったく入っていない。
22005	割り当てのエラー。	パラメーターのタイプとターゲットのデータ・タイプに互換性がない。
22011	サブストリング・エラーが発生した。	抽出すべき BLOB 値の第 1 バイトに対して無効な順序位置である。
S1010	関数のシーケンス・エラー。	`registerOutParameter` が先に呼び出されずに、`CallableStatement` get メソッドが呼び出された。

関連した解説:

- 296 ページの『DB2 Everyplace JDBC サポートの概要』

サポートされている DB2 CLI 関数

この章では、DB2 Everyplace がサポートする DB2 コール・レベル・インターフェース (DB2 CLI) の関数について説明します。

- 『DB2 CLI 関数の要約』では、各関数の目的について簡単に説明するとともに、DB2 Everyplace によってサポートされる DB2 CLI 関数と標準の DB2 CLI 関数との相違点について簡単に要約します。
- 206 ページの『DB2 CLI 関数の説明の要点』では、各 CLI 関数の機能説明を解説します。
- 295 ページの『DB2 CLI 関数によるデータ変換』には、C と SQL の双方のデータ・タイプ間でサポートされるデータ変換を示す表があります。

DB2 CLI 関数の要約

203 ページの表 29 では、DB2 Everyplace によってサポートされる DB2 CLI 関数が要約してあります。要約の中で、各関数の目的のほか DB2 Everyplace によってサポートされる DB2 CLI 関数と標準の DB2 CLI 関数の相違点を示しています。

表 29. DB2 CLI 関数リスト

関数名	目的	相違点の要約
SQLAllocConnect	接続ハンドルを入手する。	
SQLAllocEnv	環境ハンドルを入手する。	
SQLAllocHandle	ハンドルを入手する。	
SQLAllocStmt	ステートメント・ハンドルを割り当てる。	
SQLBindCol	結果列用のストレージを割り当て、データ・タイプを指定する。	ターゲット・タイプは、サポートされるデータ・タイプに制限される。 LOB ロケーターをサポートしない。
SQLBindParameter	SQL ステートメントのパラメーター用のストレージを割り当てる。	アプリケーション変数または LOB ロケーターの配列に対するバインディングをサポートしない。SQLPutData() をサポートしないので、アプリケーションはパラメーターの値を *ParameterValuePtr* に入れてから、SQLExecute() を呼び出す必要がある。ストアード・プロシージャーがサポートされていないので、パラメーター・タイプは INPUT のみに制限される。
SQLColumns	指定された表に列名のリストを戻す。	*CatalogName*、*NameLength1*、*SchemaName*、*NameLength2* は無視される。戻された結果セットの列 2、12、および 15 は常に NULL である。戻りコード SQL_STILL_EXECUTING をサポートしない。
SQLConnect	データ・ソース名、ユーザー ID、およびパスワードによって特定のドライバーに接続する。	
SQLDescribeCol	結果セット内の列について記述する。	列情報はサポートされる列データ・タイプに限定される。
SQLDisconnect	接続をクローズする。	
SQLEndTran	接続に関連するすべてのステートメントのすべての操作に対し COMMIT または ROLLBACK を要求する。	SQLEndTran() を呼び出す前に、接続の属性 SQL_ATTR_AUTOCOMMIT を SQL_AUTOCOMMIT_OFF にセットしなければならない。
SQLError	追加のエラーまたは状況情報を戻す。	

表 29. DB2 CLI 関数リスト (続き)

関数名	目的	相違点の要約
SQLExecDirect	ステートメントを実行する。	戻りコードの SQL_STILL_EXECUTING と SQL_NEED_DATA をサポートしない。非同期 CLI 呼び出しをサポートしない。
SQLExecute	準備済みステートメントを実行する。	SQLExecute() を呼び出す前に、すべてのパラメーターをバインドする必要がある。SQL 呼び出しの非同期実行をサポートしない。
SQLFetch	結果行を戻す。	結果は、行セットごとではなく、一度に 1 行ずつ取り出される。ステートメント記述子をサポートしない。戻りコード SQL_STILL_EXECUTING をサポートしない。
SQLFetchScroll	結果行セットを戻す。	結果は行セットで取り出される。戻りコード SQL_STILL_EXECUTING をサポートしない。
SQLForeignKeys	指定された表の外部キーに関する情報を戻す。	*PKCatalogName*、*NameLength1*、*PKSchemaName*、*NameLength2*、*FKCatalogName*、*NameLength4*、*FKSchemaName*、*NameLength5* は無視されます。戻された結果セットの列 1、2、5、6、12、および 13 は、常に長さがゼロのストリングです。戻された結果セットの列 10、11、および 14 は常にゼロ。戻りコード SQL_STILL_EXECUTING をサポートしない。
SQLFreeConnect	接続ハンドルを解放する。	
SQLFreeEnv	環境ハンドルを解放する。	
SQLFreeHandle	ハンドル・リソースを解放する。	
SQLFreeStmt	ステートメント処理を終了し、保留中の結果を廃棄し、オプショナルで、ステートメント・ハンドルに関連したすべてのリソースを解放する。	SQL_DROP および SQL_RESET_PARAMS オプションのみをサポートする。

表 29. DB2 CLI 関数リスト (続き)

関数名	目的	相違点の要約
SQLGetConnectAttr	接続属性の現行の設定を戻す。	DB2 Everyplace は、DB2 でサポートされている接続属性のサブセットをサポートする。また DB2 Everyplace は、DB2 でサポートされていない接続属性もいくつかサポートしている。
SQLGetCursorName	ステートメント・ハンドルと関連したカーソル名を戻す。	内部で生成されたカーソル名は常に CUR で始まる。
SQLGetData	結果セットの 1 行の 1 列の一部または全部を戻す。	ターゲット・タイプは、サポートされるデータ・タイプに制限される。 LOB ロケーターをサポートしない。戻りコード SQL_STILL_EXECUTING をサポートしない。
SQLGetDiagRec	診断データの複数フィールドを入手する。	ステートメント・ハンドルまたは接続ハンドルに関連付けられた診断レコードのみをサポートする。単一診断レコードのみをサポートする。
SQLGetInfo	指定されたドライバーとデータ・ソースについての情報を戻す。	DB2 Everyplace は、DB2 でサポートされている情報タイプのサブセットをサポートする。
SQLGetStmtAttr	ステートメント属性の現在の設定を戻す。	DB2 Everyplace は、DB2 でサポートされているステートメント属性のサブセットをサポートする。また DB2 Everyplace は、DB2 でサポートされていないステートメント属性もいくつかサポートしている。
SQLNumParams	SQL ステートメント内のパラメーター・マーカーの数を戻す。	戻りコード SQL_STILL_EXECUTING をサポートしない。
SQLNumResultCols	結果セット内の列数を戻す。	
SQLPrepare	後で実行するために SQL ステートメントを準備する。	
SQLPrimaryKeys	表の主キーが入っている列名のリストを戻す。	*CatalogName*、*NameLength1*、*SchemaName*、*NameLength2* は無視される。戻された結果セットの列 1、2、6 は、常に長さがゼロのストリングになる。戻りコード SQL_STILL_EXECUTING をサポートしない。

表 29. DB2 CLI 関数リスト (続き)

関数名	目的	相違点の要約
SQLRowCount	挿入、更新、または削除の各要求による影響を受ける行数を戻す。	
SQLSetConnectAttr	接続に関するオプションを設定する。	DB2 Everyplace は、DB2 でサポートされている接続属性のサブセットをサポートする。また DB2 Everyplace は、DB2 でサポートされていない接続属性もいくつかサポートしている。
SQLSetStmtAttr	ステートメントに関するオプションを設定する。	DB2 Everyplace は、DB2 でサポートされているステートメント属性のサブセットをサポートする。また DB2 Everyplace は、DB2 でサポートされていないステートメント属性もいくつかサポートしている。
SQLTables	特定のデータ・ソースに格納されている表名のリストを戻す。	*CatalogName*、*NameLength1*、*SchemaName*、*NameLength2*、*TableType*、*NameLength4* は無視される。DB2 Everyplace は「TABLE」タイプのみサポートする。戻りコード SQL_STILL_EXECUTING をサポートしない。

関連した解説:

- 295 ページの『DB2 CLI 関数によるデータ変換』
- 『DB2 CLI 関数の説明の要点』

DB2 CLI 関数の説明の要点

各関数の説明は、以下のセクションで構成されています。

目的 このセクションでは、関数によって何が行われるかを概説します。また、説明対象の関数を呼び出す前および後に、なんらかの関数を呼び出す必要があるかどうかについても示します。

各関数には、その関数がどの仕様または標準に準拠しているかを示す表も備わっています。

この表は、関数のサポートを示します。一部の関数では、すべての仕様または標準には適用されないようなオプションのセットを使用します。重要な相違点があれば、関数の制約事項のところで明記します。

構文 このセクションには、汎用 'C' プロトタイプが含まれています。汎用プロトタイプは、Windows を含むすべての環境で使用されます。

ポインターである関数の引き数はすべて、マクロ FAR を使用して定義されます。このマクロは、Windows を除くすべてのプラットフォームの場合、ブランクに設定されます。Windows では、FAR は、ポインター引き数を far ポインターとして定義するために使用されます。

引き数 このセクションでは、関数の各引き数をリストアップし、その引き数に関するデータ・タイプ、説明のほか、入力引き数であるか出力引き数であるかの区別も示します。

関数によっては、*据え置き* 引き数または*バインド* 引き数と呼ばれる入力引き数または出力引き数を含んでいます。

これらの引き数はアプリケーションが割り当てるバッファーへのポインターであり、SQL ステートメント内のパラメーター、または結果セット内の列のいずれかに関連付け (またはバインド) されます。関数が指定したデータ域は、後から DB2 CLI によってアクセスされます。これらの据え置きデータ域は、DB2 CLI によってアクセスされた時点でも、引き続き有効なままでなければなりません。

使用法 このセクションでは、関数の使用法に関して説明するとともに、特殊な考慮事項を示します。起こり得るエラー条件については、ここではなく、診断のセクションにリストされています。

戻りコード

このセクションでは、関数について発生し得るすべての戻りコードをリストアップしています。SQL_ERROR または SQL_SUCCESS_WITH_INFO が戻された場合、エラー情報は、SQLError() または SQLGetDiagRec() を呼び出すことによって入手できます。

診断 このセクションには、DB2 CLI によって明示的に戻される SQLSTATE (DBMS によって生成される SQLSTATE も戻される場合があります) をリストアップするとともに、エラーの原因を示す表が含まれています。これらの値は、関数が SQL_ERROR または SQL_SUCCESS_WITH_INFO を戻した後で、SQLError() または SQLGetDiagRec() を呼び出すことによって入手できます。

制約事項

このセクションでは、アプリケーションに影響を与える可能性がある、DB2 Everyplace CLI と ODBC の相違点または制限を示します。

戻りコード、診断、例、CLI 環境の設定、およびサンプル・アプリケーションへのアクセスについての情報を含めた、 DB2 CLI の詳細情報については、「*IBM DB2 Universal Database コール・レベル・インターフェースの手引きおよび解説書*」を参照してください。

関連した解説:

- 295 ページの『DB2 CLI 関数によるデータ変換』
- 202 ページの『DB2 CLI 関数の要約』

SQLAllocConnect - 接続ハンドルの割り当て

ODBC バージョン 3 では、SQLAllocConnect() は使用すべきでない関数となり、SQLAllocHandle() に置き換えられました。詳細については、『SQLAllocHandle - ハンドルの割り当て』を参照してください。

推奨事項: このバージョンの DB2 CLI では、引き続き SQLAllocConnect() をサポートしますが、最新の標準に準拠するよう、DB2 CLI プログラムでは SQLAllocHandle() を使用してください。

新しい関数へのマイグレーション

例えば、以下のステートメントは、

```
SQLAllocConnect(henv, hdbc);
```

新しい関数を使用して次のように書き直すことができます。

```
SQLAllocHandle(SQL_HANDLE_DBC, henv, hdbc);
```

関連した解説:

- 206 ページの『DB2 CLI 関数の説明の要点』
- 202 ページの『DB2 CLI 関数の要約』

SQLAllocEnv - 環境ハンドルの割り当て

ODBC バージョン 3 では、SQLAllocEnv() は使用すべきでない関数となり、SQLAllocHandle() に置き換えられました。詳細については、『SQLAllocHandle - ハンドルの割り当て』を参照してください。

推奨事項: このバージョンの DB2 CLI では、引き続き SQLAllocEnv() をサポートしますが、最新の標準に準拠するよう、DB2 CLI プログラムでは SQLAllocHandle() を使用してください。

新しい関数へのマイグレーション

例えば、以下のステートメントは、

```
SQLAllocEnv(henv);
```

新しい関数を使用して次のように書き直すことができます。

```
SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, henv);
```

関連した解説:

- 206 ページの『DB2 CLI 関数の説明の要点』
- 202 ページの『DB2 CLI 関数の要約』

SQLAllocHandle - ハンドルの割り当て

目的:

仕様:	DB2 CLI 5.0	ODBC 3.0	ISO CLI

SQLAllocHandle() は環境ハンドル、接続ハンドル、またはステートメント・ハンドルを割り当てます。

この関数は、ハンドル割り当て用の汎用関数であり、バージョン 2 の使用すべきでない関数 SQLAllocConnect()、SQLAllocEnv()、および SQLAllocStmt() を置き換えます。

構文:

```
SQLRETURN   SQLAllocHandle   (SQLSMALLINT       HandleType,
                              SQLHANDLE         InputHandle,
                              SQLHANDLE         *OutputHandlePtr);
```

関数の引き数:

表 30. SQLAllocHandle の引き数

データ・タイプ	引き数	用途	説明
SQLSMALLINT	*HandleType*	入力	SQLAllocHandle() によって割り当てるハンドルのタイプ。以下のいずれかの値でなければならない。 SQL_HANDLE_ENV SQL_HANDLE_DBC SQL_HANDLE_STMT
SQLHANDLE	*InputHandle*	入力	割り当てようとする新しいハンドルのコンテキストとして使用する既存のハンドル。*HandleType* が SQL_HANDLE_ENV である場合、SQL_NULL_HANDLE である。*HandleType* が SQL_HANDLE_DBC である場合、環境ハンドルでなければならない。SQL_HANDLE_STMT の場合は、接続ハンドルでなければならない。
SQLHANDLE	*OutputHandlePtr*	出力	新しく割り当てたデータ構造にハンドルを戻すためのバッファーへのポインター。

使用法:

SQLAllocHandle() は、以下に示すように、環境ハンドル、接続ハンドル、およびステートメント・ハンドルを割り当てるために使用します。

1 つのアプリケーションによって一度に複数のステートメント・ハンドルを割り当てることができます。

すでに存在している環境、接続、ステートメント、または記述子ハンドルに対して設定されている **OutputHandlePtr* を指定してアプリケーションで SQLAllocHandle() を呼び出すと、DB2 CLI はハンドルに関連する情報を上書きします。DB2 CLI では、**OutputHandlePtr* に入力されたハンドルがすでに使用中であるかどうかのチェックは行いません。また、上書きの前に、ハンドルの直前の内容をチェックすることもしません。

DB2 Everyplace の場合、ステートメント・ハンドルを除くすべてのハンドルはダミー・ハンドルであり、使用可能な情報を持っていません。

ステートメント・ハンドルでは、エラー・メッセージ、および SQL ステートメント処理に関する状況情報などの、ステートメント情報へのアクセスが行えます。ステートメント・ハンドルを要求するには、アプリケーションでデータ・ソースに接続した後 SQLAllocHandle() を呼び出して、その後で SQL ステートメントを実行

依頼します。この呼び出しのとき、*HandleType* を SQL_HANDLE_STMT に設定するとともに、*InputHandle* に関しては、そのハンドルを割り当てた SQLAllocHandle() への呼び出しが戻してきた接続ハンドルに設定する必要があります。 DB2 CLI は、ステートメント・ハンドルを割り当てて、そのステートメント・ハンドルを指定された接続に関連付け、関連付けされたハンドルの値を **OutputHandlePtr* に戻します。アプリケーションでは、ステートメント・ハンドルを要求した後続のすべての呼び出し時に **OutputHandlePtr* 値を渡します。

アプリケーションが終了すると、そのアプリケーションに割り当てられたすべての DB2 Everyplace リソースが解放されるため、そのアプリケーションが使用していたハンドルは無効になります。

DB2 Everyplace の場合、アプリケーションによって変更される可能性がある属性を持つステートメント・ハンドルに関連する記述子はありません。

Visual Basic と DB2 Everyplace CLI/ODBC インターフェースで DB2 Everyplace を使用する場合は、sqlcli.h で定義されているマップ/基礎 CLI 関数を明示的に呼び出さなければなりません。例えば、SQLAllocHandle() の呼び出しは失敗します。SQLAllocHandleVer(SQL_HANDLE_STMT, hdbc, hstmt, DB2eVersion) の呼び出しは成功します。

戻りコード:

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_INVALID_HANDLE
- SQL_ERROR

環境ハンドル以外のハンドルを割り当てる際に SQLAllocHandle() が SQL_ERROR を戻す場合、 SQLAllocHandle() は出力引き数がヌル・ポインターでなければ、*OutputHandlePtr* を SQL_NULL_HENV、SQL_NULL_HDBC、または SQL_NULL_HSTMT に設定します。このうちのどれに設定されるかは、*HandleType* の値によって決まります。この後、アプリケーションは、*InputHandle* 引き数内のハンドルに関連する診断データ構造から追加情報を入手することができます。

診断:

表 31. SQLAllocHandle の SQLSTATE

SQLSTATE	説明	解説
01000	警告。	通知メッセージ。(関数は、SQL_SUCCESS_WITH_INFO を戻す。)
08003	接続がクローズしている。	*HandleType* 引き数が SQL_HANDLE_STMT であるが、 *InputHandle* 引き数によって指定された接続がオープンされていない。 DB2 CLI でステートメント・ハンドルを割り当てるためには、接続処理を正常に完了する (接続をオープンする) 必要がある。
HY000	一般エラー。	エラーは発生したが、特定の SQLSTATE は無い。SQLGetDiagRec() が **MessageText* バッファーに戻したエラー・メッセージが、エラーとその原因を示しています。

表 31. SQLAllocHandle の SQLSTATE (続き)

SQLSTATE	説明	解説
HY001	メモリー割り振りエラー。	DB2 CLI が、指定されたハンドル用のメモリーを割り当てることができない。
HY013	予期しないメモリー処理エラー。	*HandleType* 引き数は SQL_HANDLE_DBC または SQL_HANDLE_STMT であるが、低メモリー状態が原因で、基礎となるメモリー・オブジェクトにアクセスできないため、関数呼び出しを処理できない。
HY014	ハンドル数の限界に到達済み。	*HandleType* 引き数によって示されたハンドルのタイプに割り当てることができるハンドル数の限界に到達した。
HY092	オプション・タイプが範囲外。	*HandleType* 引き数が以下のいずれでもない。 SQL_HANDLE_ENV SQL_HANDLE_DBC SQL_HANDLE_STMT

関連した解説:

- 206 ページの『DB2 CLI 関数の説明の要点』
- 202 ページの『DB2 CLI 関数の要約』
- 234 ページの『SQLExecDirect - ステートメントの直接の実行』
- 251 ページの『SQLFreeHandle - ハンドル・リソースの解放』

SQLAllocStmt - ステートメント・ハンドルの割り当て

ODBC バージョン 3 では、SQLAllocStmt() は使用すべきでない関数となり、SQLAllocHandle() に置き換えられました。詳細については、 208 ページの『SQLAllocHandle - ハンドルの割り当て』を参照してください。

推奨事項: このバージョンの DB2 CLI では、引き続き SQLAllocStmt() をサポートしますが、最新の標準に準拠するよう、DB2 CLI プログラムでは SQLAllocHandle() を使用してください。

新しい関数へのマイグレーション

例えば、以下のステートメントは、

```
SQLAllocStmt(hdbc, hstmt);
```

新しい関数を使用して次のように書き直すことができます。

```
SQLAllocHandle(SQL_HANDLE_STMT, hdbc, hstmt);
```

関連した解説:

- 206 ページの『DB2 CLI 関数の説明の要点』
- 202 ページの『DB2 CLI 関数の要約』

SQLBindCol - アプリケーション変数への列のバインド

目的:

仕様:	DB2 CLI 1.1	ODBC 1.0	ISO CLI

SQLBindCol() は、すべての C データ・タイプについて、結果セット内の列をアプリケーション変数に関連付ける (バインドする) ために使用されます。SQLFetch() を呼び出すと、データは DBMS からアプリケーションに転送されます。データの転送時に、データ変換が行われる場合があります。

SQLBindCol() は、アプリケーションで検索を必要とする結果セット内の列ごとに 1 回呼び出されます。

一般に、この関数の前に SQLPrepare() または SQLExecDirect() を呼び出し、この関数の後に SQLFetch() を呼び出します。列属性は、SQLBindCol() の呼び出し前に必要になる場合もありますが、SQLDescribeCol() を使用して入手することができます。

構文:

```
SQLRETURN   SQLBindCol       (SQLHSTMT          StatementHandle,   /* hstmt */
                              SQLUSMALLINT      ColumnNumber,      /* icol */
                              SQLSMALLINT       TargetType,        /* fCType */
                              SQLPOINTER        TargetValuePtr,    /* rgbValue */
                              SQLINTEGER        BufferLength,      /* cbValueMax */
                              SQLINTEGER   *FAR StrLen_or_IndPtr); /* pcbValue */
```

関数の引き数:

表 32. SQLBindCol の引き数

データ・タイプ	引き数	用途	説明
SQLHSTMT	*StatementHandle*	入力	ステートメント・ハンドル。
SQLUSMALLINT	*ColumnNumber*	入力	列を示す番号。列は左から右へ順番に番号が付けられる。列番号は 1 から始まる。
SQLSMALLINT	*TargetType*	入力	結果セット内の列番号 *ColumnNumber* の C データ・タイプ。以下のタイプがサポートされる。 SQL_C_BINARY SQL_C_BIT SQL_C_CHAR SQL_C_DOUBLE SQL_C_FLOAT SQL_C_LONG SQL_C_SHORT SQL_C_TYPE_DATE SQL_C_TYPE_TIME SQL_C_TYPE_TIMESTAMP SQL_C_TINYINT SQL_C_DEFAULT を指定すると、データがそのデフォルトの C データ・タイプに転送される。

表 32. SQLBindCol の引き数 (続き)

データ・タイプ	引き数	用途	説明
SQLPOINTER	*TargetValuePtr*	入出力 (据え置き)	列データの取り出し時に DB2 CLI が列データを保管するバッファーへのポインター。 *TargetValuePtr* がヌルである場合、列はアンバインドされる。
SQLINTEGER	*BufferLength*	入力	列データの保管に使用可能な *TargetValuePtr* バッファーのバイト単位のサイズ。 *TargetType* がバイナリー・ストリングまたは文字ストリングを示すか、あるいは SQL_C_DEFAULT である場合、*BufferLength* は > 0 でなければならない。0 以下であれば、エラーが戻される。それ以外の場合、この引き数は無視される。
SQLINTEGER *	*StrLen_or_IndPtr*	入出力 (据え置き)	DB2 CLI が *TargetValuePtr* バッファーに戻すことができるバイト数を示す値へのポインター。 SQLFetch() は、列のデータ値がヌルの場合に、この引き数に SQL_NULL_DATA を戻す。 SQL_NO_LENGTH も戻される場合がある。詳細については、使用法のセクションを参照。

この関数の場合、*TargetValuePtr* と *StrLen_or_Ind* は両方とも据え置き出力であり、これらのポインターが指し示す保管場所は、結果セット行が取り出されるまでは更新されません。結果として、これらのポインターによって参照される場所は、SQLFetch() が呼び出されるまで有効のままでなければなりません。例えば、ローカル関数内で SQLBindCol() を呼び出す場合、SQLFetch() はこの関数の同じ効力範囲内から呼び出される必要があります。あるいは、*TargetValuePtr* バッファーが静的またはグローバルとして割り当てられているか、または宣言されている必要があります。

使用法:

アプリケーションでの SQLBindCol() 呼び出しは、データを検索する必要がある結果セット内の列ごとに 1 回です。結果セットは、SQLExecute() または SQLExecDirect() を呼び出すことによって生成されます。SQLFetch() を呼び出すと、このようなバインド された各列内のデータは、割り当てられた場所 (ポインター *TargetValuePtr* と *StrLen_or_Ind* によって示される) に入れられます。

列は番号によって識別され、左から右へ順番に割り当てられます。列番号は 1 から始まります。

結果セット内の列数は、SQLNumResultCols() を呼び出すことによって判別することができます。

アプリケーションでは、最初に SQLDescribeCol() を呼び出すことによって、列の属性 (データ・タイプや長さなど) を照会することができます。次に、この情報を使用して、正しいデータ・タイプと長さの保管場所を割り当て、別のデータ・タイプへのデータ変換を指示することができます。

アプリケーションでは各列だけをバインドしないように選択することもできますし、どの列もバインドしないように選択することさえできます。現在行に関してバインドされた列を取り出した後に、SQLGetData() を使用することにより、どの列のデータを検索することもできます。

以後の取り出しで、アプリケーションでは、SQLBindCol() を呼び出すことによって、これらの列のバインディングを変更したり、以前にアンバインドした列をバインドすることができます。新しいバインディングは、すでに取り出されたデータには適用されずに、次の取り出し時に使用されます。単一の列をアンバインドするためには、*TargetValuePtr* ポインターを NULL に設定して SQLBindCol() を呼び出します。すべての列をアンバインドするためには、アプリケーションで SQLFreeStmt() を呼び出す必要があります。

アプリケーションでは、取り出すべきデータのために確実に十分なストレージを割り当てておかねばなりません。バッファーに可変長データを入れる場合、アプリケーションではバインドされた列が必要とする最大長のサイズのストレージを割り当てる必要があります。さもないと、データが切り捨てられる可能性があります。バッファーに固定長データを入れる場合、DB2 CLI は、バッファーのサイズが C データ・タイプの長さであると見なします。データ変換を指定した場合、必要とされるサイズが影響を受ける可能性があります。

ストリングの切り捨てが発生した場合、SQL_SUCCESS_WITH_INFO が戻され、*StrLen_or_IndPtr* が、アプリケーションへの戻りのために使用可能な *TargetValuePtr* の実際のサイズに設定されます。

戻りコード:

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

診断:

表 33. SQLBindCol の SQLSTATE

SQLSTATE	説明	解説
07009	無効な記述子索引。	引き数 *ColumnNumber* に指定した値が、結果セット内の最大列数を超えている。
40003 **08**S01	通信リンク・エラー。	関数の完了前に、アプリケーションとデータ・ソース間の通信リンクに障害が生じた。
58004	予期しないシステム障害。	回復不能なシステム・エラー。
HY001	メモリーの割り振りの失敗。	DB2 CLI が、関数の実行または完了のサポートに必要なメモリーを割り当てることができない。
HY002	無効な列番号。	引き数 *ColumnNumber* に指定した値が 0 より小さい。 引き数 *ColumnNumber* に指定した値が、データ・ソースによってサポートされている最大列数を超えた。

表 33. *SQLBindCol* の *SQLSTATE* (続き)

SQLSTATE	説明	解説
HY003	プログラム・タイプが範囲外。	*TargetType* が有効なデータ・タイプまたは SQL_C_DEFAULT ではない。
HY013	予期しないメモリー処理エラー。	DB2 CLI が、関数の実行または完了のサポートに必要なメモリーにアクセスできない。
HY090	無効なストリング長またはバッファー長。	引き数 *BufferLength* に指定した値が 1 より小さく、引き数 *TargetType* が SQL_C_CHAR、SQL_C_BINARY または SQL_C_DEFAULT のいずれかである。
HYC00	ドライバーが使用不可である。	引き数 *TargetType* に指定したデータ・タイプは、DB2 CLI によって認識されるが、サポートされない。

バインド列に関する追加の診断メッセージが、取り出し時に報告される場合があります。

制約事項:

出力バッファーは、ワードで (均等に) 位置合わせする必要があります。 Motorola 68000 などの多くのプロセッサーにワード位置合わせの規則があり、文字以外のデータ・タイプの場合、アプリケーションはバッファーを正しく位置合わせする必要がある。

関連した解説:

- 206 ページの『DB2 CLI 関数の説明の要点』
- 202 ページの『DB2 CLI 関数の要約』

SQLBindParameter - バッファーへのパラメーター・マーカーのバインド

目的:

仕様:	**DB2 CLI** 2.1	**ODBC** 2.0	

SQLBindParameter() は、すべての C データ・タイプに関して、SQL ステートメント内のパラメーター・マーカーをアプリケーション変数に関連付ける (バインドする) ために使用します。この場合、SQLExecute() または SQLExecDirect() の呼び出し時に、データはアプリケーションから DBMS に転送されます。データの転送時に、データ変換が行われる場合があります。

構文:

```
SQLRETURN  SQL_API SQLBindParameter(
                SQLHSTMT          StatementHandle,  /* hstmt */
                SQLUSMALLINT      ParameterNumber,  /* ipar */
                SQLSMALLINT       InputOutputType,  /* fParamType */
                SQLSMALLINT       ValueType,        /* fCType */
                SQLSMALLINT       ParameterType,    /* fSqlType */
                SQLUINTEGER       ColumnSize,       /* cbColDef */
                SQLSMALLINT       DecimalDigits,    /* ibScale */
```

```
SQLPOINTER        ParameterValuePtr,/* rgbValue */
SQLINTEGER        BufferLength,     /* cbValueMax */
SQLINTEGER *FAR   StrLen_or_IndPtr);/* pcbValue */
```

関数の引き数:

表 34. SQLBindParameter の引き数

データ・タイプ	引き数	用途	説明
SQLHSTMT	*StatementHandle*	入力	ステートメント・ハンドル。
SQLUSMALLINT	*ParameterNumber*	入力	1 から始まり、左から右へ順番に並べられるパラメーター・マーカー番号。
SQLSMALLINT	*InputOutputType*	入力	パラメーターのタイプ。サポートされるタイプは次のとおり。 • SQL_PARAM_INPUT: ステートメントの実行時に、パラメーターの実際のデータ値がサーバーに送信される。*ParameterValuePtr* バッファーには、有効な入力データ値が入っていなければならない。*StrLen_or_IndPtr* バッファーには、対応する長さ値、SQL_NTS、あるいは SQL_NULL_DATA が入っていなければならない。 DB2 Everyplace は、SQLPutData() をサポートしないため、*ParameterValuePtr* バッファーにパラメーター値を保管しないでください。 • SQL_PARAM_INPUT_OUTPUT: パラメーター・マーカーは、呼び出されたストアード・プロシージャーの入出力パラメーターに関連付けられている。ステートメントの実行時に、パラメーターの実際のデータ値がサーバーに送信される。 *ParameterValuePtr* バッファーには、有効な入力データ値が入っていなければならない。*StrLen_or_IndPtr* バッファーには、対応する長さ値、SQL_NTS、あるいは SQL_NULL_DATA が入っていなければならない。 • SQL_PARAM_OUTPUT: パラメーター・マーカーは、呼び出されたストアード・プロシージャーの出力パラメーター、またはストアード・プロシージャーの戻り値に関連付けられている。 ステートメントが実行された後で、出力パラメーターのデータが *ParameterValuePtr* および *StrLen_or_IndPtr* で指定されたアプリケーション・バッファーに戻されます。これは、どちらも NULL ポインターでないときに限ります。どちらも NULL ポインターの場合、出力データは廃棄されます。出力パラメーターに戻り値がないと、SQL_NULL_DATA には *StrLen_or_IndPtr* がセットされます。
SQLSMALLINT	*ValueType*	入力	パラメーターの C データ・タイプ。以下のタイプがサポートされる。 • SQL_C_BINARY • SQL_C_BIT • SQL_C_CHAR • SQL_C_DOUBLE • SQL_C_FLOAT • SQL_C_LONG • SQL_C_SHORT • SQL_C_TYPE_DATE • SQL_C_TYPE_TIME • SQL_C_TYPE_TIMESTAMP • SQL_C_TINYINT SQL_C_DEFAULT を指定すると、データがそのデフォルトの C データ・タイプから、*ParameterType* に示したタイプに転送される。
SQLSMALLINT	*ParameterType*	入力	パラメーターの SQL データ・タイプ。サポートされるタイプは次のとおり。 • SQL_BLOB • SQL_CHAR • SQL_DECIMAL • SQL_INTEGER • SQL_SMALLINT • SQL_TYPE_DATE • SQL_TYPE_TIME • SQL_TYPE_TIMESTAMP • SQL_VARCHAR

表 34. *SQLBindParameter* の引き数 (続き)

データ・タイプ	引き数	用途	説明
SQLUINTEGER	*ColumnSize*	入力	対応するパラメーター・マーカーの精度。 • *ParameterType* がバイナリー・ストリングまたは 1 バイト文字ストリング (SQL_CHAR、SQL_BLOB など) を示す場合は、このパラメーター・マーカーのバイト単位の最大長。 • それ以外の場合、この引き数は無視される。
SQLSMALLINT	*DecimalDigits*	入力	*ParameterType* が SQL_DECIMAL である場合、対応するパラメーターの位取り。
SQLPOINTER	*ParameterValuePtr*	入力 (据え置き)、出力 (据え置き)、あるいはその両方	• 入力の場合 (*InputOutputType* を SQL_PARAM_INPUT または SQL_PARAM_INPUT_OUTPUT に設定): 実行時、*StrLen_or_IndPtr* に SQL_NULL_DATA が含まれていない場合、*ParameterValuePtr* はパラメーターの実際のデータが入っているバッファーを指す。 • 出力の場合 (InputOutputType を SQL_PARAM_OUTPUT または SQL_PARAM_INPUT_OUTPUT に設定): *ParameterValuePtr* は、ストアード・プロシージャーの出力パラメーター値を保管するバッファーを指す。 • ヌルの *ParameterValuePtr* はパラメーターのアンバインドを示す。
SQLINTEGER	*BufferLength*	入力	文字データおよびバイナリー・データの場合、*BufferLength* は *ParameterValuePtr* バッファーの長さを指定する。文字データとバイナリー・データ以外の場合、この引き数は無視され、*ParameterValuePtr* バッファーの長さは、C データ・タイプに関連する長さであると見なされる。出力パラメーターの場合、*BufferLength* を使用してデータを切り捨てるかどうかを決定する。
SQLINTEGER *	*StrLen_or_IndPtr*	入力 (据え置き)、出力 (据え置き)、あるいはその両方	• 入力パラメーターまたは入出力パラメーターの場合: *ParameterValuePtr* に保管されているパラメーター・マーカー値の長さを含む (ステートメントの実行時に) 場所へのポインター。 パラメーター・マーカーにヌル値を指定するためには、この保管場所に SQL_NULL_DATA を入れる必要がある。 *ValueType* が SQL_C_CHAR の場合、この保管場所に、*ParameterValuePtr* に保管したデータの正確な長さ、または *ParameterValuePtr* の内容がヌルで終了している場合は、SQL_NTS が入っていなければならない。この保管場所に正確な長さが入っている場合は、ParameterValuePtr に保管したデータにヌル文字が入っていてはならない。 *ValueType* が文字データを示し (明示的に、または SQL_C_DEFAULT を使用して暗黙的に)、このポインターが NULL に設定されている場合、アプリケーションは *ParameterValuePtr* にヌル終了ストリングを提供しなければならない。これにより、このパラメーター・マーカーが NULL 値を決して持たないことも暗黙指定されます。 • 出力パラメーターの場合 (*InputOutputType* が SQL_PARAM_OUTPUT にセットされている): これは、出力パラメーターまたはストアード・プロシージャー CALL の戻り値となり、ストアード・プロシージャーの実行後、以下のいずれかのポインターとなります。 – *ParameterValuePtr* に戻すことができるバイト数。ヌル終了文字は含まない。 – SQL_NULL_DATA

使用法:

パラメーター・マーカーは、SQL ステートメント内の ? 文字によって表され、ステートメントを実行する時に、アプリケーションが提供した値を置き換えたい、ステートメント内の位置を指定するために使用します。この値は、アプリケーション変数から入手することができます。アプリケーションのストレージ域をパラメーター・マーカーにバインドするには、`SQLBindParameter()` を使用します。

SQL ステートメントの実行前に、アプリケーションでは SQL ステートメント内の各パラメーター・マーカーに変数をバインドする必要があります。この関数の場合、*ParameterValuePtr* と *StrLen_or_IndPtr* は据え置き引き数です。ステートメントを実行する時、保管場所は有効であり、かつ入力データ値を含んでいなければなりません。これは、SQLExecDirect() または SQLExecute() 呼び出しを SQLBindParameter() 呼び出しと同じプロシージャー効力範囲内に保持しておく必要があること、あるいはこれらの保管場所を動的に割り当てるか、静的またはグローバルとして宣言する必要があることを意味します。

パラメーター・マーカーは番号 (*ColumnNumber*) によって参照されます。番号は 1 から始まり、左から右へ順に付けられます。

この関数によってバインドされたすべてのパラメーターは、以下のいずれかの関数を呼び出すまで、引き続き有効です。

- SQL_RESET_PARAMS オプションを指定して SQLFreeStmt() を呼び出す。
- *HandleType* を SQL_HANDLE_STMT に設定して SQLFreeHandle() を呼び出す。
- 同じパラメーター *ParameterNumber* 番号に関して、SQLBindParameter() を再び呼び出す。

SQL ステートメントを実行し結果を処理した後、アプリケーションでは、ステートメント・ハンドルを再使用して別の SQL ステートメントを実行することができます。パラメーター・マーカー指定 (パラメーター数、長さ、またはタイプ) が異なる場合、パラメーターのバインディングをリセットまたは消去するために、SQL_RESET_PARAMS を指定して SQLFreeStmt() を呼び出す必要があります。

ValueType によって指定された C バッファーのデータ・タイプは、*ParameterType* によって指定された SQL データ・タイプと互換性がなければなりません。さもないと、エラーが発生します。

ParameterValuePtr と *StrLen_or_IndPtr* が参照する変数内のデータは、ステートメントの実行時まで検査されないため、データの内容または形式のエラーは、SQLExecute() または SQLExecDirect() を呼び出すまで、検出または報告されません。

この関数の場合、*ParameterValuePtr* と *StrLen_or_IndPtr* は据え置き引き数です。*InputOutputType* を SQL_PARAM_INPUT に設定した場合、ステートメントの実行時に、保管場所は有効であり、かつ入力データ値を含んでいなければなりません。これは、SQLExecDirect() または SQLExecute() 呼び出しを SQLBindParameter() 呼び出しと同じプロシージャー効力範囲内に保持しておく必要があること、あるいはこれらの保管場所を動的に割り当てるか、静的またはグローバルとして宣言する必要があることを意味します。

DB2 Everyplace は SQL_PARAM_INPUT、SQL_PARAM_INPUT_OUTPUT、および SQL_PARAM_OUTPUT をサポートします。 DB2 Everyplace は、SQLPutData() をサポートしないため、*ParameterValuePtr* バッファーにパラメーター値を保管しないでください。

文字データおよびバイナリー C データの場合、*BufferLength* 引き数は *ParameterValuePtr* バッファーの長さを指定します。他のすべてのタイプの C データの場合、*BufferLength* 引き数は無視されます。

戻りコード:

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

診断:

表 35. SQLBindParameter の SQLSTATE

SQLSTATE	説明	解説
07006	無効な変換。	*ValueType* 引き数によって示したデータ値から *ParameterType* 引き数によって示したデータ・タイプへの変換が無意味な変換である。(例えば、SQL_C_DATE から SQL_DOUBLE への変換。)
40003 **08**S01	通信リンク障害。	関数の完了前に、アプリケーションとデータ・ソース間の通信リンクに障害が生じた。
58004	予期しないシステム障害。	回復不能なシステム・エラー。
HY001	メモリーの割り振りの失敗。	DB2 CLI が、関数を実行または完了させるのに必要なメモリーを割り当てることができない。
HY003	プログラム・タイプが範囲外。	引き数 *ParameterNumber* によって指定した値が、有効なデータ・タイプまたは SQL_C_DEFAULT ではない。
HY004	SQL データ・タイプが範囲外。	引き数 *ParameterType* に指定した値が、無効な SQL データ・タイプである。
HY009	無効な引き数値。	引き数 *ParameterValuePtr* と *StrLen_or_IndPtr* がヌル・ポインターであり、*InputOutputType* が SQL_PARAM_OUTPUT ではない。
HY013	予期しないメモリー処理エラー。	DB2 CLI が、関数を実行または完了させるのに必要なメモリーにアクセスできない。
HY090	無効なストリング長またはバッファー長。	引き数 *BufferLength* に指定した値が 0 より小さい。
HY093	無効なパラメーター番号。	引き数 *ValueType* に指定した値が、1 より小さいか、サーバーによってサポートされるパラメーターの最大数を超えている。
HY094	無効な位取り値。	*ParameterType* に指定した値が SQL_DECIMAL または SQL_NUMERIC であり、 *DecimalDigits* に指定した値が 0 より小さいか、引き数 *ParamDef* (精度) の値より大きい。
HY104	無効な精度値。	*ParameterType* に指定した値が SQL_DECIMAL または SQL_NUMERIC のいずれかであるが、 *ParamDef* に指定した値が 1 より小さい。
HY105	無効なパラメーター・タイプ。	*InputOutputType* が SQL_PARAM_INPUT ではない。

表 35. SQLBindParameter の SQLSTATE (続き)

SQLSTATE	説明	解説
HYC00	ドライバーが使用不可である。	DB2 CLI またはデータ・ソースが、引き数 *ValueType* に指定した値と引き数 *ParameterType* に指定した値の組み合わせによって指定した変換をサポートしない。 引き数 *ParameterType* に指定した値が、DB2 CLI またはデータ・ソースによってサポートされない。

関連した解説:

- 185 ページの『割り当てと比較に関するデータ・タイプの互換性』
- 206 ページの『DB2 CLI 関数の説明の要点』
- 202 ページの『DB2 CLI 関数の要約』
- 234 ページの『SQLExecDirect - ステートメントの直接の実行』
- 235 ページの『SQLExecute - ステートメントの実行』

SQLConnect - データ・ソースへの接続

目的:

仕様:	DB2 CLI 2.1	ODBC 1.0	ISO CLI

SQLConnect() は、ターゲット・データベースへの接続を確立します。

接続ハンドルは、この関数の呼び出し前に、SQLAllocHandle() を使用して割り当てる必要があります。

この関数は、SQLAllocHandle() を使用してステートメント・ハンドルを割り当てる前に呼び出す必要があります。

構文:

```
SQLRETURN   SQLConnect       (
                SQLHDBC          ConnectionHandle,  /* hdbc */
                SQLCHAR      *FAR ServerName,       /* szDSN */
                SQLSMALLINT      NameLength1,       /* cbDSN */
                SQLCHAR      *FAR UserName,         /* szUID */
                SQLSMALLINT      NameLength2,       /* cbUID */
                SQLCHAR      *FAR Authentication,   /* szAuthStr */
                SQLSMALLINT      NameLength3);      /* cbAuthStr */
```

関数の引き数:

表 36. SQLConnect の引き数

データ・タイプ	引き数	用途	説明
SQLHDBC	*ConnectionHandle*	入力	接続ハンドル。
SQLCHAR *	*ServerName*	入力	データベースの場所と名前。名前はオプショナル。名前は、DB2 Everyplace では無視。
SQLSMALLINT	*NameLength1*	入力	*ServerName* 引き数の内容の長さ。

表 36. SQLConnect の引き数 (続き)

データ・タイプ	引き数	用途	説明
SQLCHAR *	*UserName*	入力	許可名 (ユーザー ID)。このストリングは、暗号化で使用します。それ以外の場合、DB2 Everyplace はこれを無視します。
SQLSMALLINT	*NameLength2*	入力	*UserName* 引き数の内容の長さ。
SQLCHAR *	*Authentication*	入力	認証ストリング (パスワード)。このストリングは、暗号化で使用します。それ以外の場合、DB2 Everyplace はこれを無視します。
SQLSMALLINT	*NameLength3*	入力	*Authentication* 引き数の内容の長さ。

注意事項:

未登録 ユーザー (DB2eSYSUSERS 表に存在しないユーザー) が、SQLGetDiagRec() CLI 関数の呼び出し中に暗号化された DB2 Everyplace データベースへ接続しようとすると、このユーザーは警告メッセージ 42704 (オブジェクトが未定義) を受け取ります。*登録済み* ユーザーは、この警告を受け取りません。対照的に、SQLConnect() 関数の呼び出し中には未登録ユーザーと登録済みユーザーの両者がデータベースに接続でき、警告メッセージも受け取りません。

使用法:

`SQLConnect()` を使用して、異なる場所のデータ・ソースを接続することができます。

ローカル・デバイスのデータ・ソースにアクセスするには、*ServerName* 引き数にデータ・ソース名を設定します。 DB2 Everyplace はデータ・ソース名を無視し、ローカル・データ・ソースにアクセスします。

2 次ストレージ装置を使用するアプリケーションの場合、引き数 *ServerName* には、*DataSource* の位置を指すストリングを入れることができます。DataSource は、ローカルにあるかまたは IBM マイクロドライブ、Sony メモリー・スティック、コンパクト・フラッシュ、SD メモリー・カード、MultiMediaCard のような 2 次ストレージ装置にあります。*ServerName* ストリングの形式は、次のとおりです。

`ServerName=Device:/Path/DataSource`

Device これは、*DataSource* が保管されているデバイスの名前です。コンパクト・フラッシュをサポートしている Palm OS デバイスのコンパクト・フラッシュ (CF) タイプ II ストレージ・デバイスにアクセスするには、予約済み文字 `#` を使用します。 2 次ストレージは、予約済み文字 `#` で指定されます。`#0` および `#1` は、アクセスする 2 次ストレージ・スロットを指定します。`#` は `#0` と同じです。以下に例を示します。

```
ServerName=#:/storage/
```

DB2 Everyplace は、最初の CF スロットに入っている IBM マイクロドライブの `storage` ディレクトリーにある *DataSource* に接続します。

Path これは、*Device* にある *DataSource* へのパスです。 *Path* を *`Device:/`* なしで指定すると、アプリケーションの位置との相対的なローカル・ファイル・システム・パスが使用されます。ファイルを、ボリュームのルート・ディレクトリーに書き込んではいけません。ルート・ディレクトリー内のファイルは、一部のメディア・タイプでサポートされません。例えば、次の場合、

```
ServerName=dir1/dir2/DATA1
```

注: DB2 Everyplace ではパスの長さに制限はありません。
アプリケーションがローカル・ファイル・システムの /myapp にある場合、DB2 Everyplace は /myapp/dir1/dir2/ にある *DataSource* に接続します。*DataSource* 名の DATA1 は無視されます。

DataSource
オプショナル: 接続するデータ・ソースの名前。この名前は、DB2 Everyplace では無視されます。

リモート照会およびストアード・プロシージャー・アダプターを使用してリモート・ストアード・プロシージャーにアクセスするには、*ServerName* 引き数を使用してデータベースの場所と名前を指定します。リモート照会およびストアード・プロシージャー・アダプターを使用してリモート・データベースにアクセスするアプリケーションの場合、引き数 *ServerName* には以下の URL 形式を入れることができます。

```
http://IPAddress:portNumber/path?DB=DataSource
```

IPAddress および *Authentication* は必須です。

標準的なファイル・システムではなく、Windows CE のオブジェクト記憶を使用している場合は、以下のいずれかの手順を実行します。

- CLI 関数 SQLConnect のパス・パラメーターを "@:¥" に設定する

または

- CLP で、'connect to @:¥' を実行する

Windows CE のオブジェクト記憶には、「ディレクトリー」という概念はありません。オブジェクト記憶を使用している場合、ユーザーは表の作成先のディレクトリー、つまりパスを指定することはできません。オブジェクト記憶内のすべての表は、同じネーム・スペースに作成されます。こうした制限から、オブジェクト記憶への複数の同時接続は確立できません。接続のシリアライゼーションを目的としたロック・ファイルが、ファイル・システムのルート・パスに作成されます。

オブジェクト記憶を使用している場合は、標準的なファイル・システムの場合とは異なり、DB2 Everyplace ファイルを手動で削除することはできません。

例:

ローカルの c:¥dir1¥dir2¥ にあるデータ・ソースに接続します。データ・ソース名 DS1 は無視されます。

```
ServerName=c:/dir1/dir2/DS1
```

ローカルの /dir1/dir2/ にあるデータ・ソースに、UNIX のファイル・システム表記で接続します。

```
ServerName=/dir1/dir2/
```

アプリケーション・パスから見て、ローカルの dir1¥ ディレクトリーにあるデータ・ソースに接続します。アプリケーションが c:¥myapp¥ にある場合は、c:¥myapp¥dir1¥ データ・ソースがアクセスされます。

```
ServerName=dir1¥
```

2 次ストレージ・スロット 1 に入っているストレージ・メモリーの /dir1/ ディレクトリーにあるデータ・ソースに接続します。

```
ServerName=#1:/dir1/
```

リモート照会およびストアード・プロシージャー・アダプターを使用して、DB2 Everyplace 同期サーバー 192.168.0.1 のポート 8080 でデータベース mysample に接続します。

```
ServerName=
 http://192.168.0.1:8080/db2e/servlet/com.ibm.mobileservices.adapter
                .agent.AgentServlet?DB=mysample
```

Windows CE のストレージを使用して、データ・ソースに接続します。

```
ServerName=@:¥
```

接続のシリアライゼーション:

接続のシリアライゼーションについては、 75 ページの『接続のシリアライゼーション』を参照してください。

接続の認証:

データベースの暗号化には、基礎となるユーザー認証が必要です。 DB2 Everyplace では、UserName および Authentication を使用して、接続時にユーザーの認証を実行します。

