DB2 Everywhere (Windows CE 版 および Palm OS 版) ソフトウェア バージョン 1.2

管理およびアプリケーション・プログラミングの手引き

SC88-8509-00
(英文原典：SC26-9675-00)

DB2 Everywhere (Windows CE 版 および Palm OS 版) ソフトウェア バージョン 1.2

管理およびアプリケーション・プログラミングの手引き

SC88-8509-00
(英文原典：SC26-9675-00)

お願い

本書、および本書でサポートする製品をご使用になる際は、前もって必ず 119ページの『特記事項』をお読みください。

本書は、DB2 Everywhere (Windows CE 版 および Palm OS 版) ソフトウェアのバージョン 1 リリース 2 (プログラム番号: 5648-C61) に適用されます。また、特に断りがない限り、後続のすべてのリリースにも適用されます。

本マニュアルについてご意見やご感想がありましたら

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

からお送りください。今後の参考にさせていただきます。
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.infocr.co.jp/ifc/books/`

をご覧ください。（URL は、変更になる場合があります）

原　典： SC26-9675-00
DB2 Everywhere for Windows CE and Palm OS Software
Version 1.2
Administration and Application
Programming Guide

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 1999.11

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

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

目次

本書について

IBM® 提供の DB2® Everywhere (Windows® CE 版 および Palm OS 版) ソフトウェア (DB2 Everywhere) を用いることにより、Palm OS または Windows CE ベースのパーソナル・ディジタル・アシスタント (PDA) 装置上で DB2 リレーショナル・データベースの利用が可能となり、データをエンタープライズ内の任意の場所にある DB2 データと同期化できるようになります。本書は、ユーザーが DB2 Everywhere を使用してどこからでも重要なデータにアクセスできるようにアプリケーションをセットアップしたり、開発したりする際のお役に立ちます。

本書の対象読者

本書は、下記の方々のために作成されています。

- DB2 Everywhere、および DB2 Everywhere 使用のアプリケーションを多数の PDA 装置に配置するシステム管理者。
- DB2 Everywhere 使用のアプリケーションを開発するアプリケーション・プログラマー。
- 例示照会 (QBE) アプリケーションを使用してデータにアクセスするか、またはデータをエンタープライズ・データベースと同期化する必要があるエンド・ユーザー。

本書の読者には、前提条件として Palm OS または Windows CE オペレーティング・システムおよび PDA 装置の基本操作に関する知識が必要です。

アプリケーション・プログラマーの場合は、DB2 Everywhere によってアクセスされ、更新されるリレーショナル・データベース管理システム (RDBMS) の基本機能に関する知識も必要です。

本書の使用方法

本書は、DB2 Everywhere Synchronization Server の資料と一緒に使用するようになっております。本書における作業およびトピックに関する詳細な説明が、DB2 Everywhere Synchronization Server の資料の方で述べられているものもあります。 *DB2 Everywhere Synchronization Server Administration Guide* では、DB2 Everywhere とエンタープライズ・データベース間でデータを同期化するためには DB2 Everywhere Synchronization Server をどのように構成すべきかに

ついて説明しています。DB2 Everywhere の使用時に行なわねばならぬかも知れぬ PDA 装置の基本操作の一部について本書では説明していません。これらの操作を行う場合の指示については、PDA 装置に添付されている資料をご参照ください。

表1 は一般的な作業をリストアップしたものであり、これらの作業を行う場合の説明個所を示しています。

表 1.

必要な作業	参照個所
IBM のモバイル・コンピューティング・ソリューションについて学習する	1ページの『第1章 DB2 Everywhere (Windows CE 版 および Palm OS 版) ソフトウェアの紹介』 または以下の Web サイトを参照。 http://www.ibm.com/software/data/db2/everywhere/
DB2 Everywhere 環境をセットアップする	7ページの『第2章 DB2 Everywhere の導入とセットアップ』
DB2 Everywhere データベースをアクセスする	12ページの『QBE の使用による DB2 Everywhere 表へのアクセス』
エンド・ユーザーとして、自分のアプリケーション、データ、およびファイルを自分のエンタープライズ・データベースと同期化する	11ページの『DB2 Everywhere データの同期化』
システム管理者として、PDA 装置とエンタープライズ・データベース間でアプリケーション、データ、およびファイルを同期化するようにサブスクリプションを定義する	*DB2 Everywhere Synchronization Server Administration Guide*。
アプリケーションを開発する	以下の章および付録 • 21ページの『第4章 DB2 Everywhere 用のアプリケーションの開発』 • 39ページの『付録A. サポートされる DB2 CLI 関数』 • 93ページの『付録B. サポートされる SQL ステートメント』
DB2 Everywhere によってサポートされる DB2 CLI 関数を使用する	39ページの『付録A. サポートされる DB2 CLI 関数』

表 1. (続き)

必要な作業	参照個所
DB2 Everywhere によってサポートされる SQL ステートメントを使用する	93ページの『付録B. サポートされる SQL ステートメント』
製品の最新の変更について学習する	製品と一緒に導入される CD-ROM 上の README ファイルまたは以下の Web サイトを参照。 http://www.ibm.com/software/data/db2/everywhere/

構文図の読み方

構文図は、左から右へ、上から下へと、行に沿って読み進みます。

►►── 記号はステートメントの始まりを示します。

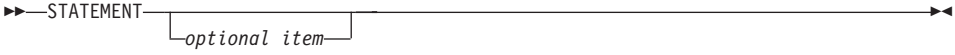
──► 記号は、ステートメント構文が次の行に続くことを示します。

►── 記号は、ステートメントが前の行から続いていることを示します。

──►◄ 記号は、ステートメントの終わりを示します

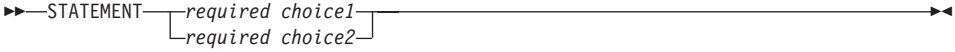
必須項目 (required item) は横線（メインパス）上に示してあります。

任意選択項目 (optional item) はメインパスの下に示されます。



2 個以上の項目から選択できる場合には、それらの項目はスタックの中に示されます。

必ず 1 つの項目を選択しなければならない (required choice) 場合は、1 つの項目がメインパス上に示されています。以下に例を示します。



複数の項目からなにも選択しなくてもよい場合には、スタック全体がメインパスの下に示されます。

項目のうちの 1 つが省略時値 (default) である場合には、メインパスの上に示され、残りの選択項目 (optional choice) は下に示されます。