認証は次のように行われます。SQLConnect の接続先のデータベースに DB2eSYSUSERS カタログ表が存在しない場合、UserName と Authentication の情報は無視されます。DB2 Everyplace では登録済み ユーザーと未登録 ユーザーとを区別しています。登録済みユーザーとは、GRANT SQL ステートメントを使用して追加された DB2eSYSUSERS 表にリストされているユーザーのことです。接続時において、DB2eSYSUSERS 表が存在し、UserName が登録済みユーザーのものであれば、認証は行われます。 Authentication パラメーターに指定されたパスワードが正しくない場合は、エラー (42505) が戻されます。 UserName が未登録である場合は、SQLConnect 関数は正常に実行されます。ただし、後続の SQLGetDiagRec への呼び出しでは、警告 42704 (オブジェクトが未定義) が戻されます。これにより、正常に接続している登録済みユーザーと正常に接続された未登録ユーザーとの区別が、アプリケーションで可能となります。詳しくは、 89 ページの『ローカル・データ暗号化の概要』、368 ページ、および 166 ページの『GRANT』を参照してください。

戻りコード:

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

診断:

表 37. SQLConnect の SQLSTATE

SQLSTATE	説明	解説
08001	データ・ソースに接続できない。	DB2 CLI がデータ・ソース (サーバー) との接続を確立できない。
08002	接続が使用中。	指定した *ConnectionHandle* が、データ・ソースとの接続を確立するためにすでに使用されていて、その接続がまだオープンされている。
08004	アプリケーション・サーバーが接続の確立をリジェクトした。	データ・ソース (サーバー) が接続の確立をリジェクトした。
58004	予期しないシステム障害。	回復不能なシステム・エラー。
HY001	メモリーの割り振りの失敗。	DB2 CLI が、関数の実行または完了のサポートに必要なメモリーを割り当てることができない。
HY013	予期しないメモリー処理エラー。	DB2 CLI が、関数の実行または完了のサポートに必要なメモリーにアクセスできない。
HY501	*DataSource* 名が無効。	指定された *DataSource* 名が無効。
HYT00	接続がタイムアウトになった。	アプリケーションがデータ・ソースへの接続を確立する前に、タイムアウト期間を過ぎた。タイムアウト期間は、*SQLSetConnectAttr()* の SQL_ATTR_LOGIN_TIMEOUT 属性を使用して設定することができる。このエラーは、データベースが他のアプリケーションによって使用中の場合に戻される。

制約事項:

SQL ステートメントの実行前に、`SQLConnect()` を呼び出す必要があります。

関連した概念:

- 75 ページの『接続のシリアライゼーション』
- 208 ページの『SQLAllocHandle - ハンドルの割り当て』
- 230 ページの『SQLDisconnect - データ・ソースからの切断』

関連した解説:

- 206 ページの『DB2 CLI 関数の説明の要点』
- 202 ページの『DB2 CLI 関数の要約』
- 208 ページの『SQLAllocHandle - ハンドルの割り当て』
- 230 ページの『SQLDisconnect - データ・ソースからの切断』

SQLColumns - 表の列情報の取得

目的:

仕様:	**DB2 CLI** 2.1	**ODBC** 1.0	

SQLColumns() は指定された表の列のリストを戻します。この情報は SQL 結果セットに入れて戻され、照会で生成された結果セットを取り出すのに使用される関数と同じ関数を使用して検索できます。

構文:

```
SQLRETURN   SQLColumns       (
               SQLHSTMT          StatementHandle,   /* hstmt */
               SQLCHAR      FAR  *CatalogName,      /* szCatalogName */
               SQLSMALLINT       NameLength1,       /* cbCatalogName */
               SQLCHAR      FAR  *SchemaName,       /* szSchemaName */
               SQLSMALLINT       NameLength2,       /* cbSchemaName */
               SQLCHAR      FAR  *TableName,        /* szTableName */
               SQLSMALLINT       NameLength3,       /* cbTableName */
               SQLCHAR      FAR  *ColumnName,       /* szColumnName */
               SQLSMALLINT       NameLength4);      /* cbColumnName */
```

関数の引き数:

表 38. SQLColumns の引き数

データ・タイプ	引き数	用途	説明
SQLHSTMT	*StatementHandle*	入力	ステートメント・ハンドル。
SQLCHAR	*CatalogName*	入力	結果セットを修飾する *pattern-value* を入れることのできるバッファー。*Catalog* は 3 部分からなる表名の最初の部分である。 この引き数は DB2 Everyplace では無視される。
SQLSMALLINT	*NameLength1*	入力	*CatalogName* の長さ。 この引き数は DB2 Everyplace では無視される。
SQLCHAR	*SchemaName*	入力	結果セットをスキーマ名で修飾する *pattern-value* を入れることのできるバッファー。 この引き数は DB2 Everyplace では無視される。
SQLSMALLINT	*NameLength2*	入力	*SchemaName* の長さ。 この引き数は DB2 Everyplace では無視される。
SQLCHAR	*TableName*	入力	結果セットを表名で修飾する *pattern-value* を入れることのできるバッファー。
SQLSMALLINT	*NameLength3*	入力	*TableName* の長さ。
SQLCHAR	*ColumnName*	入力	結果セットを列名で修飾する *pattern-value* を入れることのできるバッファー。
SQLSMALLINT	*NameLength4*	入力	*ColumnName* の長さ。

使用法:

この関数は、1 つの表または 1 組の表の列についての情報を検索するときに呼び出します。一般的なアプリケーションでこの関数を呼び出すのは、SQLTables() を呼び出した後、表の列を判別するときです。アプリケーションでは、SQLTables() 結果セットの TABLE_NAME に戻された文字ストリングをこの関数の入力として使用する必要があります。

SQLColumns() は、TABLE_NAME と ORDINAL_POSITION の順に配列された標準の結果セットを戻します。226 ページに結果セットの列をリストしてあります。

TableName 引き数と *ColumnName* 引き数は検索パターンを受け入れます。

この関数は結果セット内の列についての情報は戻しません。代わりに、SQLDescribeCol() または SQLColAttribute() を使用してください。

SQLColumns() の呼び出しは、多くの場合システム・カタログに対する複雑で時間のかかる照会となるため、控えめに行うようにしてください。呼び出しを繰り返さず、結果を保管するようにしてください。

カタログ関数の結果セットの VARCHAR 列は、SQL92 の制限と一貫性を持たせるために最大長の属性は 128 で宣言されています。 DB2 の名前は 128 より短いため、128 文字 (NULL 終止符を含む) を常に出力バッファー用にとっておくか、または SQL_MAX_TABLE_NAME_LEN と SQL_MAX_COLUMN_NAME_LEN とを指定した SQLGetInfo() を呼び出して、接続されている DBMS がサポートする TABLE_NAME 列と COLUMN_NAME 列の実際の長さを判別することができます。

将来のリリースで新規の列が追加されたり、既存の列の名前が変更されても、現在の列の位置は変わりません。

SQLColumns が戻す列:

列 1 TABLE_CAT (VARCHAR(128))
これは常に NULL です。

列 2 TABLE_SCHEM (VARCHAR(128))
これは常に NULL です。

列 3 TABLE_NAME (VARCHAR(128) 非ヌル)
表の名前。

列 4 COLUMN_NAME (VARCHAR(128) 非ヌル)
列 ID。指定された表の列の名前、ビュー、別名、または同義語。

列 5 DATA_TYPE (SMALLINT 非ヌル)
COLUMN_NAME で識別された列の SQL データ・タイプ。これは、 186 ページの『SQL のシンボリックおよびデフォルトのデータ・タイプ』の「シンボリック SQL データ・タイプ」の欄の値の 1 つです。

列 6 TYPE_NAME (VARCHAR(128) 非ヌル)
DATA_TYPE に対応するデータ・タイプの名前を表す文字ストリング。

列 7 COLUMN_SIZE (INTEGER)

DATA_TYPE 列の値が文字ストリングかバイナリー・ストリングを表す場合、この列には列の最大長 (文字数) が入ります。

DATE、TIME、または TIMESTAMP のデータ・タイプの場合、これは、文字に変換された値を表示するのに必要な文字の合計数になります。

数値データ・タイプの場合、これは、列内で許可される合計の桁数になります。

186 ページの『データ・タイプ属性』も参照してください。

列 8 BUFFER_LENGTH (INTEGER)
SQL_C_DEFAULT が SQLBindCol()、SQLGetData()、および SQLBindParameter() の呼び出しで指定された場合、この列からのデータを関連した C バッファーに格納するときのバイトの最大数。この長さは NULL 終止符を含んでいません。また、厳密な数値データ・タイプの場合、長さには 10 進数と符号が含まれます。 186 ページの『データ・タイプ属性』も参照してください。

列 9 DECIMAL_DIGITS (SMALLINT)
列の位取り。位取りが適用されないデータ・タイプの場合は、NULL が戻されます。 186 ページの『データ・タイプ属性』も参照してください。

列 10 NUM_PREC_RADIX (SMALLINT)

10 または NULL のいずれか。

DATA_TYPE が厳密な数値データ・タイプである場合、この列には値 10 が入り、COLUMN_SIZE には列に許可されている 10 進数の桁数が入ります。

数値データ・タイプの場合、DBMS は 10 の NUM_PREC_RADIX を戻します。

基数が適用されないデータ・タイプの場合は、NULL が戻されます。

列 11 NULLABLE (SMALLINT 非ヌル)

列が NULL 値を受け入れない場合、SQL_NO_NULLS。

列が NULL 値を受け入れる場合、SQL_NULLABLE。

列 12 REMARKS (VARCHAR(254))
これは常に NULL です。

列 13 COLUMN_DEF (VARCHAR(254))

列のデフォルト値。デフォルト値が数値リテラルである場合は、この列には単一引用符で囲まれていない数値リテラルの文字表現が入ります。デフォルト値が文字ストリングである場合は、この列には単一引用符で囲まれたそのストリングが入ります。デフォルト値が DATE、TIME、および TIMESTAMP といった列の疑似リテラルである場合は、引用符で囲まれていない疑似リテラルのキーワード (例えば、CURRENT DATE) が入ります。

NULL がデフォルト値として指定された場合は、この列には引用符で囲まれていない「NULL」という単語が戻されます。デフォルト値が指定されなかった場合は、この列は NULL となります。

列 14 SQL_DATA_TYPE (SMALLINT 非ヌル)
この列は DATA_TYPE 列と同じです。

列 15 SQL_DATETIME_SUB (SMALLINT)
この列は常に NULL です。

列 16 CHAR_OCTET_LENGTH (INTEGER)
文字データ・タイプ列のオクテット単位での最大長が入ります。 1 バイト文字セットの場合、これは COLUMN_SIZE と同じです。その他のデータ・タイプの場合は、NULL です。

列 17 ORDINAL_POSITION (INTEGER 非ヌル)
表内の列の順序を示す位置。表の第 1 列は番号 1 です。

列 18 IS_NULLABLE (VARCHAR(254))

列が NULL 可能でないことが既知である場合はストリング「NO」が入り、それ以外の場合は、「YES」が入ります。

この結果セットは X/Open CLI の *Columns()* 結果セット仕様と同一です。この仕様は、ODBC V2 で規定されている *SQLColumns()* 結果セットの拡張バージョンです。ODBC の *SQLColumns()* 結果セットには同じ位置に各列が含まれます。

注: この結果セットは X/Open CLI の `Columns()` 結果セット仕様と同一です。この仕様は、ODBC V2 に規定されている `SQLColumns()` 結果セットの拡張バージョンです。ODBC の `SQLColumns()` 結果セットには同じ位置に各列が含まれます。

戻りコード:

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

診断:

表 39. SQLColumns の SQLSTATE

SQLSTATE	説明	解説
24000	カーソル状態が無効。	カーソルがすでにこのステートメント・ハンドルでオープンされている。
40003 **08**S01	通信リンク障害。	関数の完了前に、アプリケーションとデータ・ソース間の通信リンクに障害が生じた。
HY001	メモリーの割り振りの失敗。	DB2 CLI が、関数を実行または完了させるのに必要なメモリーを割り当てることができない。
HY014	ハンドル数の限界に到達済み。	内部リソースが原因で、DB2 CLI がハンドルを割り振ることができない。
HY090	無効なストリング長またはバッファー長。	名前の長さの引き数のいずれかの値が、ゼロより小さいが、SQL_NTS と等しくない。

制約事項:

なし。

関連した解説:

- 292 ページの『SQLTables - 表情報の取得』

SQLDescribeCol - 列の属性セットの戻し

目的:

仕様:	DB2 CLI 1.1	ODBC 1.0	ISO CLI

`SQLDescribeCol()` は、照会によって生成される結果セットに、指定した列に関する一般的に使用される記述子情報 (列名、タイプ、精度、位取り、ヌル指定の可否) のセットを戻します。

この関数の呼び出し前に、`SQLPrepare()` または `SQLExecDirect()` を呼び出す必要があります。

この関数は通常、バインディング列関数 (SQLBindCol()) の前に呼び出され、列をアプリケーション変数にバインディングする前、列の属性を判別します。

構文:

```
SQLRETURN   SQLDescribeCol   (
                SQLHSTMT          StatementHandle,   /* hstmt */
                SQLUSMALLINT      ColumnNumber,      /* icol */
                SQLCHAR      *FAR ColumnName,        /* szColName */
                SQLSMALLINT       BufferLength,      /* cbColNameMax */
                SQLSMALLINT  *FAR NameLengthPtr,     /* pcbColName */
                SQLSMALLINT  *FAR DataTypePtr,       /* pfSqlType */
                SQLUINTEGER  *FAR ColumnSizePtr,     /* pcbColDef */
                SQLSMALLINT  *FAR DecimalDigitsPtr,  /* pibScale */
                SQLSMALLINT  *FAR NullablePtr);      /* pfNullable */
```

関数の引き数:

表 40. SQLDescribeCol の引き数

データ・タイプ	引き数	用途	説明
SQLHSTMT	*StatementHandle*	入力	ステートメント・ハンドル。
SQLUSMALLINT	*ColumnNumber*	入力	記述する列番号。列には、1 から始まり、左から右へ順に番号が付けられる。
SQLCHAR *	*ColumnName*	出力	列名バッファーへのポインター。これは、列名を判別できない場合には NULL に設定される。
SQLSMALLINT	*BufferLength*	入力	*ColumnName* バッファーのサイズ。
SQLSMALLINT *	*NameLengthPtr*	出力	*ColumnName* 引き数の値を戻すことができるバイト数。 *NameLengthPtr* が *BufferLength* バイト以上の場合、列名 (*ColumnName*) の *BufferLength - 1* バイトへの切り捨てが行われる。
SQLSMALLINT *	*DataTypePtr*	出力	列のベース SQL データ・タイプ。
SQLUINTEGER *	*ColumnSizePtr*	出力	データベース内に定義された列の精度。
SQLSMALLINT *	*DecimalDigitsPtr*	出力	データベース内に定義された列の位取り (SQL_DECIMAL にのみ適用される)。
SQLSMALLINT *	*NullablePtr*	出力	この列に NULL が許可されるかどうかを示す。以下のいずれか。 SQL_NO_NULLS SQL_NULLABLE

使用法:

列は番号によって識別され、左から右へ順番に番号が付けられるので、任意の順序で記述できます。列番号は 1 から始まります。

ポインター引き数のいずれかにヌル・ポインターを指定した場合、DB2 CLI は、アプリケーションが情報を必要としていないと見なし、何も戻しません。

戻りコード:

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

診断:

SQLDescribeCol() が SQL_ERROR または SQL_SUCCESS_WITH_INFO を戻した場合、SQLError() 関数を呼び出すことによって以下の SQLSTATE を入手できます。

表 41. SQLDescribeCol の SQLSTATE

SQLSTATE	説明	解説
01004	データが切り捨てられた。	引き数 *ColumnName* に戻された列名が、引き数 *BufferLength* に指定した値より長い。引き数 *NameLengthPtr* に、列名全体の長さが入っている。(関数は、SQL_SUCCESS_WITH_INFO を戻す。)
07005	ステートメントが結果セットを戻さなかった。	*StatementHandle* に関連するステートメントが結果セットを戻さなかった。記述する列がなかった。(最初に SQLNumResultCols() を呼び出して、結果セット内に行があるかどうかを判別する。)
07009	無効な記述子索引。	*ColumnNumber* に指定した値が 0 以下である。引き数 *ColumnNumber* に指定した値が、結果セット内の列数より大きい。
40003 **08**S01	通信リンク障害。	関数の完了前に、アプリケーションとデータ・ソース間の通信リンクに障害が生じた。
58004	予期しないシステム障害。	回復不能なシステム・エラー。
HY001	メモリーの割り振りの失敗。	DB2 CLI が、関数の実行または完了のサポートに必要なメモリーを割り当てることができない。
HY002	無効な列番号。	引き数 *ColumnNumber* に指定した値が 1 より小さい。または、引き数 *ColumnNumber* に指定した値が、結果セット内の列数より大きい。
HY090	無効なストリング長またはバッファー長。	引き数 *BufferLength* に指定した長さが 1 より小さい。
HY010	関数のシーケンス・エラー。	*StatementHandle* に対して、SQLPrepare() または SQLExecDirect() が呼び出される前に関数が呼び出された。
HY013	予期しないメモリー処理エラー。	DB2 CLI が、関数の実行または完了のサポートに必要なメモリーにアクセスできない。
HYC00	ドライバーが使用不可である。	列 *ColumnNumber* の SQL データ・タイプが DB2 CLI によって認識されない。

制約事項:

DB2 Everyplace は以下の ODBC 定義のデータ・タイプしかサポートしません。

- SQL_BLOB
- SQL_CHAR
- SQL_DECIMAL
- SQL_INTEGER
- SQL_SMALLINT
- SQL_TYPE_DATE
- SQL_TYPE_TIME
- SQL_TYPE_TIMESTAMP
- SQL_VARCHAR

関連した解説:

- 206 ページの『DB2 CLI 関数の説明の要点』
- 202 ページの『DB2 CLI 関数の要約』
- 234 ページの『SQLExecDirect - ステートメントの直接の実行』
- 274 ページの『SQLNumResultCols - 結果の列数の入手』

SQLDisconnect - データ・ソースからの切断

目的:

仕様:	DB2 CLI 1.1	ODBC 1.0	ISO CLI

SQLDisconnect() はデータベース接続ハンドルに関連する接続をクローズします。

この関数を呼び出した後で、SQLConnect() を呼び出して別のデータベースに接続するか、または SQLFreeHandle() を呼び出してください。

構文:

```
SQLRETURN   SQLDisconnect    (SQLHDBC          ConnectionHandle;) /* hdbc */
```

関数の引き数:

表 42. SQLDisconnect の引き数

データ・タイプ	引き数	用途	説明
SQLHDBC	*ConnectionHandle*	入力	接続ハンドル。

使用法:

アプリケーションで、接続に関連するすべてのステートメント・ハンドルを解放する前に SQLDisconnect() を呼び出した場合、DB2 CLI は、データベースから正常に切断した後、ステートメント・ハンドルを解放します。

SQL_SUCCESS_WITH_INFO が戻された場合、データベースからの切断は成功したが、追加のエラー情報またはインプリメンテーション固有の情報があることを意味します。例えば、接続の切断に続く処理で問題が検出された場合や、アプリケーションとは関係なく発生したイベント (通信障害など) が原因で現行接続が存在しない場合などです。

SQLDisconnect() 呼び出しが成功した後、アプリケーションでは *ConnectionHandle* を再使用して、別の SQLConnect() または SQLDriverConnect() 要求を行うことができます。

戻りコード:

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

診断:

表 43. SQLDisconnect の SQLSTATE

SQLSTATE	説明	解説
01002	切断エラー。	切断中にエラーが発生した。しかし、切断は成功した。(関数は、SQL_SUCCESS_WITH_INFO を戻す。)
08003	接続がクローズしている。	引き数 *ConnectionHandle* に指定した接続がオープンしていない。
58004	予期しないシステム障害。	回復不能なシステム・エラー。
HY001	メモリーの割り振りの失敗。	DB2 CLI が、関数の実行または完了のサポートに必要なメモリーを割り当てることができない。
HY013	予期しないメモリー処理エラー。	DB2 CLI が、関数の実行または完了のサポートに必要なメモリーにアクセスできない。

制約事項:

なし。

関連した解説:

- 206 ページの『DB2 CLI 関数の説明の要点』
- 202 ページの『DB2 CLI 関数の要約』
- 208 ページの『SQLAllocHandle - ハンドルの割り当て』
- 220 ページの『SQLConnect - データ・ソースへの接続』
- 251 ページの『SQLFreeHandle - ハンドル・リソースの解放』

SQLEndTran - COMMIT または ROLLBACK の要求

目的:

仕様:	DB2 CLI	ODBC	ISO CLI

SQLEndTran() は、接続に関連するすべてのステートメントのすべてのアクティブな操作に対し COMMIT または ROLLBACK を要求します。

構文:

```
SQLRETURN   SQLEndTran      (SQLSMALLINT       HandleType,
                             SQLHANDLE         Handle,
                             SQLSMALLINT       Completion Type);
```

関数の引き数:

表 44. SQLEndTran の引き数

データ・タイプ	引き数	用途	説明
SQLSMALLINT	*HandleType*	入力	ハンドル・タイプ。
SQLHANDLE	*Handle*	入力	接続ハンドル。
SQLSMALLINT	*CompletionType*	入力	接続に関連したアクティブな操作の終了方法。

使用法:

HandleType
ハンドル・タイプ ID。SQL_HANDLE_DBC (接続ハンドル) のみが許可されます。

Handle *HandleType* が示すタイプのハンドルです。

CompletionType
以下のいずれかの値となります。

- SQL_COMMIT
- SQL_ROLLBACK

手動コミット・モードでは、SQLDisconnect() の前に SQLEndTran() を呼び出さなければなりません。SQLDisconnect() の前に SQLEndTran() を*呼び出さ*ないと、データベースを更新した (最終トランザクションの開始後に) 操作がロールバックされます。

ROLLBACK が実行されると、ステートメント・ハンドルはすべて消去されます。

アプリケーションが、手動モードの最中に破損または途中で終了すると、最後の COMMIT 以降に更新されたものは失われます。切断の前に SQLEndTran() を呼び出さなければなりません。

戻りコード:

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

診断:

表 45. SQLEndTran の SQLSTATE

SQLSTATE	説明	解説
58004	予期しないシステム障害。	回復不能なシステム・エラー。
HY001	メモリーの割り振りの失敗。	DB2 CLI が、関数の実行または完了のサポートに必要なメモリーを割り当てることができない。
HY013	予期しないメモリー処理エラー。	DB2 CLI が、関数の実行または完了のサポートに必要なメモリーにアクセスできない。
HY014	ハンドル数の限界に到達済み。	内部リソースが原因で、DB2 CLI がハンドルを割り当てることができない。

制約事項:

なし。

関連した解説:

- 206 ページの『DB2 CLI 関数の説明の要点』
- 202 ページの『DB2 CLI 関数の要約』
- 281 ページの『SQLSetConnectAttr — 接続に関するオプションの設定』

SQLError - エラー情報の検索

ODBC バージョン 3 では、SQLError() は使用すべきでない関数となり、SQLGetDiagRec() と SQLGetDiagField() に置き換えられました。詳細については、263 ページの『SQLGetDiagRec - 診断レコードの複数のフィールド設定値の取得』を参照してください。

推奨事項: このバージョンの DB2 CLI では、引き続き SQLError() をサポートしていますが、最新の標準に準拠するよう、DB2 CLI プログラムでは SQLGetDiagRec() を使用してください。

新しい関数へのマイグレーション

例えば、特定のステートメント・ハンドルに関連する診断情報を入手するためには、以下のステートメントを

```
SQLError(henv, hdbc, hstmt, szSqlState, pfNativeError, szErrorMsg,
    cbErrorMsgMax, pcbErrorMsg);
```

新しい関数を使用して次のように書き直すことができます。

```
SQLGetDiagRec(SQL_HANDLE_STMT, hstmt, 1, szSqlState, pfNativeError,
    szErrorMsg, cbErrorMsgMax, pcbErrorMsg);
```

関連した解説:

- 206 ページの『DB2 CLI 関数の説明の要点』
- 202 ページの『DB2 CLI 関数の要約』

SQLExecDirect - ステートメントの直接の実行

目的:

仕様:	DB2 CLI 1.1	ODBC 1.0	ISO CLI

SQLExecDirect() は、指定された SQL ステートメントを直接実行します。ステートメントは 1 回しか実行できません。

構文:

```
SQLRETURN   SQLExecDirect    (SQLHSTMT          StatementHandle,   /* hstmt */
                              SQLCHAR      *FAR StatementText,     /* szSqlStr */
                              SQLINTEGER        TextLength);       /* cbSqlStr */
```

関数の引き数:

表 46. SQLExecDirect の引き数

データ・タイプ	引き数	用途	説明
SQLHSTMT	*StatementHandle*	入力	ステートメント・ハンドル。
SQLCHAR *	*StatementText*	入力	SQL ステートメント・ストリング。
SQLINTEGER	*TextLength*	入力	*StatementText* 引き数の内容の長さ。長さは、ステートメントの正確な長さ、またはステートメントがヌルで終了している場合は、SQL_NTS に設定する必要がある。

使用法:

この SQL ステートメント・ストリングにはパラメーター・マーカーを入れることができません。

戻りコード:

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE
- SQL_NO_DATA_FOUND

SQL ステートメントが検索 UPDATE または検索 DELETE であり、検索条件を満たす行がない場合には、SQL_NO_DATA_FOUND が戻されます。

診断:

表 47. SQLExecDirect の SQLSTATE

SQLSTATE	説明	解説
22003	数値が範囲外。	数値タイプ列に対する数値割り当てによって、割り当て時または中間結果の計算中に、数値の整数部分の切り捨てが行われた。

表 47. SQLExecDirect の SQLSTATE (続き)

SQLSTATE	説明	解説
42xxx	構文エラーまたはアクセス規則違反。	**42**xxx SQLSTATE は、ステートメント関連のさまざまな構文またはアクセスに関する問題を示す。xxx は該当するクラス・コードを持ついずれかの SQLSTATE を示す。例えば、**42**xxx は、**42** クラスのいずれかの SQLSTATE を示す。
58004	予期しないシステム障害。	回復不能なシステム・エラー。
HY001	メモリーの割り振りの失敗。	DB2 CLI が、関数の実行または完了のサポートに必要なメモリーを割り当てることができない。
HY009	無効な引き数値。	*StatementText* がヌル・ポインターである。
HY013	予期しないメモリー処理エラー。	DB2 CLI が、関数の実行または完了のサポートに必要なメモリーにアクセスできない。
HY014	ハンドル数の限界に到達済み。	内部リソースが原因で、DB2 CLI がハンドルを割り当てることができない。
HY090	無効なストリング長またはバッファー長。	引き数 *TextLength* が 1 より小さいが、SQL_NTS と等しくない。

制約事項:

なし。

関連した解説:

- 206 ページの『DB2 CLI 関数の説明の要点』
- 202 ページの『DB2 CLI 関数の要約』
- 212 ページの『SQLBindCol - アプリケーション変数への列のバインド』

SQLExecute - ステートメントの実行

目的:

仕様:	**DB2 CLI** 1.1	**ODBC** 1.0	**ISO CLI**

SQLExecute() は、SQLPrepare() を使用して正常に準備されたステートメントを、1 回または複数回実行します。ステートメントは、SQLBindParameter() によってパラメーター・マーカーにバインドされたアプリケーション変数の現行値を使用して実行されます。

構文:

```
SQLRETURN   SQLExecute      (SQLHSTMT         StatementHandle);  /* hstmt */
```

関数の引き数:

表 48. SQLExecute の引き数

データ・タイプ	引き数	用途	説明
SQLHSTMT	*StatementHandle*	入力	ステートメント・ハンドル。

使用法:

SQL ステートメント・ストリングにはパラメーター・マーカーが含まれている場合があります。パラメーター・マーカーは ? 文字によって表され、SQLExecute() の呼び出し時に、アプリケーション提供の値が置き換えられる、ステートメント内の位置を示すために使用されます。この値は、アプリケーション変数から入手することができます。アプリケーションのストレージ域をパラメーター・マーカーにバインドするには、SQLBindParameter() を使用します。

SQLExecute() を呼び出す前に、すべてのパラメーターをバインドする必要があります。

SQLExecute() 呼び出しの結果を処理した後、アプリケーションは新規 (または同じ) パラメーター値を使って、そのステートメントを再度実行することができます。

SQLExecDirect() によって実行したステートメントは、SQLExecute() を呼び出しても再実行できません。SQLPrepare() を最初に呼び出す必要があります。

結果セットが生成されている場合、SQLFetch() は、データの次の行を取り出してバインド変数に入れます。データは、バインドされていない列に関して SQLGetData() を呼び出すことによっても取り出すことができます。

戻りコード:

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE
- SQL_NO_DATA_FOUND

SQL ステートメントが検索 UPDATE または検索 DELETE であり、検索条件を満たす行がない場合には、SQL_NO_DATA_FOUND が戻されます。

診断:

SQLExecute() に関する SQLSTATE には、**HY**009 と **HY**090 以外の、SQLExecDirect() に関するすべての SQLSTATE (234 ページの表 47 を参照) が含まれます。また、表 49 の SQLSTATE も含まれます。

表 49. SQLExecute の SQLSTATE

SQLSTATE	説明	解説
HY010	関数のシーケンス・エラー。	指定した *StatementHandle* が準備済み状態ではない。最初に SQLPrepare() を呼び出さないで、SQLExecute() を呼び出した。
08004	アプリケーション・サーバーが接続をリジェクトした。	データ・ソースへの接続に使用するユーザー名またはパスワードが正しくない。
08S01	通信リンク障害。	関数の完了前に、アプリケーションとデータ・ソース間の通信リンクに障害が生じた。
39001	ユーザー定義関数が無効な SQLSTATE を戻した。	ユーザー定義関数が無効な SQLSTATE を戻した。

表 49. SQLExecute の SQLSTATE (続き)

SQLSTATE	説明	解説
59101	ユーザーが定義されていない。	ユーザーがモバイル・デバイス管理センター制御データベースに定義されていない。
59102	パスワードが正しくない。	ユーザー・パスワードがモバイル・デバイス管理センターに定義されているパスワードと一致しない。
59103	グループが定義されていない。	グループがモバイル・デバイス管理センターに定義されていない。
59104	アプリケーションが定義されていない。	アプリケーションがモバイル・デバイス管理センターに定義されていない。
59105	サブスクリプションが定義されていない。	「AgentAdapter」を使用するサブスクリプションがモバイル・デバイス管理センターに定義されていない。
59106	サブスクリプションが完了していない。	サブスクリプションが、リモート・ストアード・プロシージャーを起動するために必要なすべての情報を保有していない。
59120	XML 変換エラー。	ユーザー入力データを XML 文書に変換するときに、AgentAdapter が失敗した。
59121	AgentAdapter の一般エラー。	AgentAdapter の一般エラー。
59122	ライブラリーのロードが失敗した。	必須ライブラリーのいくつかが、システムで見つからない。
HY501	DataSource 名が無効。	指定されたデータ・ソース名が無効。

制約事項:

なし。

関連した解説:

- 206 ページの『DB2 CLI 関数の説明の要点』
- 202 ページの『DB2 CLI 関数の要約』
- 215 ページの『SQLBindParameter - バッファーへのパラメーター・マーカーのバインド』
- 212 ページの『SQLBindCol - アプリケーション変数への列のバインド』
- 234 ページの『SQLExecDirect - ステートメントの直接の実行』
- 275 ページの『SQLPrepare - ステートメントの準備』
- 『SQLFetch - 次の行の取り出し』

SQLFetch - 次の行の取り出し

目的:

仕様:	**DB2 CLI** 1.1	**ODBC** 1.0	

SQLFetch() はカーソルを結果セットの次の行に進めて、バインドされた列を取り出します。

列はアプリケーションのストレージにバインドすることができます。

SQLFetch() が呼び出されると、適切なデータ転送が行われます。列のバインド時に変換が指示されている場合は、データ変換も同時に行われます。 SQLGetData() を呼び出すことによって、取り出しの後に、列を個別に受け取ることもできます。

SQLFetch() は、照会の実行によって (同じステートメント・ハンドルを使用して) 結果セットを生成した後でのみ、呼び出すことができます。

構文:

```
SQLRETURN   SQLFetch        (SQLHSTMT          StatementHandle);  /* hstmt */
```

関数の引き数:

表 50. SQLFetch の引き数

データ・タイプ	引き数	用途	説明
SQLHSTMT	*StatementHandle*	入力	ステートメント・ハンドル。

使用法:

SQLFetch() は、同じステートメント・ハンドル上で結果セットが生成された後でのみ、呼び出すことができます。最初に SQLFetch() を呼び出す前に、カーソルを結果セットの先頭の前に位置付けます。

SQLBindCol() にバインドされるアプリケーション変数の数は、結果セット内の列数を超えてはなりません。超えた場合、SQLFetch() は失敗します。

列のバインドのために SQLBindCol() が呼び出されていない場合、SQLFetch() はデータをアプリケーションに戻さないで、カーソルを進めるだけです。このような場合、すべての列を個別に入手するには、SQLGetData() を呼び出します。
SQLFetch() がカーソルを次の行に進めると、アンバインド列のデータは廃棄されます。

列はアプリケーションのストレージにバインドすることができます。アプリケーションのストレージを列にバインドするには、SQLBindCol() を使用します。データは取り出し時にデータベースからアプリケーションに転送されます。戻すことのできるデータの長さも設定されます。

バインド・ストレージ・バッファーの大きさが SQLFetch() によって戻されるデータを収容するのに十分でない場合、データは切り捨てられます。文字データが切り捨てられた場合 SQL_SUCCESS_WITH_INFO が戻され、切り捨てを示す SQLSTATE が生成されます。SQLBindCol() 据え置き出力引き数 *pcbValue* に、サーバーから取り出された列データの実際の長さが入ります。アプリケーションでは実際の出力の長さと入力バッファーの長さ (SQLBindCol() からの *pcbValue* と *cbValueMax* 引き数) を比較して、切り捨てられた文字列を判別する必要があります。

小数点の右側の数字を切り捨てた場合、数値データ・タイプの切り捨ては警告として報告されます。切り捨てが小数点の左側で発生した場合、エラーが戻されます (診断のセクションを参照してください)。

結果セットからすべての行を取り出す場合、または結果セットの残りの行が必要ない場合には、SQLFreeStmt() を呼び出してカーソルをクローズし、残りのデータおよび関連するリソースを廃棄します。

DB2 Everyplace は、行セットを使用する代わりに、一度に 1 行ずつ取り出します。DB2 Everyplace はステートメント記述子をサポートしません。

SQLFetch() は、アプリケーションが別々の長さバッファーと標識バッファーを指定しているかどうかを判別します。指定している場合、データが NULL でないときは、SQLFetch() は標識バッファーを 0 に設定し、長さを長さバッファーに戻します。データが NULL のときは、 SQLFetch() は標識バッファーを SQL_NULL_DATA に設定し、長さバッファーは変更しません。

カーソルの位置決め:

結果セットが作成されると、カーソルは結果セットの先頭の前に位置付けされます。 SQLFetch() は次の行を取り出します。

戻りコード:

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE
- SQL_NO_DATA_FOUND

結果セットに行がないか、または前の SQLFetch() 呼び出しが結果セットからすべての行を取り出した場合には、SQL_NO_DATA_FOUND が戻されます。

すべての行が取り出されると、カーソルは結果セットの末尾の後に位置付けされます。

診断:

表 51. SQLFetch の SQLSTATE

SQLSTATE	説明	解説
01004	データが切り捨てられた。	1 つ以上の列に対して戻されたデータが切り捨てられる。ストリング値または数値は右側が切り捨てられる。(エラーがない場合には、SQL_SUCCESS_WITH_INFO が戻される。)
07006	無効な変換。	SQLBindCol() の *fCType* により指定したデータ・タイプへのデータ値の変換が無意味である。
22002	無効な出力バッファーまたは標識バッファーの指定。	SQLBindCol() 内の引き数 *pcbValue* に指定したポインター値がヌル・ポインターで、対応する列の値がヌルである。 SQL_NULL_DATA を報告する方法がない。
58004	予期しないシステム障害。	回復不能なシステム・エラー。

表 51. SQLFetch の SQLSTATE (続き)

SQLSTATE	説明	解説
HY001	メモリーの割り振りの失敗。	DB2 CLI が、関数の実行または完了のサポートに必要なメモリーを割り当てることができない。
HY010	関数のシーケンス・エラー。	*StatementHandle* に対して、SQLPrepare() または SQLExecDirect() が呼び出される前に関数が呼び出された。
HY013	予期しないメモリー処理エラー。	DB2 CLI が、関数の実行または完了のサポートに必要なメモリーにアクセスできない。

制約事項:

なし。

関連した解説:

- 206 ページの『DB2 CLI 関数の説明の要点』
- 202 ページの『DB2 CLI 関数の要約』
- 212 ページの『SQLBindCol - アプリケーション変数への列のバインド』
- 234 ページの『SQLExecDirect - ステートメントの直接の実行』
- 259 ページの『SQLGetData - 列からのデータの入手』

SQLFetchScroll - 行セットの取り出しと、バインドされたすべての列のデータの戻し

目的:

仕様:	DB2 CLI 5.0	ODBC 3.0	

SQLFetchScroll() は、指定した行セットのデータを結果セットから取り出し、すべてのバインドされた列のデータを戻します。行セットは、絶対位置または相対位置で指定できます。

構文:

```
SQLRETURN   SQLFetchScroll  (
                  SQLHSTMT          StatementHandle,
                  SQLSMALLINT       FetchOrientation,
                  SQLINTEGER        FetchOffset);
```

関数の引き数:

表 52. SQLFetchScroll の引き数

データ・タイプ	引き数	用途	説明
SQLHSTMT	*StatementHandle*	入力	ステートメント・ハンドル。

表 52. *SQLFetchScroll* の引き数 (続き)

データ・タイプ	引き数	用途	説明
SQLSMALLINT	*FetchOrientation*	入力	フェッチのタイプ: • SQL_FETCH_NEXT • SQL_FETCH_PRIOR • SQL_FETCH_FIRST • SQL_FETCH_LAST • SQL_FETCH_ABSOLUTE • SQL_FETCH_RELATIVE
SQLINTEGER	*FetchOffset*	入力	取り出す行数。*FetchOrientation* 引き数の値によって、この引き数の解釈は変わります。

使用法:

SQLFetchScroll() は、結果セットから指定した行セットを戻します。行セットは、絶対位置または相対位置で指定できます。SQLFetchScroll() は、結果セットが存在するときだけ呼び出すことができます。つまり、結果セットを作成する呼び出しの後から、その結果セットに関するカーソルがクローズされるまでの間です。バインドされた列があれば、それらの列にデータが戻されます。取り出した行数を入れるバッファーまたは行状況配列を指すポインターをアプリケーションが指定している場合は、SQLFetchScroll() は、その情報も戻します。SQLFetchScroll() の呼び出しと SQLFetch() の呼び出しを混ぜることもできます。

カーソルの位置決め:

結果セットが作成されると、カーソルは結果セットの先頭の前に位置付けされます。次のリストに示されているように SQLFetchScroll() は、 *FetchOrientation* と *FetchOffset* 引き数の値に基づいてブロック・カーソルの位置を指定します。新規行セットの先頭を決める厳密な規則は、次のセクションで説明します。

FetchOrientation	意味
SQL_FETCH_NEXT	次の行セットを戻します。これは、SQLFetch() 呼び出しと同じです。 SQLFetchScroll() は、*FetchOffset* の値を無視します。
SQL_FETCH_PRIOR	前の行セットを戻します。 SQLFetchScroll() は、*FetchOffset* の値を無視します。
SQL_FETCH_RELATIVE	現行の行セットの先頭から、行セット *FetchOffset* を戻します。
SQL_FETCH_ABSOLUTE	行 *FetchOffset* から始まる行セットを戻します。
SQL_FETCH_FIRST	結果セット内の最初の行セットを戻します。SQLFetchScroll() は、*FetchOffset* の値を無視します。
SQL_FETCH_LAST	結果セット内の最後の完全な行セットを戻します。SQLFetchScroll() は、*FetchOffset* の値を無視します。

SQL_ATTR_ROW_ARRAY_SIZE ステートメント属性は、行セット内の行数を指定します。 SQLFetchScroll() で取り出す行セットが結果セットの終わりとオーバー

ラップしている場合、SQLFetchScroll() は部分的な行セットを戻します。つまり、S が取り出す行セットの開始行、R が行セットのサイズ、L が結果セット内の最後の行として、S + R - 1 が L より大である場合は、行セットの内の最初から L - S + 1 行が有効な行となります。残りの行の状況は SQL_ROW_NOROW で、空です。

SQLFetchScroll() が戻ると、行セット・カーソルは結果セットの最初の行に配置されます。

カーソル位置指定の規則:

以下のセクションでは、*FetchOrientation* のそれぞれの値についての厳密な規則を説明しています。これらの規則では、以下の表記を使用しています。

Before start
ブロック・カーソルが結果セットの先頭の前に位置付けられています。新規行セットの最初の行が結果セットの先頭よりも前であれば、SQLFetchScroll() は SQL_NO_DATA を戻します。

After end
ブロック・カーソルが結果セットの終わりより後に位置付けられています。新規行セットの最初の行が結果セットの終わりよりも後であれば、SQLFetchScroll() は SQL_NO_DATA を戻します。

CurrRowsetStart
現行の行セットの最初の行の番号。

LastResultRow
結果セットの最後の行の番号。

RowsetSize
行セット・サイズ。

FetchOffset
FetchOffset 引き数の値。

SQL_FETCH_NEXT の規則:

表 53. SQL_FETCH_NEXT の規則:

状態	新規行セットの最初の行
Before start	1
CurrRowsetStart + RowsetSize <= LastResultRow	CurrRowsetStart + RowsetSize
CurrRowsetStart + RowsetSize > LastResultRow	After end
After end	After end

SQL_FETCH_PRIOR の規則:

表 54. SQL_FETCH_PRIOR の規則:

状態	新規行セットの最初の行
Before start	Before start
CurrRowsetStart = 1	Before start
1 < CurrRowsetStart <= RowsetSize	1[a]

表 54. SQL_FETCH_PRIOR の規則: (続き)

状態	新規行セットの最初の行
CurrRowsetStart > RowsetSize	CurrRowsetStart - RowsetSize
After end AND LastResultRow < RowsetSize	1[a]
After end AND LastResultRow >= RowsetSize	LastResult - RowRowsetSize + 1

a SQLFetchScroll() は SQLSTATE 01S06 (結果セットが最初の行セットを戻す前に取り出そうとした) と SQL_SUCCESS_WITH_INFO を戻します。

SQL_FETCH_RELATIVE の規則:

表 55. SQL_FETCH_RELATIVE の規則:

状態	新規行セットの最初の行
(Before start AND *FetchOffset* > 0) OR (After end AND *FetchOffset* < 0)	--[a]
Before start AND *FetchOffset* <= 0	Before start
CurrRowsetStart = 1 AND *FetchOffset* < 0	Before start
CurrRowsetStart > 1 AND CurrRowsetStart + *FetchOffset* < 1 AND |*FetchOffset*| > RowsetSize	Before start
CurrRowsetStart > 1 AND CurrRowsetStart + *FetchOffset* < 1 AND |*FetchOffset*| <= RowsetSize	1[b]
1 <= CurrRowsetStart + *FetchOffset* <= LastResultRow	CurrRowsetStart + *FetchOffset*
CurrRowsetStart + *FetchOffset* > LastResultRow	After end
After end AND *FetchOffset* >= 0	After end

a SQLFetchScroll() は、 *FetchOrientation* を SQL_FETCH_ABSOLUTE にセットして呼び出した場合と同じ行セットを戻します。詳細については、SQL_FETCH_ABSOLUTE セクションを参照してください。

b SQLFetchScroll() は SQLSTATE 01S06 (結果セットが最初の行セットを戻す前に取り出そうとした) と SQL_SUCCESS_WITH_INFO を戻します。

SQL_FETCH_ABSOLUTE の規則:

表 56. SQL_FETCH_ABSOLUTE の規則:

状態	新規行セットの最初の行
FetchOffset < 0 AND |*FetchOffset*| <= LastResultRow	LastResultRow + *FetchOffset* + 1
FetchOffset < 0 AND |*FetchOffset*| > LastResultRow AND |*FetchOffset*| > RowsetSize	Before start
FetchOffset < 0 AND |*FetchOffset*| > LastResultRow AND |*FetchOffset*| <= RowsetSize	1[a]
FetchOffset = 0	Before start
1 <= *FetchOffset* <= LastResultRow	*FetchOffset*
FetchOffset > LastResultRow	After end

a SQLFetchScroll() は SQLSTATE 01S06 (結果セットが最初の行セットを戻す前に取り出そうとした) と SQL_SUCCESS_WITH_INFO を戻します。

SQL_FETCH_FIRST の規則:

表 57. SQL_FETCH_FIRST の規則:

状態	新規行セットの最初の行
すべて	1

SQL_FETCH_LAST の規則:

表 58. SQL_FETCH_LAST の規則:

状態	新規行セットの最初の行
RowsetSize <= LastResultRow	LastResultRow - RowsetSize + 1
RowsetSize > LastResultRow	1

バインドされた列へのデータの戻し:

SQLFetchScroll() は、SQLFetch() と同じようにバインドされた列にデータを戻します。詳細は、237 ページの『SQLFetch - 次の行の取り出し』を参照してください。

バインドされた列がない場合は、SQLFetchScroll() はデータは戻さずに、ブロック・カーソルを指定された位置に移動させます。このケースでは、SQLFetch() と同様に、SQLGetData() を使用して情報を取り出すことができます。

バッファー・アドレス:

SQLFetchScroll() は SQLFetch() と同じ数式を使用してデータのアドレスと、長さ/標識バッファーを判別します。詳細については、212 ページの『SQLBindCol - アプリケーション変数への列のバインド』を参照してください。

行状況配列:

行状況配列は、行セット内の各行の状況を戻すために使用されます。この配列のアドレスは、SQL_ATTR_ROW_STATUS_PTR ステートメント属性で指定します。この配列はアプリケーションで割り振り、SQL_ATTR_ROW_ARRAY_SIZE ステートメント属性で指定されただけのエレメントがなければなりません。この値は、SQLFetch() および SQLFetchScroll() でセットされます。SQL_ATTR_ROW_STATUS_PTR ステートメント属性の値がヌル・ポインターである場合は、これらの関数は行の状況を戻しません。

SQLFetch() または SQLFetchScroll() が、 SQL_SUCCESS あるいは SQL_SUCCESS_WITH_INFO を戻していない場合は、行状況配列バッファーの内容は不確定です。

以下の値が行状況配列に戻されます。

行状況配列の値	説明
SQL_ROW_SUCCESS	行が正常に取り出された。

SQL_ROW_SUCCESS_WITH_INFO
行が正常に取り出された。しかし、この行に関して警告が戻った。

SQL_ROW_ERROR
行の取り出し中にエラーが発生した。

SQL_ROW_NOROW
行セットが結果セットの終わりとオーバーラップしており、行状況配列のこのエレメントに対応する行が戻らなかった。

取り出された行バッファー:

取り出された行バッファーは、取り出された (取り出し中にエラーが起きたためにデータが戻らなかった行を含む) 行数を戻すために使用されます。これは行状況配列の値が SQL_ROW_NOROW でない行の数です。このバッファーのアドレスは、SQL_ATTR_ROWS_FETCHED_PTR ステートメント属性で指定します。このバッファーは、アプリケーションが割り振ります。これは、SQLFetch() および SQLFetchScroll() でセットされます。 SQL_ATTR_ROWS_FETCHED_PTR ステートメント属性の値がヌル・ポインターであれば、これらの関数は取り出した行の番号を戻しません。結果セット内の現在行の番号を判断するには、アプリケーションで SQLGetStmtAttr() に SQL_ATTR_ROW_NUMBER 属性を指定して呼び出します。

SQLFetch() または SQLFetchScroll() が、 SQL_SUCCESS あるいは SQL_SUCCESS_WITH_INFO を戻していない場合は、行取り出しバッファーの内容は不確定です。ただし SQL_NO_DATA が戻った場合は例外であり、行取り出しバッファーはゼロにセットされます。

戻りコード:
- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_NO_DATA
- SQL_ERROR
- SQL_INVALID_HANDLE

診断:

表 59. SQLFetchScroll SQLSTATE

SQLSTATE	説明	解説
01000	警告	通知メッセージ。(関数は、SQL_SUCCESS_WITH_INFO を戻す。)
01004	データが切り捨てられた。	1 つ以上の列に対して戻されたデータが切り捨てられる。ストリング値または数値は右側が切り捨てられる。(エラーがない場合には、SQL_SUCCESS_WITH_INFO が戻される。)

表 59. SQLFetchScroll SQLSTATE (続き)

SQLSTATE	説明	解説
01S06	結果セットが最初の行セットを戻す前に、取り出そうとした。	現在位置が最初の行より後であり、かつ、FetchOrientation が SQL_PRIOR であるか、または SQL_RELATIVE であって、絶対値が現在の SQL_ATTR_ROW_ARRAY_SIZE 以下である負の FetchOffset が指定されている際に、要求された行セットが結果セットの先頭とオーバーラップした。(関数は、SQL_SUCCESS_WITH_INFO を戻す。)
07006	無効な変換。	SQLBindCol() の *fCType* により指定したデータ・タイプへのデータ値の変換が無意味である。
22002	無効な出力バッファーまたは標識バッファーの指定。	SQLBindCol() 内の引き数 *pcbValue* に指定したポインター値がヌル・ポインターで、対応する列の値がヌルである。 SQL_NULL_DATA を報告する方法がない。
22003	数値が範囲外。	1 つ以上のバインドされた列について (数値またはストリングとして) 数値を戻すと、(部分的にではなく) その番号全体が切り捨てられる。
24000	カーソル状態が無効。	*StatementHandle* は実行された状態であるが、*StatementHandle* に関連付けられた結果セットがない。
HY000	一般エラー。	エラーは発生したが、特定の SQLSTATE は無い。SQLGetDiagRec() が **MessageText* バッファーに戻したエラー・メッセージが、エラーとその原因を示しています。
HY001	メモリーの割り振りの失敗。	DB2 CLI が、関数の実行または完了のサポートに必要なメモリーを割り当てることができない。
HY010	関数のシーケンス・エラー。	*StatementHandle* に対して、SQLPrepare() または SQLExecDirect() が呼び出される前に関数が呼び出された。
HY106	取り出しタイプが範囲外。	引き数 *FetchOrientation* に指定された値が無効である。 SQL_CURSOR_TYPE ステートメント属性の値が SQL_CURSOR_FORWARD_ONLY で、*FetchOrientation* 引き数の値が SQL_FETCH_NEXT でない。
HY107	行の値が範囲外。	SQL_ATTR_CURSOR_TYPE ステートメント属性で指定した値は SQL_CURSOR_KEYSET_DRIVEN であるが、SQL_ATTR_KEYSET_SIZE ステートメント属性で指定した値がゼロより大きく、SQL_ATTR_ROW_ARRAY_SIZE ステートメント属性で指定した値より小さい。

制約事項:

なし。

関連した解説:

- 206 ページの『DB2 CLI 関数の説明の要点』
- 202 ページの『DB2 CLI 関数の要約』
- 212 ページの『SQLBindCol - アプリケーション変数への列のバインド』
- 228 ページの『SQLDescribeCol - 列の属性セットの戻し』
- 234 ページの『SQLExecDirect - ステートメントの直接の実行』
- 237 ページの『SQLFetch - 次の行の取り出し』
- 235 ページの『SQLExecute - ステートメントの実行』
- 274 ページの『SQLNumResultCols - 結果の列数の入手』
- 285 ページの『SQLSetStmtAttr - ステートメントに関するオプションの設定』

SQLForeignKeys - 外部キー列のリストの入手

目的:

仕様:	**DB2 CLI** 2.1	**ODBC** 1.0	

SQLForeignKeys() は指定された表の外部キーに関する情報を戻します。この情報は SQL 結果セットに戻されます。この結果セットは、照会で生成された結果セットの取り出しに使用するものと同じ関数を使用して処理することができます。*PKCatalogName*、*NameLength1*、*PKSchemaName*、*NameLength2*、*FKCatalogName*、*NameLength4*、*FKSchemaName*、*NameLength5* は無視されます。戻された結果セットの列 1、2、5、6、12、および 13 は、常に長さがゼロのストリングです。戻された結果セットの列 10、11、および 14 は常にゼロ。

構文:

```
SQLRETURN   SQLForeignKeys  (
                  SQLHSTMT         StatementHandle,  /* hstmt */
                  SQLCHAR      *FAR PKCatalogName,   /* szPkCatalogName */
                  SQLSMALLINT      NameLength1,      /* cbPkCatalogName */
                  SQLCHAR      *FAR PKSchemaName,    /* szPkSchemaName */
                  SQLSMALLINT      NameLength2,      /* cbPkSchemaName */
                  SQLCHAR      *FAR PKTableName,     /* szPkTableName */
                  SQLSMALLINT      NameLength3,      /* cbPkTableName */
                  SQLCHAR      *FAR FKCatalogName,   /* szFkCatalogName */
                  SQLSMALLINT      NameLength4       /* cbFkCatalogName */
                  SQLCHAR      *FAR FKSchemaName,    /* szFkSchemaName */
                  SQLSMALLINT      NameLength5,      /* cbFkSchemaName */
                  SQLCHAR      *FAR FKTableName,     /* szFkTableName */
                  SQLSMALLINT      NameLength6);     /* cbFkTableName */
```

関数の引き数:

表 60. SQLForeignKeys の引き数

データ・タイプ	引き数	用途	説明
SQLHSTMT	*StatementHandle*	入力	ステートメント・ハンドル。
SQLCHAR*	*PKCatalogName*	入力	主キー表のカタログ修飾子。このフィールドは、DB2 Everyplace では無視されます。
SQLSMALLINT	*NameLength1*	入力	*PKCatalogName* の長さ。このフィールドは、DB2 Everyplace では無視されます。
SQLCHAR*	*PKSchemaName*	入力	主キー表のスキーマ修飾子。このフィールドは、DB2 Everyplace では無視されます。
SQLSMALLINT	*NameLength2*	入力	*PKSchemaName* の長さ。このフィールドは、DB2 Everyplace では無視されます。

表 60. *SQLForeignKeys* の引き数 (続き)

データ・タイプ	引き数	用途	説明
SQLCHAR*	*PKTableName*	入力	主キーが入っている表の名前。
SQLSMALLINT	*NameLength3*	入力	*PKTableName* の長さ。
SQLCHAR*	*FKCatalogName*	入力	外部キーが入っている表のカタログ修飾子。このフィールドは、DB2 Everyplace では無視されます。
SQLSMALLINT	*NameLength4*	入力	*FKCatalogName* の長さ。このフィールドは、DB2 Everyplace では無視されます。
SQLCHAR*	*FKSchemaName*	入力	外部キーが入っている表のスキーマ修飾子。このフィールドは、DB2 Everyplace では無視されます。
SQLSMALLINT	*NameLength5*	入力	*FKSchemaName* の長さ。このフィールドは、DB2 Everyplace では無視されます。
SQLCHAR*	*FKTableName*	入力	外部キーが入っている表の名前。
SQLSMALLINT	*NameLength6*	入力	*FKTableName* の長さ。

使用法:

PKTableName に表の名前が入っていて、*FKTableName* が空ストリングである場合は、`SQLForeignKeys()` は指定した表の主キーと、その表を参照する (他の表に入っている) すべての外部キーが入っている結果セットを戻します。

FKTableName に表の名前が入っていて、*PKTableName* が空ストリングである場合は、`SQLForeignKeys()` は指定した表にあるすべての外部キーとそれらの外部キーが参照する (他の表にある) 主キーが入った結果セットを戻します。

PKTableName と *FKTableName* の両方に表の名前が入っている場合は、`SQLForeignKeys()` は *PKTableName* に指定されている表の主キーを参照する、*FKTableName* に指定された表の中にある外部キーを戻します。このキーは、1 つのはずです。

主キーに関連付けられた外部キーが要求されると、結果セットは FKTABLE_NAME および ORDINAL_POSITION 順に配列されます。外部キーに関連付けられた主キーが要求されると、結果セットは PKTABLE_NAME および ORDINAL_POSITION 順に配列されます。

カタログ関数結果セットの VARCHAR 列は、SQL92 の制限と一貫性を持たせるために最大長属性 128 で宣言されています。

将来のリリースで新規の列が追加されたり、既存の列の名前が変更されても、現在の列の位置は変わりません。

結果セットには、以下の列が含まれています:

列 1 PKTABLE_CAT (VARCHAR(128))
これは常に長さゼロのストリングです。

列 2 PKTABLE_SCHEM (VARCHAR(128))
これは常に長さゼロのストリングです。

列 3 PKTABLE_NAME (VARCHAR(128) 非ヌル)
主キーが入っている表の名前。

列 4 PKCOLUMN_NAME (VARCHAR(128) 非ヌル)
主キーの列名。

列 5 FKTABLE_CAT (VARCHAR(128))
これは常に長さゼロのストリングです。

列 6 FKTABLE_SCHEM (VARCHAR(128))
これは常に長さゼロのストリングです。

列 7 FKTABLE_NAME (VARCHAR(128) 非ヌル)
外部キーが入っている表の名前。

列 8 FKCOLUMN_NAME (VARCHAR(128) 非ヌル)
外部キーの列名。

列 9 ORDINAL_POSITION (SMALLINT 非ヌル)
キー内の列の順序を示す位置。1 から始まります。

列 10 UPDATE_RULE (SMALLINT)
これは常にゼロです。

列 11 DELETE_RULE (SMALLINT)
これは常にゼロです。

列 12 FK_NAME (VARCHAR(128))
これは常に長さゼロのストリングです。

列 13 PK_NAME (VARCHAR(128))
これは常に長さゼロのストリングです。

列 14 DEFERRABILITY (SMALLINT)
これは常にゼロです。

DB2 CLI が使用する列名は、X/Open CLI CAE 仕様のスタイルに準拠しています。列のタイプ、内容、順序は、ODBC で `SQLForeignKeys()` 結果セット用に定義されているものと同じです。

戻りコード:
- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

診断:

表 61. SQLForeign SQLSTATE

SQLSTATE	説明	解説
24000	カーソル状態が無効。	カーソルがすでにこのステートメント・ハンドルでオープンされている。
40003 **08**S01	通信リンク障害。	関数の完了前に、アプリケーションとデータ・ソース間の通信リンクに障害が生じた。
HY001	メモリーの割り振りの失敗。	DB2 CLI が、関数の実行または完了のサポートに必要なメモリーを割り当てることができない。
HY009	無効な引き数値。	引き数 *PKTableName* と *FKTableName* がともにヌル・ポインターである。

表 61. SQLForeign SQLSTATE (続き)

SQLSTATE	説明	解説
HY010	関数のシーケンス・エラー。	実行時データ (SQLPrepare() または SQLExecDirect()) の操作中に、関数が呼び出された。
HY014	ハンドル数の限界に到達済み。	内部リソースが原因で、DB2 CLI がハンドルを割り当てることができない。
HY090	無効なストリング長またはバッファー長。	名前の長さ引き数のいずれかの値が、ゼロより小であるが、SQL_NTS と等しくない。

制約事項:

なし。

関連した解説:

- 206 ページの『DB2 CLI 関数の説明の要点』
- 202 ページの『DB2 CLI 関数の要約』
- 277 ページの『SQLPrimaryKeys - 表の主キー列の入手』

SQLFreeConnect - 接続ハンドルの解放

ODBC バージョン 3 では、SQLFreeConnect() は使用すべきでない関数となり、SQLFreeHandle() に置き換えられました。詳細については、251 ページの『SQLFreeHandle - ハンドル・リソースの解放』を参照してください。

推奨事項: このバージョンの DB2 CLI では、引き続き SQLFreeConnect() をサポートしますが、最新の標準に準拠するように、DB2 CLI プログラムでは SQLFreeHandle() を使用してください。

新しい関数へのマイグレーション

例えば、以下のステートメントは、

```
SQLFreeConnect(hdbc);
```

新しい関数を使用して次のように書き直すことができます。

```
SQLFreeHandle(SQL_HANDLE_DBC, hdbc);
```

関連した解説:

- 206 ページの『DB2 CLI 関数の説明の要点』
- 202 ページの『DB2 CLI 関数の要約』

SQLFreeEnv - 環境ハンドルの解放

ODBC バージョン 3 では、SQLFreeEnv() は使用すべきでない関数となり、SQLFreeHandle() に置き換えられました。詳細については、251 ページの『SQLFreeHandle - ハンドル・リソースの解放』を参照してください。

推奨事項: このバージョンの DB2 CLI では、引き続き SQLFreeEnv() をサポートしますが、最新の標準に準拠するよう、DB2 CLI プログラムでは SQLFreeHandle() を使用してください。

新しい関数へのマイグレーション

例えば、以下のステートメントは、

```
SQLFreeEnv(henv);
```

新しい関数を使用して次のように書き直すことができます。

```
SQLFreeHandle(SQL_HANDLE_ENV, henv);
```

関連した解説:

- 206 ページの『DB2 CLI 関数の説明の要点』
- 202 ページの『DB2 CLI 関数の要約』

SQLFreeHandle - ハンドル・リソースの解放

目的:

仕様:	DB2 CLI 5.0	ODBC 3.0	ISO CLI

SQLFreeHandle() は、特定の環境、接続、またはステートメント・ハンドルと関連したリソースを解放します。

この関数はリソースを解放するための汎用関数です。この関数は、SQLFreeConnect (接続ハンドルの解放用) と SQLFreeEnv() (環境ハンドルの解放用) に代わる関数です。SQLFreeHandle() は、ステートメント・ハンドルを解放するための SQLFreeStmt() (SQL_DROP を伴う) に代わる関数でもあります。

構文:

```
SQLRETURN   SQLFreeHandle    (SQLSMALLINT       HandleType,
                              SQLHANDLE         Handle);
```

関数の引き数:

表 62. SQLFreeHandle の引き数

データ・タイプ	引き数	用途	説明
SQLSMALLINT	*HandleType*	入力	SQLFreeHandle() によって解放されるハンドルのタイプ。以下のいずれかの値でなければならない。 SQL_HANDLE_ENV SQL_HANDLE_DBC SQL_HANDLE_STMT *HandleType* が上記のいずれかの値でない場合、SQLFreeHandle() は SQL_INVALID_HANDLE を戻す。
SQLHANDLE	*Handle*	入力	解放するハンドルの名前。

使用法:

SQLFreeHandle() は、環境、接続、およびステートメント用のハンドル解放するために使用します。

アプリケーションではハンドルの解放後にハンドルを使用してはなりません。DB2 CLI は関数呼び出しにおけるハンドルの妥当性をチェックしません。

環境ハンドルの解放:

SQL_HANDLE_ENV の *HandleType* を指定して SQLFreeHandle() を呼び出す前に、アプリケーションでは、環境に割り当てたすべての接続に関して SQL_HANDLE_DBC の *HandleType* を指定して SQLFreeHandle() を呼び出す必要があります。そうしないと、SQLFreeHandle() の呼び出しによって SQL_ERROR が戻され、この環境とアクティブな接続は有効なまま残ります。

接続ハンドルの解放:

SQL_HANDLE_DBC の *HandleType* を指定して SQLFreeHandle() を呼び出す前に、アプリケーションでは接続に関して SQLDisconnect() を呼び出す必要があります。そうしないと、SQLFreeHandle() の呼び出しによって SQL_ERROR が戻され、接続は有効のままとなります。

ステートメント・ハンドルの解放:

SQL_HANDLE_STMT の *HandleType* を指定して SQLFreeHandle() を呼び出すと、SQL_HANDLE_STMT の *HandleType* を指定した SQLAllocHandle() への呼び出しによって割り当てられたすべてのリソースが解放されます。アプリケーションで SQLFreeHandle() を呼び出して、保留中の結果を持つステートメントを解放すると、保留中の結果は削除されます。 SQLFreeHandle() の呼び出し時に保留中の結果がある場合、結果セットは廃棄されます。

SQLDisconnect() は、接続時にオープンされたステートメントをすべて自動的にドロップします。

戻りコード:

- SQL_SUCCESS
- SQL_ERROR
- SQL_INVALID_HANDLE

SQLFreeHandle() が SQL_ERROR を戻した場合は、ハンドルは引き続き有効なままです。

診断:

表 63. SQLFreeHandle の SQLSTATE

SQLSTATE	説明	解説
01000	警告。	通知メッセージ。(関数は、SQL_SUCCESS_WITH_INFO を戻す。)
08S01	通信リンク障害。	*HandleType* 引き数が SQL_HANDLE_DBC であり、DB2 CLI と接続を試みた先のデータ・ソース間の通信リンクが、関数の処理完了前に失敗した。
HY000	一般エラー。	エラーは発生したが、特定の SQLSTATE は無い。SQLGetDiagRec() が **MessageText* バッファーに戻したエラー・メッセージが、エラーとその原因を示しています。
HY001	メモリーの割り振りの失敗。	DB2 CLI が、関数の実行または完了のサポートに必要なメモリーを割り当てることができない。

表 63. *SQLFreeHandle* の *SQLSTATE (続き)*

SQLSTATE	説明	解説
HY010	関数のシーケンス・エラー。	*HandleType* 引き数が SQL_HANDLE_ENV であり、少なくとも 1 つの接続が割り当て状態または接続状態である。*HandleType* に SQL_HANDLE_ENV を指定して SQLFreeHandle() を呼び出す前に、接続ごとに、*HandleType* に SQL_HANDLE_DBC を指定して SQLDisconnect() と SQLFreeHandle() を呼び出す必要がある。 *HandleType* 引き数が SQL_HANDLE_DBC であり、接続に対して SQLDisconnect() を呼び出す前に、この関数が呼び出される。 *HandleType* 引き数が SQL_HANDLE_STMT であり、SQLExecute() または SQLExecDirect() がステートメント・ハンドルを使用して呼び出され、SQL_NEED_DATA が戻された。(DM) SQLFreeHandle() の呼び出し前に、付随するすべてのハンドルおよび他のリソースが解放されなかった。
HY013	予期しないメモリー処理エラー。	*HandleType* 引き数は SQL_HANDLE_STMT であるが、低メモリー状態が原因で基礎となるメモリー・オブジェクトにアクセスできないため、関数呼び出しを処理できない。
HY017	自動的に割り当てられた記述子ハンドルの無効な使用。	*Handle* 引き数が、自動的に割り振られた記述子またはインプリメンテーション記述子用のハンドルに設定されている。

制約事項:

なし。

関連した解説:

- 206 ページの『DB2 CLI 関数の説明の要点』
- 202 ページの『DB2 CLI 関数の要約』
- 208 ページの『SQLAllocHandle - ハンドルの割り当て』

SQLFreeStmt - ステートメント・ハンドルの解放 (またはリセット)

目的:

仕様:	DB2 CLI 1.1	ODBC 1.0	ISO CLI

SQLFreeStmt() は、ステートメント・ハンドルによって参照されたステートメントの処理を終了します。この関数を使用して以下を実行することができます。

- アプリケーション変数とパラメーターの関連付けをリセットします。

- ステートメント・ハンドルを除去し、ステートメント・ハンドルに関連付けられた DB2 CLI リソースを解放します。

SQLFreeStmt() は、SQL ステートメントの実行と結果の処理の後に呼び出します。

構文:

```
SQLRETURN   SQLFreeStmt     (SQLHSTMT          StatementHandle,   /* hstmt */
                             SQLUSMALLINT      Option);           /* fOption */
```

関数の引き数:

表 64. SQLFreeStmt の引き数

データ・タイプ	引き数	用途	説明
SQLHSTMT	*StatementHandle*	入力	ステートメント・ハンドル。
SQLUSMALLINT	*Option*	入力	ステートメント・ハンドルを解放する方法を指定するオプション。オプションの値は SQL_DROP または SQL_RESET_PARAMS でなければならない。

使用法:

SQLFreeStmt() は以下のオプションを指定して呼び出すことができます。

SQL_DROP

入力ステートメント・ハンドルに関連する DB2 CLI リソースが解放され、ハンドルが無効になります。保留中の結果はすべて廃棄されます。

このオプションは、*HandleType* が SQL_HANDLE_STMT に設定された SQLFreeHandle() の呼び出しによって置き換えられました。

推奨事項: このバージョンの DB2 CLI では、引き続きこのオプションをサポートしますが、最新の標準に準拠するよう、DB2 CLI プログラムでは SQLFreeHandle() を使用してください。

SQL_RESET_PARAMS

StatementHandle のために SQLBindParameter() によって設定されたすべてのパラメーター・バッファーを解放します。

別の方法として、ステートメント・ハンドルを除去し、新しいハンドルを割り当てることもできます。

戻りコード:

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

Option が SQL_DROP に設定されている場合、SQL_SUCCESS_WITH_INFO は戻されません。SQLError() の呼び出し時に使用するステートメント・ハンドルがないためです。

診断:

表 65. SQLFreeStmt の SQLSTATE

SQLSTATE	説明	解説
40003 **08**S01	通信リンク障害。	関数の完了前に、アプリケーションとデータ・ソース間の通信リンクに障害が生じた。
58004	予期しないシステム障害。	回復不能なシステム・エラー。
HY001	メモリーの割り振りの失敗。	DB2 CLI が、関数の実行または完了のサポートに必要なメモリーを割り当てることができない。
HY092	オプション・タイプが範囲外。	引き数 *Option* に指定した値が SQL_DROP または SQL_RESET_PARAMS ではない。

制約事項:

なし。

関連した解説:

- 206 ページの『DB2 CLI 関数の説明の要点』
- 202 ページの『DB2 CLI 関数の要約』
- 208 ページの『SQLAllocHandle - ハンドルの割り当て』
- 212 ページの『SQLBindCol - アプリケーション変数への列のバインド』

SQLGetConnectAttr — 接続属性の現行設定の取得

目的:

仕様:	DB2 CLI 5.0	ODBC 3.0	ISO CLI

SQLGetConnectAttr() は接続属性の現行の設定を戻します。

構文:

```
SQLRETURN    SQLGetConnectAttr(SQLHDBC          ConnectionHandle,
                               SQLINTEGER       Attribute,
                               SQLPOINTER       ValuePtr,
                               SQLINTEGER       BufferLength,
                               SQLINTEGER       *StringLengthPtr);
```

関数の引き数:

表 66. SQLGetConnectAttr の引き数

データ・タイプ	引き数	用途	説明
SQLHDBC	*ConnectionHandle*	入力	接続ハンドル。
SQLINTEGER	*Attribute*	入力	取り出す属性。
SQLPOINTER	*ValuePtr*	出力	*Attribute* に指定された属性の現在値を戻すメモリーへのポインター。

表 66. *SQLGetConnectAttr* の引き数 (続き)

データ・タイプ	引き数	用途	説明
SQLINTEGER	*BufferLength*	入力	• *ValuePtr* が文字ストリングを指している場合は、この引き数は **ValuePtr* の長さでなければならない。 • *ValuePtr* はポインターであるが、ストリングを指していない場合は、*BufferLength* の値は *SQL_IS_POINTER* でなければならない。 • **ValuePtr* の値がユニコード・ストリングである場合は、*BufferLength* 引き数は偶数でなければならない。
SQLINTEGER *	*StringLengthPtr*	出力	**ValuePtr* に戻す際に使用できる合計バイト数 (ヌル終了文字を除く) を戻すバッファーを指すポインター。 • *ValuePtr* がヌル・ポインターである場合、長さは戻されない。 • 属性値が文字ストリングであり、かつ、戻り時に使用できるバイト数が *BufferLength* からヌル終了文字の長さを引いた値より大きい場合、**ValuePtr* 内のデータは *BufferLength* からヌル終了文字の長さを引いた値に切り捨てられ、DB2 CLI によってヌル終了される。

使用法:

SQLGetConnectAttr() の呼び出しでは、*Attribute* に指定した接続属性の値が **ValuePtr* に戻されます。DB2 Everyplace ではこの値は 32 ビット値になり、*BufferLength* および *StringLengthPtr* 引き数は使用されません。

SQLGetConnectAttr() によって、以下の接続属性を検索することができます。これらの属性については、『SQLSetConnectAttr — 接続に関するオプションの設定』を参照してください。

- SQL_ATTR_AUTOCOMMIT (DB2 CLI/ODBC)
- SQL_ATTR_CONNECTION_DEAD (DB2 CLI/ODBC)
- SQL_ATTR_LOGIN_TIMEOUT (DB2 CLI/ODBC)
- SQL_ATTR_FILENAME_FORMAT (DB2 Everyplace)

属性によっては、SQLGetConnectAttr() の呼び出し前にアプリケーションで接続を確立する必要がないことがあります。

戻りコード:

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_NO_DATA
- SQL_ERROR

- SQL_INVALID_HANDLE

診断:

表 67. SQLGetConnectAttr SQLSTATEs

SQLSTATE	説明	解説
01000	警告。	通知メッセージ。(関数は、SQL_SUCCESS_WITH_INFO を戻す。)
01004	データが切り捨てられた。	*ValuePtr に戻されたデータは、*BufferLength* からヌル終了文字を引いた値になるように切り捨てられた。切り捨てられていないストリング値の長さが *StringLengthPtr に戻される。(関数は、SQL_SUCCESS_WITH_INFO を戻す。)
08003	接続がクローズしている。	オープン接続を必要とする *Attribute* 値が指定された。
HY000	一般エラー。	特定の SQLSTATE が何も存在しないエラーが発生した。SQLGetDiagRec() が *MessageText バッファーに戻したエラー・メッセージが、そのエラーと原因を示す。
HY001	メモリーの割り振りの失敗。	DB2 CLI が、関数を実行または完了させるのに必要なメモリーを割り当てることができない。アプリケーション・プロセスのプロセス・レベル・メモリーが使い尽くされた可能性がある。オペレーティング・システムの構成を調べて、プロセス・レベル・メモリーの制限に関する情報を入手すること。
HY090	無効なストリング長またはバッファー長。	引き数 *BufferLength* に指定した値が 0 より小さい。
HY092	オプション・タイプが範囲外。	引き数 *Attribute* に指定した値が無効である。
HYC00	ドライバーが使用不可である。	引き数 Attribute に指定した値は、DB2 CLI ドライバーのバージョンに対しては有効な接続属性かステートメント属性であるが、データ・ソースでサポートされていない。

制約事項:

なし。

関連した解説:

- 281 ページの『SQLSetConnectAttr — 接続に関するオプションの設定』

SQLGetCursorName - カーソル名の取得

目的:

仕様:	DB2 CLI 1.1	ODBC 1.0	ISO CLI

SQLGetCursorName() は、入力ステートメント・ハンドルに関連したカーソル名を戻します。SQLSetCursorName() を呼び出してカーソル名が明示的に設定されている場合は、この名前が戻されます。そうでない場合は、暗黙的に生成された名前が戻されます。

構文:

```
SQLRETURN   SQLGetCursorName (
                SQLHSTMT          StatementHandle,   /* hstmt */
                SQLCHAR      *FAR CursorName,        /* szCursor */
                SQLSMALLINT       BufferLength,      /* cbCursorMax */
                SQLSMALLINT  *FAR NameLengthPtr);    /* pcbCursor */
```

関数の引き数:

表 68. SQLGetCursorName の引き数

データ・タイプ	引き数	用途	説明
SQLHSTMT	*StatementHandle*	入力	ステートメント・ハンドル。
SQLCHAR *	*CursorName*	出力	カーソル名
SQLSMALLINT	*BufferLength*	入力	バッファー *CursorName* の長さ
SQLSMALLINT *	*NameLengthPtr*	出力	*CursorName* の値を戻すことができるバイト数。

使用法:

SQLGetCursorName() は、SQLSetCursorName() を使用して明示的に設定されたカーソル名を戻します。または名前が設定されていない場合には、DB2 CLI によって内部的に生成されたカーソル名を戻します。

SQLSetCursorName() を使用して名前が明示的に設定されている場合、このステートメントが除去されるまで、または別の明示的な名前が設定されるまで、この名前が戻されます。

内部で生成されるカーソル名は常に SQLCUR または SQL_CUR で始まります。カーソル名は常に 18 文字以下、かつ接続内で固有のものです。

戻りコード:

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

診断:

表 69. SQLGetCursorName の SQLSTATE

SQLSTATE	説明	解説
01004	データが切り捨てられた。	*CursorName* に戻されたカーソル名の長さが *BufferLength* の値を超えているため、*BufferLength* - 1 バイトに切り捨てられる。引き数 *NameLengthPtr* に、戻すことができるカーソル名全体の長さが入る。この関数は SQL_SUCCESS_WITH_INFO を戻す。

表 69. SQLGetCursorName の SQLSTATE (続き)

SQLSTATE	説明	解説
40003 **08**S01	通信リンク障害。	関数の完了前に、アプリケーションとデータ・ソース間の通信リンクに障害が生じた。
58004	予期しないシステム障害。	回復不能なシステム・エラー。
HY001	メモリーの割り振りの失敗。	DB2 CLI が、関数の実行または完了のサポートに必要なメモリーを割り当てることができない。
HY010	関数のシーケンス・エラー。	実行時データ (SQLParamData()、SQLPutData()) の操作中に、関数が呼び出された。
HY013	予期しないメモリー処理エラー。	DB2 CLI が、関数の実行または完了のサポートに必要なメモリーにアクセスできない。
HY090	無効なストリング長またはバッファー長。	引き数 *BufferLength* に指定した値が 0 より小さい。

制約事項:

なし。

関連した解説:

- 206 ページの『DB2 CLI 関数の説明の要点』
- 202 ページの『DB2 CLI 関数の要約』
- 234 ページの『SQLExecDirect - ステートメントの直接の実行』

SQLGetData - 列からのデータの入手

目的:

仕様:	DB2 CLI 1.1	ODBC 1.0	ISO CLI

SQLGetData() は、結果セットの現在行の単一列のデータを取り出します。これは、SQLFetch() 呼び出しごとにアプリケーション変数にデータを直接転送するために使用する SQLBindCol() の代替となるものです。

SQLGetData() の前に SQLFetch() を呼び出す必要があります。

必要な列ごとに SQLGetData() を呼び出した後、SQLFetch() を呼び出して次の行を取り出します。

構文:

```
SQLRETURN   SQLGetData       (
                SQLHSTMT          StatementHandle,   /* hstmt */
                SQLUSMALLINT      ColumnNumber,      /* icol */
                SQLSMALLINT       TargetType,        /* fCType */
                SQLPOINTER        TargetValuePtr,    /* rgbValue */
                SQLINTEGER        BufferLength,      /* cbValueMax */
                SQLINTEGER   *FAR StrLen_or_IndPtr); /* pcbValue */
```

関数の引き数:

表 70. SQLGetData の引き数

データ・タイプ	引き数	用途	説明
SQLHSTMT	*StatementHandle*	入力	ステートメント・ハンドル。
SQLUSMALLINT	*ColumnNumber*	入力	データ検索を要求する対象の列番号。結果セットの列は順番に番号が付いている。列番号は 1 から始まります。
SQLSMALLINT	*TargetType*	入力	*ColumnNumber* によって示される列の C データ・タイプ。以下のタイプがサポートされる。 SQL_C_BINARY SQL_C_BIT SQL_C_CHAR SQL_C_DOUBLE SQL_C_FLOAT SQL_C_LONG SQL_C_SHORT SQL_C_TYPE_DATE SQL_C_TYPE_TIME SQL_C_TYPE_TIMESTAMP SQL_C_TINYINT SQL_C_DEFAULT の指定により、データはそのデフォルトの C データ・タイプに変換される。
SQLPOINTER	*TargetValuePtr*	出力	取り出した列データの保管先のバッファーを指すポインター。 出力バッファーは、ワードで (均等に) 位置合わせする必要がある。 Motorola 68000 などの多くのプロセッサーにワード位置合わせの規則があり、文字以外のデータ・タイプの場合、アプリケーションはバッファーを正しく位置合わせする必要がある。
SQLINTEGER	*BufferLength*	入力	*TargetValuePtr* が指すバッファーの最大サイズ。 *TargetType* がバイナリーまたは文字ストリングを示している場合は、BufferLength は 0 より大きくなければならない。そうでない場合は、エラーとなる。また、この引き数は無視される。

表 70. *SQLGetData* の引き数 (続き)

データ・タイプ	引き数	用途	説明
SQLINTEGER *	*StrLen_or_IndPtr*	出力	DB2 CLI がデータを戻す際に使用できる *TargetValuePtr* バッファーのバイト数を示す値を指すポインター。データの切り捨てが発生した場合、ここには、列全体を取り出すのに必要な合計バイト数が含まれます。 バイナリー・データ・タイプと文字データ・タイプの場合、アプリケーションで部分検索モードを選択して、大きなデータを少しずつ検索できます。このモードでは、*StrLen_or_IndPtr* 引き数の内容として、列に残っている バイトの数が入っています。 列のデータ値がヌルの場合、このポインターの値は SQL_NULL_DATA である。このポインターが NULL で、SQLFetch() がヌル・データを含む列を入手した場合、この関数はこれを報告する方法がないために失敗する。 SQLFetch() がバイナリー・データを含む列を取り出した場合、*StrLen_or_IndPtr* へのポインターは NULL であってはならない。さもないと、*TargetValuePtr* バッファーに取り出されたデータの長さについてアプリケーションに知らせる方法がないため、この関数は失敗する。

使用法:

SQLFetch() を使用する場合、同じ結果セットに関する、SQLBindCol() と共に SQLGetData() を使用することができます。一般的なステップは次のとおりです。

1. SQLFetch() によって最初の行に進み、最初の行を取り出し、バインド列に関するデータを転送します。
2. SQLGetData() によって指定した列に関するデータを転送します。
3. SQLGetData() によって、必要な列に関してステップ 2 を繰り返します。
4. SQLFetch() によって次の行に進み、次の行を取り出し、バインド列に関するデータを転送します。
5. 結果セットの行ごとにステップ 2、3 および 4 を、結果セットが必要なくなるまで繰り返します。

検索を介して列データの一部を廃棄したい場合、アプリケーションでは、*ColumnNumber* を次の必要な列位置に設定して、SQLGetData() を呼び出すことができます。行全体において、取り出されなかったデータを廃棄する場合、アプリケー

ションは SQLFetch() を呼び出して次の行へ進む必要があります。あるいは、結果セットからのデータがこれ以上必要ない場合には、SQLFreeStmt() を呼び出します。

TargetType 入力引き数は、*TargetValuePtr* によって指し示されたストレージ域に列データを入れる前に必要なデータ変換がある場合には、そのタイプを指定します。

TargetValuePtr に戻される値は、検索対象の列データがバイナリー・データでない限り、ヌルで終了します。

小数点の右側の数字を切り捨てた場合、数値データ・タイプの切り捨ては警告として報告されます。切り捨てが小数点の左側で発生した場合、エラーが戻されます。

戻りコード:

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE
- SQL_NO_DATA_FOUND

SQLGetData() によって長さゼロのストリングが取り出された場合には、SQL_SUCCESS が戻されます。このような場合、*StrLen_or_IndPtr* に 0 が入り、*TargetValuePtr* にヌル終止符が入ります。

先行する SQLFetch() の呼び出しが失敗した場合、結果が未定義であるため、SQLGetData() を呼び出してはいけません。

診断:

表 71. SQLGetData の SQLSTATE

SQLSTATE	説明	解説
01004	データが切り捨てられた。	指定した列 (*ColumnNumber*) に関して戻されたデータが切り捨てられる。ストリング値または数値は右側が切り捨てられる。SQL_SUCCESS_WITH_INFO が戻される。
07006	無効な変換。	データ値は、引き数 *TargetType* によって指定した C データ・タイプに変換できない。 この関数は以前、同じ *ColumnNumber* 値に関して、異なる *TargetType* 値を使用して呼び出されている。
22002	無効な出力バッファーまたは標識バッファーの指定。	引き数 *StrLen_or_IndPtr* に指定したポインター値がヌル・ポインターで、列の値がヌルである。SQL_NULL_DATA を報告する方法がない。
22005	割り当てのエラー。	戻り値が、引き数 *TargetType* によって示されるデータ・タイプと互換性がない。
40003 **08**S01	通信リンク障害。	関数の完了前に、アプリケーションとデータ・ソース間の通信リンクに障害が生じた。

表 71. SQLGetData の SQLSTATE (続き)

SQLSTATE	説明	解説
58004	予期しないシステム障害。	回復不能なシステム・エラー。
HY001	メモリーの割り振りの失敗。	DB2 CLI が、関数の実行または完了のサポートに必要なメモリーを割り当てることができない。
HY002	無効な列番号。	指定した列が 0 より小さいか、結果の列数より大きい。
HY003	プログラム・タイプが範囲外。	*TargetType* が有効なデータ・タイプまたは SQL_C_DEFAULT ではない。
HY010	関数のシーケンス・エラー。	最初に SQLFetch() を呼び出さないで、この関数を呼び出した。
HY013	予期しないメモリー処理エラー。	DB2 CLI が、関数の実行または完了のサポートに必要なメモリーにアクセスできない。
HY090	無効なストリング長またはバッファー長。	引き数 *BufferLength* の値が 0 より小さく、引き数 *TargetType* が SQL_C_CHAR または SQL_C_BINARY であるか、あるいは *TargetType* が SQL_C_DEFAULT で、デフォルトのタイプが SQL_C_CHAR、SQL_C_BINARY、または SQL_C_DBCHAR のいずれかである。
HYC00	ドライバーが使用不可である。	指定したデータ・タイプとして SQL データ・タイプが認識されたが、DB2 CLI によってサポートされていない。 SQL データ・タイプからアプリケーション・データ *TargetType* への要求された変換は、DB2 CLI またはデータ・ソースによって実行できない。

制約事項:

なし。

関連した解説:

- 206 ページの『DB2 CLI 関数の説明の要点』
- 202 ページの『DB2 CLI 関数の要約』
- 212 ページの『SQLBindCol - アプリケーション変数への列のバインド』

SQLGetDiagRec - 診断レコードの複数のフィールド設定値の取得

目的:

仕様:	**DB2 CLI** 5.0	**ODBC** 3.0	

SQLGetDiagRec() は、エラー、警告、および状況情報が入っている診断レコードの SQLSTATE フィールドの現行値を戻します。

接続ハンドルは、この関数の呼び出し前に、SQLAllocHandle() を使用して割り当てる必要があります。

構文:

```
SQLRETURN   SQLGetDiagRec    (SQLSMALLINT       HandleType,
                              SQLHANDLE         Handle,
                              SQLSMALLINT       RecNumber,
                              SQLCHAR           *SQLState,
                              SQLINTEGER        *NativeErrorPtr,
                              SQLCHAR           *MessageText,
                              SQLSMALLINT       BufferLength,
                              SQLSMALLINT       *TextLengthPtr);
```

関数の引き数:

表 72. SQLGetDiagRec の引き数

データ・タイプ	引き数	用途	説明
SQLSMALLINT	*HandleType*	入力	診断が必要なハンドルのタイプについて記述するハンドル・タイプ ID。SQL_HANDLE_STMT または SQL_HANDLE_DBC を取ることができる。
SQLHANDLE	*Handle*	入力	*HandleType* によって示されるタイプの診断データ構造のハンドル。
SQLSMALLINT	*RecNumber*	入力	アプリケーションが情報を取り出そうとするための状況レコードを示す。状況レコードは 1 でなければならない。
SQLCHAR	*SQLState*	出力	診断レコード *RecNumber* に関する 5 文字の SQLSTATE コードを戻すバッファーへのポインター。最初の 2 文字がクラスを示し、次の 3 文字がサブクラスを示す。
SQLINTEGER	*NativeErrorPtr*	出力	データ・ソースに特定のネイティブ・エラー・コードを戻すためのバッファーへのポインター。
SQLCHAR	*MessageText*	出力	エラー・メッセージ・テキストを戻すためのバッファーへのポインター。SQLGetDiagRec() によって戻されるフィールドはテキスト・ストリングに含まれる。
SQLINTEGER	*BufferLength*	入力	*MessageText* バッファーの長さ (バイト単位)。
SQLSMALLINT	*TextLengthPtr*	出力	*MessageText* に戻すことができる合計バイト数 (ヌル終了文字に必要なバイト数を除く) を戻すためのバッファーを指すポインター。戻すことができるバイト数が *BufferLength* より大きい場合、*MessageText* 内のエラー・メッセージ・テキストは切り捨てられて、*BufferLength* からヌル終了文字の長さを引いた文字数になる。

使用法:

アプリケーションでは一般的に、DB2 CLI 関数への前の呼び出しで SQL_SUCCESS 以外のものが戻された場合に、SQLGetDiagRec() を呼び出します。

SQLGetDiagRec() は、診断データ構造レコードの複数のフィールドを含む文字ストリングを戻します。

DB2 Everyplace バージョン 8.1 では、SQLGetDiagRec() の機能が拡張されました。57011、HY024、HY092、HY000、HY012 の SQLSTATE を戻すことができるようになりました。これらの SQLSTATE の詳細については、189 ページの『SQLState のリスト』を参照してください。

SQLGetDiagRec() は、*Handle* 引き数に指定したハンドルに最後に関連付けられた診断情報のみを検索します。アプリケーションで SQLGetDiagRec() 以外の関数を呼び出すと、同じハンドルに関する前の呼び出しからの診断情報が消失します。

HandleType 引き数

各ハンドル・タイプは関連する診断情報を持つことができます。 *HandleType* 引き数は *Handle* のハンドル・タイプを示します。 DB2 Everyplace ではステートメント・ハンドルおよび接続ハンドルをサポートしています。

戻りコード:

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

診断:

SQLGetDiagRec() は自身に関するエラー値を通知しません。この関数は自身の実行の結果を報告するために、以下の戻り値を使用します。

SQL_SUCCESS
関数が診断情報を正常に戻しました。

SQL_SUCCESS_WITH_INFO
MessageText バッファーが小さすぎて、要求された診断メッセージを収容できません。診断レコードは生成されません。切り捨てが発生したかどうかを判別するには、*StringLengthPtr* に書き込まれた使用可能な実際のバイト数と *BufferLength* をアプリケーションによって比較する必要があります。

SQL_INVALID_HANDLE
HandleType と *Handle* によって示されたハンドルが有効なハンドルではありません。

SQL_ERROR
以下のいずれかの場合が考えられます。
- *RecNumber* が負または 0。
- *BufferLength* がゼロより小さい。

SQL_NO_DATA
RecNumber が、*Handle* に指定したハンドルに関して存在した診断レコード数を超えています。この関数は、*Handle* に関する診断レコードがない場合、すべての正の *RecNumber* に関して SQL_NO_DATA も戻します。

制約事項:

なし。

関連した解説:

- 206 ページの『DB2 CLI 関数の説明の要点』
- 202 ページの『DB2 CLI 関数の要約』

SQLGetInfo — 一般情報の取得

目的:

仕様:	DB2 CLI 1.1	ODBC 1.0	ISO CLI

SQLGetInfo() は、アプリケーションの接続先の DBMS に関する一般情報 (サポートされているデータ変換を含む) を戻します。

構文:

```
SQLRETURN   SQLGetInfo      (
                SQLHDBC          ConnectionHandle,   /* hdbc */
                SQLUSMALLINT      InfoType,      /* fInfoType */
                SQLPOINTER        InfoValuePtr,    /* rgbInfoValue */
                SQLSMALLINT       BufferLength,        /* cbInfoValueMax */
                SQLSMALLINT     *FAR StringLengthPtr,        /* pcbInfoValue */
```

関数の引き数:

表 73. SQLGetInfo の引き数

データ・タイプ	引き数	用途	説明
SQLHDBC	*ConnectionHandle*	入力	データベース接続ハンドル。
SQLUSMALLINT	*InfoType*	出力	欲しい情報のタイプ。この引き数は、「データ・タイプとデータ変換」表の第 1 列に入っているいずれかの値でなければならない。
SQLPOINTER	*InfoValuePtr*	出力 (入力も)	この関数が必要情報を保管するバッファーを指すポインター。検索する情報のタイプに応じて、以下の 5 つのタイプの情報が戻される。 16 ビット整数値 32 ビット整数値 32 ビット・バイナリー値 32 ビット・マスク ヌル終了文字ストリング
SQLSMALLINT	*BufferLength*	入力	*InfoValuePtr* が指すバッファーの最大サイズ。
SQLSMALLINT *	*StrLen_or_IndPtr*	出力	この関数が、必要な情報を戻すために使用可能な合計バイト数を戻す場所を指すポインター。ストリング出力の場合は、このサイズにはヌル終了文字は含まれません。 *StringLengthPtr* が指す位置にある値が *BufferLength* で指定されている *InfoValuePtr* バッファーのサイズより大きい場合は、ストリング出力情報は *BufferLength* - 1 バイトまで切り捨てられ、この関数は SQL_SUCCESS_WITH_INFO で戻ります。

使用法:

InfoType が取り得る値のリストと、SQLGetInfo() がその値に関して戻す情報については、『SQLGetInfo が戻す情報』を参照してください。

DB2 CLI は、この表に掲載されている各 *InfoType* に対する値を戻します。*InfoType* が特定の状況に当てはまらなかったりサポートされていなかったりした場合、以下のように、結果は戻りタイプにより決まります。

- 戻りタイプが「Y」か「N」を含む文字ストリングである場合は、「N」が戻される。
- 戻りタイプが「Y」または「N」以外の値を含む文字ストリングである場合は、空ストリングが戻される。
- 戻りタイプが 16 ビット整数である場合は、0 (ゼロ) が戻される。
- 戻りタイプが 32 ビット整数である場合は、0 (ゼロ) が戻される。
- 戻りタイプが 32 ビット・マスクである場合は、0 (ゼロ) が戻される。

SQLGetInfo が戻す情報

SQL_DBMS_NAME (ストリング)
アクセスされる DBMS 製品の名前。例えば「DB2 Everyplace」。

SQL_DBMS_VER (ストリング)
DB2 Everyplace DBMS 製品のバージョン。戻される情報は `DB2 Everyplace Vm.v.r Build yyyy-mm-dd` という形式のストリングです。ここで、`m` はメジャー・バージョンを、`v` はマイナー・バージョンを、`r` はリリースを、そして `yyyy-mm-dd` は ISO 形式でのビルドの日付を示します。

例えば、次のようになります。

```
'DB2 Everyplace V8.1.2 Build 2003-04-01'
は、2003 年 4 月 1 日にビルドされた DB2 Everyplace バージョン 8.1.2 です
```

注: アプリケーションは、少なくとも 39 文字を入れることのできるバッファー (BUFSIZE) を必要とします。例えば、次のようになります。

```
rc = SQLGetInfo(hdbc, SQL_DBMS_VER, buf, BUFSIZE, &len);
```

SQL_IDENTIFIER_QUOTE_CHAR (ストリング)
区切り ID を囲むために使用される文字を示します。

SQL_MAX_BINARY_LITERAL_LEN (32 ビット符号なし整数)
SQL ステートメント内の 16 進リテラルの最大長を指定する 32 ビット符号なし整数値。

SQL_MAX_CHAR_LITERAL_LEN (32 ビット符号なし整数)
SQL ステートメント内の文字リテラルの最大長 (バイト単位)。

SQL_MAX_COLUMN_NAME_LEN (16 ビット整数)
列名の最大長 (バイト単位)。

SQL_MAX_COLUMNS_IN_GROUP_BY (16 ビット整数)
サーバーが GROUP BY 文節内でサポートする列の最大数を示します。制限がなければゼロ。

SQL_MAX_COLUMNS_IN_INDEX (16 ビット整数)
サーバーが索引内でサポートする列の最大数を示します。制限がなければゼロ。

SQL_MAX_COLUMNS_IN_ORDER_BY (16 ビット整数)
サーバーが ORDER BY 文節内でサポートする列の最大数を示します。制限がなければゼロ。

SQL_MAX_COLUMNS_IN_SELECT (16 ビット整数)
サーバーが選択リスト内でサポートする列の最大数を示します。制限がなければゼロ。

SQL_MAX_CONCURRENT_ACTIVITIES (16 ビット整数)
DB2 Everyplace CLI ドライバーがサポートできるアクティブ環境の最大数。指定された制限がないか、または制限が不明である場合は、この値はゼロに設定されます。

SQL_MAX_DRIVER_CONNECTIONS (16 ビット整数)
アプリケーションごとにサポートされるアクティブ接続の最大数。

SQL_MAX_INDEX_SIZE (32 ビット符号なし整数)
サーバーが索引内で結合列に対してサポートする最大サイズを示します (バイト単位)。制限がなければゼロ。

SQL_MAX_ROW_SIZE (32 ビット符号なし整数)
サーバーが基本表の単一行内でサポートする最大長を指定します (バイト単位)。制限がなければゼロ。

SQL_MAX_STATEMENT_LEN (32 ビット符号なし整数)
ステートメント内の空白文字数も含めた、SQL ステートメント・ストリングの最大長を示します (バイト単位)。

SQL_MAX_TABLE_NAME_LEN (16 ビット整数)
表名の最大長 (バイト単位)。

SQL_MAX_TABLES_IN_SELECT (16 ビット整数)
照会指定の FROM 文節内で使用可能な表名の最大数を示します。

SQL_MAX_USER_NAME_LEN (16 ビット整数)
ユーザー ID に許可されている最大サイズを示します (バイト単位)。

SQL_SEARCH_PATTERN_ESCAPE (ストリング)
(SQLTables(), SQLColumns()) などのカタログ関数に対してドライバーがエスケープ文字として何をサポートしているかを指定するのに使用されます。

SQL_TXN_CAPABLE (16 ビット整数)
トランザクションに DDL または DML のいずれか、あるいはその両方を入れることができるかどうかを示します。

- SQL_TC_NONE = トランザクションはサポートされていない。
- SQL_TC_DML = トランザクションに DML ステートメント (SELECT、INSERT、UPDATE、DELETE など) のみを含めることができる。トランザクション内に DDL ステートメント (CREATE TABLE、DROP INDEX など) が検出されるとエラーとなります。
- SQL_TC_DDL_COMMIT = トランザクションに DML ステートメントのみを含めることができる。トランザクションに DDL ステートメントが検出されると、そのトランザクションはコミットされます。
- SQL_TC_DDL_IGNORE = トランザクションに DML ステートメントのみを含めることができる。トランザクションに DDL ステートメントが検出されても無視されます。
- SQL_TC_ALL = トランザクションに任意の順序で DDL ステートメントと DML ステートメントを含めることができる。

SQL_USER_NAME (ストリング)
特定のデータベース内で使用されているユーザー名。これは、SQLConnect() 呼び出し時に指定された ID です。

戻りコード:

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

制約事項:

なし。

関連した解説:

- 206 ページの『DB2 CLI 関数の説明の要点』
- 202 ページの『DB2 CLI 関数の要約』

SQLGetStmtAttr - ステートメント属性の現在の設定の取得

目的:

仕様:	DB2 CLI 5.0	ODBC 3.0	ISO CLI

SQLGetStmtAttr() は、ステートメント属性の現在の設定を戻します。

構文:

```
SQLRETURN   SQLGetStmtAttr       (
                SQLHSTMT             StatementHandle,
                SQLINTEGER           Attribute,
                SQLPOINTER           ValuePtr,
                SQLINTEGER           BufferLength,
                SQLINTEGER          *StringLengthPtr);
```

関数の引き数:

表 74. SQLGetStmtAttr の引き数

データ・タイプ	引き数	用途	説明
SQLHSTMT	*StatementHandle*	入力	ステートメント・ハンドル。
SQLINTEGER	*Attribute*	入力	取り出す属性。
SQLPOINTER	*ValuePtr*	出力	*Attribute* に指定された属性の値を戻すバッファーへのポインター。

表 74. *SQLGetStmtAttr* の引き数 (続き)

データ・タイプ	引き数	用途	説明
SQLINTEGER	*BufferLength*	入力	*Attribute* が ODBC 定義の属性であり、*ValuePtr* が文字ストリングまたはバイナリー・バッファーを指している場合は、この引き数は、**ValuePtr* の長さでなければならない。 *Attribute* が ODBC 定義の属性であり、**ValuePtr* が整数であれば、*BufferLength* は無視される。 *Attribute* が DB2 CLI 属性であれば、アプリケーションは属性の性質を *BufferLength* 引き数を設定することによって示す。 *BufferLength* には、以下の値を設定可能。 • **ValuePtr* が文字ストリングを指すポインターであれば、 *BufferLength* はストリングの長さまたは SQL_NTS である。 • **ValuePtr* がバイナリー・バッファーを指すポインターであれば、アプリケーションは SQL_LEN_BINARY_ATTR(length) マクロの結果を *BufferLength* に入れる。 • **ValuePtr* が文字ストリングあるいはバイナリー・ストリング以外の値を指すポインターである場合は、*BufferLength* の値は SQL_IS_POINTER でなければならない。 • **ValuePtr* に固定長のデータ・タイプが含まれている場合は、 *BufferLength* は、適宜 SQL_IS_INTEGER または SQL_IS_UINTEGER となる。
SQLSMALLINT	**StringLengthPtr*	出力	**ValuePtr* に戻す際に使用できる合計バイト数 (ヌル終了文字を除く) を戻すバッファーを指すポインター。これがヌル・ポインターである場合、長さは戻されない。属性値が文字ストリングであり、かつ、戻り時に使用できるバイト数が *BufferLength* 以上である場合、**ValuePtr* 内のデータは DB2 CLI によって *BufferLength* からヌル終了文字の長さを引いた値に切り捨てられ、ヌル終了される。