メインパスの上の左へ戻る矢印は、反復できる項目 (repeatable item) を示します。この場合、反復する項目の間には、必ず 1 つ以上のブランクを入れます。

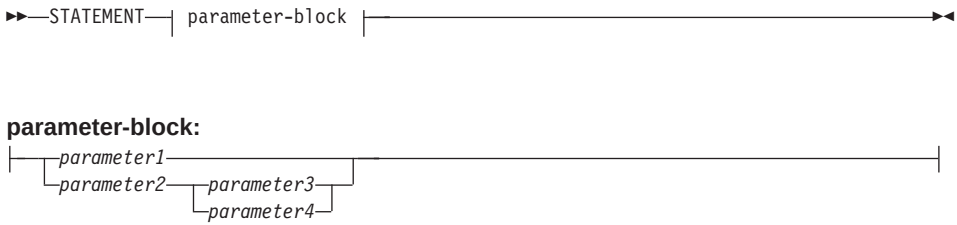
反復の矢印にコンマが含まれている場合は、反復項目をコンマで区切る必要があります。

スタックの上の反復矢印は、スタック項目の中から複数の項目を選択できること、または 1 つの項目を反復できることを示します。

キーワードは大文字で示されています (たとえば、FROM)。つづりは示されているとおり正確に入力する必要があります。変数はイタリック体の小文字で示されます (たとえば、*column-name*)。変数は構文内のユーザー提供の名前または値を表します。

構文図に句読記号、括弧、算術演算子、またはその他の記号が示されている場合は、それらを構文の一部として入力しなければなりません。

単一の変数が複数のパラメーターのセットを示す場合があります。たとえば、以下の図で、変数 parameter-block は、**parameter-block**: という見出しが付いている図によって置き換えられます。



用語

DB2 Everywhere の用語の定義が、本書のテキスト内にあります。本書で新しい用語が現れる場合には、イタリック体で示され、その定義がなされています。 121ページの『用語および省略語』 にも新出用語が定義されておりますが、そこでは DB2 Everywhere の用語と DB2 Everywhere および DB2 Everywhere Synchronization Server に関係する技術用語の定義が収めてあります。

表2 は、DB2 Everywhere の一般的な用語と本書で使用した省略名の簡単なリストです。

表 2. DB2 Everywhere の基本用語

用語	定義
DB2	IBM DATABASE 2®
DB2 CLI	DB2 コール・レベル・インターフェース、ODBC 準拠インターフェース
DB2 (AS/400® 版)	IBM DB2 (AS/400 版) バージョン 4.2.0
DB2 (OS/390 版)	IBM DB2 (MVS/ESA™ 版) バージョン 4 または IBM DB2 (OS/390® 版) バージョン 5 以降
DB2 UDB	IBM DB2 ユニバーサル・データベース バージョン 5 以降
DB2 UDB (AIX® 版)	IBM DB2 ユニバーサル・データベース (AIX 版) バージョン 5 以降
DB2 UDB (Windows NT 版)	IBM DB2 ユニバーサル・データベース (Windows NT 版) バージョン 5 以降
DB2 UDB (Solaris 版)	IBM DB2 ユニバーサル・データベース (Solaris 版) バージョン 5 以降
Palm OS	3Com Palm オペレーティング・システム、バージョン 3.0 以降
PDA 装置	パーソナル・ディジタル・アシスタント装置

表 2. DB2 Everywhere の基本用語 (続き)

用語	定義
QBE	DB2 Everywhere データベースにアクセスするための例示照会アプリケーション
Windows CE	Microsoft Windows CE オペレーティング・システム、バージョン 2.0 以降
Windows NT	Microsoft Windows NT オペレーティング・システム

強調表示の規則

本書では、下記の規則が使用されています。

太字	システムによって名前があらかじめ定義されているコマンド、キーワード、およびその他の項目を示します。
イタリック体	以下のいずれかを示します。 • ユーザーが提供する必要がある名前または値 (変数) • 一般的な強調 • 新しい用語の紹介 • 情報の別のソースの参照
モノスペース	以下のいずれかを示します。 • ファイルとディレクトリー • コマンド・プロンプトまたはウィンドウで入力を指示される情報 • 特定のデータ値の例 • システムによって表示される可能性があるテキストに類似のテキストの例 • システム・メッセージの例

本書をオンラインで使用する方法

本書は、導入メディア上で PDF 形式で使用可能です。PDF ファイル PVCA1200.PDF を表示するには、Adobe Acrobat Reader 4.0 を使用します。Adobe Acrobat Reader (Windows NT 版) は Web からダウンロードできます。詳細については、9ページの『オンライン・ブックの導入および読み取りプログラムのダウンロード』を参照してください。

PVCA1200.PDF ファイルは、任意のタイプの印刷装置で印刷することができます。

第1章 DB2 Everywhere (Windows CE 版 および Palm OS 版) ソフトウェアの紹介

本章では、DB2 Everywhere (Windows CE 版 および Palm OS 版) ソフトウェア (DB2 Everywhere) V1.2 を紹介するとともに、DB2 Everywhere ソリューションの構成要素について説明し、一般的な DB2 Everywhere のシナリオの例を示します。

DB2 Everywhere とは何か

DB2 Everywhere は、真のモバイル・コンピューティングを行うための IBM ソリューションの一部です。 DB2 Everywhere を用いることによって、よく移動する知的職業の人達 (外交員や、調査官、監査人、フィールド・サービス技術者、医者、不動産業者、および保険査定員など) は、オフィスから離れた場所にいても、必要とする重要なデータにアクセスすることができます。

とりわけ、各組織は、いまや DB2 エンタープライズ・データをパーソナル・ディジタル・アシスタント (PDA) 装置 (携帯用装置またはコンパニオン装置ともいう) へと送付することができるようになりました。DB2 Everywhere を使用すれば、PDA 装置上のデータベースにアクセスして更新を行うことができますし、Windows NT マシンを介して、データをエンタープライズ内の他の DB2 データベースと同期化することができます。

DB2 Everywhere は、PDA 装置に存在するリレーショナル・データベースです。 DB2 Everywhere と共に提供される例示照会アプリケーション (QBE) を使用すれば、PDA 装置上のデータにアクセスすることもできますし、サポートされる DB2 コール・レベル・インターフェース (CLI) 関数のセットを用いて自分独自のアプリケーションを作成することもできます。

DB2 Everywhere ソリューションの構成要素

DB2 Everywhere ソリューションの主要な構成要素は次のとおりです。

- PDA 装置上の DB2 Everywhere データベース
- DB2 Everywhere データベースと任意のプラットフォーム上のエンタープライズ・データベース間でデータの移動を行う、中間層同期サーバー上で稼動する DB2 Everywhere Synchronization Server
- 任意のプラットフォーム上の DB2 データベース・サーバー

- Windows マシン上のアプリケーション開発ツール

図1 は、DB2 Everywhere ソリューションの重要なソフトウェアとハードウェアの構成要素および構成要素間の関係を示しています。

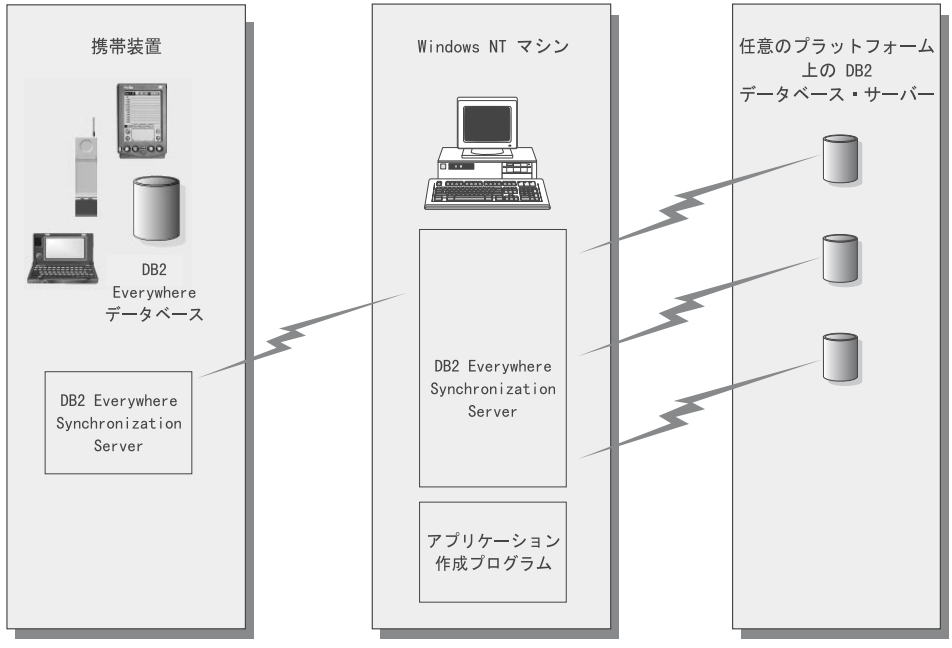


図 1. DB2 Everywhere ソリューションの構成要素

DB2 Everywhere データベース

DB2 Everywhere データベースは 3Com Palm OS 3.0 以降および Microsoft® Windows CE 2.0 以降のものを使用できる装置上で稼動可能です。 DB2 Everywhere には、簡単な例示照会 (QBE) アプリケーションが含まれている関係で、データベースを照会するための SQL ステートメントを作成するのに使用できます。 DB2 Everywhere がサポートする SQL ステートメントによって、表の作成や除去、表の行に関する削除、挿入、選択、または更新を行うことができます。

3ページの図2 は、DB2 Everywhere データベース内のデータを表示するために、QBE アプリケーションを使用した例を示しています。

図 2. QBE アプリケーション

DB2 Everywhere Synchronization Server

DB2 Everywhere とエンタープライズ・データベース間のデータの同期化は、DB2 Everywhere Synchronization Server を用いて行われます。 DB2 Everywhere Synchronization Server は Windows NT マシン上で稼動します。単一の Windows NT マシンを、多数の PDA 装置を同期化するために使用することが可能です。 DB2 Everywhere Synchronization Server を使用すれば、PDA 装置とサーバー間のデータに関する下記のソースを同期化できます。

- オープン・データベース・コネクティビティー (ODBC) データベース (DB2、Oracle、および SQL サーバーを含む)
- ファイル
- システム設定値
- アプリケーション・ソフトウェア

データの同期化は両方向です。すなわち、データは DB2 Everywhere データベー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。たとえば、 DB2 (OS/390 版) データベースからのデータのサブセットを PDA 装置にダウンロードし、データを表示して、データに変更を加えてから、変更済みのデータを OS/390 サーバーにアップロードすることができます。 DB2 Everywhere Synchronization Server は、競合解決に関するオプションを提供しています。

エンタープライズ・データベース

エンタープライズ・データベースは、ODBC 準拠のリレーショナル・データベースである必要があります。 サポートされる DB2 プラットフォームは次のとおりです。

- DB2 (OS/390 版)
- DB2 (AS/400 版)
- DB2 ユニバーサル・データベース (UNIX® 版、OS/2® 版、および Windows NT 版)

アプリケーション開発ツール

DB2 Everywhere で動くアプリケーションは、Palm OS および Windows CE プラットフォーム用の業界標準ツールを使用して、Windows NT ワークステーション上で開発することができます。 Palm OS プラットフォームの場合、アプリケーション開発者は、Palm コンピューティング・プラットフォーム用の Metrowerks CodeWarrior などの C/C++ 開発ツールが使用できます。 Windows CE プラットフォームの場合、アプリケーション開発者は Microsoft C/C++ が使用できます。

インターフェースは DB2 CLI のサブセットです。DB2 Everywhere がサポートする DB2 CLI 関数の詳細については、39ページの『付録A. サポートされる DB2 CLI 関数』を参照してください。

DB2 Everywhere のシナリオ

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。大部分の会社では、保険査定員が請求者の所有物を実際に見に行き、請求の妥当性を検査し間違いをただすなどして用紙に記入し、請求者に支払うべき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・システムに手操作で入力しますが、これは時間と費用のかかるプロセスです。

DB2 Everywhere アプリケーションの実行可能 PDA 装置を持っていると、保険査定員は、このプロセスを大幅に合理化することができます。保険査定員は、どこであれ現在自分のいるところから PDA 装置を使用することにより、査定のスケジュールや手順、および請求者の保険証券の情報にアクセスすることが可能になります。保険査定員は、PDA 装置上で査定用紙に記入することもできます。保険査定員はオフィスに戻ったとき、新しい査定用紙の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、PDA 装置上のデータを会社のコンピューター・システムと同期化することができます。また、保険査定員が現場で情報を必要とする場合には、モデムを介して即時に PDA 装置上のデータを会社のコンピューター・システムと同期化することもできます。請求査定プロセスが紙をまったく使用しないで行えるようになったため、保険会社のコストは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章 DB2 Everywhere の導入とセットアップ