使用法:

SQLGetStmtAttr() の呼び出しでは、*Attribute* に指定したステートメント属性の値が **ValuePtr* に戻されます。 DB2 Everyplace ではこの値は 32 ビット値になり、*BufferLength* および *StringLengthPtr* 引き数は使用されません。

SQLGetStmtAttr() によって、以下のステートメント属性を取り出すことができます。属性の説明については、285 ページの『SQLSetStmtAttr - ステートメントに関するオプションの設定』を参照してください。

- SQL_ATTR_CURSOR_SCROLLABLE (DB2 CLI/ODBC)
- SQL_ATTR_CURSOR_SENSITIVITY (DB2 CLI/ODBC)
- SQL_ATTR_CURSOR_TYPE (DB2 CLI/ODBC)
- SQL_ATTR_ROW_ARRAY_SIZE (DB2 CLI/ODBC)
- SQL_ATTR_ROW_BIND_TYPE (DB2 CLI/ODBC)
- SQL_ATTR_ROW_NUMBER (DB2 CLI/ODBC)
- SQL_ATTR_DELETE_MODE (DB2 Everyplace)
- SQL_ATTR_DIRTYBIT_SET_MODE (DB2 Everyplace)
- SQL_ATTR_READ_MODE (DB2 Everyplace)
- SQL_ATTR_REORG_MODE (DB2 Everyplace)

戻りコード:

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

診断:

表 75. SQLGetStmtAttr の SQLSTATE

SQLSTATE	説明	解説
01000	警告。	通知メッセージ。(関数は、SQL_SUCCESS_WITH_INFO を戻す。)
01004	データが切り捨てられた。	**ValuePtr* に戻されたデータは、*BufferLength* からヌル終了文字の長さを引いたものに切り捨てられる。切り捨てられていないストリング値の長さは **StringLengthPtr* に戻される。 (関数は、SQL_SUCCESS_WITH_INFO を戻す。)
24000	カーソル状態が無効。	引き数 *Attribute* が SQL_ATTR_ROW_NUMBER であり、カーソルがオープンしていないか、カーソルが結果セットの初めより前あるいは結果セットの終わりより後ろに位置している。
HY000	一般エラー。	エラーは発生したが、特定の SQLSTATE は無い。SQLGetDiagRec() が **MessageText* バッファーに戻したエラー・メッセージが、エラーとその原因を示しています。
HY001	メモリーの割り振りの失敗。	DB2 CLI が、関数を実行または完了させるのに必要なメモリーを割り当てることができない。

表 75. SQLGetStmtAttr の SQLSTATE (続き)

SQLSTATE	説明	解説
HY010	関数のシーケンス・エラー。	非同期で実行されている関数が *StatementHandle* に対して呼び出され、この関数が呼び出された時点でも引き続き実行されている。 SQLExecute() または SQLExecDirect() が *StatementHandle* に対して呼び出され、SQL_NEED_DATA が戻された。この関数は、実行時データのすべてのパラメーターまたは列に対してデータが送信される前に呼び出される。
HY013	予期しないメモリー処理エラー。	DB2 CLI が、関数を実行または完了させるのに必要なメモリーにアクセスできない。
HY090	無効なストリング長またはバッファー長。	引き数 *BufferLength* に指定した値が 0 より小さい。
HY092	オプション・タイプが範囲外。	引き数 *Attribute* に指定した値が、このバージョンの DB2 CLI では無効である。
HY109	カーソル位置が無効である。	*Attribute* 引き数が SQL_ATTR_ROW_NUMBER であるが、行が削除されているか、取り出すことができない。
HYC00	ドライバーが使用不可である。	引き数 *Attribute* に指定した値は、このバージョンの DB2 CLI においては有効な DB2 CLI 属性であるが、データ・ソースではサポートされていない。

制約事項:

なし。

関連した解説:

- 206 ページの『DB2 CLI 関数の説明の要点』
- 202 ページの『DB2 CLI 関数の要約』
- 281 ページの『SQLSetConnectAttr — 接続に関するオプションの設定』
- 285 ページの『SQLSetStmtAttr - ステートメントに関するオプションの設定』

SQLNumParams - SQL ステートメントのパラメーターの数を取得

目的:

仕様:	DB2 CLI 2.1	ODBC 1.0	

SQLNumParams() は SQL ステートメント内のパラメーター・マーカーの数を戻します。

構文:

```
SQLRETURN   SQLNumParams     (SQLHSTMT          StatementHandle,
                              SQLSMALLINT  FAR  *ParameterCountPtr);
```

関数の引き数:

表 76. SQLNumParams の引き数

データ・タイプ	引き数	用途	説明
SQLHSTMT	*StatementHandle*	入力	ステートメント・ハンドル。
SQLSMALLINT	*ParameterCountPtr*	出力	ステートメント内のパラメーターの数。

使用法:

StatementHandle と関連したステートメントが準備された後にのみ、この関数を呼び出すことができます。ステートメントにパラメーター・マーカーが含まれていない場合、 *ParameterCountPtr* はゼロに設定されます。

アプリケーションではこの関数を呼び出して、ステートメント・ハンドルと関連した SQL ステートメントに対して必要な、SQLBindParameter() 呼び出しの回数を判別することができます。

戻りコード:

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

診断:

表 77. SQLNumParams の SQLSTATE

SQLSTATE	説明	解説
HY001	メモリーの割り振りの失敗。	DB2 CLI が、関数を実行または完了させるのに必要なメモリーを割り当てることができない。
HY010	関数のシーケンス・エラー。	指定された *StatementHandle* に対して SQLPrepare() が呼び出される前に、この関数が呼び出された。
HY013	予期しないメモリー処理エラー。	DB2 CLI が、関数を実行または完了させるのに必要なメモリーにアクセスできなかった。

制約事項:

なし。

関連した解説:

- 215 ページの『SQLBindParameter - バッファーへのパラメーター・マーカーのバインド』
- 275 ページの『SQLPrepare - ステートメントの準備』

SQLNumResultCols - 結果の列数の入手

目的:

仕様:	DB2 CLI 1.1	ODBC 1.0	

SQLNumResultCols() は、入力ステートメント・ハンドルに関連した結果セット内の列数を戻します。

この関数を呼び出す前に、SQLPrepare() または SQLExecDirect() を呼び出す必要があります。

この関数を呼び出した後に、SQLColAttribute()、またはバインド列関数の 1 つを呼び出すことができます。

構文:

```
SQLRETURN   SQLNumResultCols (SQLHSTMT          StatementHandle,   /* hstmt */
                              SQLSMALLINT  FAR  *ColumnCountPtr);  /* pccol */
```

関数の引き数:

表 78. SQLNumResultCols の引き数

データ・タイプ	引き数	用途	説明
SQLHSTMT	*StatementHandle*	入力	ステートメント・ハンドル。
SQLSMALLINT *	*ColumnCountPtr*	出力	結果セット内の列数。

使用法:

この関数は、入力ステートメント・ハンドルに関して実行された最後のステートメントまたは関数が結果セットを生成しなかった場合に、出力引き数をゼロに設定します。

戻りコード:

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

診断:

表 79. SQLNumResultCols の SQLSTATE

SQLSTATE	説明	解説
40003 **08**S01	通信リンク障害。	関数の完了前に、アプリケーションとデータ・ソース間の通信リンクに障害が生じた。
58004	予期しないシステム障害。	回復不能なシステム・エラー。
HY001	メモリーの割り振りの失敗。	DB2 CLI が、関数の実行または完了のサポートに必要なメモリーを割り当てることができない。

表 79. *SQLNumResultCols* の *SQLSTATE* (続き)

SQLSTATE	説明	解説
HY010	関数のシーケンス・エラー。	*StatementHandle* に対して、SQLPrepare() または SQLExecDirect() が呼び出される前に関数が呼び出された。
HY013	予期しないメモリー処理エラー。	DB2 CLI が、関数の実行または完了のサポートに必要なメモリーにアクセスできない。

制約事項:

なし。

関連した解説:

- 206 ページの『DB2 CLI 関数の説明の要点』
- 202 ページの『DB2 CLI 関数の要約』
- 212 ページの『SQLBindCol - アプリケーション変数への列のバインド』
- 228 ページの『SQLDescribeCol - 列の属性セットの戻し』
- 234 ページの『SQLExecDirect - ステートメントの直接の実行』
- 259 ページの『SQLGetData - 列からのデータの入手』

SQLPrepare - ステートメントの準備

目的:

仕様:	DB2 CLI 1.1	ODBC 1.0	ISO CLI

SQLPrepare() は、SQL ステートメントを入力ステートメント・ハンドルと関連付け、そのステートメントを DBMS に送信して準備します。アプリケーションでは、ステートメント・ハンドルを他の関数に渡すことによって、この準備済みステートメントを参照することができます。

ステートメント・ハンドルが照会ステートメント (または結果セットを戻すいずれかの関数) と一緒に以前使用されていた場合は、SQLPrepare() を呼び出す前に、SQLFreeStmt() を呼び出す必要があります。

構文:

```
SQLRETURN   SQLPrepare      (SQLHSTMT         StatementHandle,   /* hstmt */
                             SQLCHAR      FAR *StatementText,   /* szSqlStr */
                             SQLINTEGER        TextLength);     /* cbSqlStr */
```

関数の引き数:

表 80. *SQLPrepare* の引き数

データ・タイプ	引き数	用途	説明
SQLHSTMT	*StatementHandle*	入力	ステートメント・ハンドル。
SQLCHAR	*StatementText*	入力	SQL ステートメント・ストリング。

表 80. *SQLPrepare* の引き数 (続き)

データ・タイプ	引き数	用途	説明
SQLINTEGER	*TextLength*	入力	*StatementText* 引き数の内容の長さ。 これは、*szSqlstr* 内の SQL ステートメントの正確な長さに設定するか、ステートメント・テキストがヌルで終了する場合には、SQL_NTS に設定する必要がある。

使用法:

SQLPrepare() を使用してステートメントを準備した後、アプリケーションでは (ステートメントが照会であった場合)、以下のいずれかの関数を呼び出して、結果セットの形式に関する情報を要求することができます。

- SQLNumResultCols()
- SQLDescribeCol()

SQL ステートメント・ストリングにはパラメーター・マーカーが含まれている場合があります。パラメーター・マーカーは、? 文字によって表され、SQLExecute() の呼び出し時に、アプリケーション提供の値が置き換えられる、ステートメント内の位置を示すために使用されます。バインド・パラメーター関数 SQLBindParameter() は、アプリケーション値を各パラメーター・マーカーにバインドし (関連付け)、データの転送時にいずれかのデータ変換を実行すべきかどうかを示すために使用します。

SQLExecute() を呼び出す前に、すべてのパラメーターをバインドする必要があります。詳細については、235 ページの『SQLExecute - ステートメントの実行』を参照してください。

パラメーター・マーカーに関する規則については、「*DB2 Universal Database SQL リファレンス*」の PREPARE ステートメントのセクションを参照してください。

SQLExecute() 呼び出しの結果を処理した後、アプリケーションは新規 (または同じ) パラメーター値を使って、そのステートメントを再度実行することができます。

戻りコード:

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

診断:

表 81. SQLPrepare の SQLSTATE

SQLSTATE	説明	解説
42nnn	構文エラー。	**42**nnn SQLSTATE は、ステートメント関連のさまざまな構文またはアクセスに関する問題を示す。文字 nnn は、そのクラス・コードを持ついずれかの SQLSTATE を示す。例えば、**42**nnn は、**42** クラスのいずれかの SQLSTATE を示す。
58004	予期しないシステム障害。	回復不能なシステム・エラー。
HY001	メモリーの割り振りの失敗。	DB2 CLI が、関数を実行または完了させるのに必要なメモリーを割り当てることができない。
HY009	無効な引き数値。	*StatementText* がヌル・ポインターである。
HY013	予期しないメモリー処理エラー。	DB2 CLI が、関数を実行または完了させるのに必要なメモリーにアクセスできない。
HY014	ハンドル数の限界に到達済み。	DB2 CLI が、内部リソースが原因でハンドルを割り振ることができない。
HY090	無効なストリング長またはバッファー長。	引き数 *TextLength* が 1 より小さいが、SQL_NTS と等しくない。

制約事項:

なし。

関連した解説:

- 206 ページの『DB2 CLI 関数の説明の要点』
- 202 ページの『DB2 CLI 関数の要約』
- 215 ページの『SQLBindParameter - バッファーへのパラメーター・マーカーのバインド』
- 228 ページの『SQLDescribeCol - 列の属性セットの戻し』
- 234 ページの『SQLExecDirect - ステートメントの直接の実行』
- 235 ページの『SQLExecute - ステートメントの実行』
- 274 ページの『SQLNumResultCols - 結果の列数の入手』

SQLPrimaryKeys - 表の主キー列の入手

目的:

仕様:	**DB2 CLI** 2.1	**ODBC** 1.0	

`SQLPrimaryKeys()` は、表の主キーを構成する列名のリストを戻します。この情報は SQL 結果セットに戻されます。この結果セットは、照会で生成された結果セットの処理に使用するものと同じ関数を使用して、取り出すことができます。
CatalogName、*NameLength1*、*SchemaName*、*NameLength2* は無視されます。戻された結果セットの列 1、2、6 は、常に長さがゼロのストリングになります。

構文:

```
SQLRETURN   SQLPrimaryKeys  (
                  SQLHSTMT          StatementHandle,  /* hstmt */
                  SQLCHAR      FAR  *CatalogName,     /* szCatalogName */
                  SQLSMALLINT       NameLength1,      /* cbCatalogName */
                  SQLCHAR      FAR  *SchemaName,      /* szSchemaName */
                  SQLSMALLINT       NameLength2,      /* cbSchemaName */
                  SQLCHAR      FAR  *TableName,       /* szTableName */
                  SQLSMALLINT       NameLength3);     /* cbTableName */
```

関数の引き数:

表 82. SQLPrimaryKeys の引き数

データ・タイプ	引き数	用途	説明
SQLHSTMT	*StatementHandle*	入力	ステートメント・ハンドル。
SQLCHAR*	*CatalogName*	入力	3 つの部分で構成される表名のカタログ修飾子。 このフィールドは、DB2 Everyplace では無視されます。
SQLSMALLINT	*NameLength1*	入力	*CatalogName* の長さ。このフィールドは、DB2 Everyplace では無視されます。
SQLCHAR*	*SchemaName*	入力	表名のスキーマ修飾子。このフィールドは、DB2 Everyplace では無視されます。
SQLSMALLINT	*NameLength2*	入力	*SchemaName* の長さ。このフィールドは、DB2 Everyplace では無視されます。
SQLCHAR*	*TableName*	入力	表名。
SQLSMALLINT	*NameLength3*	入力	*TableName* の長さ。

使用法:

SQLPrimaryKeys() は単一の表の主キー列を戻します。表名を指定するために、検索パターンを使用することはできません。

指定した表に主キーが含まれていない場合、空の結果セットが戻されます。

多くの場合、SQLPrimaryKeys() の呼び出しは、システム・カタログに対する複雑な (そのために高コストの) 照会にマップされます。

将来のリリースで新規の列が追加されたり、既存の列の名前が変更されても、現在の列の位置は変わりません。

結果セットには、以下の列が TABLE_NAME および ORDINAL_POSITION 順に含まれています:

列 1 TABLE_CAT (VARCHAR(128))
これは常に長さゼロのストリングです。

列 2 TABLE_SCHEM (VARCHAR(128))
これは常に長さゼロのストリングです。

列 3 TABLE_NAME (VARCHAR(128) 非ヌル)
指定した表の名前。

列 4 COLUMN_NAME (VARCHAR(128) 非ヌル)
主キーの列名。

列 5 ORDINAL_POSITION (SMALLINT 非ヌル)
1 から始まる、主キー内の列シーケンス番号。

列 6 PK_NAME (VARCHAR(128))
これは常に長さゼロのストリングです。

DB2 CLI/ODBC が使用する列名は、X/Open CLI CAE 仕様のスタイルに準拠しています。

戻りコード:
- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

診断:

表 83. SQLPrimaryKey SQLSTATE

SQLSTATE	説明	解説
24000	カーソル状態が無効。	カーソルがすでにこのステートメント・ハンドルでオープンされている。
40003 **08**S01	通信リンク障害。	関数の完了前に、アプリケーションとデータ・ソース間の通信リンクに障害が生じた。
HY001	メモリーの割り振りの失敗。	DB2 CLI が、関数の実行または完了のサポートに必要なメモリーを割り当てることができない。
HY010	関数のシーケンス・エラー。	実行時データ (SQLPrepare() または SQLExecDirect()) の操作中に、関数が呼び出された。
HY014	ハンドル数の限界に到達済み。	内部リソースが原因で、DB2 CLI がハンドルを割り当てることができない。
HY090	無効なストリング長またはバッファー長。	名前の長さ引き数のいずれかの値が、ゼロより小であるが、SQL_NTS と等しくない。

制約事項:

SQLPrimaryKeys() の呼び出しは慎重に使用し、呼び出しを繰り返し実行するのではなく、結果を保管するようにしてください。

関連した解説:
- 206 ページの『DB2 CLI 関数の説明の要点』
- 202 ページの『DB2 CLI 関数の要約』
- 247 ページの『SQLForeignKeys - 外部キー列のリストの入手』

SQLRowCount - 行カウントの入手

目的:

仕様:	DB2 CLI 1.1	ODBC 1.0	ISO CLI

SQLRowCount() は、表に対して実行された、両方向スクロール・カーソル・ステートメントでの UPDATE、INSERT、DELETE、または SELECT に影響を受ける、表中の行数を戻します。

SQLExecute() または SQLExecDirect() を、この関数を呼び出す前に呼び出す必要があります。

構文:

```
SQLRETURN   SQLRowCount      (SQLHSTMT          StatementHandle,   /* hstmt */
                              SQLINTEGER   FAR  *RowCountPtr);     /* pcrow */
```

関数の引き数:

表 84. SQLRowCount の引き数

データ・タイプ	引き数	用途	説明
SQLHSTMT	*StatementHandle*	入力	ステートメント・ハンドル。
SQLINTEGER	*RowCountPtr*	出力	影響された行数が保管される場所へのポインター。

使用法:

入力ステートメント・ハンドルによって参照される、最後の実行ステートメントが UPDATE、INSERT、または DELETE ステートメントではない場合、あるいはこのステートメントが正常に実行されなかった場合、この関数は *RowCountPtr* の内容を -1 に設定します。

ステートメントによって影響を受けた可能性のある他の表の行は、このカウントには含まれていません。

戻りコード:

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

診断:

表 85. SQLRowCount の SQLSTATE

SQLSTATE	説明	解説
40003 **08**S01	通信リンク障害。	関数の完了前に、アプリケーションとデータ・ソース間の通信リンクに障害が発生した。
58004	予期しないシステム障害。	回復不能なシステム・エラー。
HY001	メモリー割り振りの失敗。	DB2 CLI が、関数の実行または完了のサポートに必要なメモリーを割り当てることができない。アプリケーション・プロセス用のプロセス・レベル・メモリーをすべて使い切った可能性があります。オペレーティング・システムの構成を調べてプロセス・レベル・メモリーの制限についての情報を入手してください。

表 85. SQLRowCount の SQLSTATE (続き)

SQLSTATE	説明	解説
HY010	関数のシーケンス・エラー。	*StatementHandle* に対して、SQLExecute() または SQLExecDirect() が呼び出される前に関数が呼び出された。
HY013	予期しないメモリーの処理エラー。	DB2 CLI が、関数の実行または完了のサポートに必要なメモリーにアクセスできなかった。

制約事項:

なし。

関連した解説:

- 206 ページの『DB2 CLI 関数の説明の要点』
- 202 ページの『DB2 CLI 関数の要約』
- 234 ページの『SQLExecDirect - ステートメントの直接の実行』
- 235 ページの『SQLExecute - ステートメントの実行』
- 274 ページの『SQLNumResultCols - 結果の列数の入手』

SQLSetConnectAttr — 接続に関するオプションの設定

目的:

仕様:	DB2 CLI	ODBC 1.0	ISO CLI

SQLSetConnectAttr() は接続に関するオプションを設定します。

構文:

```
SQLRETURN   SQLSetConnectAttr   (SQLHDBC        ConnectionHandle,
                                 SQLINTEGER     Attribute,
                                 SQLPOINTER     ValuePtr,
                                 SQLINTEGER     StringLength);
```

関数の引き数:

表 86. SQLSetConnectAttr の引き数

データ・タイプ	引き数	用途	説明
SQLHDBC	*ConnectionHandle*	入力	接続ハンドル。
SQLINTEGER	*Attribute*	入力	セットするオプション。

表 86. *SQLSetConnectAttr* の引き数 (続き)

データ・タイプ	引き数	用途	説明
SQLPOINTER	*ValuePtr*	入力	*Attribute* が ODBC 定義の属性であり、*ValuePtr* が文字ストリングまたはバイナリー・バッファーを指している場合は、この引き数は、*ValuePtr* の長さでなければならない。 *Attribute* が ODBC 定義の属性であり、ValuePtr が整数であれば、*StringLength* は無視される。 *Attribute* が DB2 CLI 属性であれば、アプリケーションは属性の性質を *StringLength* 引き数を設定することによって示す。 *StringLength* には、以下の値を設定可能。 • *ValuePtr* が文字ストリングを指すポインターであれば、 *StringLength* はストリングの長さまたは SQL_NTS である。 • *ValuePtr* がバイナリー・バッファーを指すポインターであれば、アプリケーションは SQL_LEN_BINARY_ATTR(length) マクロの結果を *StringLength* に入れる。これにより、負の値が *StringLength* に入る。 • *ValuePtr* が文字ストリングあるいはバイナリー・ストリング以外の値を指すポインターであれば、*StringLength* の値は SQL_IS_POINTER でなければならない。 • *ValuePtr* に固定長の値が含まれている場合は、 *StringLength* は、適宜、SQL_IS_INTEGER または SQL_IS_UINTEGER となる。
SQLINTEGER	*StringLength*	入力	*ValuePtr* が文字ストリングまたはバイナリー・バッファーを指している場合は、この引き数は *ValuePtr* の長さでなければならない。 *ValuePtr* がポインターであるがストリングまたはバイナリー・バッファーを指していない場合は、*StringLength* の値は SQL_IS_POINTER でなければならない。 *ValuePtr* がポインターでない場合は、 *StringLength* の値は SQL_IS_NOT_POINTER でなければならない。

使用法:

接続の接続属性は、別の SQLSetConnectAttr() 呼び出しで変更されるか、SQLDisconnect() 呼び出しでその接続が除去されるまでは有効です。

SQLSetConnectAttr() には、属性情報が入ります。属性情報の形式は、ヌル終了文字ストリングまたは 32 ビット整数値のいずれかとなります。それぞれの形式は、属性の説明のところで述べられています。SQLSetConnectAttr() の *ValuePtr* 引き数が指す文字ストリングの長さは *StringLength* です。

接続属性:

現在定義されている属性は以下のとおりです。

SQL_ATTR_AUTOCOMMIT (DB2 CLI/ODBC)
モード・タイプを指定する 32 ビット整数値。サポートされる値は次のとおりです。

- SQL_AUTOCOMMIT_ON = それぞれのステートメントは自動的にコミットされます。これはデフォルトです。

  自動コミット・モードでは、ステートメントによって行われたすべての更新が、そのステートメントの実行後に自動的に永続化されます。自動コミット・モード はデフォルトです。デフォルトにより、トランザクション・サポートは使用不可となっており、さらにはステートメント・レベルのアトミシティは保証されません。例えば、以下の UPDATE ステートメントは、処理の途中で失敗する可能性があり、行のサブセットだけが更新される可能性があります。

  ```
  UPDATE T SET A = A + 1
  ```

  更新、削除、挿入の操作が失敗する理由はたくさんあります。例えば、更新中にチェック制約に違反することがあります。その結果、表の一部が正常に更新できても、残りの部分が更新できないことがあります。このため、変更はロールバックできません。

- SQL_AUTOCOMMIT_OFF = アプリケーションは、手動で明示的にコミットするか、トランザクションをロールバックしなければなりません。SQLEndTran() を呼び出すと、コミットまたはトランザクションのロールバックが行えます。SQLEndTran() の使用法については、 232 ページの『SQLEndTran - COMMIT または ROLLBACK の要求』を参照してください。

  手動コミット・モードの場合、トランザクションは暗黙的に開始され、初めに SQLPrepare()、SQLExecDirect() を使用してデータベースにアクセスします。この時点では、呼び出しが失敗してもトランザクションは開始されます。 SQLEndTran() を使用してトランザクションを ROLLBACK または COMMIT すると、トランザクションは終了します。

  手動コミット・モードでは、DDL および DML (例えば、CREATE TABLE ステートメントや UPDATE ステートメント) を含む任意の SQL ステートメントをトランザクションで発行できます。

SQL_ATTR_CONNECTION_DEAD (DB2 CLI/ODBC)
接続がまだアクティブであるかどうかを示す読み取り専用の 32 ビット整数値。DB2 CLI は、以下のいずれかの値を戻します。

- SQL_CD_FALSE - 接続がまだアクティブである。

- SQL_CD_TRUE - 接続は非活動である。

SQL_ATTR_LOGIN_TIMEOUT (DB2 CLI/ODBC)

アプリケーションに制御権を戻す前に、ログイン要求の完了を待機する秒数に対応する 32 ビット整数値です。

SQL_ATTR_FILENAME_FORMAT (DB2 Everyplace)

DB2e データベース・エンジンがファイル名を長形式または 8.3 形式のどちらで作成すべきかを指定する 32 ビット整数値。アプリケーションは、SQLSetConnectAttr が呼び出されたときに、接続されたパスにカタログ・ファイルが存在しない場合にのみ、ファイル名の形式を変更することができます。既にカタログ・ファイルが存在するためにファイル名の形式の変更が否認された場合は、 SQLState HY000 の SQL_ERROR が戻されます。例えば、既に DB2 Everyplace カタログ・ファイルが存在するパスにアプリケーションが接続した場合、ファイル名の形式を変更しようとするすべての試みが失敗します。アプリケーションが DB2 Everyplace カタログ・ファイルの存在しないパスに接続し、最初の CREATE TABLE ステートメントの後にファイル名の形式を変更しようとした場合にも、 SQLSetConnectAttr が SQL_ERROR を戻します。これは、最初の CREATE TABLE ステートメントの間にカタログ・ファイルが作成されており、カタログ・ファイルが作成された後はファイル名の形式の変更が許可されていないためです。デフォルトのファイル名の形式は、プラットフォームごとに異なります。現在のところ、サポートされるすべてのプラットフォームにおいて、SQL_FILENAME_FORMAT_LONG がデフォルトです。

属性値:

SQL_FILENAME_FORMAT_LONG - ファイルは長ファイル名形式で作成されます。

SQL_FILENAME_FORMAT_83- ファイルは 8.3 ファイル名形式で作成されます。

戻りコード:

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

診断:

表 87. SQLSetConnectAttr の SQLSTATE

SQLSTATE	説明	解説
HY000	一般エラー。	ファイル名の形式を変更することはできない。
HY001	メモリーの割り振りの失敗。	DB2 CLI が、関数の実行または完了のサポートに必要なメモリーを割り当てることができない。
HY014	ハンドル数の限界に到達済み。	DB2 CLI が内部リソースが原因でハンドルを割り当てることができなかった。
HY090	無効なストリング長またはバッファー長。	名前の長さ引き数のいずれかの値が、ゼロより小であるが、SQL_NTS と等しくない。

制約事項:

- トランザクション内で更新できる表の数は制限されています。DB2 Everyplace では、トランザクション内に最大 256 個のオープン・ファイルを許可しています (ただし、オペレーティング・システムも同数のオープン・ファイルを許可しているものと想定した場合)。これは通常、およそ 100 個の表が更新できることを意味しています。表の数は、索引の使用量とステートメント・ハンドルの数によって決まります。アクティブなステートメント・ハンドルの数が増えるにつれ、更新できる表の数は少なくなります。それぞれの表は、たとえトランザクション内でアクセスや更新が複数回行われても、一度しかカウントされません。
- 複数の表において複数の関連レコードを挿入したり、一貫性のある更新を行ったりするためのトランザクションが DB2 Everyplace に追加されています。変更は、アプリケーションがトランザクションをコミットした後に、データ表に書き込まれます。
- アプリケーションが現行トランザクションをコミットせずに早期に終了してしまうと、そのトランザクション内の更新内容は自動的にロールバックされます。
- SQLEndTran が戻った後、トランザクションはコミットされるかロールバックされます。
- 未完のまま (アクティブ・トランザクションの実行中) に終了したデータベースにアプリケーションが接続すると、トランザクションはリカバリーされます。データベースは、以下の論理を使用してトランザクションをリカバリーします。
  - トランザクションが完了していない場合、データベースは更新されない。
  - トランザクションが完了している場合、データベースはそのトランザクションの情報で更新される。
  - リカバリーが中断された場合、次の接続時に適切なアクションが取られる。

関連した解説:

- 206 ページの『DB2 CLI 関数の説明の要点』
- 202 ページの『DB2 CLI 関数の要約』
- 232 ページの『SQLEndTran - COMMIT または ROLLBACK の要求』

SQLSetStmtAttr - ステートメントに関するオプションの設定

目的:

仕様:	DB2 CLI 2.1	ODBC 1.0	ISO CLI

SQLSetStmtAttr() はステートメントに関するオプションを設定します。

構文:

```
SQLRETURN   SQLSetStmtAttr   (SQLHSTMT          StatementHandle,
                              SQLINTEGER        Attribute,
                              SQLPOINTER        ValuePtr,
                              SQLINTEGER        StringLength);
```

関数の引き数:

表 88. SQLSetStmtAttr の引き数

データ・タイプ	引き数	用途	説明
SQLHSTMT	*StatementHandle*	入力	ステートメント・ハンドル。

表 88. *SQLSetStmtAttr* の引き数 (続き)

データ・タイプ	引き数	用途	説明
SQLINTEGER	*Attribute*	入力	セットするオプション。
SQLPOINTER	*ValuePtr*	入力	*Attribute* が ODBC 定義の属性であり、*ValuePtr* が文字ストリングまたはバイナリー・バッファーを指している場合は、この引き数は、**ValuePtr* の長さでなければならない。 *Attribute* が ODBC 定義の属性であり、ValuePtr が整数であれば、*StringLength* は無視される。 *Attribute* が DB2 CLI 属性であれば、アプリケーションは属性の性質を *StringLength* 引き数を設定することによって示す。 *StringLength* には、以下の値を設定可能。 • *ValuePtr* が文字ストリングを指すポインターであれば、 *StringLength* はストリングの長さまたは SQL_NTS である。 • *ValuePtr* がバイナリー・バッファーを指すポインターであれば、アプリケーションは SQL_LEN_BINARY_ATTR(length) マクロの結果を *StringLength* に入れる。これにより、負の値が *StringLength* に入る。 • *ValuePtr* が文字ストリングあるいはバイナリー・ストリング以外の値を指すポインターであれば、*StringLength* の値は SQL_IS_POINTER でなければならない。 • *ValuePtr* に固定長の値が含まれている場合は、 *StringLength* は、SQL_IS_INTEGER または SQL_IS_UINTEGER となる。
SQLINTEGER	*StringLength*	入力	*ValuePtr* が文字ストリングまたはバイナリー・バッファーを指している場合は、この引き数は *ValuePtr* の長さでなければならない。 *ValuePtr* がポインターであるがストリングまたはバイナリー・バッファーを指していない場合は、*StringLength* の値は SQL_IS_POINTER でなければならない。 *ValuePtr* がポインターでない場合は、 *StringLength* の値は SQL_IS_NOT_POINTER でなければならない。

使用法:

ステートメントのステートメント属性は、別の SQLSetStmtAttr() 呼び出しで変更されるか、SQLFreeHandle() の呼び出しでそのステートメントが除去されるまでは有効です。SQL_CLOSE、SQL_UNBIND、または SQL_RESET_PARAMS オプションを指定して SQLFreeStmt() を呼び出しても、ステートメント属性はリセットされません。

ステートメント属性によっては、データ・ソースが *ValuePtr* で指定されている値をサポートしない場合、似たような値での置き換えをサポートするものがあります。そのような場合、 DB2 CLI は SQL_SUCCESS_WITH_INFO および SQLSTATE 01S02 (オプション値が変更された) を戻します。例えば、属性が SQL_ATTR_CONCURRENCY で、 *ValuePtr* が SQL_CONCUR_ROWVER であり、データ・ソースがそれをサポートしない場合、DB2 CLI は SQL_CONCUR_VALUES を置き換えて SQL_SUCCESS_WITH_INFO を戻します。置き換えられた値を判別するには、アプリケーションで SQLGetStmtAttr() を呼び出します。 *ValuePtr* に設定する情報の形式は、指定された属性によって変わります。

SQLSetStmtAttr() は、属性情報を次の 2 つ (ヌル終了文字ストリングまたは 32 ビット整数値) の形式のいずれかで受け入れます。それぞれの形式は、属性の説明のところで述べられています。この形式は、SQLGetStmtAttr() でそれぞれの属性ごとに戻される情報に適用されます。 SQLSetStmtAttr() の *ValuePtr* 引き数が指す文字ストリングの長さは *StringLength* です。

ダーティー・ビット:

DB2 Everyplace はダーティー・ビットを使用してレコードに対して行われた変更を追跡します。ダーティー・ビットの動作は SQL_ATTR_DELETE_MODE、SQL_ATTR_READ_MODE、SQL_ATTR_DIRTYBIT_SET_MODE ステートメント属性の影響を受けます。次の表は、あるレコードに対して特定のデータベース操作が行われた後のダーティー・ビットの状態を示しています。この表では、SQL_ATTR_DIRTYBIT_SET_MODE パラメーターが、システムによって保守されるダーティー・ビットを持つ SQL_DIRTYBIT_SET_BY_SYSTEM に設定されていると想定しています。

表 89. DB2 Everyplace ダーティー・ビットの状態

レコードに対するアクション	ダーティー・ビットの状態
状態をクリーン (0) にして INSERT	INSERT
状態をクリーン (0) にして DELETE	DELETE
状態をクリーン (0) にして UPDATE	UPDATE
DELETE して INSERT	UPDATE
DELETE して DELETE	該当せず
DELETE して UPDATE	該当せず
INSERT して INSERT	該当せず
INSERT して DELETE	レコードの物理的な削除
INSERT して UPDATE	INSERT
UPDATE して INSERT	該当せず
UPDATE して DELETE	DELETE

表 89. DB2 Everyplace ダーティー・ビットの状態 (続き)

レコードに対するアクション	ダーティー・ビットの状態
UPDATE して UPDATE	UPDATE

ダーティー・ビットの値は、表の $dirty 列を照会して入手します。例えば、以下のステートメントはダーティー・ビットと PHONEBOOK 表の NAME 列を戻します。

```
SELECT $dirty, NAME from PHONEBOOK
```

ダーティー・ビットの値は以下のようになります。

表 90. DB2 Everyplace ダーティー・ビット値

説明	ダーティー・ビット値
未変更レコード (CLEAN)	0
削除済みレコード (DELETE)	1
挿入済みレコード (INSERT)	2
更新済みレコード (UPDATE)	3

ステートメント属性:

現在定義されている属性は以下のとおりです。

SQL_ATTR_CURSOR_SCROLLABLE (DB2 CLI)
アプリケーションが求めるサポート・レベルを指定する 32 ビット整数。この属性の設定は、後続の SQLExecDirect() および SQLExecute() への呼び出しに影響します。サポートされる値は次のとおりです。

- SQL_NONSCROLLABLE

  ステートメント・ハンドルは両方向スクロール・カーソルを必要としません。アプリケーションがこのハンドルで SQLFetchScroll() を呼び出した場合、 FetchOrientation() の唯一の有効な値は SQL_FETCH_NEXT です。これはデフォルトです。

- SQL_SCROLLABLE

  ステートメント・ハンドルは両方向スクロール・カーソルを必要とします。 SQLFetchScroll() を呼び出す場合、カーソルを順次モード以外のモードで配置することができるように、アプリケーションは FetchOrientation の有効な値を指定します。

SQL_ATTR_CURSOR_SENSITIVITY (DB2 CLI)
カーソルが別のカーソルの書き込みアクティビティーを認識するかどうかを指定する 32 ビット整数値。サポートされる値は次のとおりです。

- SQL_UNSPECIFIED

  他のカーソルの書き込みアクティビティーにより、現行のカーソルが未定義の影響を受けます。これはデフォルトです。

- SQL_INSENSITIVE

  他のカーソルの書き込みアクティビティーによる、現行のカーソルへの影響はありません。

注: この属性値はパフォーマンスに影響するため、慎重に使用してください。

SQL_ATTR_CURSOR_TYPE (DB2 CLI)
カーソル・タイプを指定する 32 ビット整数値。サポートされる値は次のとおりです。

- SQL_CURSOR_FORWARD_ONLY = カーソルは前方にのみスクロール。これはデフォルトです。
- SQL_CURSOR_STATIC = 結果セット内のデータは静的です。

このオプションをオープン・カーソルに指定することはできません。

SQL_ATTR_ROW_ARRAY_SIZE (DB2 CLI)
行セット内の行数を指定する 32 ビット整数値。これは、SQLFetch() または SQLFetchScroll() に対する各呼び出しで戻される行数です。デフォルト値は 1 です。指定した行セットのサイズが、データ・ソースがサポートする最大行セット・サイズを超える場合、 DB2 CLI はその値を置き換え、SQLSTATE 01S02 (オプション値が変更された) を戻します。このオプションはオープン・カーソルに対して指定できます。

SQL_ATTR_ROW_BIND_TYPE (DB2 CLI)
SQLFetch() または SQLFetchScroll() が関連したステートメントで呼び出されるときに使用する、バインディング方向を設定する 32 ビットの整数値。列方向バインディングは、定義済み定数の SQL_BIND_BY_COLUMN を *ValuePtr* に設定することで選択します。 *ValuePtr* に指定する長さに、バインドされた列用のスペースと、構造体あるいはバッファーの埋め込み用のスペースをすべて含めることによって、バインドされた列のアドレスが指定された長さの分だけ増分しても、結果が次の行の同じ列の先頭を指すようにする必要があります。**sizeof** 演算子を ANSI C の共用体または構造体で使用すれば、この動作は保証されます。列方向バインディングは、SQLFetchScroll() の場合のデフォルトのバインディング方向です。

SQL_ATTR_ROW_NUMBER (DB2 CLI)
結果セット全体の中の現在行の番号を示す 32 ビット整数値。現在行の番号が判別できない場合、あるいは現在行がない場合、DB2 CLI は 0 を戻します。この属性は SQLGetStmtAttr() を呼び出すことによって取り出すことができますが、 SQLSetStmtAttr() を呼び出すことによって設定することはできません。

SQL_ATTR_ROW_STATUS_PTR (DB2 CLI)
SQLFetch() または SQLFetchScroll() の呼び出しの後、行状況値が入った UWORD 値の配列を指す、16 ビットの符号なし整数値。この配列には、行セットの行数分のエレメントがあります。このステートメント属性をヌル・ポインターとしてセットできますが、その場合は、DB2 CLI は行状況値を戻しません。この属性はいつでもセットできますが、新規の値は次に SQLFetch() または SQLFetchScroll() が呼び出されるまでは使用されません。

SQL_ATTR_ROWS_FETCHED_PTR (DB2 CLI)
SQLFetch() または SQLFetchScroll() の呼び出しの後、取り出された行数が戻るバッファーを指す、32 ビットの符号なしの整数値。

SQL_ATTR_DELETE_MODE (DB2 Everyplace)

サポートされる値は次のとおりです。

- SQL_DELETE_MARK_ONLY

  これはシステム・デフォルトです。削除 SQL ステートメントが実行されると、レコードは「delete」とだけマークされます。そのレコード内容は、SQL_READ_INCLUDE_MARKED_DELETE が設定されている場合、引き続き読み取ることができます。

- SQL_DELETE_PHYSICAL_REMOVE

  SQL の DELETE ステートメントは、レコードのダーティー・ビットとは無関係に、WHERE 文節の条件を満たすレコードを物理的に削除します。

例えば、ダーティー・ビットの状況を無視してレコードを物理的に除去するには、以下の構文を使用します。

```
SQLSetStmtAttr (stmt, SQL_ATTR_DELETE_MODE, SQL_DELETE_PHYSICAL_REMOVE, 0)
```

次に、以下の SQL ステートメントを実行して、X が 0 に等しくないすべてのレコードを表から削除します。

```
DELETE T WHERE X<>0
```

SQL_ATTR_DIRTYBIT_SET_MODE (DB2 Everyplace)

カーソル・タイプを指定する 32 ビット整数値。サポートされる値は次のとおりです。

- SQL_DIRTYBIT_SET_BY_SYSTEM

  これはシステム・デフォルトです。挿入、更新、または削除されたレコードは、それぞれ、INSERT、UPDATE、または DELETE に設定されたダーティー・ビットを持ちます。 SQL_DIRTYBIT_SET_BY_SYSTEM が設定されている場合、 $dirty 列の UPDATE は許可されていません。

- SQL_DIRTYBIT_SET_BY_APPLICATION

  レコードの挿入、更新、または削除の際のダーティー・ビットの設定は、アプリケーションが行います。各操作の意味は以下のとおりです。

  UPDATE

  システムは、アプリケーションが指定したとおりのダーティー・ビットを設定します。例えば、アプリケーションが以下のステートメントを実行した場合、表内のすべてのレコードは 0 (CLEAN) にリセットされます。

  ```
  UPDATE T SET $dirty=0 WHERE $dirty>0
  ```

  INSERT

  新しく挿入されたレコードのダーティー・ビットが CLEAN に設定されます。

  DELETE

  SQL_DELETE_PHYSICAL_REMOVE を設定すると、DELETE はデータベースからレコードを物理的に削除します。それ以外の場合は、$dirty 列の値が DELETE に設定され、レコードはデータベース内に残ります。

例えば、レコードのダーティー・ビットをクリーンにするには、以下のステートメントを使用します。

```
SQLSetStmtAttr (stmt, SQL_ATTR_DIRTYBIT_SET_MODE,
                SQL_DIRTYBIT_SET_BY_APPLICATION, 0)
```

次に、以下の SQL ステートメントを実行します。

```
UPDATE T SET $DIRTY=0 WHERE $DIRTY>0
```

一般的に、エンド・ユーザーによるデータベース更新のトラッキングにダーティー・ビットが必要とされないときは、アプリケーションで SQL_DIRTYBIT_SET_BY_APPLICATION を設定できます。

SQL_ATTR_READ_MODE (DB2 Everyplace)

カーソル・タイプを指定する 32 ビット整数値。サポートされる値は次のとおりです。

- SQL_READ_EXCLUDE_MARKED_DELETE

  これはシステム・デフォルトです。ダーティー・ビットが「delete」に設定されているすべてのレコードが SQL から隠されます。

- SQL_READ_INCLUDE_MARKED_DELETE

  これを設定すると、ダーティー・ビットが DELETE に設定されたレコードが、SQL の SELECT ステートメントから可視の状態になります。アプリケーションは、レコードのダーティー・ビットを調べることによって、アプリケーションの削除済みレコードとその他のレコードを区別することができます。

例えば、ダーティー・ビットが DELETE とマークされたレコードを含め、ダーティー・ビットが設定されているすべてのレコードを読み取るには、以下のステートメントを使用します。

```
SQLSetStmtAttr (stmt, SQL_ATTR_READ_MODE, SQL_READ_INCLUDE_MARKED_DELETE, 0)
```

次に、以下の SQL ステートメントを実行して、すべてのレコードを検索します。

```
SELECT * FROM T WHERE $dirty<>0
```

SQL_ATTR_REORG_MODE (DB2 Everyplace)

ユーザーが作成した表でのデータベースの自動再編成を実行するかどうか、および明示的な REORG SQL ステートメントを許可するかどうかを指定する 32 ビット整数値。サポートされる値は次のとおりです。

- SQL_REORG_ENABLED - これはシステム・デフォルトです。 データベースの再編成は、 DB2 Everyplace が実行することも、ユーザーが REORG SQL ステートメントを使用することによって明示的に実行することもできます。
- SQL_REORG_DISABLED - REORG SQL ステートメントには制限があり、ユーザーが作成した表の自動データベース再編成は行われません。

このオプションをオープン・カーソルに指定することはできません。

戻りコード:

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR

- SQL_INVALID_HANDLE

診断:

表 91. SQLSetStmtAttr SQLSTATE

SQLSTATE	説明	解説
24000	カーソル状態が無効。	カーソルがすでにこのステートメント・ハンドルでオープンされている。
HY001	メモリーの割り振りの失敗。	DB2 CLI が、関数の実行または完了のサポートに必要なメモリーを割り当てることができない。
HY010	関数のシーケンス・エラー。	実行時データ (SQLPrepare() または SQLExecDirect()) の操作中に、関数が呼び出された。 BEGIN COMPOUND および END COMPOUND SQL 操作中に、関数が呼び出された。
HY014	ハンドル数の限界に到達済み。	内部リソースが原因で、DB2 CLI がハンドルを割り当てることができない。
HY090	無効なストリング長またはバッファー長。	名前の長さ引き数のいずれかの値が、ゼロより小であるが、SQL_NTS と等しくない。

制約事項:

なし。

関連した解説:

- 206 ページの『DB2 CLI 関数の説明の要点』
- 202 ページの『DB2 CLI 関数の要約』

SQLTables - 表情報の取得

目的:

仕様:	**DB2 CLI** 2.1	**ODBC** 1.0	

SQLTables() は、接続されたデータ・ソースのシステム・カタログに保管されている表名と関連情報のリストを戻します。表名のリストは結果セットとして戻され、照会で生成された結果セットを処理するのに使用されている関数と同じ関数を使用して検索できます。

構文:

```
SQLRETURN   SQLTables        (
                SQLHSTMT          StatementHandle,   /* hstmt */
                SQLCHAR      FAR  *CatalogName,      /* szCatalogName */
                SQLSMALLINT       NameLength1,       /* cbCatalogName */
                SQLCHAR      FAR  *SchemaName,       /* szSchemaName */
                SQLSMALLINT       NameLength2,       /* cbSchemaName */
                SQLCHAR      FAR  *TableName,        /* szTableName */
```

```
SQLSMALLINT       NameLength3,       /* cbTableName */
SQLCHAR      FAR  *TableType,        /* szTableType */
SQLSMALLINT       NameLength4);      /* cbTableType */
```

関数の引き数:

表 92. SQLTables の引き数

データ・タイプ	引き数	用途	説明
SQLHSTMT	*StatementHandle*	入力	ステートメント・ハンドル。
SQLCHAR	*CatalogName*	入力	結果セットを修飾する *pattern-value* を入れることのできるバッファー。 *Catalog* は 3 部分からなる表名の最初の部分である。 このフィールドは DB2 Everyplace では無視される。
SQLSMALLINT	*NameLength1*	入力	*CatalogName* の長さ。 このフィールドは DB2 Everyplace では無視される。
SQLCHAR	*SchemaName*	入力	結果セットをスキーマ名で修飾する *pattern-value* を入れることのできるバッファー。 このフィールドは DB2 Everyplace では無視される。
SQLSMALLINT	*NameLength2*	入力	*SchemaName* の長さ。 このフィールドは DB2 Everyplace では無視される。
SQLCHAR	*TableName*	入力	結果セットを表名で修飾する *pattern-value* を入れることのできるバッファー。
SQLSMALLINT	*NameLength3*	入力	*TableName* の長さ。
SQLCHAR	*TableType*	入力	DB2 Everyplace はタイプ TABLE のみサポートする。このフィールドは DB2 Everyplace では無視される。
SQLSMALLINT	*NameLength4*	入力	このフィールドは DB2 Everyplace では無視される。

TableName 引き数が検索パターンを受け入れることに注意してください。

使用法:

表情報は結果セットに戻されますが、各表は結果セットの 1 行で表されます。

時には、戻される結果セットが制限されないよう、アプリケーションで *TableName* 引き数にヌル・ポインターを指定して `SQLTables()` を呼び出します。多数の表が入っているデータ・ソースの場合、この方法だと極めて大きな結果セットになり、検索時間が非常に長くなります。

SQLTables() で戻される結果セットには、指定された順序で表 93内にリストされた列が入ります。行は TABLE_NAME の順に配列されます。

SQLTables() の呼び出しは、多くの場合システム・カタログに対する複雑で時間のかかる照会となるため、控えめに行うようにしてください。呼び出しを繰り返さず、結果を保管するようにしてください。

カタログ関数の結果セットの VARCHAR 列は、SQL92 の制限と一貫性を持たせるために最大長の属性は 128 で宣言されています。 DB2 の名前は 128 より短いため、128 文字 (NULL 終止符を含む) を常に出力バッファー用にとっておくか、または SQL_MAX_TABLE_NAME_LEN を指定した SQLGetInfo() を呼び出して、接続されている DBMS がサポートする TABLE_NAME 列の実際の長さを判別することができます。

表 93. SQLTables が戻す列

列名	データ・タイプ	説明
TABLE_CAT	VARCHAR (128)	これは常に長さゼロのストリングです。
TABLE_SCHEM	VARCHAR (128)	これは常に長さゼロのストリングです。
TABLE_NAME	VARCHAR (128)	表の名前
TABLE_TYPE	VARCHAR (128)	TABLE_NAME 列内で名前で指定されたタイプを識別します。常にストリング値「TABLE」となります。
REMARKS	VARCHAR(254)	表についての記述情報が入ります。

戻りコード:

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

診断:

表 94. SQLTables の SQLSTATE

SQLSTATE	説明	解説
HY001	メモリーの割り振りの失敗。	DB2 CLI が、関数を実行または完了させるのに必要なメモリーを割り当てることができない。
HY014	ハンドル数の限界に到達済み。	内部リソースが原因で、DB2 CLI がハンドルを割り振ることができない。
HY090	無効なストリング長またはバッファー長。	名前の長さ引き数のいずれかの値が、ゼロより小であるが、SQL_NTS と等しくない。 指定された名前長さの引き数のうちの 1 つが、データ・ソースに対してサポートされている最大値を超える値を持っている。サポートされている最大値は、SQLGetInfo() 関数を呼び出すことで取得できる。

制約事項:

なし。

関連した解説:

- 265 ページの『SQLGetInfo — 一般情報の取得』

DB2 CLI 関数によるデータ変換

DB2 CLI は、アプリケーションと DB2 Everyplace 間のデータの転送および必要な変換を管理します。データ転送が実際に行われる前に、SQLBindParameter()、SQLBindCol()、または SQLGetData() の呼び出し時に、ソース、ターゲット、またはその両方のデータ・タイプが示されます。これらの関数はシンボル名 (SQL_CHAR および SQL_C_CHAR など) を使用して、関連するデータ・タイプを識別します。

例えば、SQL データ・タイプ SQL_VARCHAR に対応するパラメーター・マーカーを、アプリケーションの長整数の C バッファー・タイプにバインドしたい場合、適切な SQLBindParameter() 呼び出しは、次のようになります。

```
SQLBindParameter (hstmt, 1, SQL_PARAM_INPUT, SQL_C_LONG,
                  SQL_VARCHAR, 0, 0, long_ptr, 0, NULL);
```

表 95 は、C と SQL データ・タイプ間でサポートされるデータ変換を示しています。表 95 の最初の列には SQL データ・タイプが含まれています。残りの列は C データ・タイプを示しています。 C データ・タイプの列の文字の意味は次のとおりです。

D この変換はサポートされ、SQL データ・タイプのデフォルトの変換です。

X DB2 Everyplace はこの変換をサポートします。

ブランク
DB2 Everyplace はこの変換をサポートしません。

タイプ変換の際の精度と位取りに関する制限事項、および切り捨てと丸めに関する規則は、 SQL の構文規則に従っています。

表 95. サポートされているデータ変換

SQL データ・タイプ	デフォルトの変換	サポートされているその他の変換
BLOB	SQL C BINARY	SQL C CHAR
CHAR	SQL C CHAR	SQL C LONG SQL C SHORT SQL C TINYINT SQL C TYPE DATE SQL C TYPE TIME SQL C BINARY SQL C BIT SQL C TYPE TIMESTAMP
DATE	SQL C TYPE DATE	SQL C CHAR
DECIMAL	SQL C CHAR	SQL C LONG SQL C SHORT SQL C TINYINT SQL C BIT

表 95. サポートされているデータ変換 (続き)

SQL データ・タイプ	デフォルトの変換	サポートされているその他の変換
INTEGER	SQL_C_LONG	SQL_C_CHAR SQL_C_SHORT SQL_C_TINYINT SQL_C_FLOAT SQL_C_DOUBLE SQL_C_BIT
SMALLINT	SQL_C_SHORT	SQL_C_CHAR SQL_C_LONG SQL_C_TINYINT SQL_C_FLOAT SQL_C_DOUBLE SQL_C_BIT
TIME	SQL_C_TYPE_TIME	SQL_C_CHAR
TIMESTAMP	SQL_C_TYPE_TIMESTAMP	SQL_C_CHAR
VARCHAR	SQL_C_CHAR	SQL_C_LONG SQL_C_SHORT SQL_C_TINYINT SQL_C_TYPE_DATE SQL_C_TYPE_TIME SQL_C_BINARY SQL_C_BIT SQL_C_TYPE_TIMESTAMP

関連した解説:

- 206 ページの『DB2 CLI 関数の説明の要点』
- 202 ページの『DB2 CLI 関数の要約』

サポートされる JDBC メソッド

この章では、DB2 Everyplace がサポートする JDBC メソッドについての情報を提供します。本章には、以下のセクションがあります。

- 『DB2 Everyplace JDBC サポートの概要』
- 297 ページの『java.sql パッケージ内のインターフェース』
- 316 ページの『javax.sql パッケージ内のインターフェース』

DB2 Everyplace JDBC サポートの概要

DB2 Everyplace は、Sun Java Developer's Kit で提供される JDBC API 仕様に定義されたメソッドのサブセットをサポートします。DB2 Everyplace がサポートする JDBC メソッドの情報は、Sun の Java Development Kit Version 1.4.1 のドキュメンテーションから変更されています。また DB2 Everyplace は、拡張 Connection インターフェースおよび Statement インターフェースもサポートします。

詳しくは、314 ページの『DB2eStatement クラス』および 301 ページの『DB2eConnection クラス』を参照してください。

DB2 Everyplace JDBC ドライバーは、JSR 169 によって指定された CDC/Foundation Profile の JDBC Optional Package と互換性があります。

関連したタスク:

- 21 ページの『DB2 Everyplace Java アプリケーションの開発』

関連した解説:

- 『Blob インターフェース』
- 299 ページの『Connection インターフェース』
- 301 ページの『DB2eConnection クラス』
- 301 ページの『DatabaseMetaData インターフェース』
- 304 ページの『Driver インターフェース』
- 305 ページの『PreparedStatement インターフェース』
- 307 ページの『ResultSet インターフェース』
- 311 ページの『ResultSetMetaData インターフェース』
- 313 ページの『Statement インターフェース』
- 314 ページの『DB2eStatement クラス』
- 202 ページの『JDBC により報告される SQLState メッセージ』

java.sql パッケージ内のインターフェース

この章では java.sql パッケージ内の JDBC メソッドについての情報を提供します。説明されているトピックは、以下のとおりです。

- 『Blob インターフェース』
- 298 ページの『CallableStatement インターフェース』
- 299 ページの『Connection インターフェース』
- 301 ページの『DB2eConnection クラス』
- 301 ページの『DatabaseMetaData インターフェース』
- 304 ページの『Driver インターフェース』
- 305 ページの『PreparedStatement インターフェース』
- 307 ページの『ResultSet インターフェース』
- 311 ページの『ResultSetMetaData インターフェース』
- 313 ページの『Statement インターフェース』
- 314 ページの『DB2eStatement クラス』

Blob インターフェース

Blob インターフェースは、Java™ プログラム言語で SQL BLOB を表します (マップします)。 SQL BLOB は、データベース表の行に 2 進ラージ・オブジェクト (BLOB) を列値として保管する組み込みタイプです。BLOB オブジェクトは、そのオブジェクトが作成されたトランザクションの存続している間有効です。

プログラマーは、ResultSet および PreparedStatement インターフェースのメソッド (getBlob や setBlob など) を使用して SQL BLOB にアクセスできます。Blob イン

ターフェースには、SQL BLOB (2 進ラージ・オブジェクト) の長さの取得や、クライアント上の BLOB 値のマテリアライズを行うためのメソッドが用意されています。

java.sql パッケージ

共通インターフェース **Blob**

表 96には、DB2 Everyplace でサポートされている Blob インターフェースのメソッドがリストされています。

表 96. Blob インターフェース・メソッド

メソッドの戻り値タイプ	メソッド
`InputStream`	**getBinaryStream**() この BLOB インスタンスによって指定された BLOB をストリームとして取り出す。
`byte[]`	**getBytes**(long pos, int length) この BLOB オブジェクトが指定する BLOB 値の一部または全部を、バイトの配列として戻す。
`long`	**length**() この BLOB オブジェクトによって指定された BLOB 値のバイト数を戻す。

関連したタスク:

- 21 ページの『DB2 Everyplace Java アプリケーションの開発』

関連した解説:

- 296 ページの『DB2 Everyplace JDBC サポートの概要』
- 202 ページの『JDBC により報告される SQLState メッセージ』

CallableStatement インターフェース

リモート SQL ストアード・プロシージャーを実行するのに使用するインターフェースです。結果パラメーターは、OUT パラメーターとして登録されなければなりません。その他のパラメーターは、入力、出力、またはその両方に使用できます。パラメーターは番号順に参照されます。最初のパラメーターは 1 です。

詳しくは、「*DB2 Everyplace 同期サーバー 管理ガイド*」の『リモート照会およびストアード・プロシージャー・アダプター』を参照してください。

```
call <procedure-name> (?,?, ...)
```

IN パラメーター値は、PreparedStatement から継承される set メソッドを使用して設定されます。すべての OUT パラメーターのタイプは、ストアード・プロシージャーを実行する前に登録しなければなりません。その値は、ここで提供される get メソッドを介して実行された後に検索されます。出力パラメーターのサイズは、4K バイトに制限されます。

A `CallableStatement` は、1 つの `ResultSet` を戻せます。

java.sql パッケージ

共通インターフェース **CallableStatement**

PreparedStatement の拡張

表 97は、DB2 Everyplace がサポートする CallableStatement インターフェースのメソッドをリストしたものです。

表 97. CallableStatement インターフェースのメソッド

メソッドの戻り値タイプ	メソッド
Blob	**getBlob**(int i) JDBC 2.0 JDBC BLOB パラメーターの値を Java プログラミング言語の Blob オブジェクトとして取得する。
byte[]	**getBytes**(int parameterIndex) JDBC BINARY または VARBINARY パラメーターの値を Java プログラミング言語の byte 値の配列として取得する。
Date	**getDate**(int parameterIndex) JDBC DATE パラメーターの値を java.sql.Date オブジェクトとして取得する。
int	**getInt**(int parameterIndex) JDBC INTEGER パラメーターの値を Java プログラミング言語の int として取得する。
Object	**getObject**(int parameterIndex) パラメーターの値を Java プログラミング言語の object として取得する。
short	**getShort**(int parameterIndex) JDBC SMALLINT パラメーターの値を Java プログラミング言語の short として取得する。
String	**getString**(int parameterIndex) JDBC CHAR、VARCHAR、または LONGVARCHAR パラメーターの値を Java プログラミング言語の String として検索する。
Time	**getTime**(int parameterIndex) JDBC TIME パラメーターの値を java.sql.Time オブジェクトとして取得する。
Timestamp	**getTimestamp**(int parameterIndex) JDBC TIMESTAMP パラメーターの値を java.sql.Timestamp オブジェクトとして取得する。
void	**registerOutParameter**(int parameterIndex, int sqlType) OUT 出力パラメーターを parameterIndex で指定された順番で JDBC タイプ sqlType に登録する。
boolean	**wasNull**() 最後に読み取られた OUT パラメーターに SQL NULL 値があるか否かを示す。

関連したタスク:

- 21 ページの『DB2 Everyplace Java アプリケーションの開発』

関連した解説:

- 296 ページの『DB2 Everyplace JDBC サポートの概要』
- 202 ページの『JDBC により報告される SQLState メッセージ』

Connection インターフェース

Connection インターフェースは、特定のデータベースとの接続 (セッション) を確立します。 Connection のコンテキスト内において、SQL ステートメントが実行され、結果が戻されます。

Connection のデータベースは、そのデータベースの持つ表、サポートされている SQL の文法、ストアード・プロシージャー、この接続の機能、などを説明する情報を提供することができます。この情報は、 getMetaData メソッドによって取得します。

java.sql パッケージ

共通インターフェース **Connection**

表 98には、DB2 Everyplace でサポートされている Connection インターフェースのメソッドがリストされています。

表 98. Connection インターフェース・メソッド

メソッドの戻り値タイプ	メソッド
void	**clearWarnings**() この Connection オブジェクトについて報告されたすべての警告をクリアする。
void	**close**() 接続のデータベースおよび JDBC リソースが自動的に解放されるのを待たずに、それらをすぐに解放する。
void	**commit**() 前のコミットまたはロールバック以降に行われたすべての変更を永続的にし、Connection によって現在保持されているすべてのデータベース・ロックを解放する。
Statement	**createStatement**() SQL ステートメントをデータベースに送信するための Statement オブジェクトを作成する。
Statement	**createStatement**(int resultSetType, int resultSetConcurrency) JDBC 2.0。指定されたタイプおよび並行性で ResultSet オブジェクトを生成する Statement オブジェクトを作成する。
boolean	**isClosed**() Connection がクローズされたかどうかをテストする。
DatabaseMetaData	**getMetaData**() この Connection のデータベースに関するメタデータを取得する。
SQLWarning	**getWarnings**() この Connection での呼び出しにより報告された最初の警告を戻す。
CallableStatement	**prepareCall**(String sql) データベースのストアード・プロシージャーを呼び出すための CallableStatement オブジェクトを作成する。
PreparedStatement	**prepareStatement** (String sql) パラメーター化された SQL ステートメントをデータベースに送信するための PreparedStatement オブジェクトを作成する。
PreparedStatement	**prepareStatement**(String sql, int resultSetType, int resultSetConcurrency) JDBC 2.0。指定されたタイプおよび並行性で ResultSet オブジェクトを生成する PreparedStatement オブジェクトを作成する。
void	**rollback**() 前のコミットまたはロールバック以降に行われたすべての変更を除去し、この Connection によって現在保持されているすべてのデータベース・ロックを解放する。
void	**setAutoCommit** (boolean autoCommit) この接続の自動コミット・モードを設定する。

関連したタスク:

- 21 ページの『DB2 Everyplace Java アプリケーションの開発』

関連した解説:

- 296 ページの『DB2 Everyplace JDBC サポートの概要』
- 202 ページの『JDBC により報告される SQLState メッセージ』

DB2eConnection クラス

DB2eConnection クラスは、特定の Connection 属性を取得し、設定します。Connection オブジェクトで DB2eConnection クラス・メソッドを使用するためには、まず Connection オブジェクトが DB2eConnection オブジェクトにキャストされていなければなりません。これらのメソッドは、CLI/ODBC 関数である SQLGetConnectAttr および SQLSetConnectAttr を、適切な引数を指定して呼び出すことによりインプリメントされます。

詳しくは、第 18 章『 サポートされている DB2 CLI 関数』を参照してください。

com.ibm.db2e.jdbc パッケージ

共通クラス **DB2eConnection**

Connection のインプリメント

表 99は、DB2 Everyplace がサポートする DB2eConnection クラスのメソッドをリストしたものです。

表 99. DB2eConnection クラス・メソッド

メソッドの戻りタイプ	メソッド
`void`	**enableFilenameFormat83** (`boolean enable`) enable が true の場合にデータベース・エンジンがファイル名を 8.3 形式で作成できるようにし、そうでない場合は、ファイル名は長形式で作成できるようにする。ファイル名形式は、この接続のパスにカタログ・ファイルがない場合のみ変更できる。
`boolean`	**isEnabledFilenameFormat83** () データベース・エンジンが 8.3 形式でファイル名を作成するか? または、長形式でファイル名を作成するか?

関連したタスク:

- 21 ページの『DB2 Everyplace Java アプリケーションの開発』

関連した解説:

- 296 ページの『DB2 Everyplace JDBC サポートの概要』
- 202 ページの『JDBC により報告される SQLState メッセージ』
- 255 ページの『SQLGetConnectAttr — 接続属性の現行設定の取得』
- 281 ページの『SQLSetConnectAttr — 接続に関するオプションの設定』

DatabaseMetaData インターフェース

DatabaseMetaData インターフェースは、データベース全体に関する包括的な情報を提供します。

これらのメソッドの一部では、カタログ名およびスキーマ名に String 引き数を使用します。これらの引き数は DB2 Everyplace では無視されます。

ここに示すいくつかのメソッドは、ResultSet オブジェクトのフォームで情報のリストを戻します。 getString や getInt などの通常の ResultSet メソッドを使用して、これらの ResultSet からデータを取り出すことができます。

指定されたフォームのメタデータが利用不能な場合、これらのメソッドは SQLException をスローします。

java.sql パッケージ

共通インターフェース **DatabaseMetaData**

表 100には、DB2 Everyplace でサポートされている DatabaseMetaData インターフェース内のフィールドがリストされています。

表 100. DatabaseMetaData フィールド

フィールド・タイプ	フィールド
`static int`	**columnNoNulls** 列で NULL 値が許可されていない可能性のあることを示す。
`static int`	**columnNullable** 列で明確に NULL 値を許可していることを示す。
`static int`	**columnNullableUnknown** 列のヌル可能性が不明であることを示す。

表 101には、DB2 Everyplace でサポートされている DatabaseMetaData インターフェースのメソッドがリストされています。

表 101. DatabaseMetaData インターフェース・メソッド

メソッドの 戻り値タイプ	メソッド
`ResultSet`	**getColumns**(String `catalog`, String `schemaPattern`, String `tableNamePattern`, String `columnNamePattern`) 指定されたカタログ内で使用可能な表列の記述を取得する。
`Connection`	**getConnection**() JDBC 2.0。このメタデータ・オブジェクトを作成した接続を検索する。
`ResultSet`	**getCrossReference**(String `primaryCatalog`, String `primarySchema`, String `primaryTable`, String `foreignCatalog`, String `foreignSchema`, String `foreignTable`) 主キー表の主キー列を参照する、外部キー表の外部キー列の記述を取得する (ある表が別のキーをインポートする方法についての記述)。これは、通常、単一の外部キー/主キーのペアを戻す (ほとんどの表では、表から外部キーを一度インポートするのみとなる)。これらは、FKTABLE_NAME および KEY_SEQ 順に配列される。
`String`	**getDatabaseProductName**() このデータベース製品の名前は?
`String`	**getDatabaseProductVersion**() このデータベース製品のバージョンは?

表 101. *DatabaseMetaData* インターフェース・メソッド (続き)

メソッドの 戻り値タイプ	メソッド
int	**getDriverMajorVersion**() この JDBC ドライバーのメジャー・バージョン番号は?
int	**getDriverMinorVersion**() この JDBC ドライバーのマイナー・バージョン番号は?
String	**getDriverName**() この JDBC ドライバーの名前は?
String	**getDriverVersion**() この JDBC ドライバーのバージョンは?
ResultSet	**getExportedKeys**(String catalog, String schema, String table) 表の主キー列を参照する外部キー列 (表によってエクスポートされた外部キー) の記述を取得する。
String	**getIdentifierQuoteString**() SQL ID の引用に使用されるストリングは? これは、ID 引用がサポートされていない場合、スペース " " を戻す。
ResultSet	**getImportedKeys**(String catalog, String schema, String table) 表の外部キー列 (表によってエクスポートされた外部キー) により参照される主キー列の記述を取得する。
int	**getMaxBinaryLiteralLength**() インライン・バイナリー・リテラルに入れることのできる 16 進文字数は?
int	**getMaxCharLiteralLength**() 文字リテラルの最大長は?
int	**getMaxColumnNameLength**() 列名の長さの限度は?
int	**getMaxColumnsInGroupBy**() GROUP BY 文節の最大列数は?
int	**getMaxColumnsInIndex**() 索引に許可される最大列数は?
int	**getMaxColumnsInOrderBy**() ORDER BY 文節の最大列数は?
int	**getMaxColumnsInSelect**() SELECT ステートメントの最大列数は?
int	**getMaxConnections**() このデータベースに対して同時に保持できるアクティブ接続数は?
int	**getMaxIndexLength**() 索引の最大長は (バイト単位)?
int	**getMaxRowSize**() 単一行の最大長は?
int	**getMaxStatementLength**() SQL ステートメントの最大長は?
int	**getMaxStatements**() このデータベースに対して一度にオープンできるアクティブ・ステートメント数は?
int	**getMaxTableNameLength**() 表名の最大長は?
int	**getMaxTablesInSelect**() SELECT ステートメントの最大表数は?
int	**getMaxUserNameLength**() ユーザー名の最大長は?
ResultSet	**getPrimaryKeys**(String catalog, String schema, String table) 表の主キー列の記述を取得する。
String	**getSearchStringEscape**() ワイルドカード文字を拡張するために使用できるストリングを取得する。
ResultSet	**getTables**(String catalog, String schemaPattern, String tableNamePattern, String[] types) カタログ内で使用可能な表の記述を取得する。

表 101. DatabaseMetaData インターフェース・メソッド (続き)

メソッドの戻り値タイプ	メソッド
ResultSet	**getUDTs**(String catalog, String schemaPattern, String typeNamePattern, int[] types) JDBC 2.0。特定のスキーマ内に定義されたユーザー定義タイプの記述を取得する。DB2 Everyplace では UDT をサポートしていないため、常に空の結果セットを戻す。
String	**getURL**() このデータベースの URL は?
String	**getUserName**() データベースに認識されているユーザー名は?
boolean	**supportsColumnAliasing**() 列の別名割り当てはサポートされているか?
boolean	**supportsFullOuterJoins**() 完全ネストされた外部結合はサポートされているか?
boolean	**supportsMixedCaseIdentifiers**() データベースでは、引用符で囲まれていない大文字小文字混合の SQL ID をケース・センシティブとして扱い、大文字小文字混合でそれら ID を保管するか?
boolean	**supportsMixedCaseQuotedIdentifiers**() データベースでは、引用符で囲まれた大文字小文字混合の SQL ID をケース・センシティブとして扱い、大文字小文字の区別でそれら ID を保管するか?
boolean	**supportsNonNullableColumns**() 列はヌル不可能として定義できるか?
boolean	**supportsOrderByUnrelated**() 「ORDER BY」文節では、SELECT ステートメントに存在しない列を使用できるか?
boolean	**supportsOuterJoins**() 何らかの形式の外部結合がサポートされているか?
boolean	**supportsPositionedDelete**() 位置決めされた DELETE はサポートされているか?
boolean	**supportsPositionedUpdate**() 位置決めされた UPDATE はサポートされているか?
boolean	**supportsResultSetType**(int type) JDBC 2.0。データベースでは指定された結果セット・タイプはサポートされているか?
boolean	**supportsSchemasInTableDefinitions**() 表定義ステートメントでスキーマ名は使用できるか?
boolean	**supportsTransactions**() トランザクションはサポートされているか? サポートされていない場合、分離レベルは TRANSACTION_NONE。

関連したタスク:

- 21 ページの『DB2 Everyplace Java アプリケーションの開発』

関連した解説:

- 296 ページの『DB2 Everyplace JDBC サポートの概要』
- 202 ページの『JDBC により報告される SQLState メッセージ』

Driver インターフェース

Driver インターフェースは、複数のデータベース・ドライバーへの対処が可能な Java SQL フレームワークです。

Driver クラスをロードすると、Driver クラスは自身のインスタンスを作成して、それを DriverManager に登録します。これは、ユーザーが、以下に示す呼び出しによって DB2 Everyplace JDBC ドライバーのロードと登録を行えることを意味します。

```
Class.forName("com.ibm.db2e.jdbc.DB2eDriver")
```

java.sql パッケージ

共通インターフェース **Driver**

表 102は、DB2 Everyplace がサポートする Driver インターフェースのメソッドをリストしたものです。

表 102. Driver インターフェース・メソッド

メソッドの 戻り値タイプ	メソッド
boolean	**acceptsURL**(String url) ドライバーが、指定された URL に対する接続をオープンできると判断した場合、true を戻す。
Connection	**connect**(String url, Properties info) 指定された URL へのデータベース接続を試みる。 java.util.Properties 引き数を使用して、任意のストリング・タグ/値のペアを接続引き数として渡すことができる。 DB2 Everyplace では、次のドライバー固有のキーおよび値のペアをサポートする。 • キー: LOGIN_TIMEOUT 値: 秒数 • キー: DB2e_ENCODING 値: 文字エンコード
int	**getMajorVersion**() ドライバーのメジャー・バージョン番号を取得する。
int	**getMinorVersion**() ドライバーのマイナー・バージョン番号を取得する。
boolean	**jdbcCompliant**() このドライバーが真正な JDBC COMPLIANT™ ドライバーであるかどうかを報告する。

関連したタスク:

- 21 ページの『DB2 Everyplace Java アプリケーションの開発』

関連した解説:

- 296 ページの『DB2 Everyplace JDBC サポートの概要』
- 202 ページの『JDBC により報告される SQLState メッセージ』

PreparedStatement インターフェース

PreparedStatement インターフェースは、プリコンパイルされた SQL ステートメントを表すオブジェクトを作成します。

SQL ステートメントはプリコンパイルされて、PreparedStatement オブジェクトに保管されます。これにより、このオブジェクトは、このステートメントを複数回効率的に実行する際に使用できるようになります。

java.sql パッケージ

共通インターフェース PreparedStatement

Statement の拡張

表 103は DB2 Everyplace がサポートする PreparedStatement インターフェースのメソッドをリストしたものです。

表 103. PreparedStatement インターフェース・メソッド

メソッドの戻り値タイプ	メソッド
void	**clearParameters**() 現行パラメーター値をすぐにクリアする。
boolean	**execute**() あらゆる種類の SQL ステートメントを実行する。
ResultSet	**executeQuery**() この PreparedStatement オブジェクトにおいて SQL 照会を実行して、照会によって生成された結果セットを戻す。
int	**executeUpdate**() この PreparedStatement オブジェクトにおいて SQL INSERT、 UPDATE、または DELETE ステートメントを実行する。
void	**setBigDecimal** (int parameterIndex, BigDecimal x) 指定されたパラメーターを java.lang.BigDecimal 値に設定する。このメソッドは、Palm OS 用の DB2 Everyplace JDBC ドライバーでは使用できない。
void	**setBoolean**(int parameterIndex, boolean x) 指定されたパラメーターを Java boolean 値に設定する。DB2 Everyplace JDBC ドライバーは、これをデータベースに送信する際に SQL SMALLINT 値に変換する。
void	**setBlob**(int i, Blob x) JDBC 2.0。BLOB パラメーターを設定する。
void	**setBytes**(int parameterIndex, byte[ ]x) 指定されたパラメーターを Java のバイト配列に設定する。
void	**setDate**(int parameterIndex, Date x) 指定されたパラメーターを java.sql.Date 値に設定する。
void	**setDouble**(int parameterIndex, double x) 指定されたパラメーターを Java double 値に設定する。DB2 Everyplace JDBC ドライバーは、これをデータベースに送信する際に SQL DECIMAL 値に変換する。
void	**setFloat**(int parameterIndex, float x) 指定されたパラメーターを Java float 値に設定する。 BigDecimal が float に変換される際に BigDecimal が float を表すには大きすぎる場合、必要に応じて FLOAT.NEGATIVE_INFINITY または FLOAT.POSITIVE_INFINITY に変換される。
void	**setInt** (int parameterIndex, int x) 指定されたパラメーターを Java int 値に設定する。
void	**setLong**(int parameterIndex, long x) 指定されたパラメーターを Java long 値に設定する。
void	**setNull** (int parameterIndex, int sqlType) 指定されたパラメーターを SQL NULL に設定する。

表 103. PreparedStatement インターフェース・メソッド (続き)

メソッドの 戻り値タイプ	メソッド
void	**setObject**(int parameterIndex, Object x, int targetSqlType) 指定されたパラメーターの値を指定されたオブジェクトで設定する。 **DB2 Everyplace の制約事項**: • targetSqlType は DB2 Everyplace がサポートするデータ・タイプの 1 つに対応していなければならない。 • 基本変換およびストリング変換をサポートする。例えば、targetSqlType が Types.INTEGER の場合、 x は Integer または String オブジェクトのいずれかである。 • targetSqlType が Types.DECIMAL の場合、x は Double、Float、または Long オブジェクトにもなる。 • targetSqlType が Types.SMALLINT の場合、 x は Boolean オブジェクトにもなる。 • Palm OS では、targetSqlType が Types.DECIMAL の場合、x は String オブジェクトになる。
void	**setShort** (int parameterIndex, short x) 指定されたパラメーターを Java short 値に設定する。
void	**setString** (int parameterIndex, String x) 指定されたパラメーターを Java String 値に設定する。
void	**setTime** (int parameterIndex, Time x) 指定されたパラメーターを java.sql.Time 値に設定する。
void	**setTimestamp** (int parameterIndex, Timestamp x) 指定されたパラメーターを java.sql.Timestamp 値に設定する。

関連したタスク:

- 21 ページの『DB2 Everyplace Java アプリケーションの開発』

関連した解説:

- 296 ページの『DB2 Everyplace JDBC サポートの概要』
- 202 ページの『JDBC により報告される SQLState メッセージ』

ResultSet インターフェース

ResultSet インターフェースは、データ表へのアクセスを提供します。ResultSet オブジェクトは通常、ステートメントを実行することによって生成されます。

ResultSet は、データの現在行を指すカーソルを保存しています。最初に、カーソルは始めの行の前に置かれます。next() メソッドは、カーソルを次の行へ移動させます。

getXXX メソッドは、現在行の列の値を検索します。列の索引番号または列の名前のいずれかを使用して、値を検索することができます。通常は、列索引を使用するほうが効率的です。列は 1 から番号が付けられます。

java.sql パッケージ

共通インターフェース **ResultSet**

表 104は、DB2 Everyplace がサポートする ResultSet インターフェースをリストしたものです。

表 104. ResultSet インターフェース・フィールド

フィールド・タイプ	フィールド
`static int`	**CONCUR_READ_ONLY** 更新できない ResultSet オブジェクトの並行性モードを示す定数。**注**: CONCUR_UPDATABLE は DB2 Everyplace ではサポートされない。ステートメント・オブジェクトを作成する際に ResultSet オブジェクト用に CONCUR_UPDATABLE を並行性モードに指定すると、DB2 Everyplace JDBC ドライバーはステートメントを作成した Connection オブジェクトについて SQLWarning を発行し、代わりに CONCUR_READ_ONLY を使用する。
`static int`	**TYPE_FORWARD_ONLY** カーソルが順方向のみに動く ResultSet オブジェクト用のタイプを示す定数。
`static int`	**TYPE_SCROLL_INSENSITIVE** スクロール可能であるが、通常は他からの変更に影響されない ResultSet オブジェクトのタイプを示す定数。**注**: パフォーマンスに影響するため、このタイプの ResultSet オブジェクトは控えめに使用してください。このタイプは、CLI ステートメント属性 SQL_ATTR_CURSOR_SENSITIVITY の値の SQL_INSENSITIVE を使用します。詳細については、CLI 関数 SQLSetStmtAttr の資料を参照してください。
`static int`	**TYPE_SCROLL_SENSITIVE** スクロール可能であるが、通常は他からの変更に影響されない ResultSet オブジェクトのタイプを示す定数。**注**: このタイプは、CLI ステートメント属性 SQL_ATTR_CURSOR_SENSITIVITY の値の SQL_UNSPECIFIED を使用します。 詳細については、CLI 関数 SQLSetStmtAttr の資料を参照してください。

表 105は、DB2 Everyplace がサポートする ResultSet インターフェースのメソッドをリストしたものです。

表 105. ResultSet インターフェース・メソッド

メソッドの戻り値タイプ	メソッド
`boolean`	**absolute**(`int row`) JDBC 2.0。カーソルを結果セットの指定された行番号に移動する。
`void`	**afterLast**() JDBC 2.0。カーソルを結果セットの最後、つまり最終行の後に移動する。
`void`	**beforeFirst**() JDBC 2.0。カーソルを結果セットの先頭、つまり最初の行の前に移動する。
`void`	**clearWarnings**() この ResultSet オブジェクトで報告されるすべての警告をクリアする。
`void`	**close**() この ResultSet オブジェクトのデータベースおよび JDBC リソースを、オブジェクトが自動的にクローズしてそれらが解放されるのを待たずに、直ちに解放する。
`int`	**findColumn**(String `columnName`) 指定された ResultSet 列名を ResultSet 列索引にマップする。

表 105. ResultSet インターフェース・メソッド (続き)

メソッドの戻り値タイプ	メソッド
boolean	**first**() JDBC 2.0。カーソルを結果セットの最初の行に移動する。
BigDecimal	**getBigDecimal**(int columnIndex) JDBC 2.0。現在行の列の値を完全精度の java.math.BigDecimal オブジェクトとして取得する。このメソッドは、Palm OS 用の DB2 Everyplace JDBC ドライバーではサポートされない。
BigDecimal	**getBigDecimal**(int columnIndex, int scale) この ResultSet オブジェクトの現在行の指定された列の値を、Java プログラム言語の java.math.BigDecimal オブジェクトとして取得する。このメソッドは、Palm OS 用の DB2 Everyplace JDBC ドライバーではサポートされない。**使用すべきでない。**
BigDecimal	**getBigDecimal**(String columnName) JDBC 2.0。現在行の列の値を完全精度の java.math.BigDecimal オブジェクトとして取得する。このメソッドは、Palm OS 用の DB2 Everyplace JDBC ドライバーではサポートされない。
BigDecimal	**getBigDecimal**(String columnName, int scale) この ResultSet オブジェクトの現在行の指定された列の値を、Java プログラム言語の java.math.BigDecimal オブジェクトとして取得する。このメソッドは、Palm OS 用の DB2 Everyplace JDBC ドライバーではサポートされない。**使用すべきでない。**
Blob	**getBlob**(int columnIndex) JDBC 2.0。この ResultSet オブジェクトの現在行の BLOB 値を取得する。
Blob	**getBlob**(String columnName) JDBC 2.0。この ResultSet オブジェクトの現在行の BLOB 値を取得する。
boolean	**getBoolean**(int columnIndex) 現在行の列の値を Java boolean として取得する。
boolean	**getBoolean**(String columnName) 現在行の列の値を Java boolean として取得する。
byte	**getByte**(int columnIndex) この ResultSet オブジェクトの現在行の指定された列の値を、Java プログラム言語のバイトとして取得する。
byte	**getByte**(String columnName) この ResultSet オブジェクトの現在行の指定された列の値を、Java プログラム言語のバイトとして取得する。
byte[]	**getBytes**(int columnIndex) この ResultSet オブジェクトの現在行の指定された列の値を、Java プログラム言語のバイト配列として取得する。
byte[]	**getBytes**(String columnName) この ResultSet オブジェクトの現在行の指定された列の値を Java プログラム言語のバイト配列として取得する。
int	**getConcurrency**() JDBC 2.0。結果セットの並行性モードを戻す。
Date	**getDate**(int columnIndex) この ResultSet オブジェクトの現在行の指定された列の値を、Java プログラム言語の java.sql.Date オブジェクトとして取得する。
Date	**getDate**(int columnIndex, Calendar cal) この ResultSet オブジェクトの現在行の指定された列の値を、Java プログラム言語の java.sql.Date オブジェクトとして取得する。

表 105. *ResultSet* インターフェース・メソッド (続き)

メソッドの 戻り値タイプ	メソッド
Date	**getDate**(String columnName) この ResultSet オブジェクトの現在行の指定された列の値を Java プログラム言語の java.sql.Date オブジェクトとして取得する。
double	**getDouble**(int columnIndex) 現在行の列の値を Java double として取得する。
double	**getDouble**(String columnName) 現在行の列の値を Java double として取得する。
float	**getFloat**(int columnIndex) 現在行の列の値を Java float として取得する。
float	**getFloat**(String columnName) 現在行の列の値を Java float として取得する。
int	**getInt** (int columnIndex) この ResultSet オブジェクトの現在行の指定された列の値を、Java プログラム言語の整数として取得する。
int	**getInt** (String columnName) この ResultSet オブジェクトの現在行の指定された列の値を Java プログラム言語の整数として取得する。
long	**getLong**(int columnIndex) 現在行の列の値を Java long として取得する。
long	**getLong**(String columnName) 現在行の列の値を Java long として取得する。
ResultSetMetaData	**getMetaData**() この ResultSet オブジェクトの列の数、タイプ、プロパティーを検索する。
Object	**getObject**(int columnIndex) 現在行の列の値を Java オブジェクトとして取得する。
Object	**getObject**(String columnName) 現在行の列の値を Java オブジェクトとして取得する。
int	**getRow**() JDBC 2.0。現在行の番号を検索する。
short	**getShort**(int columnIndex) この ResultSet オブジェクトの現在行の指定された列の値を、Java プログラム言語の short として取得する。
short	**getShort**(String columnName) この ResultSet オブジェクトの現在行の指定された列の値を、Java プログラム言語の short として取得する。
Statement	**getStatement**() JDBC 2.0。この ResultSet オブジェクトを作成した Statement を戻す。
String	**getString**(int columnIndex) この ResultSet オブジェクトの現在行の指定された列の値を、Java プログラム言語の String として取得する。
String	**getString**(String columnName) この ResultSet オブジェクトの現在行の指定された列の値を、Java プログラム言語の String として取得する。
Time	**getTime**(int columnIndex) この ResultSet オブジェクトの現在行の指定された列の値を、Java プログラム言語の java.sql.Time オブジェクトとして取得する。

表 105. ResultSet インターフェース・メソッド (続き)

メソッドの 戻り値タイプ	メソッド
Time	**getTime**(String columnName) この ResultSet オブジェクトの現在行の指定された列の値を、Java プログラム言語の java.sql.Time オブジェクトとして取得する。
Timestamp	**getTimestamp**(String columnName) この ResultSet オブジェクトの現在行の指定された列の値を、Java プログラム言語の java.sql.Timestamp オブジェクトとして取得する。
Timestamp	**getTimestamp**(int columnIndex) この ResultSet オブジェクトの現在行の指定された列の値を、Java プログラム言語の java.sql.Timestamp オブジェクトとして取得する。
int	**getType**() JDBC 2.0。この結果セットのタイプを戻す。
SQLWarning	**getWarnings**() この ResultSet での呼び出しにより報告された最初の警告を戻す。
boolean	**isAfterLast**() JDBC 2.0。カーソルが結果セットの最終行の後にあるかどうかを示す。
boolean	**isBeforeFirst**() JDBC 2.0。カーソルが結果セットの最初の行の前にあるかどうかを示す。
boolean	**isFirst**() JDBC 2.0。カーソルが結果セットの最初の行にあるかどうかを示す。
boolean	**isLast**() JDBC 2.0。カーソルが結果セットの最終行にあるかどうかを示す。このメソッドは、タイプ TYPE_FORWARD_ONLY の結果セットではサポートされない。
boolean	**last**() JDBC 2.0。カーソルを結果セットの最終行に移動する。
boolean	**next**() カーソルを現在位置から 1 行下へ移動させる。
boolean	**previous**() JDBC 2.0。カーソルを結果セットの直前の行に移動する。
boolean	**relative**(int rows) JDBC 2.0。カーソルを正または負の相対行数の分移動する。
boolean	**wasNull**() 最後に読み取られた列の値が SQL NULL であるかどうかを報告する。

関連したタスク:

- 21 ページの『DB2 Everyplace Java アプリケーションの開発』

関連した解説:

- 296 ページの『DB2 Everyplace JDBC サポートの概要』
- 202 ページの『JDBC により報告される SQLState メッセージ』
- 285 ページの『SQLSetStmtAttr - ステートメントに関するオプションの設定』

ResultSetMetaData インターフェース

ResultSetMetaData インターフェースは、ResultSet 内の列のタイプおよびプロパティーを検索する際に使用できるオブジェクトを作成します。

java.sql パッケージ

共通インターフェース ResultSetMetaData

表 106は、DB2 Everyplace がサポートする ResultSetMetaData インターフェースのフィールドをリストしたものです。

表 106. ResultSetMetaData インターフェース・フィールド

フィールド・タイプ	フィールド
static int	**columnNoNulls** 列が NULL 値を許可しないことを示す定数。
static int	**columnNullable** 列が NULL 値を許可することを示す定数。
static int	**columnNullableUnknown** 列の値をヌルにできるかどうかが不明であることを示す定数。

表 107は、DB2 Everyplace がサポートする ResultSetMetaData インターフェースのメソッドをリストしたものです。

表 107. ResultSetMetaData インターフェース・メソッド

メソッドの戻り値タイプ	メソッド
String	**getCatalogName**(int column) 列の表のカタログ名を取得する。DB2 Everyplace は常に "" (適用外)を戻す。
int	**getColumnCount**() この ResultSet オブジェクトにおける列番号を戻す。
int	**getColumnDisplaySize** (int column) 指定された列の通常の最大幅を文字数で示す。
String	**getColumnLabel**(int column) 印刷出力および表示で使用する推奨される列タイトルを取得する。
String	**getColumnName** (int column) 指定された列の名前を取得する。
int	**getColumnType** (int column) 指定された列の SQL タイプを取得する。
String	**getColumnTypeName**(int column) 列のデータベース固有のタイプ名を検索する。
int	**getPrecision** (int column) 指定された列の小数桁数を取得する。
int	**getScale** (int column) 指定された列の小数点の右側の桁数を取得する。
String	**getSchemaName**(int column) 列の表のスキーマ名を取得する。DB2 Everyplace は常に "" (適用外)を戻す。
int	**isNullable** (int column) 指定された列の値のヌル指定の可否を示す。
boolean	**isWritable**(int column) 列での書き込みが続けられるかどうかを示す。

関連したタスク:

- 21 ページの『DB2 Everyplace Java アプリケーションの開発』

関連した解説:

- 296 ページの『DB2 Everyplace JDBC サポートの概要』
- 202 ページの『JDBC により報告される SQLState メッセージ』

Statement インターフェース

Statement インターフェースは、静的 SQL ステートメントの実行と、そのステートメントによって生成される結果の取得に使用されるオブジェクトを作成します。

java.sql パッケージ

共通インターフェース **Statement**

表 108は、DB2 Everyplace がサポートする Statement インターフェースのメソッドをリストしたものです。

表 108. Statement インターフェース・メソッド

メソッドの戻り値タイプ	メソッド
void	**addBatch**(String sql) JDBC 2.0 SQL コマンドをステートメント用のコマンドの現行バッチに追加する。
void	**clearBatch**() JDBC 2.0 現行バッチ内にあるコマンドのセットを空にする。
void	**close**() この Statement オブジェクトのデータベースおよび JDBC リソースを、オブジェクトが自動的にクローズしてそれらが解放されるのを待たずに、直ちに解放する。
boolean	**execute**(Stringsql) 複数の結果を戻す可能性のある SQL ステートメントを実行する。
int[]	**executeBatch**() JDBC 2.0 実行のためにコマンドのバッチをデータベースにサブミットする。
ResultSet	**executeQuery**(Stringsql) 単一の ResultSet オブジェクトを戻す SQL ステートメントを実行する。
int	**executeUpdate**(Stringsql) SQL INSERT、UPDATE、または DELETE ステートメントを実行する。
Connection	**getConnection** () JDBC 2.0。この Statement オブジェクトを作成した Connection オブジェクトを戻す。
boolean	**getMoreResults**() ステートメントの次の結果に移動する。DB2 Everyplace は常に false (結果がこれ以上ない) を戻す。
ResultSet	**getResultSet**() 現在の結果を ResultSet オブジェクトとして戻す。
int	**getResultSetConcurrency**() JDBC 2.0。結果セット並行性を検索する。
int	**getResultSetType**() JDBC 2.0。結果セット・タイプを判別する。
int	**getUpdateCount**() 現在の結果を更新カウントとして戻す。結果が ResultSet または結果がない場合、-1 を戻す。

314 ページの表 109は、DB2 Everyplace がサポートする Statement インターフェースのフィールドをリストしたものです。

表 109. Statement インターフェース・フィールド

フィールド・タイプ	フィールド
`static int`	**SUCCESS_NO_INFO** バッチ・ステートメントが正常に実行されたが、そのステートメントが影響を与える行の数のカウントがないことを示す定数。

関連したタスク:

- 21 ページの『DB2 Everyplace Java アプリケーションの開発』

関連した解説:

- 296 ページの『DB2 Everyplace JDBC サポートの概要』
- 202 ページの『JDBC により報告される SQLState メッセージ』

DB2eStatement クラス

DB2eStatement クラスは、特定の Statement 属性を取得し、設定します。 Statement オブジェクトで DB2eStatement クラス・メソッドを使用するためには、まず Statement オブジェクトが DB2eStatement オブジェクトにキャストされていなければなりません。これらのメソッドは、CLI/ODBC 関数である SQLGetStmtAttr および SQLSetStmtAttr を、適切な引き数を指定して呼び出すことによりインプリメントされます。

詳しくは、 202 ページの『DB2 CLI 関数の要約』を参照してください。

com.ibm.db2e.jdbc パッケージ

共通クラス **DB2eStatement**

Statement のインプリメント

表 110は、DB2 Everyplace がサポートする DB2eStatement クラスのメソッドをリストしたものです。

表 110. DB2eStatement クラス・メソッド

メソッドの戻りタイプ	メソッド
`void`	**enableDeletePhysicalRemove**(`boolean enable`) DELETE SQL ステートメントで、レコードのダーティー・ビット値にかかわらず、レコードの物理的な除去を使用可能または使用不可にする。
`void`	**enableDirtyBitSetByApplication**(`boolean enable`) 使用可能が真であれば、アプリケーション・モードを使用可能にする。それ以外の場合は、システム・モードを使用可能にする。
`void`	**enableReadIncludeMarkedDelete** (`boolean enable`) 論理的に削除されたレコードを、可視あるいは不可視にする。
`void`	**enableReorg** (`boolean enable`) DB2 Everyplace または ユーザーの明示的な REORG SQL ステートメントによるデータベース再編成を使用可能または使用不可にする。

表 110. *DB2eStatement* クラス・メソッド (続き)

メソッドの戻りタイプ	メソッド
boolean	**isEnabledDeletePhysicalRemove**() delete SQL ステートメントが、レコードのダーティー・ビット値にかかわらず、レコードを物理的に除去するか? あるいは、レコードに「削除」とマークするのみとするか?
boolean	**isEnabledDirtyBitSetByApplication**() データベース・システムはアプリケーション・モードであるか? あるいは、システム・モードであるか?
boolean	**isEnabledReadIncludeMarkedDelete**() 論理的に削除されたレコードは SQL ステートメントから可視であるか? あるいは、これらのレコードは SQL からは不可視であるか?
boolean	**isEnabledReorg**() データベース再編成を DB2 Everyplace またはユーザーの明示的な REORG SQL ステートメントによって行えるか? あるいは、REORG SQL ステートメントに制限があり、ユーザーが作成した表の自動データベース再編成が使用不可であるか?