本章では、PDA 装置上で DB2 Everywhere を使用するための環境の導入とセットアップについて説明します。Palm OS、Windows CE 両方の PDA 装置に関する説明が含まれています。

本章の説明は、1 個または複数の PDA 装置上に DB2 Everywhere を配置しようとしているシステム管理者またはアプリケーション・プログラマー向けに用意しました。

DB2 Everywhere 環境を導入しセットアップするステップ

DB2 Everywhere 環境をセットアップするためには以下のステップを実行してください。

1. Windows NT マシン上で、PDA 装置 に添付されている接続ソフトウェア (たとえば、Palm OS 装置の場合は hotsync、Windows CE 装置の場合は WinCE Services) を導入し構成する。
2. PDA 装置を Windows NT マシンに物理的に接続する。
3. CD-ROM 装置のフォルダー内の **Setup.exe** アイコンをクリックすることによって、DB2 Everywhere を Windows NT マシンおよび任意選択で PDA 装置に導入する。
4. セットアップ・プロセス時に DB2 Everywhere を PDA 装置に導入しなかった場合には、それをここで以下のいずれかの方法により導入する。
   - **スタート → プログラム → DB2 Everywhere → PDA デバイスへのインストール** をクリックする。 DB2 Everywhere がユーザーのマシンに接続されている PDA 装置 に導入されます。

     Windows CE PDA 装置の場合、DB2 Everywhere 動的リンク・ライブラリー (DB2e.dll) が ¥Windows ディレクトリーに導入されます。サンプル・コードが ¥Windows¥Programs¥DB2 Everywhere ディレクトリーに導入されます。
   - DB2 Everywhere ファイルを手操作で PDA 装置に導入する。

     この方法は、サンプル・コードをユーザーが選択したディレクトリーに導入したい場合に使用することができます。

     Palm OS の場合:

DB2 Everywhere 共用ライブラリー (DB2eCmp.prc と DB2eRunTime.prc) をユーザーの PDA 装置 にコピーする必要があります。詳細については、23ページの図9を参照してください。

Windows CE の場合:

DB2 Everywhere 動的リンク・ライブラリー (DB2e.dll) をユーザーの PDA 装置にコピーする必要があります。詳細については、23ページの図9を参照してください。 DB2e.dll を PDA 装置上の Windows フォルダーに導入する必要があります。

5. DB2 Everywhere アプリケーションを PDA 装置に導入する。
6. 任意選択:
   DB2 エンタープライズ・データベースからのデータを使用して、DB2 Everywhere データベースを初期設定する。
   a. ユーザーの Windows NT マシン上に DB2 Everywhere Synchronization Server を導入し構成する。DB2 Everywhere と共に使用するために DB2 Everywhere Synchronization Server を導入し構成する方法については、*DB2 Everywhere Synchronization Server Administration Guide* を参照してください。
   b. DB2 Everywhere Synchronization Server を使用して、DB2 エンタープライズ・データベース・サーバー (DB2 (OS/390 版)、DB2 (AS/400 版)、または DB2 ユニバーサル・データベースなど) からの初期 DB2 データをユーザーの DB2 Everywhere データベースに複写する。

以上で、DB2 Everywhere アプリケーションを PDA 装置でテストし使用することができます。DB2 Everywhere アプリケーションを用いて、いずれかのデータを変更する場合、DB2 Everywhere Synchronization Server を使用すれば、データを DB2 エンタープライズ・データベースと同期化することができます。データの同期化に必要なステップについては、11ページの『DB2 Everywhere データの同期化』を参照してください。

DB2 Everywhere アプリケーションの大量の配置

DB2 Everywhere アプリケーションの初期テストの後で、DB2 Everywhere アプリケーションと DB2 データを多数の PDA 装置に配置したい場合があります。これを行うのに DB2 Everywhere Synchronization Server の使用が効果的です。

DB2 Everywhere Synchronization Server は、ユーザー・グループを定義する機能を提供します。ユーザーをユーザー・グループに編成し、ユーザー・グループごとにデータ同期体系を定義することができます。たとえば、自動車保険の

査定員用に 1 つのユーザー・グループを定義し、住宅保険の査定員用にもう 1 つのユーザー・グループを定義することができます。次に、DB2 Everywhere Synchronization Server を使用すれば、DB2 Everywhere アプリケーションをユーザー・グループ内の各ユーザーに対して、そのグループに定義済みの同期体系を使用して配置することができます。これらの作業を行う方法の詳細については、*DB2 Everywhere Synchronization Server Administration Guide* を参照してください。

オンライン・ブックの導入および読み取りプログラムのダウンロード

本書は PDF 形式で提供されています。Adobe Acrobat Reader を使用すれば、本書をオンラインで表示することができます。表3 はオンライン・ブックのファイルとディレクトリーを示しています。

表 3. DB2 Everywhere オンライン・ブックのファイル

表題	ディレクトリーとファイル名
管理およびアプリケーション・プログラミングの手引き V1.2	¥DOCS¥PVCA1200.PDF

Adobe Acrobat Reader は以下の Web サイトからダウンロードできます。

http://www.adobe.com/prodindex/acrobat/readstep.html

第3章 DB2 Everywhere の使用

本章では、以下の作業を行う方法について説明します。

- DB2 Everywhere データを PDA 装置 とエンタープライズ・データベース間で同期化する。
- 例示照会アプリケーション (QBE) を使用して DB2 Everywhere 表にアクセスする。

DB2 Everywhere データの同期化

Palm OS の場合:

Palm OS PDA 装置上の DB2 Everywhere データをエンタープライズ・データベースと同期化するには、以下のステップを実行してください。

1. PDA 装置 を連結架台またはモデムに接続する。
2. **アプリケーション** アイコンをタップしてから、**接続** アイコンをタップする。 DB2 Everywhere Synchronization Server ウィンドウがオープンします。
3. **接続** をタップする。

データを同期化した後、QBE アプリケーションを使用すれば、DB2 Everywhere 表に格納されているデータを照会し変更することができます。

Windows CE の場合:

Windows CE PDA 装置上の DB2 Everywhere データをエンタープライズ・データベースと同期化するには、以下のステップを実行してください。

1. PDA 装置 を連結架台またはモデムに接続する。
2. **スタート** ボタンから、**プログラム → DB2 Everywhere Synchronization Server** をタップする。 DB2 Everywhere Synchronization Server ウィンドウがオープンします。
3. **接続** をタップする。

重要: PDA 装置上に新しい表を作成する場合、PDA 装置 をサーバーと同期化させても、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。同期化を行う前に、エンタープライズ・データベース上に表を作成しておく必要があります。

QBE の使用による DB2 Everywhere 表へのアクセス

DB2 Everywhere には QBE と呼ばれる簡単なアプリケーションがあり、これを使用すれば Palm OS PDA 装置 上の DB2 Everywhere 表にアクセスすることができます。QBE を使用して以下の操作を行うことができます。

- 『データの表示』
- 16ページの『データの変更』
- 19ページの『SQL ステートメントの入力と実行』

QBE の開始と停止

QBE を開始するには、**アプリケーション** アイコンをタップしてから、**QBE** アイコンをタップします。

QBE を終了するには、**アプリケーション** アイコンをタップするか、または装置上の任意のボタン (たとえば、**up** または **down** ボタン) を押します。

データの表示

QBE を使用すれば、DB2 Everywhere 表のデータを表示することができます。画面上のスクロール・バーまたは PDA 装置上の up および down ボタンを使用して表のデータをスクロールすることができます。up および down ボタンによって、一度に 1 ページがスクロールされます。

表のデータを表示するには、以下のステップを実行します。

1. QBE を開始する。
2. 表選択の下矢印をタップして、表示可能な表のリストを表示する。 13ページの図3 は、表示可能な表リストの例を示しています。

図3. QBE の表リスト

3. 表示したい表の名前をタップする。表の内容が表示されます。 14ページの図4 は表の内容を示しています。

図 4. 「個人情報」表の内容

右側へスクロールすれば列をさらに表示することができ、下側にスクロールすれば行をさらに表示することができます。

別の表を選択するには、表選択の下矢印をタップしてください。

データのフィルター処理

QBE を使用すれば、表示している表のデータをフィルター処理することもできます。データのフィルター処理を行うには、以下のステップを実行してください。

1. **メニュー** アイコンをタップし、メニューから **フィルター** を選択する。「フィルター」ウィンドウがオープンします。15ページの図5 は「フィルター」ウィンドウを示しています。

図 5. 「QBE フィルター」ウィンドウ

2. **列** 見出しの下で、下矢印をタップし、フィルター処理したい列を選択する。
3. **演算子** 見出しの下で、下矢印をタップし、フィルター処理したい演算子を選択する。
4. **値** 見出しの下で、ブランク・フィールドをタップし、フィルター用の値を入力する。
5. さらにフィルターが必要であれば、次の 2 つの行のうちの 1 つまたは両方の **列**、**演算子**、および **値** の各フィールドを定義する。
6. **一致する** のボックスで、定義済みのすべてのフィルターに関して (AND 条件で) フィルター処理するには **すべて** を選択し、定義済みのいずれかのフィルターに関して (OR 条件で) フィルター処理するには **任意** を選択する。

7. **OK** をタップする。 表は、フィルター処理されたデータのみが再表示されます。

前に適用したフィルターに戻したい場合には、**やり直し** ボタンをタップしてください。 入力したデータを消去したい場合には、**クリア** ボタンをタップしてください。「フィルター」ウィンドウの初期状態に戻したい場合には、**クリア** ボタンをタップしてから、**OK** をタップしてください。すべての表データが表示されます。

データの分類

表のデータを列名によって分類することができます。表のデータを分類するには、分類に使用する列名をタップしてください。表の各行が、選択した列のデータの昇順に分類されます。 分類された列名の先頭に + が表示されます。

データの変更

QBE を使用すれば、DB2 Everywhere データベースの表のデータを更新し、表に新しいデータを挿入し、表からデータを削除することができます。データに加えた変更は、同期化の後にのみ、エンタープライズ・データベースに反映されます。

制約事項: 表に基本キーが含まれていない場合には、QBE を使用しても表のデータを変更できません。

データの更新

表のデータを更新するには、以下のステップを実行します。

1. 更新したいセルをタップする。「値の更新」ウィンドウがオープンします。17ページの図6 は「値の更新」ウィンドウを示しています。

図 6. 「値の更新」ウィンドウ

2. セルの新しい値を指定する。
3. **更新** をタップする。表が更新済みのデータとともに再表示されます。

データの挿入

表に新しいデータを挿入するには、以下のステップを実行します。

1. **メニュー** アイコンをタップし、メニューから **レコードの挿入** を選択する。「値の挿入」ウィンドウがオープンします。 18ページの図7 は「値の挿入」ウィンドウを示しています。

図 7. 「値の挿入」ウィンドウ

2. フィールドで新しい値を指定する。
3. **挿入** をタップする。ウィンドウがオープンし、行が正常に挿入されたことを示すメッセージが現れます。
4. **OK** をタップする。ウィンドウがオープンし、フィルターを適用するかどうかを尋ねてくるので、新しいレコードの残りの列値を更新することができます。
5. 任意選択:
   残りの列の値を指定するためには、**Yes** をタップ。これにより、表の残りの各列に関する新しい値の入力が指示されます。

   残りの列値を更新したくない場合には、**No** をタップします。
6. 新しいレコードのすべての値の入力を完了すると、フィルターのリセットとすべての行の表示を行うかどうかを尋ねるウィンドウがオープンされるので、**Yes** をタップする。表が再表示されます。

データの削除

表からデータを削除するには、以下のステップを実行してください。

1. 削除したい行の行番号をタップする。ウィンドウがオープンし、選択した行の削除について確認してきます。
2. **Yes** をタップして、選択した行を削除する。

SQL ステートメントの入力と実行

QBE コマンド行プロセッサー (CLP) を使用すれば、ユーザー指定の SQL ステートメントを入力し実行することができます。

SQL ステートメントを入力し実行するためには、以下のステップに従ってください。

1. **メニュー** アイコンをタップし、メニューから **CLP** を選択する。 DB2 Everywhere CLP ウィンドウがオープンします。
2. 照会フィールドに SQL ステートメントを指定する。20ページの図8 は、DB2 Everywhere CLP ウィンドウに入力された SQL ステートメントの例を示しています。

図 8. DB2 Everywhere CLP ウィンドウ

3. **SQL の実行** をタップする。

DB2 Everywhere がサポートする SQL ステートメントの詳細については、93 ページの『付録B. サポートされる SQL ステートメント』を参照してください。

第4章 DB2 Everywhere 用のアプリケーションの開発

本章では、DB2 Everywhere でのアプリケーションを C/C++ を使用して開発する方法について説明します。 最初の節では、業界標準のアプリケーション開発ツールを使用して C/C++ でアプリケーションを開発する方法について説明します。次の節には、C で開発したアプリケーションの例が記載されています。将来の強化点として、アプリケーションの開発がサテライト形式を使用して行えるようになる可能性があります。

アプリケーションの開発

C または C++ を使用して DB2 Everywhere 用のアプリケーションを開発するには、以下のステップを実行してください。

1. アプリケーションとそのデータ要件を定義する。

   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 Everywhere データベースで検索し、格納し、更新する方法を決定してください。

2. DB2 CLI インターフェースおよび使用する DB2 CLI 関数について理解する。

   DB2 Everywhere には、アプリケーションの開発で使用できる DB2 CLI 関数セットが用意されています。 これらの関数の説明は 39ページの『付録A. サポートされる DB2 CLI 関数』 にあります。

3. DB2 Everywhere によってサポートされる DB2 CLI 関数を使用して C/C++ アプリケーション・プログラムを作成する。

   アプリケーションを開発する際、PDA 装置に使用できる標準の C/C++ 開発ツールを使用することができます。

   Palm OS 開発環境:

   Palm OS アプリケーションの場合、Palm Computing プラットフォーム用の Metrowerks CodeWarrior を使用することができます。この市販の開発環境によって、ANSI C および C++ プログラムを Windows NT システム上で作成することができます。この開発環境は以下のものから構成されています。

   Metrowerks Constructor
   : アプリケーションのユーザー・インターフェース要素を作成するために使用できるグラフィカル資源エディター。

CodeWarrior Integrated Development Environment (IDE)
Motorola 68000 C/C++ コンパイラー、リンカー、および PalmRez が含まれているプロジェクト・ベースの IDE。PalmRez はオブジェクト・コードから PRC ファイルを作成し、Constructor 形式から PRC 形式に資源を変換します。

CodeWarrior Debugger
Palm OS アプリケーションをデバッグするために使用するソース・レベル・デバッガー。このデバッガーは、シリアル・ケーブルを介して Windows NT マシンに接続されている Palm OS PDA 装置上または Palm OS エミュレーター (POSE) 上で実行中のアプリケーションをデバッグすることができます。

重要: SELECT ステートメントを使用すると、照会の実行が長時間行われる可能性があります。 Palm OS PDA 装置上で実行されている長時間の照会を停止するには、装置上の任意のボタン (たとえば、**up** または **down** ボタン) を押します。これにより、SQLSTATE 57014 が DB2 Everywhere によって出されます。この後でアプリケーションの方ではどのように割り込みを処理するかを決めることができます。

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、作成者 ID を 3Com に登録してください。 DB2 Everywhere の表とアプリケーションは DB2*x* のような作成者 ID を持っています。ここで *x* は a から z の英字です。

Windows CE 開発環境:

Windows CE アプリケーションの場合、Microsoft Visual C++ WCE ツールキット (Windows CE 版) を使用することができます。 Visual C++ WCE ツールキット (Windows CE 版) は、グラフィカル・コンストラクターおよびソース・レベルのデバッガーを含むプロジェクト・ベースの IDE です。

4. DB2 Everywhere が Windows NT マシンに導入済みであることを確認する。

   DB2 Everywhere ヘッダー・ファイル (sqlcli.h、sqlcli1.h、sqlext.h、sqlsystm.h) および CLI 静的ライブラリー (DB2e.lib) をプロジェクト・ファイルに組み込んでください。ヘッダー・ファイルには、DB2 Everywhere とともに提供される定数、データ・タイプおよび C 関数のプロトタイプが入っています。23ページの図9 は、各種装置の DB2 Everywhere ファイルのディレクトリー構造とファイル名を示しています。

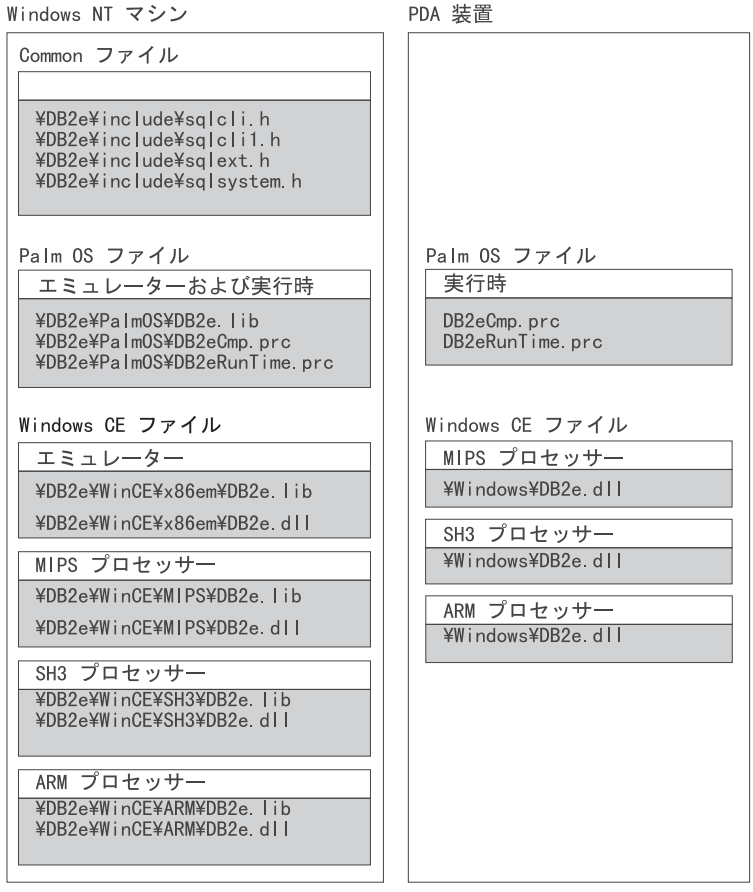


図 9. DB2 Everywhere ファイル

5. DB2 Everywhere ヘッダー・ファイルを使用してアプリケーション・コードをコンパイルする。

   Palm OS アプリケーションの場合:

   CodeWarrior IDE を使用してプログラムをコンパイルならびにリンクするには、sample.mcp のようなプロジェクト・ファイルを定義する必要があります。DB2 Everywhere ヘッダー・ファイルである sqlcli.h、sqlcli1.h、sqlext.h、および sqlsystm.h にアクセスするためのパスを設定する必要があります。プロジェクトに DB2 Everywhere CLI 静的ライブラリー DB2e.lib を組み込んでください。 25ページの図10 は、アプリケーションのコンパイル、リンク、および PDA 装置へのダウンロードのプロセスを示しています。

Palm OS アプリケーションの省略時アプリケーション・スタック・サイズは非常に制限されています。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生じる可能性があります。この問題を回避するために、DB2 Everywhere 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・サイズを指定することができます。 palm-pref.r ファイル内の指示に従って、これをプロジェクト・ファイルに組み込んでください。

推奨事項: スタック・サイズを 8 KB に増加してください。省略時値は 4 KB です。

Windows CE アプリケーションの場合:

Microsoft Visual C++ を使用してプログラムをコンパイルならびにリンクするには、sample.dsp のようなプロジェクト・ファイルを定義する必要があります。 DB2 Everywhere ヘッダー・ファイルである sqlcli.h、sqlcli1.h、sqlext.h、および sqlsystm.h にアクセスするためのパスを設定する必要があります。プロジェクトに DB2 Everywhere CLI 静的ライブラリー DB2e.lib を組み込んでください。各環境 (たとえば、エミュレーション、MIPS プロセッサー、SH3 プロセッサー、ARM プロセッサー) の設定値を定義してください。25ページの図10 は、アプリケーションのコンパイル、リンク、および PDA 装置へのダウンロードのプロセスを示しています。

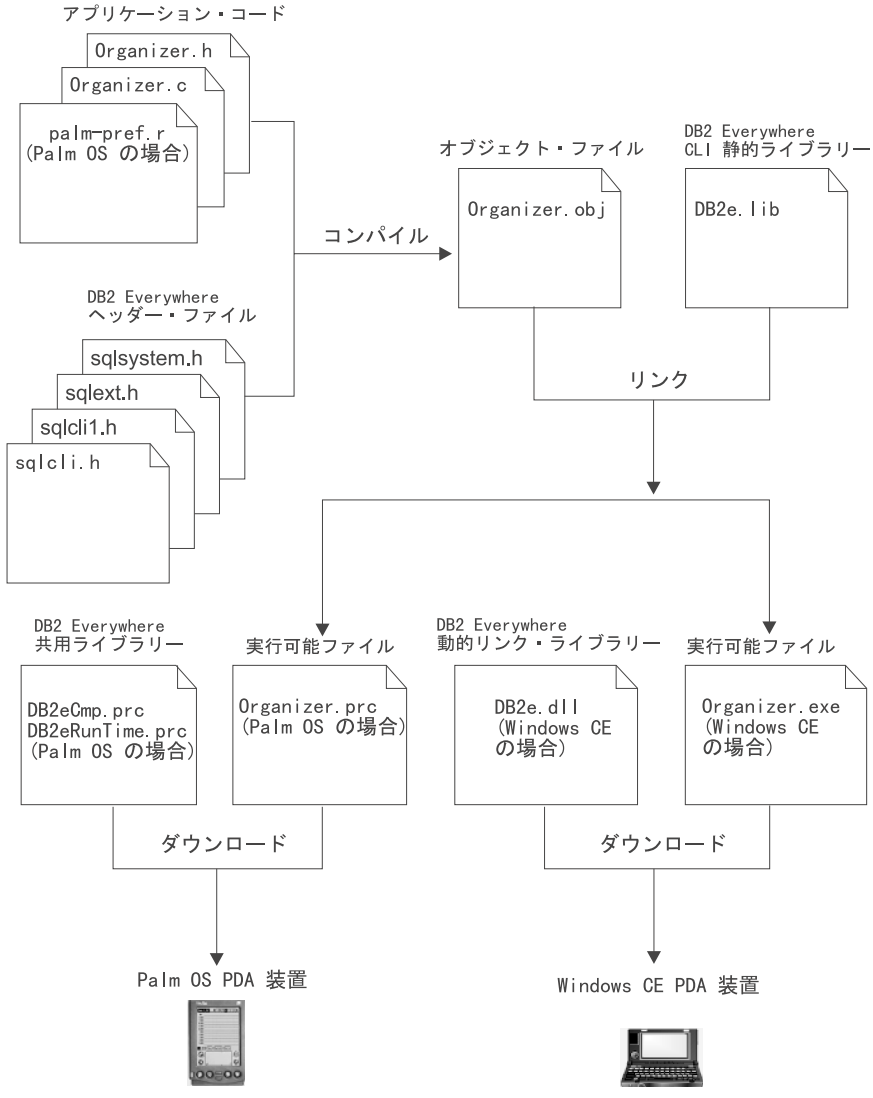


図 10. アプリケーションのコンパイル、リンク、およびダウンロード

6. オブジェクト・コードを DB2 Everywhere CLI 静的ライブラリー (DB2e.lib) にリンクする。 DB2e.lib には様々なバージョンがあります。たとえば、Palm OS 環境用のバージョンと、各 Windows CE エミュレーション環境 (Windows CE MIPS プロセッサー装置、Windows CE SH3 プロセッサー装置および Windows CE ARM プロセッサー装置) 用のバージョンがあります。

7. Windows NT マシン上でエミュレーター・モードでアプリケーションをテストする。
   a. アプリケーション用の適切なエミュレーターをセットアップする。

      Palm OS アプリケーションの場合:

      エミュレーターは CodeWarrior 開発環境に組み込まれています。エミュレーターの最新バージョンを入手するには、エミュレーターを以下の Web サイトからダウンロードしてください。

      http://www.palm.com/devzone/pose

      Windows CE アプリケーションの場合:

      エミュレーターは Microsoft Visual C++ WCE ツールキット (Windows CE 版) の一部です。

   b. エミュレーターが使用するための適切な DB2 Everywhere 共用ライブラリーまたは動的リンク・ライブラリーをロードする。共用ライブラリーまたは動的リンク・ライブラリーがない場合には、DB2 CLI 関数の呼び出し時に、エラー・メッセージ 『Cannot Load DB2e library (DB2e ライブラリーをロードできません)』 が表示されます。