以下の例で、st は Statement オブジェクトを、 rs は ResultSet オブジェクトを表します。

ダーティー・ビットの状況を無視してレコードを表 T から物理的に除去するには、次のようにします。

```
DB2eStatement db2e_st = (DB2eStatement) st;
db2e_st.enableDeletePhysicalRemove(true);
st.executeUpdate("DELETE FROM T WHERE X<>0");
```

ダーティー・ビットが DELETE とマークされたレコードを含め、ダーティー・ビットが設定されている表 T のすべてのレコードを読み取るには、次のようにします。

```
DB2eStatement db2e_st = (DB2eStatement) st;
db2e_st.enableReadIncludeMarkedDelete(true);
rs = st.executeQuery("SELECT * FROM T WHERE $dirty<>0");
```

表 T のレコードのダーティー・ビットをクリーンにするには、次のようにします。

```
DB2eStatement db2e_st = (DB2eStatement) st;
db2e_st.enableDirtyBitSetByApplication(true);
st.executeUpdate("UPDATE T SET $dirty=0 WHERE $dirty>0");
```

関連したタスク:

- 21 ページの『DB2 Everyplace Java アプリケーションの開発』

関連した解説:

- 296 ページの『DB2 Everyplace JDBC サポートの概要』
- 202 ページの『JDBC により報告される SQLState メッセージ』
- 269 ページの『SQLGetStmtAttr - ステートメント属性の現在の設定の取得』
- 285 ページの『SQLSetStmtAttr - ステートメントに関するオプションの設定』

javax.sql パッケージ内のインターフェース

この章では javax.sql パッケージ内の JDBC メソッドについての情報を提供します。説明されているトピックは、以下のとおりです。

- 『DataSource インターフェース』

DataSource インターフェース

この DataSource オブジェクトが表す物理データ・ソースへ接続するためのファクトリー。DriverManager ファシリティーに置き換わるもので、接続する方法として DataSource オブジェクトが優先されます。

DataSource オブジェクトのインスタンスは、Connection オブジェクトを作成するために独立型プログラムの中で使用できます。以下の例では、DB2eDataSource のインスタンスが、URL「jdbc:db2e:myDataSource」の DB2 Everyplace データベースへの Connection を作成するために使用されます。

```
com.ibm.db2e.jdbc.DB2eDataSource ds = new com.ibm.db2e.jdbc.DB2eDataSource();
    ds.setUrl("jdbc:db2e:myDataSource");
    Connection con = ds.getConnection();
```

javax.sql パッケージ

共通インターフェース **DataSource**

表 111および表 112は、DB2 Everyplace がサポートする DataSource インターフェースのプロパティーをリストしたものです。プロパティーは、「getter」および「setter」メソッドを使用してアクセスできます。(DataSource プロパティーは、JavaBeans™ 1.01 仕様の中の JavaBeans コンポーネントのプロパティーで指定された規則に従います。)

表 111. DB2 Everyplace がサポートする標準 DataSource プロパティー

プロパティー名	タイプ	説明
説明	String	このデータ・ソースの説明
password	String	データベースのパスワード
user	String	ユーザーのアカウント名

表 112は、DB2 Everyplace 固有の DataSource インターフェースでサポートされているプロパティーをリストしたものです。

表 112. DataSource インターフェースでの DB2 Everyplace 固有のプロパティー

プロパティー名	タイプ	説明
encoding	String	文字のエンコード方式
URL	String	データ・ソース

表 113は、DB2 Everyplace がサポートする DataSource インターフェースのメソッドをリストしたものです。

表 113. DataSource インターフェース・メソッド

メソッドの戻り値タイプ	メソッド
Connection	**getConnection**() この DataSource オブジェクトが表すデータ・ソースとの接続を確立しようとする試行。
Connection	**getConnection** (java.lang.String username, java.lang.String password) この DataSource オブジェクトが表すデータ・ソースとの接続を確立しようとする試行。

表 113. DataSource インターフェース・メソッド (続き)

メソッドの戻り値タイプ	メソッド
int	**getLoginTimeout**() データベースへの接続試行中にこのデータ・ソースが待機できる最大秒数を取得する。
java.io.PrintWriter	**getLogWriter**() この DataSource オブジェクト用のログ・ライターを検索する。
void	**setLoginTimeout**(int seconds) データベースへの接続試行中にこのデータ・ソースが待機できる最大秒数を設定する。
void	**setLogWriter**(java.io.PrintWriter out) この DataSource オブジェクト用のログ・ライターを指定された java.io.PrintWriter オブジェクトに設定する。

関連したタスク:

- 21 ページの『DB2 Everyplace Java アプリケーションの開発』

関連した解説:

- 296 ページの『DB2 Everyplace JDBC サポートの概要』
- 202 ページの『JDBC により報告される SQLState メッセージ』

サポートされる .NET クラス

この章では、DB2 Everyplace がサポートする .NET クラスについての情報を提供します。本章には、以下のセクションがあります。

- 『DB2eCommandBuilder メンバー』
- 318 ページの『DB2eCommand メンバー』
- 319 ページの『DB2eConnection メンバー』
- 320 ページの『DB2eDataAdapter メンバー』
- 321 ページの『DB2eDataReader メンバー』
- 322 ページの『DB2eError メンバー』
- 322 ページの『DB2eException メンバー』
- 322 ページの『DB2eParameter メンバー』
- 324 ページの『DB2eTransaction メンバー』
- 324 ページの『DB2eType 列挙』

DB2eCommandBuilder メンバー

表 114. 共通静的 (共有) メソッド

メソッド	説明
DeriveParameters	*DB2eCommand* で指定されたストアード・プロシージャーからパラメーター情報を検索し、指定した *DB2eCommand* オブジェクトの Parameters コレクションを取り込む。

表 115. 共通インスタンス・コンストラクター

コンストラクター	説明
DB2eCommandBuilder()	過負荷。 *DB2eCommandBuilder* クラスの新規インスタンスを初期化します。
DB2eCommandBuilder(DB2eDataAdapter)	過負荷。 関連する *DB2eDataAdapter* オブジェクトを使用して *DB2eCommandBuilder* クラスの新規インスタンスを初期化します。

表 116. 共通インスタンス・プロパティー

プロパティー	説明
DataAdapter	この *DB2eCommandBuilder* オブジェクトが SQL ステートメントを生成する *DB2eDataAdapter* オブジェクトを取得または設定する。

表 117. 共通インスタンス・メソッド

メソッド	説明
GetDeleteCommand	データベースで削除を実行する際に必要となる、自動的に生成された *DB2eCommand* オブジェクトを取得する。
GetInsertCommand	データベースで挿入を実行する際に必要となる、自動的に生成された *DB2eCommand* オブジェクトを取得する。
GetUpdateCommand	データベースで更新を実行する際に必要となる、自動的に生成された *DB2eCommand* オブジェクトを取得する。
RefreshSchema	INSERT、UPDATE、または DELETE ステートメントの生成に使用するデータベース・スキーマ情報をリフレッシュする。

表 118. 保護インスタンス・メソッド

メソッド	説明
Dispose	過負荷。

DB2eCommand メンバー

表 119. 共通インスタンス・コンストラクター

コンストラクター	説明
DB2eCommand()	過負荷。 *DB2eCommand* クラスの新規インスタンスを初期化します。
DB2eCommand(string)	過負荷。 照会のテキストを使用して *DB2eCommand* クラスの新規インスタンスを初期化します。
DB2eCommand(string, DB2eConnection)	過負荷。 照会のテキストと *DB2eConnection* オブジェクトを使用して *DB2eCommand* クラスの新規インスタンスを初期化します。
DB2eCommand(string, DB2eConnection, DB2eTransaction)	過負荷。 照会のテキスト、*DB2eConnection* オブジェクト、および *DB2eTransaction* オブジェクトを使用して *DB2eCommand* クラスを初期化します。

表 120. 共通インスタンス・プロパティー

プロパティー	説明
CommandText	SQL ステートメントまたはストアード・プロシージャーを取得または設定して、データベースに対して実行する。
CommandType	*CommandText* プロパティーの解釈方法を示す値を取得または設定する。
Connection	この *DB2eCommand* のインスタンスが使用する *DB2eConnection* を取得または設定する。
DesignTimeVisible	コマンド・オブジェクトがカスタマイズされたインターフェース制御で可視であるべきか否かを示す値を取得または設定する。
Parameters	*DB2eParameterCollection* を取得する。
Transaction	*DB2eCommand* が実行される *DB2eTransaction* を取得または設定する。
UpdatedRowSource	*Update* メソッドがコマンド結果を *DataRow* に適用する方法を指定する値を取得または設定する。

表 121. 共通インスタンス・メソッド

メソッド	説明
CreateParameter	*DB2eParameter* オブジェクトの新規インスタンスを作成する。
Dispose	過負荷。 クリーンアップします。
EnableDeletePhysicalRemove	物理的に除去されたレコードを使用可能または使用不可にする。
EnableDirtyBitSetByApplication	使用可能が真であれば、アプリケーション・モードを使用可能にする。それ以外の場合は、システム・モードを使用可能にする。
EnableReadIncludeMarkedDelete	論理的に削除されたレコードを可視あるいは不可視にする。
EnableReorg	データベース再編成を、DB2 Everyplace によるか、または REORG SQL ステートメントを使用してユーザーによって明示的に、使用可能または使用不可にする。
ExecuteNonQuery	*Connection* に対して SQL ステートメントを実行し、影響のある行の数を戻す。

表 121. 共通インスタンス・メソッド (続き)

メソッド	説明
ExecuteReader	過負荷。 接続に対して *CommandText* を送信し、*DB2eDataReader* をビルドします。
ExecuteScalar	照会を実行し、照会が戻した結果セットにある先頭行の先頭列を戻す。余分の列または行は無視されます。
IsEnabledDeletePhysicalRemove	物理的な削除が使用可能な否かをチェックする。使用可能な場合は true を戻し、そうでない場合は false を戻します。
IsEnabledDirtyBitSetByApplication	データベース・システムがアプリケーション・モードにあるかシステム・モードにあるかをチェックする。使用可能な場合は true を戻し、そうでない場合は false を戻します。
IsEnabledReadIncludeMarkedDelete	論理的に削除されたレコードがアプリケーションで可視であるか、または不可視であるかをチェックする。 論理的に削除されたレコードがアプリケーションで可視であれば true を戻し、そうでない場合は false を戻します。
IsEnabledReorg	データベース再編成が使用可能であるかどうかをチェックする。使用可能であれば true を戻し、そうでない場合は false を戻します。
Prepare	コマンドの準備済み (つまりコンパイル済み) バージョンをデータベースで作成する。

DB2eConnection メンバー

表 122. 共通静的 (共有) メソッド

メソッド	説明
ReleaseObjectPool	最後の低レベル層接続が解放される際に DB2e 環境ハンドルが解放できることを示す。

表 123. 共通インスタンス・コンストラクター

コンストラクター	説明
DB2eConnection()	過負荷。 *DB2eConnection* クラスの新規インスタンスを初期化します。
DB2eConnection(string)	過負荷。 指定した接続ストリングを使用して *DB2eConnection* クラスの新規インスタンスを初期化します。

表 124. 共通インスタンス・プロパティー

プロパティー	説明
ConnectionString	データベースをオープンするのに使用するストリングを取得または設定する。
ConnectionTimeout	接続の確立において、接続の試行を終了してエラーを生成するまでに、接続の確立の試行を待機する時間を取得または設定する。
Database	現在のデータベースまたは接続がオープンした後に使用されるデータベースの名前を取得する。
ServerVersion	クライアントが接続しているサーバーのバージョンが入っているストリングを取得する。
State	現在の接続状態を取得する。

表 125. 共通インスタンス・メソッド

メソッド	説明
BeginTransaction	過負荷。 データベースでトランザクションを開始します。
ChangeDatabase	オープンしている *DB2eConnection* に関連した現行データベースを変更する。
Close	データベースへの接続をクローズする。これは、どのオープン接続をクローズするメソッドよりも優先します。
CreateCommand	*DB2eConnection* に関連した *DB2eCommand* オブジェクトを作成し戻す。
Open	*ConnectionString* が指定したプロパティー設定を持つデータ・ソースへの接続をオープンする。

表 126. 共通インスタンス・イベント

イベント	説明
InfoMessage	DB2 Everyplace が警告メッセージまたは通知メッセージを送信する際に発生する。
StateChange	接続の状態が変更されたときに発生する。

DB2eDataAdapter メンバー

表 127. 共通インスタンス・コンストラクター

コンストラクター	説明
DB2eDataAdapter()	過負荷。 *DB2eDataAdapter* クラスの新規インスタンスを初期化します。
DB2eDataAdapter(DB2eCommand)	過負荷。 指定した SQL SELECT ステートメントを使用して *DB2eDataAdapter* クラスの新規インスタンスを初期化します。
DB2eDataAdapter(string, DB2eConnection)	過負荷。 指定した SQL SELECT ステートメントおよび *DB2eConnection* オブジェクトを使用して *DB2eDataAdapter* クラスの新規インスタンスを初期化します。
DB2eDataAdapter(string, string)	過負荷。 指定した SQL SELECT ステートメントおよび接続ストリングを使用して *DB2eDataAdapter* クラスの新規インスタンスを初期化します。

表 128. 共通インスタンス・プロパティー

プロパティー	説明
AcceptChangesDuringFill (**DataAdapter** から継承)	*AcceptChanges* が *DataTable* に追加された後 *DataRow* で呼び出されたか否かを示す値を取得または設定する。
ContinueUpdateOnError (**DataAdapter** から継承)	行の更新中にエラーが発生した場合、例外の生成またはエラーの行を指定する値を取得または設定する。
DeleteCommand	データベース内のレコードの削除に使用される SQL ステートメントまたはストアード・プロシージャーを取得または設定する。
InsertCommand	新規レコードをデータ・ソースに挿入するのに使用する SQL ステートメントまたはストアード・プロシージャーを取得または設定する。
MissingMappingAction (**DataAdapter** から継承)	着信データと一致する表または列がない場合に取る処置を決定する。
MissingSchemaAction (**DataAdapter** から継承)	既存の *DataSet* スキーマが着信データと一致しない場合に取る処置を決定する。
SelectCommand	データベース内のレコードの選択に使用する SQL ステートメントまたはストアード・プロシージャーを取得または設定する。
TableMappings (**DataAdapter** から継承)	ソース表と *DataTable* の間のマスター・マッピングを提供するコレクションを取得する。
UpdateCommand	データベース内のレコードの更新に使用する SQL ステートメントまたはストアード・プロシージャーを取得または設定する。

表 129. 共通インスタンス・メソッド

メソッド	説明
Clone	リモート・オブジェクトとの通信に使用するプロキシーを生成するのに必要なすべての関連情報が入ったオブジェクトを作成する。
Fill (**DbDataAdapter** から継承)	*DataSet* または *DataTable* に対して行を追加またはリフレッシュして、データ・ソース内で突き合わせる。
FillSchema (**DbDataAdapter** から継承)	*DataTable* を *DataSet* へ追加し、スキーマを構成して、データ・ソース内で突き合わせる。
GetFillParameters (**DbDataAdapter** から継承)	SQL SELECT ステートメントを実行する際にユーザーが設定したパラメーターを取得する。
Update (**DbDataAdapter** から継承)	DataSet 内の挿入、更新、または削除されるそれぞれの行に対して、それぞれ INSERT、UPDATE、または DELETE ステートメントを呼び出す。

表 130. 共通インスタンス・イベント

イベント	説明
FillError (**DbDataAdapter** から継承)	充てん操作中にエラーが発生した際に戻される。
RowUpdated	データベースでコマンドが実行された後の更新操作で発生する。
RowUpdating	データベースでコマンドが実行される前の*更新* で発生する。

DB2eDataReader メンバー

表 131. 共通インスタンス・プロパティー

プロパティー	説明
Depth	現在行のネストの深さを示す値を取得する。
FieldCount	現在行の列数を取得する。
IsClosed	*DB2eDataReader* がクローズしているか否かを示す。
Item	過負荷。 列の序数を指定して、指定した列の値をネイティブ・フォーマットで取得します。
RecordsAffected	SQL ステートメントの実行で変更、挿入、削除された行数を取得する。

表 132. 共通インスタンス・メソッド

メソッド	説明
Close	*DB2eDataReader* オブジェクトをクローズする。
GetByte	指定した列の値をバイトとして取得する。
GetBytes	指定したバッファー・オフセットから開始して、指定した列オフセットからのバイト・ストリームを、配列としてバッファーに読み取る。
GetDataTypeName	ソース・データ・タイプの名前を取得する。
GetDate	指定した列の値を *DateTime* オブジェクトとして取得する。
GetDateTime	指定した列の値を *DateTime* オブジェクトとして取得する。
GetDecimal	指定した列の値を *Decimal* オブジェクトとして取得する。
GetDouble	指定した列の値を倍精度浮動小数点数として取得する。
GetFieldType	オブジェクトのデータ・タイプである *Type* を取得する。
GetFloat	指定した列の値を、単精度浮動小数点数として取得する。
GetInt16	指定した列の値を 16 ビット符号付き整数として取得する。
GetInt32	指定した列の値を 32 ビット符号付き整数として取得する。
GetInt64	指定した列の値を 64 ビット符号付き整数として取得する。
GetName	指定した列の名前を取得する。
GetOrdinal	列の名前を指定して、列の序数を取得する。
GetSchemaTable	*DB2eDataReader* の列メタデータを記述する *DataTable* を戻す。
GetString	指定した列の値をストリングとして取得する。
GetTime	指定した列の値を *TimeSpan* オブジェクトとして取得する。

表 132. 共通インスタンス・メソッド (続き)

メソッド	説明
GetValue	指定した序数の列の値をネイティブ・フォーマットで取得する。
GetValues	現在行のすべての属性列を取得する
IsDBNull	列に存在しない値、つまり欠落値が含まれているかどうかを示す値を取得する。
NextResult	バッチ SQL ステートメントの結果を読み込むときに、*DB2eDataReader* を次の結果に進める。DB2 Everyplace では、現在、バッチ SQL ステートメントはサポートしていません。
Read	*DB2eDataReader* を次のレコードに進める。

DB2eError メンバー

表 133. 共通インスタンス・プロパティー

プロパティー	説明
Message	エラーの簡略説明を取得する。
NativeError	DB2 Everyplace からエラー情報を取得する。
SQLState	データベースに関する ANSI SQL 標準に準拠した 5 文字のエラー・コードを取得する。

DB2eException メンバー

表 134. 共通インスタンス・プロパティー

プロパティー	説明
Errors	DB2 Everyplace .NET Data Provider が生成する例外についての詳細情報を提供する、1 つまたは複数の *DB2eError* オブジェクトの集合を取得する。
Message	エラーを説明したテキスト記述を取得する。

DB2eParameter メンバー

表 135. 共通インスタンス・コンストラクター

コンストラクター	説明
DB2eParameter()	過負荷。 *DB2eParameter* クラスの新規インスタンスを初期化します。
DB2eParameter(string, object)	過負荷。 パラメーター名およびパラメーターの値を指定して *DB2eParameter* クラスの新規インスタンスを初期化します。
DB2eParameter(string, DB2eType)	過負荷。 パラメーター名およびデータ・タイプを指定して *DB2eParameter* クラスの新規インスタンスを初期化します。

表 135. 共通インスタンス・コンストラクター (続き)

コンストラクター	説明
DB2eParameter(string, DB2eType, int)	過負荷。 パラメーター名、データ・タイプ、および幅を指定して *DB2eParameter* クラスの新規インスタンスを初期化します。
DB2eParameter(string, DB2eType, int, string)	多重定義。パラメーター名、データ・タイプ、幅、およびソース列名を指定して *DB2eParameter* クラスの新規インスタンスを初期化します。
DB2eParameter(string, DB2eType, int, ParameterDirection, bool, byte, byte, string, DataRowVersion, object)	過負荷。 パラメーター名、データ・タイプ、幅、パラメーター方向、NULL 可能ブール、数値精度、位取り、ソース列名、ソース・バージョン、およびパラメーターの値を指定して *DB2eParameter* クラスの新規インスタンスを初期化します。

表 136. 共通インスタンス・プロパティー

プロパティー	説明
DB2eType	パラメーターの *DB2eType* を取得または設定する。
DbType	
Direction	パラメーターが、入力のみ、出力のみ、双方向、またはストアード・プロシージャー戻り値パラメーターかどうかを示す値を取得または設定する。
IsNullable	パラメーターが NULL 値を受け入れるかどうかを示す値を取得または設定する。
ParameterName	*DB2eParameter* の名前を取得または設定する。
Precision	*Value* プロパティーを表すのに使用する最大桁数を取得または設定する。
Scale	*Value* が解決される小数点以下の桁数を取得または設定する。
Size	列内のデータのバイト数での最大サイズを取得または設定する。
SourceColumn	*DataSet* にマッピングされ、*Value* をロードまたは戻すために使用するソース列の名前を取得または設定する。
SourceVersion	*Value* をロードする際に使用する *DataRowVersion* を取得または設定する。
Value	パラメーターの値を取得または設定する。

表 137. 共通インスタンス・メソッド

メソッド	説明
ToString	*ParameterName* が入っているストリングを取得する。

DB2eTransaction メンバー

表 138. 共通インスタンス・プロパティー

プロパティー	説明
Connection	トランザクションに関連した *DB2eConnection* オブジェクトを指定する。
IsolationLevel	このトランザクションの *IsolationLevel* を指定する。

表 139. 共通インスタンス・メソッド

メソッド	説明
Commit	データベース・トランザクションをコミットする。
Rollback	トランザクションを保留状態からロールバックする。

DB2eType 列挙

フィールド、プロパティー、または *DB2eParameter* のデータ・タイプを指定します。

[Visual Basic]

Public Enum DB2eType

[C#]

public enum DB2eType

以下の表に、*DB2eType* データ・タイプ、 DB2 Everyplace データ・タイプ(括弧内)、および .NET Framework タイプ間のマッピングを示します。

表 140. データ・タイプ・マッピング

メンバー	説明
SmallInt	精度 5、位取り 0 の厳密な数値 (符号付き: -32,768 <= n <= 32,767、符号なし: 0 <= n <= 65,535) (SMALLINT)。 これは、Int16 にマッピングされます。
整数	精度 10、位取り 0 の厳密な数値 (符号付き: -2[31] <= n <= 2[31] - 1、符号なし:0 <= n <= 2[32] - 1) (INTEGER)。 これは、Int32 にマッピングされます。
Char	固定長文字ストリング (CHAR)。これは、String にマッピングされます。
VarChar	可変長文字ストリング (VARCHAR)。これは String にマッピングされます。
Decimal	少なくとも精度 p、位取り s (ここで、1 <= p <= 31 および s <= p) の符号付きの厳密な数値 (DECIMAL)。これは Decimal にマッピングされます。

表 140. データ・タイプ・マッピング (続き)

メンバー	説明
Date	yyyy-mm-dd という形式の日付データ (DATE)。これは DateTime にマッピングされます。
Time	hh:mm:ss という形式の時間データ (TIME)。これは TimeSpan にマッピングされます。
Timestamp	yyyy-mm-dd-hh.mm.ss.zzzzzz という形式のタイムスタンプ・データ (TIMESTAMP)。これは DateTime にマッピングされます。
Blob	バイナリー・データのストリーム (BLOB)。これはタイプ Byte の配列にマッピングされます。

要件:

ネーム・スペース: IBM.Data.DB2.DB2e Namespace

アセンブリー: IBM.Data.DB2.DB2e.dll

IBM 同期クライアント C-API

この章では IBM 同期クライアント C-API についての情報を提供します。説明されているトピックは、以下のとおりです。

- 『IBM 同期クライアント C-API バージョン 8.1 と バージョン 7.2 の比較』
- 328 ページの『IBM 同期クライアント C-API の関数の要約』
- 329 ページの『IBM 同期クライアント C-API のデータ・タイプ』
- 331 ページの『IBM 同期クライアント C-API の関数の説明』

IBM 同期クライアント C-API バージョン 8.1 と バージョン 7.2 の比較

本セクションでは、バージョン 8.1 で IBM 同期クライアント C-API に加えられた主な変更点を要約します。

- サービス、構成、およびエンジンという 3 つのハンドルが使用できるようになりました。 (同期化を実行しない場合、同期エンジン・ハンドルをオープンする必要はありません。)
- IBM 同期クライアント C-API バージョン 8.1 ではプリファレンスが永続的なものではなくなり、実際には必須の情報である一部のプリファレンスが廃止されました。例えば、旧 `isyncSetPref` 関数のホスト名、ポート、ユーザー名、およびパスワードは、サービス・ハンドルをオープンする `iscOpenService` 関数の必要パラメーターになりました。
- 同期モードはアプリケーションに対して暗黙的なものとなり、同期化を実行する際、同期モードのパラメーターは不要になりました。
- 同期リスナーへのインターフェースが、イベント・ベースとなりました。アプリケーションには、イベント情報を含むイベント構造体が渡されるようになりました。

- サブスクリプション・セットの同期状況 (その最後の同期化以降の) が永続的なものとなり、後から照会を行うことができるようになりました。
- デフォルトのリスナーが廃止されました。イベントのデフォルトのアクションが必要な場合、アプリケーションは単に ISCRTNCB_Default コードを戻します。
- DB2e Everyplace が、機密データを含む表を保護するためのデータ暗号化をサポートするようになりました。暗号化された表を同期化するための同期クライアント・アプリケーションを開発する際、 DB2 Everyplace のユーザー名とパスワードを問い合わせる、同期エンジンからの照会を (リスナーに) インプリメントすることができます。
- リジェクトされたレコード (競合または無許可の操作が発生したレコードを含む) が、リスナーを介してアプリケーションに渡されるようになりました。
- ログ・ファイル (LOGDB-ISYNC) がアプリケーションによって管理されるようになりました。つまり、バージョン 8.1 の同期エンジンでは、バージョン 7.2.1 のような、各国語でのログ・ファイルの生成は行われません。代わりに、サービスを目的として、同期エンジンがトレース・ファイル (TRACE-ISYN) を生成します (英語のみ)。
- IBM 同期クライアントのエンジンでは、すべてのファイル (構成、トレース・ファイル、データ、およびプリファレンス (該当する場合)) を、以下に示す 1 つのディレクトリーに保管します。
  - Windows CE® オペレーティング・システムの場合: ¥ (ルート・ディレクトリー)
  - EPOC オペレーティング・システムの場合: C:¥Systems¥Data¥ISync¥
  - Palm オペレーティング・システムの場合: メイン・メモリー
  - その他のオペレーティング・システムの場合: 現行ディレクトリー
- IBM 同期クライアント API バージョン 7.2.1 の機能も、 API の後方互換性を処理する API ラッパー (isynce ライブラリー) を通して、引き続きサポートされます。また、API ラッパーは、バージョン 7.2.1 と同一の以下に示すディレクトリーに、ネイティブ言語のログ・ファイル (LOGDB-ISYN) を生成します。
  - Windows CE® オペレーティング・システムの場合: ¥Progran Files¥ISync¥
  - EPOC オペレーティング・システムの場合: C:¥Systems¥Apps¥ISync¥
  - Palm オペレーティング・システムの場合: メイン・メモリー
  - その他のオペレーティング・システムの場合: 現行ディレクトリー

  また、isyncGo 関数および isyncSetSyncMoe 関数では、ISYNCOPTION_SkipConfig オプションおよび ISYNCOPTION_UseAppSignature オプションは機能しません。

  注: IBM 同期クライアント API バージョン 8.1 を使用する場合は、 API ラッパー (isynce) ライブラリーをインストールする必要があります。

327 ページの表 141 は、IBM 同期クライアント C-API バージョン 8.1 と IBM 同期クライアント API バージョン 7.2 の間における関数の主な違いをリストしたものです。

表 141. IBM 同期クライアント C-API バージョン 8.1 とバージョン 7.2 の比較

バージョン 8.1	バージョン 7.2	注釈
iscGetVersion	isyncGetVersion	iscGetVersion ではハンドルは必要ありません。
iscServiceOpen iscConfigOpeniscEngineOpen	isyncOpen	3 つのハンドルをオープンする必要があります。 ホスト、ポート、ユーザー名、およびパスワードは iscServiceOpen に指定され、永続的なものではありません。
iscServiceClose iscConfigClose iscEngineClose	isyncClose	3 つのハンドルをクローズする必要があります。
iscEngineSetListener	isyncSetListener	リスナーのプロトタイプおよびインターフェースが変更されました。
(なし)	isyncDefaultListener	外部デフォルト・リスナーは廃止されました。デフォルトのイベント処理のためには、ISCRTNCB_Default コードが戻されます。
iscEngineSetPref iscEngineGetPref	isyncSetPref isyncGetPref	2 つのプリファレンス (トレースおよびタイムアウト) のみが必要です。これらのプリファレンスは永続的ではありません。
iscEngineSync iscEngineSyncConfig	isyncGo	同期モードは必須ではなくなりました。 構成は、iscEngineSyncConfig でのみ更新できます。
iscConfigEnableSubsSet iscConfigDisableSubsSet iscConfigResetSubsSet	isyncSetSyncMode	汎用の同期モード設定は廃止されました。 サブスクリプション・セットの同期化は、iscConfigDisableSubsSet によってスキップ (使用不可に) することができます。
iscConfigOpenCursor iscConfigCloseCursor iscConfigGetNextSubsSet iscConfigSubsSetIsEnabled iscConfigSubsSetIsReset	isyncGetFirstApp isyncGetNextApp	サブスクリプション・セットを繰り返す前にカーソルをオープンします。 サブスクリプション・セットの照会を行うには、サブスクリプション・セット ID が必要です。
iscEngineGetInfo iscConfigPurge iscConfigGetSubsSetStatus		バージョン 8.1 からの新規 C-API です。

関連した概念:

- 129 ページの『同期クライアント C/C++ サンプル・アプリケーション』

関連したタスク:

- 18 ページの『C/C++ を使用した DB2 Everyplace 同期クライアント・アプリケーションの開発』

関連した解説:

- 『IBM 同期クライアント C-API の関数の要約』
- 329 ページの『IBM 同期クライアント C-API のデータ・タイプ』
- 331 ページの『IBM 同期クライアント C-API 関数の説明の要点』

IBM 同期クライアント C-API の関数の要約

表 142 は、DB2 Everyplace がサポートする IBM 同期クライアント C-API の関数とそれぞれの関数の目的を示したものです。

表 142. IBM 同期クライアント C-API の関数リスト

関数名	目的
iscGetVersion	同期クライアント C-API のバージョン番号を取得します。

表 143. IBM サービス API 関数のリスト

関数名	目的
iscServiceOpen	新規サービスをオープンします。
iscServiceOpenEx	プロパティーを使用して新規サービスをオープンします。
iscServiceClose	サービスをクローズします。

表 144. IBM 構成 API 関数のリスト

関数名	目的
iscConfigOpen	構成ストアへの接続をオープンします。
iscConfigClose	構成ストアへの接続をクローズします。
iscConfigPurge	構成を再初期化します。
iscConfigOpenCursor	サブスクリプション・セットを繰り返すためにカーソル (のハンドル) を取得します。
iscConfigCloseCursor	オープン・カーソルをクローズします。
iscConfigGetNextSubsSet	次のサブスクリプション・セットが存在する場合、その記述を取得します。
iscConfigEnableSubsSet	サブスクリプション・セットの同期化を使用可能にします。
iscConfigDisableSubsSet	サブスクリプション・セットの同期化を使用不可にします。
iscConfigResetSubsSet	サブスクリプション・セットをリセット・モードに戻します。
iscConfigSubsSetIsEnabled	サブスクリプション・セットの同期化が使用可能であるかどうかを照会します。

表 144. IBM 構成 API 関数のリスト (続き)

関数名	目的
iscConfigSubsSetIsReset	サブスクリプション・セットがリセットされているかどうかを照会します。
iscConfigGetSubsSetStatus	前の同期の同期状況を照会します。

表 145. IBM 同期エンジン API 関数のリスト

関数名	目的
iscEngineOpen	同期エンジンへのハンドルをオープンします。
iscEngineClose	同期エンジンへのオープン・ハンドルをクローズします。
iscEngineGetInfo	同期エンジンに関する一般情報を取得します。
iscEngineSetListener	使用するユーザー定義のリスナー関数について、同期化に通知します。
iscEngineListenerPF	ユーザー定義のリスナー関数のデータ・タイプです。
iscEngineSetPref	プリファレンスを設定します。
iscEngineGetPref	プリファレンス値を取り出します。
iscEngineSync	同期セッションを立ち上げます。
iscEngineSyncConfig	指定された構成とサーバーとを同期させます。

関連した概念:

- 129 ページの『同期クライアント C/C++ サンプル・アプリケーション』

関連したタスク:

- 18 ページの『C/C++ を使用した DB2 Everyplace 同期クライアント・アプリケーションの開発』

関連した解説:

- 325 ページの『IBM 同期クライアント C-API バージョン 8.1 と バージョン 7.2 の比較』
- 『IBM 同期クライアント C-API のデータ・タイプ』
- 331 ページの『IBM 同期クライアント C-API 関数の説明の要点』

IBM 同期クライアント C-API のデータ・タイプ

表 146 は、IBM 同期クライアント C-API で定義された新規のデータ・タイプをリストしたものです。 C-API の関数を呼び出す際は、引き数のタイプが関数のプロトタイプに適合していることを確認してください。

表 146. IBM 同期クライアント C-API のデータ・タイプ

データ・タイプ	説明
isy_VOID	void タイプ
isy_INT	整数
isy_UINT	符号なし整数
isy_INT16	2 バイト整数
isy_UINT16	2 バイト符号なし整数

表 146. IBM 同期クライアント C-API のデータ・タイプ (続き)

データ・タイプ	説明
isy_INT32	4 バイト整数
isy_UINT32	4 バイト符号なし整数
isy_ULONG	符号なし長整数
isy_BYTE	1 バイト・タイプ
isy_WORD	1 ワード・タイプ
isy_DWORD	2 ワード・タイプ
isy_TCHAR	文字タイプ
isy_BOOL	ブール・タイプ
HISCSERV	サービス・ハンドルのデータ・タイプ
HISCCONF	構成ハンドルのデータ・タイプ
HISCENG	同期エンジン・ハンドルのデータ・タイプ
HISCCSR	サブスクリプション・セットの反復カーソルのデータ・タイプ

ISCEVT	リスナー・イベントのデータ・タイプ
	`typedef struct {`
	`  isy_INT32  code;`
	`  isy_UINT32  type;`
	`  isy_INT32  retry;`
	`  ISCSTATE  state;`
	`  ISCLISTENARG  *info;`
	`} ISCEVT;`

ISCSTATE	イベント状態のデータ・タイプ
	`typedef struct {`
	`  isy_TCHAR  currSubsSet[ISCLEN_SubsSetName];`
	`  isy_TCHAR  currSubs[ISCLEN_SubsName];`
	`  isy_UINT32  subsType;`
	`  isy_INT32  syncProg;`
	`} ISCSTATE;`

ISCLISTENARG	同期リスナーの情報のデータ・タイプで、ストリング引き数 (argc、argv) のリストから構成されます。
	`typedef struct {`
	`  isy_INT32  argc;`
	`  isy_TCHAR  **argv;`
	`} ISCLISTENARG;`

ISCLISTENCOLUMN	同期リスナーの情報のデータ・タイプで、列の位置、主キー・シーケンス、列タイプ、データ・サイズ、および実際の列データを含む表の列から構成されます。
	`typedef struct {`
	`  isy_INT32  pos;`
	`  isy_INT32  pkseq;`
	`  isy_INT32  type;`
	`  isy_INT32  size;`
	`  isy_BYTE  *data;`
	`} ISCLISTENCOLUMN;`
	
	DB2 Everyplace のヘッダー・ファイル `sqlcli.h` には、列タイプのさまざまな列タイプ定数が定義されています。列データはヌル終了のテキスト・ストリングとして表されています。これは、実際の列データ (データ・フィールド) がプレーン・バイト・ストリングとして表され、かつ、ヌル終了でない blob 列タイプ以外に当てはまります。また、サイズ・フィールドにそのサイズ (バイト数) が指定されます。

表 146. IBM 同期クライアント C-API のデータ・タイプ (続き)

データ・タイプ	説明

ISCLISTENCONFLICT	同期リスナーの情報のデータ・タイプで、表名、操作、列数、および列情報の配列 (ISCLISTENCOLUMN) を含む表レコードから構成されます。
	`typedef struct {`
	`  isy_TCHAR  table[ISCLEN_Table];`
	`  isy_INT32  op;`
	`  isy_INT32  colc;`
	`  ISCLISTENCOLUMN *colv;`
	`} ISCLISTENCONFLICT;`
	
	op フィールドはリジェクトされた操作を示します。この操作は、以下の操作定数のいずれかです (括弧内は実際の値)。
	• ISCCONST_OpDelete (1)
	• ISCCONST_OpInsert (2)
	• ISCCONST_OpUpdate (3)

関連した概念:

- 129 ページの『同期クライアント C/C++ サンプル・アプリケーション』

関連したタスク:

- 18 ページの『C/C++ を使用した DB2 Everyplace 同期クライアント・アプリケーションの開発』

関連した解説:

- 325 ページの『IBM 同期クライアント C-API バージョン 8.1 と バージョン 7.2 の比較』
- 328 ページの『IBM 同期クライアント C-API の関数の要約』
- 『IBM 同期クライアント C-API 関数の説明の要点』

IBM 同期クライアント C-API の関数の説明

本章では、IBM 同期クライアント C-API の関数について説明します。

IBM 同期クライアント C-API 関数の説明の要点

それぞれの同期クライアント C-API 関数の説明には、以下のセクションが含まれています。

目的 関数の機能についての概要です。

構文 汎用 C プロトタイプを示しています。汎用プロトタイプは、Windows を含むすべての環境で使用されます。

関数の引き数
各関数の引き数を、各引き数のデータ・タイプ、説明、使用タイプ (入力または出力) と共にリストしています。

使用法 関数の使用法に関する情報を示し、特別な考慮事項について説明しています。

戻りコード
関数のすべての戻りコードをリストしています。

制約事項

各同期クライアント C-API の関数を適用する際の相違点または制限を示しています。

参照 関連する同期クライアント C-API の関数をリストしています。

注: IBM 同期クライアント C-API には『診断』セクションはありません。

関連した概念:

- 129 ページの『同期クライアント C/C++ サンプル・アプリケーション』

関連したタスク:

- 18 ページの『C/C++ を使用した DB2 Everyplace 同期クライアント・アプリケーションの開発』

関連した解説:

- 325 ページの『IBM 同期クライアント C-API バージョン 8.1 と バージョン 7.2 の比較』
- 329 ページの『IBM 同期クライアント C-API のデータ・タイプ』
- 328 ページの『IBM 同期クライアント C-API の関数の要約』

iscGetVersion()

目的: `iscGetVersion()` は、同期クライアント C-API のバージョン番号を取得します。

構文:

```
isy_UINT32 iscGetVersion();
```

関数の引き数:

なし。

使用法:

`iscGetVersion()` は、同期クライアント C-API のバージョン番号を取り出すために使用します。バージョン番号は、*0xmmnnrrxx* という形式の 32 ビット符号なし整数として戻されます。ここで *mm*、*nn*、および *rr* は、それぞれメジャー、マイナー、およびモディフィケーションの各バージョン番号の 16 進表記です。 *xx* は予約値です。

例::

```
isy_UINT32  version;
       int verMajor, verMinor, verModi;
       version = iscGetVersion();
       verMajor = (int) (version >> 24);
       verMinor = (int) ((version >> 16) & 0x000000FF);
       verModi  = (int) ((version >> 8)  & 0x000000FF);
```

戻りコード:

同期クライアント C-API のバージョン番号。

制約事項:

なし。

関連した概念:

- 129 ページの『同期クライアント C/C++ サンプル・アプリケーション』

関連したタスク:

- 18 ページの『C/C++ を使用した DB2 Everyplace 同期クライアント・アプリケーションの開発』

関連した解説:

- 351 ページの『iscEngineGetInfo()』
- 325 ページの『IBM 同期クライアント C-API バージョン 8.1 と バージョン 7.2 の比較』
- 329 ページの『IBM 同期クライアント C-API のデータ・タイプ』
- 331 ページの『IBM 同期クライアント C-API 関数の説明の要点』
- 328 ページの『IBM 同期クライアント C-API の関数の要約』

iscServiceOpen()

目的: `iscServiceOpen()` は、新規サービス・ハンドルをオープンします。

構文:

```
isy_INT32 iscServiceOpen(
    isy_CONST isy_TCHAR* host,
    isy_CONST isy_TCHAR* port,
    isy_CONST isy_TCHAR* username,
    isy_CONST isy_TCHAR* password,
    isy_CONST isy_VOID*  reserved,
    HISCSERV*            phServ);
```

関数の引き数:

表 147 は、`iscServiceOpen()` 関数で使用される有効な引き数をリストしたものです。

表 147. iscServiceOpen() の引き数

データ・タイプ	引き数	用途	説明
isy_CONST isy_TCHAR*	*host*	入力	ホスト名または IP
isy_CONST isy_TCHAR*	*port*	入力	ポート番号
isy_CONST isy_TCHAR*	*username*	入力	要求されたサービスのユーザー名
isy_CONST isy_TCHAR*	*password*	入力	要求されたサービスのパスワード
isy_CONST isy_TCHAR*	*reserved*	入力	(予約済み)
HISCSERV*	*phServ*	出力	サービスへのハンドル

使用法:

`iscServiceOpen()` は、ホスト名およびポート番号によって識別される特定のサービスの新規ハンドルを要求するために使用します。ユーザー名およびパスワードは、サービスを要求する際に指定します。処理が正常に行われると、サービス・ハンド

ル (HISCSERV) が *phServ を介して戻されます。処理が正常に行われなかった場合は、*phServ は NULL となり、エラー・コードが戻されます。

戻りコード:

- ISCRTN_Succeeded: 正常
- ISCRTN_OutOfMemory: メモリー不足
- ISCRTN_ResourceBusy: リソースがロック状態である (例えば、別のアプリケーションなどによって)
- ISCRTN_NotPermitted: リソースにアクセス不能である (例えば、読み取り不可の場合など)
- ISCRTN_NotFound: リソースが見つからない (例えば、パスが見つからない場合など)
- ISCRTN_Failed: その他

制約事項:

なし。

関連した概念:

- 129 ページの『同期クライアント C/C++ サンプル・アプリケーション』

関連したタスク:

- 18 ページの『C/C++ を使用した DB2 Everyplace 同期クライアント・アプリケーションの開発』

関連した解説:

- 325 ページの『IBM 同期クライアント C-API バージョン 8.1 と バージョン 7.2 の比較』
- 329 ページの『IBM 同期クライアント C-API のデータ・タイプ』
- 328 ページの『IBM 同期クライアント C-API の関数の要約』
- 336 ページの『iscServiceClose()』
- 『iscServiceOpenEx()』

iscServiceOpenEx()

目的: `iscServiceOpenEx()` は、プロパティー配列に基づいて新規サービス・ハンドルをオープンします。

構文:

```
isy_INT32 iscServiceOpenEx(
    isy_CONST isy_TCHAR* URL,
    ISCPROPERTY*          property,
    isy_INT32            propertyNum,
    HISCSERV*            phServ);
```

関数の引き数:

335 ページの表 148 は `iscServiceOpenEx()` 関数で使用する有効な引き数をリストしたものです。

表 148. iscServiceOpenEx() の引き数

データ・タイプ	引き数	用途	説明
isy_CONST isy_TCHAR	*URL*	入力	URL ストリングとしてのサーバー情報

ISCPROPERTY	property	入力	以下の ISCPROPERTY タイプのプロパティー配列
			`typedef struct {`
			`  isy_TCHAR *key;  //property ID string`
			`  isy_TCHAR *value; //property value string`
			`} ISCPROPERTY;`
			以下の 3 つのプロパティーが使用可能です。
			• *isync.user* — 同期クライアントのユーザー名
			• *isync.password* — 同期クライアントのパスワード
			• *isync.encoding* — ターゲット・データの文字エンコード
			ユーザー名およびパスワードのプロパティーは必須です。

isy_INT32	propertyNum	入力	プロパティー数。
HISCSERV	phServ	出力	サービスへのハンドル。

使用法:

`iscServiceOpenEx()` は、プロパティー配列で表された設定を持つサーバーから、特定のサービスの新規ハンドルを要求するために使用します。サーバーは URL (Uniform Resource Locator) ストリングによって識別され、プロトコル、ホスト名 (または IP)、およびポート番号が含まれています。同期サーバーが Secure Socket Layer (SSL) で構成されている場合は URL のプロトコルの部分は「https://」となり、それ以外の場合は「http://」となります。ポート番号は省略できますが、SSL および 非 SSL のデフォルト・ポートはそれぞれポート 443 およびポート 80 です。すべての設定 (ユーザー名およびパスワードを含む) はプロパティー配列の中に指定されます。処理が正常に行われると、サービス・ハンドル (HISCSERV) が *phServ* を介して戻されます。そうでない場合、*phServ* は NULL となり、エラー・コードが戻されます。完了したら、サービス・ハンドルは `iscServiceClose()` でクローズされます。

例:

```
int rc = 0;
    HISCSERV hSyncServ;
    ISCPROPERTY properties[3] = {{"isync.user", "myUserName"},
                    {"isync.password", "myPassword"},
                    {"isync.encoding", "ISO8859_1"}}
    rc = iscServiceOpenEx("http://localhost.mycom.com:80", properties, 3, &hSyncServ);
```

戻りコード:

- ISCRTN_Succeeded: 正常
- ISCRTN_OutOfMemory: メモリー不足
- ISCRTN_ResourceInUse : リソースがロック状態である (例えば、別のアプリケーションなどによって)
- ISCRTN_NotPermitted: リソースにアクセス不能である (例えば、読み取り不可の場合など)
- ISCRTN_NotFound: リソースが見つからない (例えば、パスが見つからない場合など)
- ISCRTN_Failed: その他

制約事項:

なし。

関連した解説:

- 338 ページの『iscConfigClose()』

iscServiceClose()

目的: `iscServiceClose()` は、オープン・サービス・ハンドルをクローズします。

構文:

```
isy_INT32 iscServiceClose(
    HISCSERV  hServ);
```

関数の引き数:

表 149 は、`iscServiceClose()` 関数で使用される有効な引き数をリストしたものです。

表 149. iscServiceClose() の引き数

データ・タイプ	引き数	用途	説明
HISCSERV	*hServ*	入力	サービス・ハンドル

使用法:

`iscServiceClose()` は、以前にオープンされたサービス・ハンドルのストレージを解放する際に使用します。

戻りコード:

- ISCRTN_Succeeded: 正常
- ISCRTN_Failed: その他

制約事項:

`iscServiceClose()` を複数回呼び出すとエラーが発生する可能性があるため、複数回呼び出さないようにしてください。

関連した概念:

- 129 ページの『同期クライアント C/C++ サンプル・アプリケーション』

関連したタスク:

- 18 ページの『C/C++ を使用した DB2 Everyplace 同期クライアント・アプリケーションの開発』

関連した解説:

- 325 ページの『IBM 同期クライアント C-API バージョン 8.1 と バージョン 7.2 の比較』
- 329 ページの『IBM 同期クライアント C-API のデータ・タイプ』
- 328 ページの『IBM 同期クライアント C-API の関数の要約』
- 333 ページの『iscServiceOpen()』

iscConfigOpen()

目的: `iscConfigOpen()` は、構成ストアへの接続をオープンします。

構文:

```
isy_INT32 iscConfigOpen(
    HISCSERV   hServ,
    isy_TCHAR *path,
    HISCCONF  *phConf);
```

関数の引き数:

表 150 は、`iscConfigOpen()` 関数で使用される有効な引き数をリストしたものです。

表 150. iscConfigOpen() の引き数

データ・タイプ	引き数	用途	説明
HISCSERV	*hServ*	入力	サービス・ハンドル
isy_TCHAR*	*path*	入力	作業ディレクトリーのパス
HISCCONF*	*phConf*	出力	構成接続

使用法:

`iscConfigOpen()` は、特定サービスの所定のパスに指定されたとおりに、構成ストアへの接続をオープンします。処理が正常に行われると、構成接続 (HISCCONF) が *phServ を介して戻されます。処理が正常に行われなかった場合は、*phServ は NULL となり、エラー・コードが戻されます。これが新規のサービス (新規のホストまたは新規のポートのいずれか) である場合、このサービス用の新しい空の構成が作成されます。

戻りコード:

- ISCRTN_Succeeded: 正常
- ISCRTN_OutOfMemory: メモリー不足
- ISCRTN_ResourceBusy: リソースがロック状態である (例えば、別のアプリケーションなどによって)
- ISCRTN_NotPermitted: リソースにアクセス不能である (例えば、読み取り不可の場合など)
- ISCRTN_NotFound: リソースが見つからない (例えば、パスが見つからない場合など)
- ISCRTN_Failed: その他

制約事項:

なし。

関連した概念:

- 129 ページの『同期クライアント C/C++ サンプル・アプリケーション』

関連したタスク:

- 18 ページの『C/C++ を使用した DB2 Everyplace 同期クライアント・アプリケーションの開発』

関連した解説:

- 325 ページの『IBM 同期クライアント C-API バージョン 8.1 と バージョン 7.2 の比較』
- 329 ページの『IBM 同期クライアント C-API のデータ・タイプ』
- 328 ページの『IBM 同期クライアント C-API の関数の要約』
- 『iscConfigClose()』

iscConfigClose()

目的: `iscConfigClose()` は、オープン構成ストア接続をクローズします。

構文:

```
isy_INT32 iscConfigClose(
    HISCCONF    hConf);
```

関数の引き数:

表 151 は、`iscConfigClose()` 関数で使用される有効な引き数をリストしたものです。

表 151. iscConfigClose() の引き数

データ・タイプ	引き数	用途	説明
HISCCONF	*hConf*	入力	構成接続

使用法:

`iscConfigClose()` は、以前にオープンされた構成ストア接続をクローズします。

戻りコード:

- ISCRTN_Succeeded: 正常
- ISCRTN_Failed: その他

制約事項:

なし。

関連した概念:

- 129 ページの『同期クライアント C/C++ サンプル・アプリケーション』

関連したタスク:

- 18 ページの『C/C++ を使用した DB2 Everyplace 同期クライアント・アプリケーションの開発』

関連した解説:

- 325 ページの『IBM 同期クライアント C-API バージョン 8.1 と バージョン 7.2 の比較』
- 329 ページの『IBM 同期クライアント C-API のデータ・タイプ』
- 328 ページの『IBM 同期クライアント C-API の関数の要約』
- 337 ページの『iscConfigOpen()』

iscConfigPurge()

目的: `iscConfigPurge()` は、構成ストアのサブスクリプション情報をすべて空にします。

構文:

```
isy_INT32 iscConfigPurge(
    HISCCONF    hConf);
```

関数の引き数:

表 152 は、`iscConfigPurge()` 関数で使用される有効な引き数をリストしたものです。

表 152. iscConfigPurge() の引き数

データ・タイプ	引き数	用途	説明
HISCCONF	*hConf*	入力	構成接続

使用法:

`iscConfigPurge()` は、構成ストアにあるすべてのユーザー・サブスクリプション情報を除去します。エンジンは、次回に同期化を実行する際に再度構成をサーバーから取り出し、すべてのサブスクリプション・セットについて全体のリフレッシュを実行します。

戻りコード:

- ISCRTN_Succeeded: 正常
- ISCRTN_Failed: その他

制約事項:

なし。

関連した概念:

- 129 ページの『同期クライアント C/C++ サンプル・アプリケーション』

関連したタスク:

- 18 ページの『C/C++ を使用した DB2 Everyplace 同期クライアント・アプリケーションの開発』

関連した解説:

- 325 ページの『IBM 同期クライアント C-API バージョン 8.1 と バージョン 7.2 の比較』
- 329 ページの『IBM 同期クライアント C-API のデータ・タイプ』
- 328 ページの『IBM 同期クライアント C-API の関数の要約』
- 345 ページの『iscConfigResetSubsSet()』

iscConfigOpenCursor()

目的: `iscConfigOpenCursor()` は、すべてのサブスクリプション・セットを反復処理するためにカーソルを取得します。

iscConfigOpenCursor()

構文:

```
isy_INT32 iscConfigOpenCursor(
    HISCCONF    hConf,
    HISCCSR    *phCursor);
```

関数の引き数:

表 153 は、iscConfigOpenCursor() 関数で使用される有効な引き数をリストしたものです。

表 153. iscConfigOpenCursor() の引き数

データ・タイプ	引き数	用途	説明
HSYNCCONF	*hConf*	入力	構成接続
HISCCSR*	*phCursor*	出力	サブスクリプション・セットの反復処理のために戻されるカーソル

使用法:

すべてのサブスクリプション・セットに対して反復処理が必要な場合は、iscConfigOpenCursor() を使用して適切なカーソルを取得してください。次に、iscConfigGetNextSubsSet() を使用して各サブスクリプション・セットおよびそれに対応する記述を取得します。

戻りコード:

- ISCRTN_Succeeded: 正常
- ISCRTN_Failed: その他

制約事項:

iscConfigOpenCursor() が呼び出されると、以前にオープンされたカーソルはすべて無効になるため、これらのカーソルをクローズしなければなりません。クローズされたカーソルを使用してサブスクリプション・セットを処理しようとすると、ISCRTN_Failed 戻りコードが生成されます。つまり、サブスクリプション・セットの反復を他の反復の中にネストすることはできません。同様に、構成が (iscEngineSync() または iscEngineSyncConfig() を使用して) 同期化されると、オープン・カーソルは無効になります。

関連した概念:

- 129 ページの『同期クライアント C/C++ サンプル・アプリケーション』

関連したタスク:

- 18 ページの『C/C++ を使用した DB2 Everyplace 同期クライアント・アプリケーションの開発』

関連した解説:

- 325 ページの『IBM 同期クライアント C-API バージョン 8.1 と バージョン 7.2 の比較』
- 329 ページの『IBM 同期クライアント C-API のデータ・タイプ』
- 328 ページの『IBM 同期クライアント C-API の関数の要約』
- 341 ページの『iscConfigCloseCursor()』
- 342 ページの『iscConfigGetNextSubsSet()』

iscConfigCloseCursor()

目的: `iscConfigCloseCursor()` は、オープン・カーソルをクローズします。

構文:

```
isy_INT32 iscConfigCloseCursor(
    HISCCONF    hConf,
    HISCCSR     hCursor);
```

関数の引き数:

表 154 は、`iscConfigCloseCursor()` 関数で使用される有効な引き数をリストしたものです。

表 154. iscConfigCloseCursor() の引き数

データ・タイプ	引き数	用途	説明
HISCCONF	*hConf*	入力	構成接続
HISCCSR	*hCursor*	入力	サブスクリプション・セットの反復処理のためのカーソル

使用法:

`iscConfigOpenCursor()` を使用してカーソルをオープンしたものの、カーソルが不要である場合には、 `iscConfigCloseCursor()` を使用してこのカーソルをクローズしてください。クローズしない場合、オープン・カーソルが原因となって、メモリー・リークやその他の構成の整合性に関する問題が発生する可能性があります。予期しないエラーが発生する可能性があるため、カーソルがクローズされた後に、クローズされたハンドルを使用しようとしないでください。

戻りコード:

- ISCRTN_Succeeded: 正常
- ISCRTN_Failed: その他

制約事項:

なし。

関連した概念:

- 129 ページの『同期クライアント C/C++ サンプル・アプリケーション』

関連したタスク:

- 18 ページの『C/C++ を使用した DB2 Everyplace 同期クライアント・アプリケーションの開発』

関連した解説:

- 325 ページの『IBM 同期クライアント C-API バージョン 8.1 と バージョン 7.2 の比較』
- 329 ページの『IBM 同期クライアント C-API のデータ・タイプ』
- 328 ページの『IBM 同期クライアント C-API の関数の要約』
- 339 ページの『iscConfigOpenCursor()』
- 342 ページの『iscConfigGetNextSubsSet()』

iscConfigGetNextSubsSet()

目的: iscConfigGetNextSubsSet() は、次のサブスクリプション・セットの記述が存在する場合はそれを取得し、カーソルをそのセットへ移動します。

構文:

```
isy_INT32 iscConfigGetNextSubsSet(
    HISCCONF    hConf,
    HISCCSR     hCursor,
    isy_TCHAR*  id,
    isy_TCHAR*  name);
```

関数の引き数:

表 155 は、iscConfigGetNextSubsSet() 関数で使用される有効な引き数をリストしたものです。

表 155. iscConfigGetNextSubsSet() の引き数

データ・タイプ	引き数	用途	説明
HISCCONF	*hConf*	入力	構成接続
HISCCSR	*hCursor*	入力	サブスクリプション・セットの反復処理のためのカーソル
isy_TCHAR*	*id*	出力	サブスクリプション・セットの ID
isy_TCHAR*	*name*	出力	サブスクリプション・セットの名前

使用法:

iscConfigGetNextSubsSet() は、サーバーからサブスクリプション・セットの ID を取得し、サブスクリプション・セット名が存在する場合はそれを取り出して、次のサブスクリプション・セットへカーソルを移動します。

例:

```
isy_TCHAR  id[ISCLEN_SubsSetID];
isy_TCHAR  name[ISCLEN_SubsSetName];
isy_INT32  isReset, isEnabled;
HISCCSR    hCursor;
isy_INT32  rc;

// start iteration of all subscription sets
rc = iscConfigOpenCursor(hConf, &hCursor);
while (rc == ISCRTN_Succeeded) {
    rc = iscConfigGetNextSubsSet(hConf, hCursor, id, name);
    if (rc == ISCRTN_Succeeded) {
        isReset = iscConfigSubsSetIsReset(hConf, id);
        isEnabled = iscConfigSubsSetIsEnabled(hConf, id);
        // processing the subscription set
        ...
        // get next subscription
    } // end of processing
} // end of iteration
iscConfigCloseCursor(hConf, hCursor);
```

戻りコード:

- ISCRTN_Succeeded: 正常
- ISCRTN_Empty: サブスクリプション・セットがこれ以上存在しない
- ISCRTN_Failed: その他

制約事項:

なし。

関連した概念:

- 129 ページの『同期クライアント C/C++ サンプル・アプリケーション』

関連したタスク:

- 18 ページの『C/C++ を使用した DB2 Everyplace 同期クライアント・アプリケーションの開発』

関連した解説:

- 325 ページの『IBM 同期クライアント C-API バージョン 8.1 と バージョン 7.2 の比較』
- 329 ページの『IBM 同期クライアント C-API のデータ・タイプ』
- 328 ページの『IBM 同期クライアント C-API の関数の要約』
- 347 ページの『iscConfigSubsSetIsReset()』
- 346 ページの『iscConfigSubsSetIsEnabled()』

iscConfigEnableSubsSet()

目的: `iscConfigEnableSubsSet()` は、構成内でサブスクリプション・セットの同期化を使用可能にします。

構文:

```
isy_INT32 iscConfigEnableSubsSet(
    HISCCONF    hConf,
    isy_TCHAR*  id);
```

関数の引き数:

表 156 は、`iscConfigEnableSubsSet()` 関数で使用される有効な引き数をリストしたものです。

表 156. iscConfigEnableSubsSet() の引き数

データ・タイプ	引き数	用途	説明
HISCCONF	*hConf*	入力	構成接続
isy_TCHAR*	*id*	入力	サブスクリプション・セットの ID

使用法:

初期状態では、すべてのサブスクリプション・セットで同期化が使用可能になっています。 `iscConfigEnableSubsSet()` 関数および `iscConfigDisableSubsSet()` 関数は、所定の ID で指定されるサブスクリプション・セットの同期機能を使用可能または使用不可にします。

戻りコード:

- ISCRTN_Succeeded: 正常
- ISCRTN_NotFound: サブスクリプション・セットが見つからない
- ISCRTN_Failed: その他

制約事項:

なし。

関連した概念:

- 129 ページの『同期クライアント C/C++ サンプル・アプリケーション』

関連したタスク:

- 18 ページの『C/C++ を使用した DB2 Everyplace 同期クライアント・アプリケーションの開発』

関連した解説:

- 325 ページの『IBM 同期クライアント C-API バージョン 8.1 と バージョン 7.2 の比較』
- 329 ページの『IBM 同期クライアント C-API のデータ・タイプ』
- 328 ページの『IBM 同期クライアント C-API の関数の要約』
- 『iscConfigDisableSubsSet()』
- 346 ページの『iscConfigSubsSetIsEnabled()』

iscConfigDisableSubsSet()

目的: `iscConfigDisableSubsSet()` は、サブスクリプション・セットの同期化を使用不可にします。

構文:

```
isy_INT32 iscConfigDisableSubsSet(
    HISCCONF    hConf,
    isy_TCHAR*  id);
```

関数の引き数:

表 157 は、`iscConfigDisableSubsSet()` 関数で使用される有効な引き数をリストしたものです。

表 157. iscConfigDisableSubsSet() の引き数

データ・タイプ	引き数	用途	説明
HISCCONF	*hConf*	入力	構成接続
isy_TCHAR*	*id*	入力	サブスクリプション・セットの ID

使用法:

初期状態では、すべてのサブスクリプション・セットで同期化が使用可能になっています。 `iscConfigEnableSubsSet()` 関数および `iscConfigDisableSubsSet()` 関数は、所定の ID で指定されるサブスクリプション・セットの同期機能を使用可能または使用不可にします。

戻りコード:

- ISCRTN_Succeeded: 正常
- ISCRTN_NotFound: サブスクリプション・セットが見つからない
- ISCRTN_Failed: その他

制約事項:

なし。

関連した概念:

- 129 ページの『同期クライアント C/C++ サンプル・アプリケーション』

関連したタスク:

- 18 ページの『C/C++ を使用した DB2 Everyplace 同期クライアント・アプリケーションの開発』

関連した解説:

- 325 ページの『IBM 同期クライアント C-API バージョン 8.1 と バージョン 7.2 の比較』
- 329 ページの『IBM 同期クライアント C-API のデータ・タイプ』
- 328 ページの『IBM 同期クライアント C-API の関数の要約』
- 343 ページの『iscConfigEnableSubsSet()』
- 346 ページの『iscConfigSubsSetIsEnabled()』

iscConfigResetSubsSet()

目的: `iscConfigResetSubsSet()` は、構成内でサブスクリプション・セットをリセット・モードにリセットします。

構文:

```
isy_INT32 iscConfigResetSubsSet(
    HISCCONF    hConf,
    isy_TCHAR*  id);
```

関数の引き数:

表 158 は、`iscConfigResetSubsSet()` 関数で使用される有効な引き数をリストしたものです。

表 158. iscConfigResetSubsSet() の引き数

データ・タイプ	引き数	用途	説明
HISCCONF	*hConf*	入力	構成接続
isy_TCHAR*	*id*	入力	サブスクリプション・セットの ID

使用法:

サブスクリプション・セットが同期化時にリセット・モードである場合、同期エンジンはそのサブスクリプション・セットのクライアント・データを除去します。同期エンジンは単にサーバー・データのみを取り出します (または再取り出しします)。このプロセスはリフレッシュと呼ばれます。サブスクリプション・セットが同期化された後には、このサブスクリプション・セットはリセット・モードではなくなります。指定したサブスクリプション・セットをリセット・モードに戻すには、`iscConfigResetSubsSet()` を使用してください。

戻りコード:

- ISCRTN_Succeeded: 正常

- ISCRTN_NotFound: サブスクリプション・セットが見つからない
- ISCRTN_Failed: その他

制約事項:

なし。

関連した概念:

- 129 ページの『同期クライアント C/C++ サンプル・アプリケーション』

関連したタスク:

- 18 ページの『C/C++ を使用した DB2 Everyplace 同期クライアント・アプリケーションの開発』

関連した解説:

- 325 ページの『IBM 同期クライアント C-API バージョン 8.1 と バージョン 7.2 の比較』
- 329 ページの『IBM 同期クライアント C-API のデータ・タイプ』
- 328 ページの『IBM 同期クライアント C-API の関数の要約』
- 347 ページの『iscConfigSubsSetIsReset()』

iscConfigSubsSetIsEnabled()

目的: `iscConfigSubsSetIsEnabled()` は、サブスクリプション・セットの同期化が使用可能であるかどうかを照会します。

構文:

```
isy_INT32 iscConfigSubsSetIsEnabled(
    HISCCONF    hConf,
    isy_TCHAR*  id);
```

関数の引き数:

表 159 は、`iscConfigSubsSetIsEnabled()` 関数で使用される有効な引き数をリストしたものです。

表 159. iscConfigSubsSetIsEnabled() の引き数

データ・タイプ	引き数	用途	説明
HISCCONF	*hConf*	入力	構成接続
isy_TCHAR*	*id*	入力	サブスクリプション・セットの ID

使用法:

`iscConfigSubsSetIsEnabled()` は、所定の ID で指定されたサブスクリプション・セットで同期化が使用可能になっているかどうかを照会するために使用します。初期状態では、すべてのサブスクリプション・セットで同期化が使用可能になっています。

戻りコード:

- ISCRTN_True: サブスクリプション・セットの同期化は使用可能
- ISCRTN_False: サブスクリプション・セットの同期化は使用不可

- ISCRTN_NotFound: サブスクリプション・セットが見つからない
- ISCRTN_Failed: その他

制約事項:

なし。

関連した概念:

- 129 ページの『同期クライアント C/C++ サンプル・アプリケーション』

関連したタスク:

- 18 ページの『C/C++ を使用した DB2 Everyplace 同期クライアント・アプリケーションの開発』

関連した解説:

- 325 ページの『IBM 同期クライアント C-API バージョン 8.1 と バージョン 7.2 の比較』
- 329 ページの『IBM 同期クライアント C-API のデータ・タイプ』
- 328 ページの『IBM 同期クライアント C-API の関数の要約』
- 『iscConfigSubsSetIsReset()』

iscConfigSubsSetIsReset()

目的: `iscConfigSubsSetIsReset()` は、サブスクリプション・セットがリセット・モードであるかどうかの照会を実行します。

構文:

```
isy_INT32 iscConfigSubsSetIsReset(
    HISCCONF    hConf,
    isy_TCHAR*  id);
```

関数の引き数:

表 160 は、`iscConfigSubsSetIsReset()` 関数で使用される有効な引き数をリストしたものです。

表 160. iscConfigSubsSetIsReset() の引き数

データ・タイプ	引き数	用途	説明
HISCCONF	*hConf*	入力	構成接続
isy_TCHAR*	*id*	入力	サブスクリプション・セットの ID

使用法:

初期状態では、すべてのサブスクリプション・セットがリセット・モードに設定されています。しかし、サブスクリプション・セットが同期化されると、サブスクリプション・セットのモードは変更されます。所定の ID で指定されたサブスクリプション・セットをリセット・モードに戻すには、`iscConfigResetSubsSet()` を使用します。

戻りコード:

- ISCRTN_True: サブスクリプション・セットはリセット・モードである

- ISCRTN_False: サブスクリプション・セットはリセット・モードではない
- ISCRTN_NotFound: サブスクリプション・セットが見つからない
- ISCRTN_Failed: その他

制約事項:

なし。

関連した概念:

- 129 ページの『同期クライアント C/C++ サンプル・アプリケーション』

関連したタスク:

- 18 ページの『C/C++ を使用した DB2 Everyplace 同期クライアント・アプリケーションの開発』

関連した解説:

- 325 ページの『IBM 同期クライアント C-API バージョン 8.1 と バージョン 7.2 の比較』
- 329 ページの『IBM 同期クライアント C-API のデータ・タイプ』
- 328 ページの『IBM 同期クライアント C-API の関数の要約』
- 346 ページの『iscConfigSubsSetIsEnabled()』

iscConfigGetSubsSetStatus()

目的: `iscConfigGetSubsSetStatus()` は、サブスクリプション・セットの同期状況を取得します。

構文:

```
isy_INT32 iscConfigGetSubsSetStatus(
    HISCCONF    hConf,
    isy_TCHAR*  id);
```

関数の引き数:

表 161 は、`iscConfigGetSubsSetStatus()` 関数で使用される有効な引き数をリストしたものです。

表 161. iscConfigGetSubsSetStatus() の引き数

データ・タイプ	引き数	用途	説明
HISCCONF	*hConf*	入力	構成接続
isy_TCHAR*	*id*	入力	サブスクリプション・セットの ID

使用法:

`iscConfigGetSubsSetStatus()` は、(指定された ID を持つ) サブスクリプション・セットの最新の同期化実行時の同期状況を照会するために使用します。

戻りコード:

- ISCRTN_Succeeded: サブスクリプション・セットの同期化が正常に終了した
- ISCRTN_Ready: サブスクリプション・セットは使用可能。同期化処理は開始されたが、サブスクリプション・セットは同期化されていない

- ISCRTN_Canceled: サブスクリプション・セットの同期化はキャンセルされた
- ISCRTN_Failed: サブスクリプション・セットの同期化が失敗した
- ISCRTN_NotFound: サブスクリプション・セットが見つからない

制約事項:

なし。

関連した概念:

- 129 ページの『同期クライアント C/C++ サンプル・アプリケーション』

関連したタスク:

- 18 ページの『C/C++ を使用した DB2 Everyplace 同期クライアント・アプリケーションの開発』

関連した解説:

- 325 ページの『IBM 同期クライアント C-API バージョン 8.1 と バージョン 7.2 の比較』
- 329 ページの『IBM 同期クライアント C-API のデータ・タイプ』
- 328 ページの『IBM 同期クライアント C-API の関数の要約』
- 363 ページの『iscEngineSync()』
- 346 ページの『iscConfigSubsSetIsEnabled()』

iscEngineOpen()

目的: `iscEngineOpen()` は、同期エンジンへのハンドルをオープンします。

構文:

```
isy_INT32 iscEngineOpen(
    HISCCONF     hConf,
    HISCENG     *phEngine);
```

関数の引き数:

表 162 は、`iscEngineOpen()` 関数で使用される有効な引き数をリストしたものです。

表 162. iscEngineOpen() の引き数

データ・タイプ	引き数	用途	説明
HISCCONF	*hConf*	入力	構成ハンドル
HISCENG*	*phEngine*	出力	同期エンジンへのハンドル

使用法:

`iscEngineOpen()` は、指定された構成を同期化する際に、同期エンジン (HISCENG) へのハンドルをオープンするために使用します。同期化が正常に終了すると、ハンドルが *phEngine を介して戻されます。同期化が正常に終了しなかった場合、*phEngine 値は NULL となり、エラー・コードが戻されます。

戻りコード:

- ISCRTN_Succeeded: 正常

- ISCRTN_OutOfMemory: メモリー不足
- ISCRTN_ResourceBusy: リソースがロック状態である (例えば、別のアプリケーションなどによって)
- ISCRTN_NotPermitted: リソースにアクセス不能である (例えば、リソースが読み取り不可の場合など)
- ISCRTN_NotFound: リソースが見つからない (例えば、パスが見つからない場合など)
- ISCRTN_Failed: その他

制約事項:

`iscEngineOpen()` を複数回呼び出すと、同期エンジンへの複数のハンドルがオープンして整合性に関する問題が発生する可能性があるため、複数回呼び出さないようにしてください。

関連した概念:

- 129 ページの『同期クライアント C/C++ サンプル・アプリケーション』

関連したタスク:

- 18 ページの『C/C++ を使用した DB2 Everyplace 同期クライアント・アプリケーションの開発』

関連した解説:

- 325 ページの『IBM 同期クライアント C-API バージョン 8.1 と バージョン 7.2 の比較』
- 329 ページの『IBM 同期クライアント C-API のデータ・タイプ』
- 328 ページの『IBM 同期クライアント C-API の関数の要約』
- 『iscEngineClose()』

iscEngineClose()

目的: `iscEngineClose()` は、同期エンジンへのオープン・ハンドルをクローズします。

構文:

```
isy_INT32 iscEngineClose(
   HISCENG    hEngine);
```

関数の引き数:

表 163 は、`iscEngineClose()` 関数で使用される有効な引き数をリストしたものです。

表 163. iscEngineClose() の引き数

データ・タイプ	引き数	用途	説明
HISCENG	*hEngine*	入力	同期エンジンへのハンドル

使用法:

`iscEngineClose()` は、同期エンジンへのオープン・ハンドルをクローズするために使用します。

戻りコード:

- ISCRTN_Succeeded: 正常
- ISCRTN_Failed: その他

制約事項:

`iscEngineClose()` を複数回呼び出すとエラーが発生する可能性があるため、複数回呼び出さないようにしてください。

関連した概念:

- 129 ページの『同期クライアント C/C++ サンプル・アプリケーション』

関連したタスク:

- 18 ページの『C/C++ を使用した DB2 Everyplace 同期クライアント・アプリケーションの開発』

関連した解説:

- 325 ページの『IBM 同期クライアント C-API バージョン 8.1 と バージョン 7.2 の比較』
- 329 ページの『IBM 同期クライアント C-API のデータ・タイプ』
- 328 ページの『IBM 同期クライアント C-API の関数の要約』
- 349 ページの『iscEngineOpen()』

iscEngineGetInfo()

目的: `iscEngineGetInfo()` は、同期エンジンに関する一般情報を取得します。

構文:

```
isy_INT32 iscEngineGetInfo(
    HISCENG       hEngine,
    isy_TCHAR    *info,
    isy_INT32     infoLen);
```

関数の引き数:

表 164 は、`iscEngineGetInfo()` 関数で使用される有効な引き数をリストしたものです。

表 164. iscEngineGetInfo() の引き数

データ・タイプ	引き数	用途	説明
HISCENG	*hEngine*	入力	同期エンジンへのハンドル
isy_TCHAR*	*info*	出力	戻り情報が保管されるバッファーへのポインター
isy_INT32	*infoLen*	入力	指定のバッファーのサイズ

使用法:

`iscEngineGetInfo()` は、サービス目的のために、同期エンジンに関する情報を取得します。情報の内容およびフォーマットは、今後変更される場合があります。した

がって、アプリケーションではこの情報を表示目的またはログ目的にのみ使用してください。この情報をアプリケーション・プログラム処理の入力としては使用しないでください。

戻りコード:

- ISCRTN_Succeeded: 正常
- ISCRTN_ValTruncated: 情報の実際の長さが infoLen より長い
- ISCRTN_Failed: その他

制約事項:

なし。

関連した概念:

- 129 ページの『同期クライアント C/C++ サンプル・アプリケーション』

関連したタスク:

- 18 ページの『C/C++ を使用した DB2 Everyplace 同期クライアント・アプリケーションの開発』

関連した解説:

- 325 ページの『IBM 同期クライアント C-API バージョン 8.1 と バージョン 7.2 の比較』
- 329 ページの『IBM 同期クライアント C-API のデータ・タイプ』
- 328 ページの『IBM 同期クライアント C-API の関数の要約』
- 332 ページの『iscGetVersion()』

iscEngineSetListener()

目的: `iscEngineSetListener()` は、ユーザー定義のリスナー関数を同期エンジンに登録します。リスナー関数は、同期セッション中に、同期イベント (同期化の開始など) またはエラーが発生した際に呼び出されます。

構文:

```
isy_INT32 iscEngineSetListener(
    HISCENG            hEngine,
    iscEngineListenerPF syncListener,
    isy_UINT32         syncListenerData);
```

関数の引き数:

表 165 は、`iscEngineSetListener()` 関数で使用される有効な引き数をリストしたものです。

表 165. iscEngineSetListener() の引き数

データ・タイプ	引き数	用途	説明
HISCENG	*hEngine*	入力	同期エンジンへのハンドル
iscEngineListenerPF	*syncListener*	入力	ユーザー定義のリスナー関数のアドレス
isy_UINT32	*syncListenerData*	入力	アプリケーションがユーザー定義のリスナー関数へ転送するデータ

使用法:

ユーザー定義のリスナー関数を登録することによって、アプリケーションは同期化処理を表示することができます。同期化中にイベントまたはエラーが発生すると、アプリケーションに通知されます。アプリケーションでは、これらのイベントまたはエラーをユーザーに表示するメソッドをカスタマイズすることができます。

例:

```
// Function syncListener is defined with the following prototype:
isy_INT32 mySyncListener(
    isy_UINT32 listenerData,
    ISCEVT*  event,
    isy_VOID*  pExtraInfo);
...
// Handle to the synchronization engine is passed to the listener function
iscEngineSetListener(hEngine, mySyncListener, (isy_UINT32) hEngine);
```

戻りコード:

- ISCRTN_Succeeded: 正常
- ISCRTN_Failed: その他

制約事項:

ユーザー定義のリスナー関数は、同期エンジンのプロトコルに従わなければなりません。従わない場合、同期エンジンが正しく動作しない可能性があります。

関連した概念:

- 129 ページの『同期クライアント C/C++ サンプル・アプリケーション』

関連したタスク:

- 18 ページの『C/C++ を使用した DB2 Everyplace 同期クライアント・アプリケーションの開発』

関連した解説:

- 325 ページの『IBM 同期クライアント C-API バージョン 8.1 と バージョン 7.2 の比較』
- 329 ページの『IBM 同期クライアント C-API のデータ・タイプ』
- 328 ページの『IBM 同期クライアント C-API の関数の要約』
- 363 ページの『iscEngineSync()』

iscEngineListenerPF

目的: iscEngineListenerPF は、iscEngineSetListener() で登録されたユーザー定義のリスナー関数が準拠しなければならないプロトタイプを定義します。

構文:

```
typedef isy_INT32 (*iscEngineListenerPF)(
    isy_UINT32  listenerData,
    ISCEVT*     event,
    isy_VOID*   pExtraInfo);
```

関数の引き数:

354 ページの表 166 は、iscEngineSetListenerPF 関数タイプで使用される有効な引き数をリストしたものです。

表 166. iscEngineListenerPF の引き数

データ・タイプ	引き数	用途	説明
isy_UINT32	*listenerData*	入力	*iscEngineSetListener()* による *syncListenerData* 引き数のデータ・セットは、リスナー関数に転送されて戻されます
ISCEVT*	*event*	入力	イベント・オブジェクト
isy_VOID*	*pExtraInfo*	入力	予約済み

使用法:

同期化の進行状況をモニターするためにユーザー定義のリスナー関数を使用するには、まず、関数が `iscEngineSetListenerPF` 関数タイプに準拠するようにしなければなりません。次に、`iscEngineSetListener()` 関数を使用してリスナー関数を登録します。これで、同期イベントが発生するとユーザー定義のリスナー関数に通知されます。 `event` 引き数は、イベントに関するさまざまな情報を含む構造体です。

表 167 は、イベント構造体の全フィールドと、各フィールドの目的をリストしたものです。

表 167. iscEngineListenerPF イベントのフィールド

フィールド	説明
type	イベント・タイプは、以下のいずれかの値にすることができます (括弧内は実際の値)。 **ISCEVTTYPE_Info (1)** 同期化の進行状況に関する情報。 **ISCEVTTYPE_Conflict (2)** 同期化処理における競合または操作のリジェクト。 **ISCEVTTYPE_Query (3)** 同期化を継続するには、多少の情報が必要です。同期エンジンの処理を継続させるには、アプリケーション側がいくつかの必須情報 (イベント・コードに基づく) を指定しなければなりません。 **ISCEVTTYPE_Retry (4)** 例外が発生しており、同期化を継続するには再試行命令またはキャンセル命令が必要です。 **ISCEVTTYPE_Error (5)** エラーが発生しており、同期エンジンは現在のサブスクリプション・セットの同期化処理を継続できません。 **ISCEVTTYPE_Fatal (6)** 致命的エラーが発生しており、同期エンジンはサブスクリプション・セットの同期化処理を継続できません。

表 167. iscEngineListenerPF イベントのフィールド (続き)

フィールド	説明
state	以下のサブフィールドを含む、イベントの状態です。 **currSubsSet** 空ではない場合は、サブスクリプション・セットの名前。 **currSubs** 空ではない場合は、サブスクリプションの名前。 **subsType** 0 以外の場合は、サブスクリプション・タイプ。数字の範囲は以下のとおり。 • 100 ～ 999: 予約済み • 1000 ～ 9999: 登録済みサブスクリプション・タイプ • 10000+: カスタム・サブスクリプション・タイプ 事前定義の発信元は以下のとおり (括弧内は実際の値)。 – ISCSUBSTYPE_Config (100): 構成 – ISCSUBSTYPE_File (101): ファイル・サブスクリプション – ISCSUBSTYPE_DB2e (102): DB2 Everyplace の表サブスクリプション **syncProg** % で表される同期化の進行状況。
retry	0 以外の場合は、同じイベントに対する再試行の回数。
info	イベント固有のオプショナル情報 (NULL 以外の場合)。これは、非競合イベントの場合、ストリング引き数の配列です。競合イベントの場合、データ・タイプは ISCLISTENCONFLICT です。

event.info フィールドには、イベント固有のオプショナル情報がいくつか含まれています。この情報を識別して解釈するには、イベント・コードを使用します。

表 168 は、イベント・タイプのカテゴリー別にすべてのイベント・コードをリストしたものです。

表 168. iscEngineListenerPF のイベント・コード： イベント・タイプ: ISCEVTTYPE_Info

イベント・コード	イベント情報 (argc)	説明
ISCEVT_InfGeneral (1000)	NULL	一般情報 (デバッグ目的) です。
ISCEVT_InfSyncStarted (1001)	NULL	同期化が開始されました。
ISCEVT_InfPrepMsg (1002)	NULL	メッセージの準備中。
ISCEVT_InfSendMsg (1003)	NULL	メッセージの送信中。
ISCEVT_InfWaitMsg (1004)	NULL	サーバー応答の待機中。
ISCEVT_InfApplyMsg (1005)	NULL	サーバー・メッセージを適用中です。
ISCEVT_InfCancelingSync (1006)	NULL	同期の取り消し中。
ISCEVT_InfSubsSetStarted (1007)	NULL	サブスクリプション・セットの同期化が開始されました。
ISCEVT_InfSyncingSubs (1008)	NULL	サブスクリプションの同期化が開始されました。
ISCEVT_InfSubsSetFailed (1009)	NULL	サブスクリプション・セットの同期化が失敗しました。
ISCEVT_InfSubsSetCanceled (1010)	NULL	サブスクリプション・セットの同期化がキャンセルされました。
ISCEVT_InfSubsSetSucceeded (1011)	NULL	サブスクリプション・セットの同期化が正常に終了しました。
ISCEVT_InfSyncSucceeded (1012)	NULL	同期化が正常に終了しました。

表 168. *iscEngineListenerPF* のイベント・コード *(続き)*： イベント・タイプ: ISCEVTTYPE_Info

イベント・コード	イベント情報 (argc)	説明
SCEVT_InfSyncFailed (1013)	NULL	(一部のサブスクリプション・セットで) 同期化が失敗しました。
ISCEVT_InfSyncCanceled (1014)	NULL	同期化が (ユーザーによって) キャンセルされました。
ISCEVT_InfSyncProg (1015)	NULL	% で表される同期化の進行状況です。
ISCEVT_InfNoNewChange (1016)	NULL	新規のサーバー変更はありません。プルおよび確認フェーズをスキップします。
ISCEVT_InfLoginFailed (1017)	NULL	指定したログイン情報が認証プロセスをパスしました。

表 169. *iscEngineListenerPF* のイベント・コード： イベント・タイプ: ISCEVTTYPE_Conflict

イベント・コード	イベント情報 (argc)	説明
ISCEVT_CftReject (2000)	ISCLISTENCONFLICT	データ競合が同期内で発見されました。実際の競合データは ISCLISTENCONFLICT 構造体で表され、その参照ポインターが *event.info* を介してアプリケーションに戻されます。

表 170. *iscEngineListenerPF* のイベント・コード： イベント・タイプ: ISCEVTTYPE_Retry

イベント・コード	イベント情報 (argc)	説明
ISCEVT_TryNetConn (4601)	NULL	サーバーへの接続を再試行します。
ISCEVT_TrySendRequest (4602)	NULL	要求の送信を再試行します。
ISCEVT_TryRecvReply (4603)	NULL	応答の受信を再試行します。
ISCEVT_TryRecvTimeout (4604)	NULL	受信応答を待機する時間を延ばします。
SCEVT_TryRecvAck (4605)	NULL	確認通知の受信を再度試みます。

表 171. *iscEngineListenerPF* のイベント・コード： イベント・タイプ: ISCEVTTYPE_Query

イベント・コード	イベント情報 (argc)	説明
ISCEVT_QueCancel (5000)	NULL	ユーザーがキャンセルしたかどうかを照会して、次に示す値を戻します (括弧内は実際の値)。 • ISCRTNCB_ReplyYes (3): ユーザーがキャンセルした場合 • ISCRTNCB_ReplyNo (2): ユーザーが処理の継続を選択した場合 • ISCRTNCB_Default (0): デフォルト (つまり、ISCRTNCB_ReplyNo)
ISCEVT_QueCancelUponError (5001)	NULL	ユーザーがキャンセルしたかどうかを照会して、次に示す値を戻します (括弧内は実際の値)。 • ISCRTNCB_ReplyYes (3): ユーザーがキャンセルした場合 • ISCRTNCB_ReplyNo (2): ユーザーが処理の継続を選択した場合 • ISCRTNCB_Default (0): デフォルト (つまり、ISCRTNCB_ReplyNo)
ISCEVT_QueLogin (5002)	ISCLISTENARG(3) info->argv[0] info->argv[1] info->argv[2]	アダプターによって要求されたログイン情報です。リスナーはイベント情報で要求された情報を提供し、実際の値 (1) と共に *ISCRTNCB_Done* を戻さなければなりません。 データ・ソースのターゲット名 ユーザー名を保持するブランク・バッファー パスワードを保持するブランク・バッファー
ISCEVT_QueSubsTarget (5003)	ISCLISTENARG(1) info->argv[0]	アダプターによって要求されたデータベース情報。リスナーは、 イベント情報で要求された情報を提供し、デフォルトのターゲット・ディレクトリーを使用するためにISCRTNCB_Done または ISCRTNCB_Default を戻します。 サブスクリプション用ディレクトリー。

表 172. *iscEngineListenerPF* のイベント・コード： イベント・タイプ: ISCEVTTYPE_Error

イベント・コード	イベント情報 (argc)	説明
ISCEVT_ErrOpenAdapter (300)	NULL	アダプター *<adapter name>* のオープンに失敗しました。
ISCEVT_ErrLoadAdapter (301)	NULL	アダプター *<adapter name>* のロードに失敗しました。

表 172. *iscEngineListenerPF* のイベント・コード *(続き)*： イベント・タイプ: ISCEVTTYPE_Error

イベント・コード	イベント情報 (argc)	説明
ISCEVT_ErrCloseAdapter (302)	NULL	アダプター *<adapter name>* のクローズに失敗しました。
ISCEVT_ErrAuthenticateKey (306)	NULL	認証が失敗しました (無効な暗号鍵)。同期化は打ち切られました。
ISYNCEVT_ErrClientCryptoFailed (307)	NULL	クライアントの暗号化または暗号化解除が失敗しました。同期化は打ち切られました。
ISCEVT_ErrEncryptNotAvail (308)	NULL	暗号化が利用不可です。
ISCEVT_ErrEncryptLibOpen (309)	NULL	暗号化ライブラリーのオープンに失敗しました。
ISCEVT_ErrSubsNotFound (311)	NULL	サーバーがサブスクリプションを検出できません。
ISCRTN_ErrSubsNotAvail (312)	NULL	サブスクリプションがサーバーによってブロックされています。
ISCRTN_ErrSubsDefAltered (316)	NULL	同期エンジンが最後に構成を同期化した後に、サブスクリプション定義が変更されています。
ISCEVT_ErrAllocResource (400)	NULL	アダプター・リソースの割り振りに失敗しました。
ISCEVT_ErrConnectData (401)	NULL	ターゲット・データへの接続に失敗しました。
ISCEVT_ErrDisconnectData (402)	NULL	ターゲット・データからの切断に失敗しました。
ISCEVT_ErrNoData (403)	NULL	データが見つかりません。
ISCEVT_ErrMessageFormat (412)	NULL	予期しないメッセージ・フォーマットです。
ISCEVT_ErrNotFound (413)	ISCLISTENARG(2) info->argv[0] info->argv[1]	要求されたデータが見つかりません。 データ・ソースのターゲット名 データ名
ISCEVT_ErrEndOfData (414)	NULL	予期しないデータの終わりです。
ISCEVT_ErrDataTooLong (415)	ISCLISTENARG(3) info->argv[0] info->argv[1] info->argv[2]	データが長すぎるため、切り捨てられました。 データ・ソースのターゲット名 データ名 データ・エレメント名 (空ではない場合)
ISCEVT_ErrSyncDisabled (417)	NULL	サーバーは、ユーザーが使用可能ではないと報告しています。
ISCEVT_ErrServerException (418)	NULL	サーバーは、不明な例外を報告しています。
ISCEVT_ErrReadOnly (420)	ISCLISTENARG(2) info->argv[0] info->argv[1]	読み取り専用データを更新しようとしました。 データ・ソースのターゲット名 データ名
ISCEVT_ErrOperation (421)	NULL	データに対する正しくない操作です。
ISCEVT_ErrUnauthorized (423)	NULL	ターゲット・データへのアクセスが許可されていません。
ISCEVT_ErrNotAvailable (424)	ISCLISTENARG(2) info->argv[0] info->argv[1]	要求されたデータが利用不能です。 データ・ソースのターゲット名 データ名
ISCEVT_ErrNotSupported (425)	ISCLISTENARG(3) info->argv[0] info->argv[1] info->argv[2]	要求されたデータはサポートされていません。 データ・ソースのターゲット名 データ名 データ・エレメント名 (空ではない場合)
ISCEVT_ErrNetConn (601)	NULL	サーバーへの接続に失敗しました。
ISCEVT_ErrSendRequest (602)	NULL	要求の送信に失敗しました。
ISCEVT_ErrRecvReply (603)	NULL	応答の受信に失敗しました。
ISCEVT_ErrRecvTimeout (604)	NULL	応答の受信中にタイムアウトになりました。
ISCEVT_ErrRecvAck (605)	NULL	確認通知の受信が失敗しました。
ISCRTN_ErrCloseNetLib (608)	NULL	ネットワーク・ライブラリーのクローズに失敗しました。
ISCEVT_ErrOutOfMemory (610)	NULL	メモリー不足。

表 172. iscEngineListenerPF のイベント・コード (続き)： イベント・タイプ: ISCEVTTYPE_Error

イベント・コード	イベント情報 (argc)	説明
ISCEVT_ErrInternal (698)	ISCLISTENARG(1) info->argv[0]	その他の内部エラーが発生しました。 エラー状態 (ストリング)。

表 173. iscEngineListenerPF のイベント・コード： イベント・タイプ: ISCEVTTYPE_Fatal

イベント・コード	イベント情報 (argc)	説明
ISCEVT_FatSyncCfgAbort (303)	NULL	構成の同期化が失敗しました。同期化は打ち切られました。
ISCEVT_FatAuthenticateFailed (304)	NULL	認証が失敗しました。同期化は打ち切られました。
ISCEVT_FatIncompVersion (310)	NULL	非互換バージョンの同期クライアントです。
ISCEVT_FatInvalidSession (313)	NULL	無効なセッション ID です。
ISCEVT_FatSyncGroup (314)	NULL	ユーザーがどの同期グループにも属していません。
ISCEVT_FatRegisterDevice (315)	NULL	ユーザー用のデバイスの登録に失敗しました。
ISCEVT_FatNetOpenConn (600)	NULL	サーバーへの接続のオープンに失敗しました。
ISCEVT_FatOpenNetLib (606)	NULL	ネットワーク・ライブラリーのロードに失敗しました。
ISCEVT_FatResolveHost (609)	NULL	ホスト名の解決に失敗しました。
ISCEVT_FatServerForbidden (611)	NULL	サーバーへの同期化が禁止されています。
ISCEVT_FatServerNotFound (612)	NULL	サーバーが見つかりません。
ISCEVT_FatServer (613)	NULL	サーバーのエラー。
ISCEVT_FatServerNotAvail (614)	NULL	サーバーから応答がありません。
ISCEVT_FatNetUnknown (699)	NULL	不明なネットワーク・エラーです。

例:

```
isy_INT32 mySyncListener(
    isy_UINT32 listenerData,
    ISCEVT*  event,
    isy_VOID*  pExtraInfo)
{
    char *statusMsg = appEventCodeToMessage(event);
    int   timesRetried;

    switch (event->type) {
      case ISCEVTTYPE_Info:
        appStatusBar(statusMsg);
        // appStatusBar can be any routine which shows the statusMsg (e.g., in a
        // status bar)
        return ISCRTNCB_Done;

      case ISCEVTTYPE_Retry:
        timesRetried = event->retry;
        if (timesRetried >= 3) // Try no more than 3 times
            return ISCRTNCB_ReplyNo;
        else
            return appRetryCancelBox(statusMsg, 10); // 10 sec timeout
        // appRetryCancelBox can be any routine which shows a window with two
        // buttons: Cancel and Retry.  It returns
        //   ISCRTNCB_ReplyYes, if user clicks Retry
        //   ISCRTNCB_ReplyNo, if user clicks Cancel
        // If the user doesn't make choice, it returns ISCRTNCB_Default.
        break;

      // all other event types, don't care
      default:
        return ISCRTNCB_Default;
    } // switch (event->type)
} // mySyncListener
```

戻りコード:

- ISCRTNCB_ReplyYes: ユーザーは照会に `Yes` と応答する
- ISCRTNCB_ReplyNo*: ユーザーは照会に `No` と応答する

- ISCRTNCB_Default: 応答なし。デフォルトのアクションを実行する

イベント・タイプが ISCEVTTYPE_Retry である場合、リスナー関数は以下のいずれかのコードを戻します。

イベント・タイプが ISCEVTTYPE_Query である場合、戻りコードの意味はイベント・コードの値によって決まります。つまり、リスナーはイベント・コードをチェックし、該当する値を戻します。しかし、ユーザーが照会に応答しない場合、アプリケーションは以下のコードを戻します。

- ISCRTNCB_Default: 応答なし。デフォルトのアクションを実行する

ISCEVTTYPE_Retry および ISCEVTTYPE_Query 以外のイベント・タイプの場合、同期エンジンは戻りコードを無視します。リスナーは単に ISCRTNCB_Done を戻します。

注: 評価対象外のイベントの場合、リスナー関数は単に ISCRTNCB_Default を戻し、同期エンジンでデフォルトのアクションを実行します。

注: 上記の中で示されているアスタリスク (*) は、各イベント・タイプのデフォルトのアクションを示しています。

制約事項:

ユーザー定義のリスナー関数は、同期エンジンのプロトコルに従わなければなりません。従わない場合、同期エンジンが正しく動作しない可能性があります。

関連した概念:

- 129 ページの『同期クライアント C/C++ サンプル・アプリケーション』

関連したタスク:

- 18 ページの『C/C++ を使用した DB2 Everyplace 同期クライアント・アプリケーションの開発』

関連した解説:

- 325 ページの『IBM 同期クライアント C-API バージョン 8.1 と バージョン 7.2 の比較』
- 329 ページの『IBM 同期クライアント C-API のデータ・タイプ』
- 328 ページの『IBM 同期クライアント C-API の関数の要約』
- 363 ページの『iscEngineSync()』
- 352 ページの『iscEngineSetListener()』

iscEngineSetPref()

目的: `iscEngineSetPref()` は、同期エンジンのプリファレンスを設定します。

構文:

```
isy_INT32 iscEngineSetPref(
    HISCENG              hEngine,
    isy_CONST isy_INT32  prefID,
    isy_CONST isy_TCHAR *prefVal);
```

関数の引き数:

表 174 は、iscEngineSetPref() 関数で使用される有効な引き数をリストしたものです。

表 174. iscEngineSetPref() の引き数

データ・タイプ	引き数	用途	説明
HISCENG	*hEngine*	入力	同期エンジンへのハンドル
isy_CONST isy_INT32	*prefID*	入力	プリファレンス ID。以下のいずれかの値です。 • ISCPREF_Timeout: メッセージ受信時のタイムアウトの長さ • ISCPREF_Trace: 詳細トレース
isy_CONST isy_TCHAR*	*prefVal*	入力	設定する新しいプリファレンス値。事前定義されたプリファレンス定数もあります。 ISCPREF_Trace プリファレンスの場合は以下のとおりです。 • ISCCONST_TraceON: 詳細デバッグ・トレースをオンにします。 • ISCCONST_TraceOFF: 詳細デバッグ・トレースをオフにします。 ISCPREF_Timeout プリファレンスの場合は以下のとおりです。 • ISCCONST_TimeoutNever: サーバー応答の待機中にタイムアウトになりません。 • ISCCONST_TimeoutMinimum: タイムアウトの最短時間。

使用法:

iscEngineSetPref() は、同期エンジンのプリファレンスを設定するために使用します。これらのプリファレンスは永続的なものではなく、同期エンジンへの新規ハンドルをオープンするたびにリセットしなければなりません。

戻りコード:

- ISCRTN_Succeeded: 正常
- ISCRTN_UnknownID: 不明
- ISCRTN_ValTooLong: 指定された prefVal の長さが長すぎる
- ISCRTN_Failed: その他のエラー

制約事項:

指定されたプリファレンス値は、所定のプリファレンス制限内に収まらなければなりません。

- ISCPREF_Trace: 1
- ISCPREF_Timeout: 11

関連した概念:

- 129 ページの『同期クライアント C/C++ サンプル・アプリケーション』

関連したタスク:

- 18 ページの『C/C++ を使用した DB2 Everyplace 同期クライアント・アプリケーションの開発』

関連した解説:

- 325 ページの『IBM 同期クライアント C-API バージョン 8.1 と バージョン 7.2 の比較』
- 329 ページの『IBM 同期クライアント C-API のデータ・タイプ』
- 328 ページの『IBM 同期クライアント C-API の関数の要約』
- 『iscEngineGetPref()』

iscEngineGetPref()

目的: `iscEngineGetPref()` は、現在のプリファレンス設定を取り出します。

構文:

```
isy_INT32 iscEngineGetPref(
    HISCENG              hEngine,
    isy_CONST isy_INT32  prefID,
    isy_TCHAR           *prefVal,
    isy_CONST isy_INT32  prefLen);
```

関数の引き数:

表 175 は、`iscEngineGetPref()` 関数で使用される有効な引き数をリストしたものです。

表 175. iscEngineGetPref() の引き数

データ・タイプ	引き数	用途	説明
HISCENG	*hEngine*	入力	同期エンジンへのハンドル
isy_CONST isy_INT32	*prefID*	入力	プリファレンス ID。以下のいずれかの値です。 • ISCPREF_Timeout: メッセージ受信時のタイムアウトの長さ • ISCPREF_Trace: 詳細トレース

表 175. iscEngineGetPref() の引き数 (続き)

データ・タイプ	引き数	用途	説明
isy_TCHAR*	*prefVal*	出力	戻されたプリファレンス値を保管するためのバッファーへのポインター。事前定義されたプリファレンス定数もあります。 ISCPREF_Trace プリファレンスの場合は以下のとおりです。 • ISCCONST_TraceON: 詳細デバッグ・トレースをオンにします。 • ISCCONST_TraceOFF: 詳細デバッグ・トレースをオフにします。 ISCPREF_Timeout プリファレンスの場合は以下のとおりです。 • ISCCONST_TimeoutNever: サーバー応答の待機中にタイムアウトになりません。 • ISCCONST_TimeoutMinimum: タイムアウトの最短時間。
isy_CONST isy_INT32	*prefLen*	入力	提供されたバッファー (prefVal) のサイズ。

使用法:

`iscEngineGetPref()` は、同期エンジンのプリファレンス設定 (デフォルト値または `iscEngineSetPref()` によって設定された値のいずれか) を取得するために使用します。

戻りコード:

- ISCRTN_Succeeded: 正常
- ISCRTN_UnknownID: 不明な prefID が指定された
- ISCRTN_ValTruncated: プリファレンス値の実際の長さが prefLen より長い
- SCRTN_Failed: その他のエラー

制約事項:

提供されるバッファーは、各プリファレンス値を保管することのできる十分な大きさがなければなりません。

- ISCPREF_Trace: 1
- ISCPREF_Timeout: 11
- ISCPREF_CodePage: 15

関連した概念:

- 129 ページの『同期クライアント C/C++ サンプル・アプリケーション』

関連したタスク:

- 18 ページの『C/C++ を使用した DB2 Everyplace 同期クライアント・アプリケーションの開発』

関連した解説:

- 325 ページの『IBM 同期クライアント C-API バージョン 8.1 と バージョン 7.2 の比較』
- 329 ページの『IBM 同期クライアント C-API のデータ・タイプ』
- 328 ページの『IBM 同期クライアント C-API の関数の要約』
- 359 ページの『iscEngineSetPref()』

iscEngineSync()

目的: `iscEngineSync()` は、同期セッションを立ち上げます。

構文:

```
isy_INT32 iscEngineSync(
    HISCENG       hEngine);
```

関数の引き数:

表 176 は、`iscEngineSync()` 関数で使用される有効な引き数をリストしたものです。

表 176. iscEngineSync() の引き数

データ・タイプ	引き数	用途	説明
HISCENG	*hEngine*	入力	同期エンジンへのハンドル

使用法:

`iscEngineSync()` は、`iscEngineOpen()` で指定された構成を同期化する同期セッションを立ち上げるために使用します。サブスクリプション・セットが同期化されていない場合、そのサブスクリプション・セットはリセット・モードにあります。同期エンジンがそのサブスクリプション・セットに同期化を実行すると、同期クライアントが同期サーバーからデータを取り出します。このプロセスはリフレッシュと呼ばれます。リフレッシュの完了後、サブスクリプション・セットが再び同期化されると、同期エンジンは変更されたデータを同期化します。このプロセスは同期化と呼ばれます。同期エンジンは常に構成を最初に同期化します。構成の同期化が失敗すると、同期エンジンは後続のサブスクリプション・セットの処理を継続せず、同期セッションは停止します。同期エンジンが、あるサブスクリプション・セットで失敗した (ただし、構成では失敗していない) 場合、同期エンジンは残りのサブスクリプション・セットがある場合は、その処理を継続します。

戻りコード:

- ISCRTN_Succeeded: 同期化が正常に終了した
- ISCRTN_Failed: 同期化が失敗した
- ISCRTN_Canceled: 同期化がユーザーによってキャンセルされた

`iscEngineSync()` の戻りコードは、同期化したすべてのサブスクリプション・セットの同期状況の集合 (以下に示す優先順位に従う) です。

ISCRTN_Canceled > ISCRTN_Failed > ISCRTN_Succeeded

制約事項:

なし。

関連した概念:

- 129 ページの『同期クライアント C/C++ サンプル・アプリケーション』

関連したタスク:

- 18 ページの『C/C++ を使用した DB2 Everyplace 同期クライアント・アプリケーションの開発』

関連した解説:

- 325 ページの『IBM 同期クライアント C-API バージョン 8.1 と バージョン 7.2 の比較』
- 329 ページの『IBM 同期クライアント C-API のデータ・タイプ』
- 328 ページの『IBM 同期クライアント C-API の関数の要約』
- 339 ページの『iscConfigPurge()』
- 『iscEngineSyncConfig()』

iscEngineSyncConfig()

目的: `iscEngineSyncConfig()` は、構成のみを同期化する同期セッションを立ち上げます。

構文:

```
isy_INT32 iscEngineSyncConfig(
    HISCENG        hEngine);
```

関数の引き数:

表 177 は、`iscEngineSyncConfig()` 関数で使用される有効な引き数をリストしたものです。

表 177. iscEngineSyncConfig() の引き数

データ・タイプ	引き数	用途	説明
HISCENG	*hEngine*	入力	同期エンジンへのハンドル

使用法:

サーバー上の構成が変更されると、`iscEngineSyncConfig()` はすべてのサブスクリプション・セットを再同期化することなく、構成を更新します。

戻りコード:

- ISCRTN_Succeeded: 同期化が正常に終了した
- ISCRTN_Failed: 同期化が失敗した
- ISCRTN_Canceled: 同期化がユーザーによってキャンセルされた

制約事項:

なし。

関連した概念:

- 129 ページの『同期クライアント C/C++ サンプル・アプリケーション』

関連したタスク:

- 18 ページの『C/C++ を使用した DB2 Everyplace 同期クライアント・アプリケーションの開発』

関連した解説:

- 325 ページの『IBM 同期クライアント C-API バージョン 8.1 と バージョン 7.2 の比較』
- 329 ページの『IBM 同期クライアント C-API のデータ・タイプ』
- 328 ページの『IBM 同期クライアント C-API の関数の要約』
- 363 ページの『iscEngineSync()』
- 339 ページの『iscConfigPurge()』

第 19 章 DB2 Everyplace システム・カタログ基本表

データベース・マネージャーは、一連のシステム・カタログ基本表を作成し、保守します。この付録には、各システム・カタログ基本表の説明が列名およびデータ・タイプとともに記載されています。すべてのシステム・カタログ基本表は、データベース・マネージャーが作成します。システム・カタログ基本表を、明示的に作成したりドロップしたりすることはできません。システム・カタログ基本表は、通常のオペレーションで SQL データ定義ステートメント、環境ルーチン、および特定のユーティリティーの結果、更新されます。システム・カタログ基本表に入っているデータは、通常の SQL 照会機能で入手できます。システム・カタログ基本表は、通常の SQL データ操作コマンドでは変更できません。システム・カタログ表にアクセスするには、区切り ID を使用する必要があります。

表 178. システム・カタログ基本表

説明	カタログ基本表
表	367 ページ
列	367 ページ
参照制約	368 ページ
ユーザー	368 ページ

DB2eSYSTABLES:

このシステム・カタログ基本表には、作成されたそれぞれの表ごとに 1 行ずつ記録があります。すべてのカタログ表の項目が、DB2eSYSTABLES カタログにあります。

表 179. DB2eSYSTABLES システム・カタログ基本表

列名	データ・タイプ	ヌル可能	説明
TNAME	VARCHAR (19)		表名
NUMCOLS	INTEGER (4)		列数
FLAGS	INTEGER (4)		(内部使用専用)
NUMKEY	INTEGER (4)		主キー内の列数
CHK	BLOB (512)	はい	チェック制約 (内部使用専用)
IDXINFO	BLOB (700)	はい	索引 (内部使用専用)
NUMREFS	INTEGER (4)	はい	主キーおよび外部キー (内部使用専用)
F_ID	INTEGER (4)	はい	(内部使用専用)
PD	BLOB (4096)	はい	(内部使用専用)

DB2eSYSCOLUMNS:

このシステム・カタログ基本表には、表に定義されたそれぞれの列ごとに 1 行ずつ記録があります。

表 180. DB2eSYSCOLUMNS システム・カタログ基本表

列名	データ・タイプ	ヌル可能	説明
CNAME	VARCHAR (19)		列名
TNAME	VARCHAR (19)		表名
TYPE	INTEGER (4)		データ・タイプ

表 180. DB2eSYSCOLUMNS システム・カタログ基本表 (続き)

列名	データ・タイプ	ヌル可能	説明
ATTR	INTEGER (4)		(内部使用専用)
LENGTH	INTEGER (4)		列の長さ
POS	INTEGER (4)		列番号
FLAGS	INTEGER (4)		(内部使用専用)
KEYSEQ	INTEGER (4)		主キー内の列の順序を示す位置
SCALE	INTEGER (4)		10 進数列の位取り
DEF	VARCHAR (128)	はい	デフォルト値 (内部使用)

DB2eSYSRELS:

このシステム・カタログ基本表には、それぞれの参照制約ごとに 1 行ずつ記録があります。

表 181. DB2eSYSRELS システム・カタログ基本表

列名	データ・タイプ	ヌル可能	説明
RMD_ID	INTEGER (4)		主キーおよび外部キー (内部使用専用)
PKTABLE_NAME	VARCHAR (19)		親表名
PKCOLUMN_NAME	VARCHAR (19)		親表の主キー列
FKTABLE_NAME	VARCHAR (19)		子表名
FKCOLUMN_NAME	VARCHAR (19)		子表の外部キー列名
ORDINAL_POSITION	INTEGER (4)		外部キー参照内の列の位置

DB2eSYSUSERS:

DB2eSYSUSERS 表は、暗号化された表が初めて作成される際、または GRANT ステートメントが初めて実行される際に、自動的に作成されます。この表はデータベースおよび暗号化されたデータに緊密にバインドされており、暗号化された異なるデータを含む別の DB2 Everyplace データベースへ移動することはできません。

このシステム・カタログ基本表には、データベースに定義されたそれぞれの登録済みユーザー名ごとに 1 行ずつ記録があります。

表 182. DB2eSYSCOLUMNS システム・カタログ基本表

列名	データ・タイプ	ヌル可能	説明
USERNAME	VARCHAR (19)		主キーの一部 (大文字小文字は区別されます)。この行に関連付けられたユーザーの名前。
DATABASENAME	VARCHAR (19)		将来的に使用される予定です。空ストリングが保管されます。主キーの一部。
TABLENAME	VARCHAR (19)		将来的に使用される予定です。空ストリングが保管されます。主キーの一部。
ENCMETHOD	VARCHAR (198)		将来的に使用される予定です。空ストリングが保管されます。主キーの一部。
PRIVILEGES	CHAR (19)	はい	ユーザー特権を定義します。現在は、値「E」(暗号化 (encryption) を示す) のみが使用可能です。
ENCKEYDATA	BLOB (64)	はい	暗号鍵を再生成するために使用します。
ATTIME	TIMESTAMP (26)	はい	ユーザーの追加またはレコードの変更のいずれかが行われた最新の時刻。
VALIDATE	BLOB (64)	はい	レコードが信頼できるかどうか、およびユーザーが認証済みユーザーによって追加されたかどうかを検証します。
GRANTOR	VARCHAR (19)	はい	列 1 にユーザー名を登録したユーザーの名前。
INTERNALDATA	BLOB (255)	はい	(将来的に内部使用)

第 20 章 DB2 Everyplace の制限

次の表は、DB2 Everyplace および SQL の制限をいくつか説明しています。最もきびしい制限を守った場合は、プログラマーが移植が容易にできるアプリケーション・プログラムを設計するのに役立ちます。これらの制限の多くは、デバイスでの物理的なメモリーまたはシステムの限界により、さらに大きく制限されている場合があります。

表 183. DB2 Everyplace データベースと SQL の制限

説明	制限
最大表サイズ (32 ビット・システム)	2 ギガバイト
データベースの最大長	75 バイト
データ保管での表の最大数	65535
表の索引の最大数	15
表内の外部キーの最大数	8
索引内の列の最大数	8
主キー内の列の最大数	8
SQL ステートメントの最大長	64 K バイト
データ保管パスに対する接続の最大数	1
表内の行の最大数	表のサイズによって制限される
表内の列の最大数	256
CHAR 列の最大長	32767 バイト
VARCHAR または BLOB 列の最大長	32767 バイト
行の 32767 固定長列の最大累積長	32767 バイト
接続あたりのステートメント・ハンドルの最大数	20
チェック制約の最大長	512 バイト
10 進数の最大サイズ	31 桁
単一索引における各列の最大長	1024 バイト

関連した解説:

- 147 ページの『DB2 Everyplace SQL ステートメント・サポートの概要』

第 21 章 DB2 Everyplace 予約語

以下の DB2 Everyplace 予約語は、区切り ID として指定されない限り、ID として使用することはできません。例えば、次のようになります。

以下のステートメントは SQL エラーの原因となります。

```
CREATE TABLE tab1 (select int)
```

二重引用符を使用すれば、SQL エラーは発生しません。

```
CREATE TABLE tab1 ("select" int)
```

DB2 Everyplace 予約語:

```
ALL、AND、AS、ASC、
BEGIN、BLOB、BY、
DATABASE
CALL、CHAR、CHAR、CHECK、COMMIT、CONCAT、CREATE、CURRENT、
DATE、DECIMAL、DEFAULT、DELETE、DESC、DISTINCT、DROP、
ENCRYPT
FETCH、FOR、FOREIGN、FROM、
GRANT、GROUP、
IN、INDEX、INSERT、INT、INTEGER、INTO、IS、
KEY、
LIKE、LIMIT、
NEW、NOT、NULL、
OF、ON、ONLY、OR、ORDER、
PRIMARY、
READ、REFERENCES、REORG、REVOKE、ROLLBACK、
SELECT、SET、SMALLINT、
TABLE、TIME、TIMESTAMP、TO、TRANSACTION
UPDATE、UPSERT、USING
VALUES、VARCHAR、
WHERE、WITH
```

将来的な互換性を保つため、以下の IBM SQL および ISO/ANSI SQL92 の予約語を ID として使用しないでください。 IBM SQL の予約語で、現在 DB2 Everyplace では使用されていないものを、以下に示します。

```
ACQUIRE ADD AFTER ALIAS ALLOCATE
ALLOW ALTER ANY ASUTIME AUDIT AUTHORIZATION
AUX AUXILIARY AVG BEFORE BETWEEN BINARY
BUFFERPOOL CALLED CAPTURE CASCADED CASE
CAST CCSID CHARACTER CLOSE CLUSTER COLLECTION
COLLID COLUMN COMMENT CONDITION CONNECT
CONNECTION CONSTRAINT CONTAINS CONTINUE
COUNT COUNT_BIG CROSS CURRENT_DATE CURRENT_LC_PATH
CURRENT_PATH CURRENT_SERVER CURRENT_TIME
CURRENT_TIMESTAMP CURRENT_TIMEZONE CURRENT_USER
CURSOR DATA DATABASE DAY DAYS DBA DBINFO
DBSPACE DB2GENERAL DB2SQL DECLARE DESCRIPTOR
DETERMINISTIC DISALLOW DISCONNECT DO DOUBLE
DSSIZE DYNAMIC EDITPROC ELSE ELSEIF END
END-EXEC ERASE ESCAPE EXCEPT EXCEPTION
EXCLUSIVE EXECUTE EXISTS EXIT EXPLAIN
EXTERNAL FENCED  FIELDPROC FILE FINAL FREE
FULL FUNCTION GENERAL GENERATED GO GOTO
GRANT GRAPHIC HANDLER HAVING HOUR HOURS
IDENTIFIED IF IMMEDIATE INDICATOR INNER
INOUT INSENSITIVE INTEGRITY INTERSECT
```

```
ISOBID ISOLATION JAVA JOIN LABEL LANGUAGE
LC_CTYPE LEAVE LEFT LINKTYPE LOCAL LOCALE
LOCATOR LOCATORS LOCK LOCKSIZE LONG LOOP
MAX MICROSECOND MICROSECONDS MIN MINUTE
MINUTES MODE MODIFIES MONTH MONTHS NAME
NAMED NHEADER NO NODENAME NODENUMBER NULLS
NUMPARTS OBID OPEN OPTIMIZATION OPTIMIZE
OPTION OUT OUTER PACKAGE PAGE PAGES
PARAMETER PART PARTITION PATH PCTFREE
PCTINDEX PIECESIZE PLAN POSITION PRECISION
PREPARE PRIQTY PRIVATE PRIVILEGES PROCEDURE
PROGRAM PSID PUBLIC QUERYNO READS RECOVERY
RELEASE RENAME REPEAT RESET RESOURCE RESTRICT
RESULT RETURN RETURNS REVOKE RIGHT ROW ROWS
RRN RUN SCHEDULE SCHEMA SCRATCHPAD SECOND
SECONDS SECQTY SECURITY SHARE SIMPLE SOME
SOURCE SPECIFIC SQL STANDARD STATIC STATISTICS
STAY STOGROUP STORES STORPOOL STYLE SUBPAGES
SUBSTRING SUM SYNONYM TABLESPACE THEN TO
TRANSACTION TRIGGER TRIM TYPE UNDO UNION
UNIQUE UNTIL USAGE USER USING VALIDPROC
VARIABLE VARIANT VCAT VIEW VOLUMES WHEN
WHILE WLM WORK WRITE YEAR YEARS
```

ISO/ANS SQL92 の予約語で、IBM SQL では使用されていないものを、以下に示します。

```
ABSOLUTE ACTION ARE ASSERTION AT BIT_LENGTH
BOTH CATALOG CHAR_LENGTH CHARACTER_LENGTH
COALESCE COLLATE COLLATION CONSTRAINTS CONVERT
CORRESPONDING DEALLOCATE DEC DEFERRABLE DEFERRED
DESCRIBE DIAGNOSTICS DOMAIN EXEC EXTRACT FALSE
FIRST FLOAT FOUND FULL GET GLOBAL IDENTITY
INITIALLY INPUT INTERVAL LAST LEADING LEVEL
LOWER MATCH MODULE NAMES NATIONAL NATURAL
NCHAR NEXT NULLIF NUMERIC OCTET_LENGTH OUTPUT
OVERLAPS PAD PARTIAL PRESERVE PRIOR REAL
RELATIVE SCROLL SECTION SESSION SESSION_USER
SIZE SPACE SQLCODE SQLERROR SQLSTATE SYSTEM_USER
TEMPORARY TIMEZONE_HOUR TIMEZONE_MINUTE
TRAILING TRANSLATION TRUE UNKNOWN UPPER
VALUE VARYING WHENEVER ZONE
```

関連した解説:

- 147 ページの『DB2 Everyplace SQL ステートメント・サポートの概要』

第 22 章 各国語サポート (NLS)

この章では、DB2 Everyplace で提供されている各国語サポート (NLS) について説明します。この章には、国、言語、およびサポートされているコード・ページ (コード・セット) に関する情報や、ご使用のデバイスおよびアプリケーションで DB2 Everyplace NLS フィーチャーを構成し使用する方法に関する情報が含まれています。DB2 Everyplace は、1 バイト、2 バイト、およびマルチバイトの文字セットをサポートします。 Win32 オペレーティング・システムはすべて、ユニコードと非ユニコード (ANSI) の両方をサポートしています。コード・ページとユニコードがどのようにサポートされているかについては、以下のトピックを参照してください。

- 『オペレーティング・システムによる DB2 Everyplace NLS サポート』
- 375 ページの『Java アプリケーションでの文字エンコード』
- 376 ページの『DB2 Everyplace 言語イネーブラー』
- 377 ページの『DB2 Everyplace ユニコード・サポート』

オペレーティング・システムによる DB2 Everyplace NLS サポート

表 184 は、NLS サポートが提供されるオペレーティング・システムおよび対応する言語をリストしたものです。

表 184. NLS サポート

言語	Win32	WinCE	Linux	Palm OS	Symbian OS	Neutrino
英語	コード・ページ / ユニコード	ユニコード	コード・ページ / UTF-8	コード・ページ	ユニコード	UTF-8
フランス語	コード・ページ / ユニコード	ユニコード	コード・ページ / UTF-8	コード・ページ	ユニコード	UTF-8
ドイツ語	コード・ページ / ユニコード	ユニコード	コード・ページ / UTF-8	コード・ページ	ユニコード	UTF-8
イタリア語	コード・ページ / ユニコード	ユニコード	コード・ページ / UTF-8	コード・ページ	ユニコード	UTF-8
スペイン語	コード・ページ / ユニコード	ユニコード	コード・ページ / UTF-8	コード・ページ	ユニコード	UTF-8
中国語 (簡体字)	コード・ページ / ユニコード	ユニコード	コード・ページ / UTF-8	コード・ページ • イネーブラーのインストール	適用外	UTF-8

表 184. NLS サポート (続き)

言語	Win32	WinCE	Linux	Palm OS	Symbian OS	Neutrino
中国語 (繁体字)	コード・ページ / ユニコード	ユニコード • Pocket PC 用のイネーブラーのインストール	コード・ページ / UTF-8	コード・ページ • イネーブラーのインストール Acer S60 には、中国語 (繁体字) Palm OS が組み込まれています。	適用外	UTF-8
韓国語	コード・ページ / ユニコード	ユニコード • イネーブラーのインストール	コード・ページ / UTF-8	コード・ページ • イネーブラーのインストール	適用外	UTF-8
日本語	コード・ページ / ユニコード	ユニコード	コード・ページ / UTF-8	コード・ページ	適用外	UTF-8
ヘブライ語	適用外	適用外	適用外	コード・ページ • イネーブラーのインストール	適用外	適用外
チェコ語	コード・ページ / ユニコード	ユニコード • イネーブラーのインストール	適用外	コード・ページ • イネーブラーのインストール	ユニコード	適用外
アラビア語	適用外	適用外	適用外	コード・ページ • イネーブラーのインストール	適用外	適用外

ユニコード・サポートのない、Palm OS、QNX Neutrino、Linux、Windows NT および Windows 2000 の各オペレーティング・システムでは、ロケール情報を使用して正しいコード・ページを判別します。 DB2 Everyplace の中では内部ストリング変換を行いません。渡された各ストリングは、そのままで保管されます。照会アプリケーションは、保管時に使用されたのと同じコード・ページ設定を使用しなければなりません。これは、DB2 Universal Database が NLS を提供する方法と似ています。DB2 Everyplace は、コード・ページ変換機能を提供していません。あるシステム上に特定のコード・ページを用いて作成された DB2 Everyplace データベースは、同じコード・ページを使用するシステムにのみ配置できます。特定のコード・ページを用いて作成された表は、そのコード・ページをサポートするすべてのデバイスで使用可能です。ただし、特定のランゲージ・イネーブラーが必要な場合を除きます。データベースにアクセスするアプリケーションが、文字データを正しく解釈する責任を負います。

DB2 Everyplace は、現在設定されているロケール、または現在使用可能なロケールを調べることによって、現在使用されているエンコード形式を検出します。

Palm OS では、コード・ページを判別するために、ランゲージ・イネーブラーも必要です。

関連した解説:

- 『Java アプリケーションでの文字エンコード』
- 376 ページの『DB2 Everyplace 言語イネーブラー』
- 377 ページの『DB2 Everyplace ユニコード・サポート』

Java アプリケーションでの文字エンコード

Java プログラムでは、内部的にはユニコード・テキストを使用しますが、 DB2 Everyplace の表内の文字データは、表を作成したときのオペレーティング・システムや言語によっては、ユニコード以外の形式である可能性があります。 Windows CE および Symbian OS オペレーティング・システムの場合、 DB2 Everyplace JDBC ドライバーはテキスト取り出しと挿入を UTF-8 形式で行います。サポートされるその他のオペレーティング・システムの場合、システムのデフォルトの文字エンコード方式が使用されます。デフォルト値は、通常、Java システム・プロパティーの「file.encoding」属性によって決定されます。

例えば、Win32 オペレーティング・システムの場合、同一のマシン上で、ユニコード・バージョンの CLI インターフェースを使用するか、非ユニコード・バージョンの CLI インターフェースを使用するかを選択することができるため、あるデータベースでは UTF-8 形式のエンコードを使用し、別のデータベースではローカルのコード・ページのエンコード方式を使用するといったことが可能です。 JDBC アプリケーションが両方のデータベースのデータにアクセスできるようにするために、 DB2 Everyplace には、アプリケーションが使用するデータ・エンコード形式を動的に指示できるようにする機能が用意されています。

DB2 Everyplace JDBC ドライバーは、アプリケーションによって指定された形式に従って、 Java ストリングをバイトに変換します。アプリケーションで指定された形式は、オペレーティング・システムのデフォルトの文字エンコード方式をオーバーライドします。

ユーザーは JDBC インターフェースを介して、アプリケーションのデータ・エンコード形式を動的に指定することができます。これを行う方法は以下のとおりです。

1. java.util.Properties オブジェクトを作成する。
   - キー: DB2e_ENCODING
   - 値: 文字エンコード方式

   値 UTF-8 を使用して、UTF-8 コーディングを使用する DB2 Everyplace を指定するか、あるいは、 JVM がサポートする任意の文字エンコード方式を使用してください。
2. 以下のいずれかの方法によって、java.util.properties オブジェクトを渡す。
   - 指定のデータベース URL への接続を確立する。

java.sql パッケージの DriverManager クラスにある静的メソッド Connection getConnection(String url, Properties info) を使用します。

- 指定の URL へのデータベース接続を作成する。

  java.sql パッケージの Interface Driver クラスにあるメソッド Connection connect(String url, Properties info) を使用します。

関連した解説:

- 373 ページの『オペレーティング・システムによる DB2 Everyplace NLS サポート』
- 『DB2 Everyplace 言語イネーブラー』
- 377 ページの『DB2 Everyplace ユニコード・サポート』

DB2 Everyplace 言語イネーブラー

使用している言語のすべての文字をモバイル・デバイスで表示できるようにするには、ランゲージ・イネーブラーをモバイル・デバイスにインストールします。以下の表は、DB2 Everyplace で使用できるイネーブラーをリストしたものです。

表 185. モバイル・デバイス用の言語イネーブラー

言語	イネーブラーおよびオペレーティング・システム
アラビア語	Sakhr Arabic Palm 2.0
中国語 (簡体字)	CWP v1.0 for Palm
中国語 (繁体字)	• CJKOS 3.21 for Palm OS カラー・デバイス (CJK オプションのソート・レコードは、予期しない結果を生じさせることがあります。) • Gismosoft Chinese Small_Knife 2.0 for Pocket PC のみ • Acer S60 には、中国語 (繁体字) Palm OS が組み込まれています。
チェコ語	• RedGrep GNU-czech0.71 for Palm OS • Sunnysoft InterWrite5.5P Pro for Windows CE
ヘブライ語	Penticon Technologies Ltd. Hebrew Support+3.20c for Palm OS
韓国語	• HANME 2.0 for Palm OS • HANTIP 2.01for Palm OS CessHan for Casio E-115 1.0 on Windows CE

関連した解説:

- 375 ページの『Java アプリケーションでの文字エンコード』
- 373 ページの『オペレーティング・システムによる DB2 Everyplace NLS サポート』
- 377 ページの『DB2 Everyplace ユニコード・サポート』

DB2 Everyplace ユニコード・サポート

ユニコードをサポートするオペレーティング・システム (Windows CE、Symbian OS、Windows NT、および Windows 2000) の場合、 DB2 Everyplace は入出力ストリングとしてのみユニコード・ストリングを使用します。これらのユニコード・ストリングは、DB2 Everyplace エンジンの内部では UTF-8 形式で保管されます。ユニコード文字は、UTF-8 変換を行った後は、1 文字につき 1 から 3 バイトのストレージ・スペースが必要になります。IBM DB2 Universal Database などのデータベース・サーバーに保管された文字ストリングは、ダウンロードして Windows CE デバイス上の DB2 Everyplace に保管すると、元より大きなスペースが必要になる場合があります。

CLI ユニコード・インターフェースの注:

- DB2 Everyplace CLI UNICODE 関数は、終わりに「W」の文字が付加されています。マクロ UNICODE (これは Windows CE でのシステム・デフォルトです) を定義することによって、通常の CLI 関数が、対応する UNICODE 関数に自動的にマップされます。移植可能なコードを作成するためには、マクロ「UNICODE」を定義して、システムによる変換を実行してください。
- ユニコード・サポートが使用可能になると、データ・タイプ SQL_C_CHAR、SQL_C_TCHAR、および SQL_C_WCHAR は同じ意味になります。
- 多くの CLI 関数は、ストリング (または、バッファー) の長さを入出力パラメーターとして持っています。
  - *引き数タイプ* が SQLCHAR* (または、W 関数では SQLWCHAR*) である関数の場合、長さは文字数です。例えば次の場合、

    ```
    SQLRETURN   SQLExecDirect    (SQLHSTMT          hstmt,
                                  SQLCHAR     FAR   *szSqlStr,
                                  SQLINTEGER        cbSqlStr);
    ```

    ユニコード・ストリング `L"ABCD"` は 4 文字です。
  - *引き数タイプ* が SQLPOINTER である関数の場合、長さはバイト数です。例えば次の場合、

    ```
    SQLRETURN  SQLGetData       (SQLHSTMT          hstmt,
                                 SQLUSMALLINT      icol,
                                 SQLSMALLINT       fCType,
                                 SQLPOINTER        rgbValue,
                                 SQLINTEGER        cbValueMax,
                                 SQLINTEGER  FAR   *pcbValue);
    ```

    入力パラメーター `cbValueMax` および出力パラメーター `*pcbValue` の長さは、バイトで示します。ユニコード・ストリング `L"ABCD"` は 8 バイトです。
- UNICODE 関数では、ヌル終了ストリングを示すために SQL_NTS も使用できます。

移植可能なコードを作成するためのヒントを、以下に挙げます。

- SQLCHAR または SQLWCHAR の代わりに、SQLTCHAR を使用します。
- `str`*XXXX* (ANSI) または `wcs`*XXXX* (UNICODE) の代わりに、`_tcs`*XXXX* 関数を使用します。例えば、`wcslen()` または `strlen()` の代わりに、`_tcslen()` を使用します。

- リテラル・ストリングの折り返しには、_TEXT() (または、TEXT()) を使用します。例えば、_TEXT("ABCD") は、マクロ定義によって ANSI またはユニコードのいずれかのストリングとして解釈できます。
- 文字配列のサイズを検出するには、sizeof(ArrayName)/sizeof(TCHAR) を使用します。

例については、DB2 Everyplace に組み込まれている「Windows CE SampleCLP のサンプル・コード」を参照してください。

関連した解説:

- 375 ページの『Java アプリケーションでの文字エンコード』
- 376 ページの『DB2 Everyplace 言語イネーブラー』
- 373 ページの『オペレーティング・システムによる DB2 Everyplace NLS サポート』

第 23 章 DB2 Everyplace 情報集

DB2 Everyplace ライブラリーは、HTML 形式のオンライン・ヘルプと、PDF および HTML 形式の資料から構成されています。このセクションでは、提供されている情報とそれらにアクセスする方法について説明します。

また、すべての製品情報が www.ibm.com/software/data/db2/everyplace/library.html でオンラインで使用可能でます。

DB2 Everyplace PDF および HTML ファイル

DB2 Everyplace 資料およびリリース情報は、CD-ROM から直接、HTML または PDF 形式で参照できます。DB2 Everyplace 情報はさまざまな言語に翻訳されています。ただし、すべての情報がすべての言語に翻訳されているわけではありません。特定の言語で情報が利用できない場合は、英語による情報が提供されます。

DB2 Everyplace をご使用のワークステーションにインストールすると、`¥DB2everyplace¥docs` に資料が保管されます。次の表に、**docs** ディレクトリーに保管されている資料をリストします。

表 186. DB2 Everyplace の資料

資料の名称	説明	PDF ファイル名	HTML ディレクトリー
DB2 Everyplace インストールおよびユーザーズ・ガイド (SC88-9478)	• DB2 Everyplace コンポーネントのワークステーションへのインストール。 • DB2 Everyplace データベースとサンプル・アプリケーションのモバイルまたは組み込みデバイスへのインストール。 • モバイルまたは組み込みデバイスの構成および保守。 • DB2 Everyplace サンプル・アプリケーションの使用。	dsyiug.pdf	dsyiug

表 186. *DB2 Everyplace* の資料 (続き)

資料の名称	説明	PDF ファイル名	HTML ディレクトリー
DB2 Everyplace アプリケーション 開発ガイド (SC88-9479)	• 使用可能なプラットフォームでの DB2 Everyplace アプリケーションの作成。 • DB2 Everyplace サンプル・アプリケーションおよびソース・コード。 • サポートされている SQL ステートメント、SQLState、DB2 CLI/ODBC、JDBC メソッド、IBM 同期クライアント C-API、IBM Java 同期 API、および各国語サポート。 • DB2 Everyplace データベースへのアクセス。 • ローカル・データ暗号化の使用。	dsyadg.pdf	dsyadg
DB2 Everyplace 同期サーバー 管理ガイド (SC88-9480)	• 同期サーバーの構成および保守。 • 同期サーバーのデータ・ソースへの接続。 • 同期サーバー、モバイル・デバイス、および組み込みデバイス間での通信の構成。 • ローカル・データベースおよびリモート・データベースの構成および保守。 • ユーザーおよびデータの管理。	dsysag.pdf	dsysag

DB2 Everyplace オンライン資料

オンライン・ヘルプは、DB2 Everyplace 同期サーバーのモバイル・デバイス管理センターおよび DB2 Everyplace Mobile Application Builder で利用できます。

第 5 部 付録

特記事項

本書に記載の製品、サービス、または機能が日本においては提供されていない場合があります。日本で利用可能な製品、サービス、および機能については、日本 IBM の営業担当員にお尋ねください。本書で IBM 製品、プログラム、またはサービスに言及していても、その IBM 製品、プログラム、またはサービスのみが使用可能であることを意味するものではありません。これらに代えて、IBM の知的所有権を侵害することのない、機能的に同等の製品、プログラム、またはサービスを使用することができます。ただし、IBM 以外の製品とプログラムの操作またはサービスの評価および検証は、お客様の責任で行っていただきます。

IBM は、本書に記載されている内容に関して特許権 (特許出願中のものを含む) を保有している場合があります。本書の提供は、お客様にこれらの特許権について実施権を許諾することを意味するものではありません。実施権についてのお問い合わせは、書面にて下記宛先にお送りください。

〒106-0032
東京都港区六本木 3-2-31
IBM World Trade Asia Corporation
Licensing

以下の保証は、国または地域の法律に沿わない場合は、適用されません。 IBM およびその直接または間接の子会社は、本書を特定物として現存するままの状態で提供し、商品性の保証、特定目的適合性の保証および法律上の瑕疵担保責任を含むすべての明示もしくは黙示の保証責任を負わないものとします。国または地域によっては、法律の強行規定により、保証責任の制限が禁じられる場合、強行規定の制限を受けるものとします。

この情報には、技術的に不適切な記述や誤植を含む場合があります。本書は定期的に見直され、必要な変更は本書の次版に組み込まれます。 IBM は予告なしに、随時、この文書に記載されている製品またはプログラムに対して、改良または変更を行うことがあります。

本書において IBM 以外の Web サイトに言及している場合がありますが、便宜のため記載しただけであり、決してそれらの Web サイトを推奨するものではありません。それらの Web サイトにある資料は、この IBM 製品の資料の一部ではありません。それらの Web サイトは、お客様の責任でご使用ください。

IBM は、お客様が提供するいかなる情報も、お客様に対してなんら義務も負うことのない、自ら適切と信ずる方法で、使用もしくは配布することができるものとします。

本プログラムのライセンス保持者で、(i) 独自に作成したプログラムとその他のプログラム（本プログラムを含む）との間での情報交換、および (ii) 交換された情報の相互利用を可能にすることを目的として、本プログラムに関する情報を必要とする方は、下記に連絡してください。

IBM Canada Limited

Office of the Lab Director
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、適切な使用条件の下で使用することができますが、有償の場合もあります。

本書で説明されているライセンス・プログラムまたはその他のライセンス資料は、IBM 所定のプログラム契約の契約条項、IBM プログラムのご使用条件、またはそれと同等の条項に基づいて、 IBM より提供されます。

この文書に含まれるいかなるパフォーマンス・データも、管理環境下で決定されたものです。そのため、他の操作環境で得られた結果は、異なる可能性があります。一部の測定が、開発レベルのシステムで行われた可能性がありますが、その測定値が、一般に利用可能なシステムのものと同じである保証はありません。さらに、一部の測定値が、推定値である可能性があります。実際の結果は、異なる可能性があります。お客様は、お客様の特定の環境に適したデータを確かめる必要があります。

IBM 以外の製品に関する情報は、その製品の供給者、出版物、もしくはその他の公に利用可能なソースから入手したものです。IBM は、それらの製品のテストは行っておりません。したがって、他社製品に関する実行性、互換性、またはその他の要求については確証できません。 IBM 以外の製品の性能に関する質問は、それらの製品の供給者にお願いします。

IBM の将来の方向または意向に関する記述については、予告なしに変更または撤回される場合があり、単に目標を示しているものです。

本書には、日常の業務処理で用いられるデータや報告書の例が含まれています。より具体性を与えるために、それらの例には、個人、企業、ブランド、あるいは製品などの名前が含まれている場合があります。これらの名称はすべて架空のものであり、名称や住所が類似する企業が実在しているとしても、それは偶然にすぎません。

著作権使用許諾:

本書には、様々なオペレーティング・プラットフォームでのプログラミング手法を例示するサンプル・アプリケーション・プログラムがソース言語で掲載されています。お客様は、サンプル・プログラムが書かれているオペレーティング・プラットフォームのアプリケーション・プログラミング・インターフェースに準拠したアプリケーション・プログラムの開発、使用、販売、配布を目的として、いかなる形式においても、IBM に対価を支払うことなくこれを複製し、改変し、配布することができます。このサンプル・プログラムは、あらゆる条件下における完全なテストを経ていません。従って IBM は、これらのサンプル・プログラムについて信頼性、利便性もしくは機能性があることをほのめかしたり、保証することはできません。

それぞれの複製物、サンプル・プログラムのいかなる部分、またはすべての派生した創作物には、次のように、著作権表示を入れていただく必要があります。

© (お客様の会社名) (西暦年). このコードの一部は、IBM Corp. のサンプル・プログラムから取られています。 © Copyright IBM Corp. _年を入れる_. All rights reserved.

この製品には、3Com およびその寄稿者により開発されたソフトウェアが含まれています:

Copyright (c) 1998 3Com/Palm Computing Division. All rights reserved.
ソースおよびバイナリー形式での再配布および使用は、変更の有無に拘らず、次の条件を満たす場合に許可されます。

1. ソース・コードを再配布する場合には、上記の著作権表示、この使用条件および以下の免責表示を含める必要があります。
2. バイナリー形式で再配布する場合には、上記の著作権表示、以下の使用条件および免責表示を、配布に際して提供する関連文書および資料に記載する必要があります。
3. このソフトウェアの機能および使用についての広告には、以下の表示を行う必要があります。「この製品には、3Com およびその寄稿者が開発したソフトウェアが含まれています。」
4. なお、3Com および寄稿者の名称は、事前の書面による承諾がある場合を除き、このソフトウェアをもとに開発した製品を保証または推奨する目的で使用することはできません。

3COM およびその寄稿者は、このソフトウェアを特定物として現存するままの状態で提供し、法律上の瑕疵担保責任、商品性の保証および特定目的適合性の保証を含むすべての明示もしくは黙示の保証責任を負いません。起こりうる損害について予見の有無を問わず、「ソフトウェア」を使用したために生じる、直接的、間接的、付帯的、特別、懲罰的、または結果的損害 (代替の製品またはサービスの調達、データまたは利益の喪失、事業の中断などを含み、他のいかなる場合も含む) については、それが契約、厳格な責任、不法行為 (過失の場合もそうでない場合も含む) など、いかなる責任の理論においても、3Com および寄稿者はその責任を負いません。

商標

アスタリスク (*) 付きの以下の用語は、IBM Corporation の商標です。

ACF/VTAM
AISPO
AIX
AIX/6000
AIXwindows
AnyNet
APPN
AS/400
BookManager
CICS
C Set++
C/370
DATABASE 2
DataHub
DataJoiner
DataPropagator
DataRefresher
DB2DB2 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
iSeries
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
VisualAgeVM/ESA
VSE/ESA
VTAM
WebExplorer
WIN-OS/2
z/OS

以下は、それぞれ各社の商標または登録商標です。

Microsoft、Windows、Windows NT および Windows ロゴは、Microsoft Corporation の米国およびその他の国における商標です。

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。

Tivoli および NetView は、Tivoli Systems Inc. の米国およびその他の国における商標および登録商標です。

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などはそれぞれ各社の商標または登録商標です。

用語集

［ア行］

アプライ修飾子 (Apply qualifier). DataPropagator アプライ・プログラムの各インスタンスに固有のサブスクリプション定義を示す、文字ストリング。

一時表 (temporary table). SQL ステートメントの処理中に、中間結果を収容するために作成される表。

エンタープライズ・サーバー (enterprise server). ソース・サーバー を参照。

エンタープライズ・データベース (enterprise database). ソース・データベース を参照。

オブジェクト (object).

1. SQL を使用して作成または操作できるもの。例えば、表、視点、索引、またはパッケージ。
2. オブジェクト指向の設計またはプログラミングの場合、データおよびそのデータに関連する操作から構成される抽象的概念。

［カ行］

キー (key). 表、索引、または参照制約の記述の中で示されている 1 つの列または一連の順序付けされた列。

競合検出 (conflict detection). ユーザー・アプリケーションで更新された、ターゲット表内の古い行を検出するプロセス。競合が検出されると、競合を引き起こしたトランザクションはリジェクトされる。

クライアント (client). データベース・サーバーにアクセスする、または、それとやりとりを行うプログラム、あるいはユーザー。クライアントはアドミニストレーターを使用して定義する。

グループ (group). モバイル・データ同期の必要性が似通っているクライアントの集合。仕事をこなすためにグループ内のユーザーがアクセスする必要があるアプリケーションは何か、また、エンタープライズ・データのどのサブセットにそれらのユーザーがアクセスする必要があるかなど、それぞれのグループごとに同期特性を定義する。

携帯情報端末 (personal digital assistant (PDA)). 個人的な計画および整理のための作業 (例えば、予定表作成やメモ作成) に使用でき、電話、ファックス、およびネットワーキング機能を備えたハンドヘルド・デバイス。

結合 (join). リレーショナル演算の 1 つ。マッチング列値を基準にして 2 つまたはそれ以上の表からデータを検索することができる。

権限 (authorization). コンピューター・セキュリティーに関して、コンピューター・システムとやりとりを行う、あるいはそれを使用するためにユーザーに与えられた権利。

構造化照会言語 (SQL) (Structured Query Language (SQL)). リレーショナル・データベース内のデータを定義し操作するために使用するプログラミング言語。

コントロール・センター (Control Center). データベース・オブジェクト (例えば、データベースや表) および、それらのお互いの関係を表示する、グラフィカル・インターフェース。コントロール・センターから、DBA ユーティリティー、Visual Explain、および、パフォーマンス・モニター・ツールが提供するタスクを実行することができる。

［サ行］

サブスクリプション (subscription). ソース・データベースにある情報をどのようにターゲット・データベースに複写するかの仕様。サブスクリプションにより、どのデータ・サブセットおよびファイルをソース・データベースからコピーできるかを定義できる。ソース・サーバーに保管されているファイル用のファイル・サブスクリプションと、ソース・データベースにある表のための表サブスクリプションの、2 つのタイプのサブスクリプションが作成可能。

サブスクリプション・セット (subscription set). レプリケーション・サブスクリプションが入っているアドミニストレーター・オブジェクト。レプリケーション・サブスクリプションに定義されているデータおよびファイルに、グループ・メンバーがアクセスできるようにするには、サブスクリプション・セットを作成して、それにサブスクリプションを割り当ててから、そのサブスクリプション・セットをグループに割り当てる。サブスクリプション・セット・オブジェクトは、アプリケーション・オブジェクトを置き換える。

持続 (persistent). セッション境界間で、通常はデータベース・システムまたはディレクトリーなどの不揮発性ストレージに保持されるデータに関する用語。

主キー (primary key). 表定義の一部をなす固有のキー。主キーは、参照制約定義のデフォルトの親キー。DB2 Everyplace 同期サーバー バージョン 7 では、各レプリケーション・ソースに唯一無二の基本キーが必要。

ソース表 (source table). ターゲット表にコピーするデータが入っている表。ソース表は、レプリケーション・ソース表でなければならない。*ターゲット表* と対比。

ソース・サーバー (source server). レプリケーション・ソースがあるデータベースのロケーション。

ソース・データベース (source database). ターゲット・システムにコピーするデータが入っている、ソース・サーバーにあるデータベース。

［タ行］

ターゲット表 (target table). ソース表からデータがコピーされる表。中間層サーバーにあるミラー表も、モバイル・デバイス上の DB2 Everyplace 表も、ターゲットである。

ターゲット・データベース (target database). ソース・データベースからデータがコピーされる、モバイル・デバイス上にある DB2 Everyplace データベース。

タップ (tap). ハンドヘルド・デバイスと対話するためにスタイラスを使用すること。

中間層システム (mid-tier system). DB2 Everyplace 同期サーバーがインストールされているマシン。2 層同期構成では、中間層とソース・システムは同じマシンを指す。

データの同期 (data synchronization). *モバイル・データの同期* を参照。

データベース管理システム (database management system (DBMS)). 中央制御、データ独立性、および複雑な物理構造などのサービスを提供することによってデータを管理し、アクセスや、保全性、リカバリー、並行制御、プライバシー、セキュリティーの効率化が図れるコンピューター・プログラム。

データベース・サーバー (database server). データベースにデータベース・サービスを提供する機能単位。

データ・フィルター (data filter). *フィルター* を参照。

同期 (synchronization). *モバイル・データの同期* を参照。

同期オブジェクト (synchronization object). お客様の環境における同期化処理についての情報が入っている、アドミニストレーター内にある管理可能な項目。同期オブジェクトには、グループ、クライアント、サブスクリプション・セット、サブスクリプション、ログの 5 つのタイプがある。

同期セッション (synchronization session). モバイル・ユーザーまたはクライアント が、前回に同期をとった後にソース・データのローカル・コピーに対して行った変更をサブミットし、(リモート・サーバーにある) ソース・データに対して行われた変更があれば、それを受け取るトランザクション。

動的ホスト構成プロトコル (DHCP). TCP/IP を使用するコンピューターの構成を自動化するためのインターネット・プロトコル。

特権 (privilege). 特定の方法で特定のデータベース・オブジェクトにアクセスするための権利。これらの権利は、SYSADM (システム管理者) 権限、または DBADM (データベース管理者) 権限を持つユーザー、あるいは、オブジェクトの作成者がコントロールする。特権には、表の作成、削除、および表からデータを選択する権利も含まれる。

［ナ行］

認証 (authentication). ユーザー ID とパスワードを管理制御データベースにある項目と比較して、同期サーバーを使用してデータの同期をとることをユーザーに許可しているかどうかを検証するプロセス。

［ハ行］

パーベイシブ・コンピューティング (pervasive computing (PVC)). 情報機器と呼ばれる特殊な機器を含むコンピューティング基本設備の使用。これにより、ユーザーは広範囲のネットワークに基づくサービス (一般的にインターネットによって提供されるサービスを含む) にアクセスできる。このような情報機器としては、テレビ、自動車、電話、冷蔵庫、および電子レンジがある。パーベイシブ・コンピューティングにより、関連情報へのアクセスが便利になり、その情報に基づいた処置が行えるようになる。

バインド (bind). SQL において SQL プリコンパイラーからの出力を、アクセス・プランと呼ばれる使用可能な構造に変換するためのプロセス。このプロセスの際、データへのアクセス・パスが選択され、ある種の許可検査が行われる。

ハンドヘルド・デバイス (handheld device). 携帯可能なコンピューティング・デバイス。ハンドヘルド・デバイスとしては、手のひらサイズの PC および携帯情報端末 (PDA) がある。

ビュー (view). 照会によって生成されるデータから構成される論理表。

フィルター (filter). 指定された基準に従って、データ、シグナル、物体を分離するデバイスまたはプログラム。

[マ行]

マスター・データベース (master database). ソース・データベース を参照。

ミラー・データベース (mirror database). 同期とレプリケーションに必要なデータを保管するために、同期サーバーが内部で使用するデータベース。

無線 LAN (wireless LAN). 無線の使用の場合、モバイル・ユーザーは無線接続を介してローカル・エリア・ネットワーク (LAN) に接続できる。LAN 接続用の無線テクノロジーとしては、スピード・スペクトル、マイクロ波、および赤外線がある。

モバイル (mobile). さまざまなロケーションへ頻繁に移動し、さまざまなタイプのネットワーク接続 (例えば、ダイヤルアップ、LAN、または無線) を使用するユーザーが、携帯用コンピューターまたはハンドヘルド・デバイス上で行うコンピューターを使った作業に関する用語。

モバイル・データの同期 (mobile data synchronization). モバイル・ユーザーまたはクライアント が、ソース・データのローカル・コピーに対して行った変更をサブミットし、前回の同期以降に (リモート・データベース内の) ソース・データに対して行われた変更があればそれを受け取る、2 ステップのプロセス。

モバイル・デバイス管理センター (Mobile Devices Administration Center (MDAC)). 同期オブジェクト作成したり、編集したり、それらの互いの関係を見るためのグラフィカル・インターフェース。また、モバイル・デバイス管理センターにより、個別のクライアントの同期状況やエラー・メッセージを見ることができる。

[ラ行]

ラージ・オブジェクト (large object (LOB)). 連続したバイトで、長さは 2 ギガバイトまで可能。タイプとしては、BLOB (2 進)、CLOB (1 バイト文字または混合)、あるいは DBCLOB (2 バイト文字) の 3 つがある。

リフレッシュ (refresh). ユーザー表にある、関係するすべてのデータをターゲット表にコピーして、既存のデータを置き換えるプロセス。

リモート・アクセス・サービス (Remote Access Service (RAS)). 2 つのシステムの間の接続を管理する Windows プログラム。

リモート・データベース (remote database). 使用中のコンピューター以外のコンピューターに物理的に存在するデータベース。ローカル・データベースとの対比。リモート・コンピューティング・デバイスは、据え付けで移動できないものである場合も、また、移動できるものの場合もある。

レプリケーション (replication). ソース・サーバーのデータベース・ログまたはジャーナルに保管されている変更点を取り出し、それらをターゲット・サーバーに適用するプロセス。

レプリケーション・ソース (replication source). レプリケーションのためのソースとして定義されるデータベース表。レプリケーション・ソースとしてデータベース表を定義した後、その表はコピー要求を受け入れることができる。

ローカル・データベース (local database). 使用中のコンピューターに物理的に存在するデータベース。リモート・データベース との対比。

ログ (log). 同期エラー・メッセージとそれらの説明が入っているアドミニストレーター・オブジェクト。

[数字]

2 進ラージ・オブジェクト (BLOB) (binary large object (BLOB)). 連結したバイトで、そのサイズの範囲は 0 バイトから 2 ギガバイトまで。このバイト・シーケンスには、関連するコード・ページおよび文字セットはない。イメージ、オーディオ、およびビデオの各オブジェクトは BLOB に保管される。

2 バイト文字セット (DBCS). 各文字が 2 バイトで表現される文字セット。

B

BLOB. *2 進ラージ・オブジェクト* を参照。

D

DB2 DataPropagator. ソースからターゲットへの、データの自動的な複製を行う手段を提供するレプリケーション製品。モバイル・データの同期をとる際は、ミラー・データベースとリモート・データベースはソースとターゲットの両方の働きをする。DataPropagator は、クライアントの変更内容をミラーからリモート・データベースへ複製し、また、リモート・データベースからミラー・データベースへも変更内容を複製する。

DB2 コントロール・センター (DB2 Control Center). *コントロール・センター* を参照。

DBCS. *2 バイト文字セット (double-byte character set)* を参照してください。

DHCP. *動的ホスト構成プロトコル* を参照。

DPROP. *DB2 DataPropagator* を参照。

I

IBM Sync. DB2 Everyplace 同期サーバー・ソフトウェアのクライアント・コンポーネントを表すアイコンの名前。

L

LOB. *ラージ・オブジェクト* を参照。

M

MDAC. *モバイル・デバイス管理センター* を参照。

O

ODBC. *Open Database Connectivity* を参照。

Open Database Connectivity (ODBC). 呼び出し可能 SQL を使用してデータベース管理システムへのアクセスを可能にする API であり、SQL プリプロセッサーを使用する必要がない。 ODBC アーキテクチャーにより、ユーザーは、実行時にユーザーが選択したデータベース管理システムにアプリケーションをリンクする、データベース・ドライバーと呼ばれるモジュールを追加できる。アプリケーションは、サポートされているすべてのデータベース管理システムのモジュールに直接リンクする必要はない。

P

PDA. *携帯情報端末* を参照。

PVC. *パーベイシブ・コンピューティング* を参照。

R

RAS. *リモート・アクセス・サービス* を参照。

S

SQL. *構造化照会言語* を参照。

索引

日本語, 数字, 英字, 特殊文字の順に配列されています。なお, 濁音と半濁音は清音と同等に扱われています。

[ア行]

[カ行]

[サ行]

[タ行]

[ナ行]

[ハ行]

［マ行］

［ヤ行］

［ラ行］

［ワ行］

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

Q

R

S

T

U

V

W

[特殊文字]

弊社へのお問い合わせ

DB2 Everyplace 製品についてのお問い合わせや注文については、IBM 担当員または代理店にお申し付けください。

米国内では、次の番号にお願いします。

- 1-800-237-5511 (カスタマー・サポート)
- 1-888-426-4343 (サービスについての問い合わせ)

製品情報

米国内では、次の番号にお願いします。

- 1-800-IBM-CALL (1-800-426-2255) または 1-800-3IBM-OS2 (1-800-342-6672) (製品のご注文、お問い合わせ)
- 資料のご注文方法については、http://www.infocreate.co.jp/ をご覧ください。 (URL は、変更になる場合があります)

http://www.ibm.com/software/data/db2/everyplace/
DB2 Everyplace のワールド・ワイド・ウェブ (WWW) ページで、製品説明、講習会その他の最新の DB2 Everyplace 情報がご覧になれます。

http://www.ibm.com/software/data/db2/everyplace/library.html
この DB2 Everyplace Technical Library からは、よくある質問と回答をまとめた Q&A 集 (FAQ)、修正、資料、および DB2 Everyplace に関する最新の技術的な情報にアクセスできます。

注: この情報の提供は、英語版のみとなります。

http://www.ibm.com/software/data/
DB2 のワールド・ワイド・ウェブ (WWW) ページで、製品説明、講習会その他の最新の DB2 情報がご覧になれます。

http://www.ibm.com/software/data/db2/library/
この DB2 Product and Service Technical Library からは、よくある質問と回答をまとめた Q&A 集 (FAQ)、修正、資料、および DB2 に関する最新の技術的な情報にアクセスできます。

注: この情報の提供は、英語版のみとなります。

http://www.infocreate.co.jp/
資料のご注文方法については、この Web サイトをご覧ください。

http://www.ibm.com/education/certify/
IBM Web サイトの Professional Certification Program では、DB2 を含め、いろいろな IBM 製品についての認証テスト情報を提供しています。

ftp.software.ibm.com
無名 (anonymous) でログオンします。ディレクトリー `/ps/products/db2` に、DB2 およびその他の多数の製品についてのデモ、修正、情報ならびにツールがあります。

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 製品に関しての経験などの情報交換の場となっています。

Compuserve では、GO IBMDB2
このコマンドを入力して IBM DB2 ファミリー・フォーラムにアクセスします。すべての DB2 製品が、これらのフォーラムでサポートされています。

米国外での IBM に対するお問い合わせは、 *IBM Software Support Handbook* の付録 A を参照してください。この資料を見るには、http://www.ibm.com/support/ にアクセスして、画面の下部にある IBM Software Support Handbook へのリンクを選択してください。

注: 国によっては、IBM 特約店は IBM サポート・センターではなく、独自のサポート・センターに問い合わせていただく必要があります。

プログラム番号: 5724-D04

Printed in Japan

SC88-9479-01

日本アイ・ビー・エム株式会社
〒106-8711　東京都港区六本木3-2-12