      Palm OS アプリケーションの場合

      ¥DB2e¥PalmOS¥DB2eCmp.prc および
      ¥DB2e¥PalmOS¥DB2eRunTime.prc

      Windows CE アプリケーションの場合

      ¥DB2e¥WinCE¥x86em¥DB2e.dll

   c. 適切なエミュレーターを使用して Windows NT マシン上でアプリケーションをテストする。
8. 実際の PDA 装置上でアプリケーションをテストする。
   a. PDA 装置 に添付されている接続ソフトウェア (たとえば、Palm OS 装置の場合は hotsync、Windows CE 装置の場合は WinCE Services) を導入し構成する。
   b. PDA 装置を Windows NT マシンに物理的に接続する。
   c. 適切な DB2 Everywhere 共用ライブラリーまたは動的リンク・ライブラリーを PDA 装置にロードする。 以下はその例です。

      Palm OS PDA 装置 の場合

      ¥DB2e¥PalmOS¥DB2eCmp.prc および
      ¥DB2e¥PalmOS¥DB2eRunTime.prc

Windows CE MIPS プロセッサー PDA 装置の場合
¥DB2e¥WinCE¥MIPS¥DB2e.dll

Windows CE SH3 プロセッサー PDA 装置の場合
¥DB2e¥WinCE¥SH3¥DB2e.dll

Windows CE ARM プロセッサー PDA 装置の場合
¥DB2e¥WinCE¥ARM¥DB2e.dll

Windows CE PDA 装置の場合、DB2e.dll 動的リンク・ライブラリーが Windows フォルダーに組み込まれていなければなりません。

d. 実行可能ファイルを PDA 装置に導入する。

e. 任意選択:
DB2 Everywhere Synchronization Server を Windows NT マシンに導入し構成する。DB2 Everywhere と共に使用するように DB2 Everywhere Synchronization Server を導入し構成する方法については、 *DB2 Everywhere Synchronization Server Administration Guide* を参照してください。

f. 任意選択:
DB2 Everywhere Synchronization Server を使用して、初期 DB2 データを DB2 エンタープライズ・データベース・サーバー (DB2 (OS/390 版)、DB2 (AS/400 版)、または DB2 ユニバーサル・データベースなど) から DB2 Everywhere データベースに複写する。

g. PDA 装置上でアプリケーションをテストする。

C/C++ を使用して開発するアプリケーションの例

Visiting Nurse (訪問看護婦) のデモは、DB2 Everywhere を使用してアプリケーションを作成する方法の例を示すために作成されています。このデモは、Metrowerk's CodeWarrior (Palm OS 版) および Microsoft Visual C++ WCE ツールキット (Windows CE 版) を使用して C で作成されています。 Visiting Nurse デモのソース・コードは製品の CD-ROM に入っています。

Visiting Nurse デモの説明

このデモは、家庭にいる患者を訪問する出張介護看護婦用のアプリケーションです。 DB2 Everywhere アプリケーションがない場合、看護婦はメモ用紙にメモし、後でメモをオフィス内ワークステーションのデータベースに転記するする必要があります。DB2 Everywhere アプリケーションを使用すれば、訪問看護婦は以下のことが行えます。

- 患者の一般情報 (名前、住所、電話番号、病状など) にアクセスする

- 患者の病状 (血圧、脈拍、体温、および血糖値など) を収集する
- 新しい病状レコードに関する自動的な時刻と日付のスタンプを入手する
- 緊急時に連絡すべき人のリストにアクセスする

1 日の終わりに、訪問看護婦は PDA 装置上のデータを中央のデータベースと同期化して、以下の処理を行うことができます。

- 患者の状態を使用して中央のデータベースを更新する
- 次の日に訪問すべき患者のリストを入手する

この例のデータベースには、以下の 4 つの表があります。

スケジュール
: 看護婦の予約情報が入っています。この表には、看護婦 ID、患者 ID、日付、および時刻などの情報が含まれています。

個人情報
: 患者および緊急時連絡先に関するデータが入っています。この表には、名前、社会保険番号、住所、および電話番号などの情報が含まれています。社会保険番号が基本キーとして使用されます。

医療記録
: 患者の毎日の病状レコードが入っています。この表には、血圧、脈拍、および体温などの情報が含まれています。レコード ID が基本キーとして使用されます。

連絡先 各患者の緊急時連絡先のリストが入っています。この表には、患者名、緊急時連絡先の名前、および患者との関係が入っています。

サンプル・コードに含まれているファイル

Palm OS の場合:

Palm OS の Visiting Nurse デモは、Palm OS プラットフォーム上で DB2 Everywhere 用のアプリケーションを作成する方法を示すための、2 つの完全なプロジェクト・ファイルから構成されています。 2 つのプロジェクト・ファイルを以下に示します。

DemoInit アプリケーション
: DB2 Everywhere Synchronization Server を使用しないで、定義済みのレコードと表のセットを使用して、DB2 Everywhere データベースを定義しデータを入力するために使用します。 DB2 Everywhere Synchronization Server を使用してデータを入手することもできます。 DemoInit アプリケーションは、以下の 6 つのファイルからなります。

¥DB2e¥PalmOS¥Samples¥DemoInit¥Src¥DemoInit.mpc
プロジェクト・ファイル

¥DB2e¥PalmOS¥Samples¥DemoInit¥DemoInit.prc
実行可能ファイル

¥DB2e¥PalmOS¥Samples¥DemoInit¥Src¥DemoInit.c
C ソース・コード

¥DB2e¥PalmOS¥Samples¥DemoInit¥Src¥DemoInitRsc.h
資源生成ヘッダー・ファイル

¥DB2e¥PalmOS¥Samples¥DemoInit¥Src¥DemoInit.rsrc
資源ファイル

¥DB2e¥PalmOS¥Samples¥DemoInit¥Src¥Resource.frk¥DemoInit.rsrc
資源ファイル

デモ・アプリケーション
Visiting Nurse アプリケーション。このアプリケーションには、DemoInit アプリケーションから入手できるか、または DB2 Everywhere Synchronization Server を使用してサーバーと同期化することによって入手できるデータのセットが必要です。デモ・アプリケーションは以下の 7 つのファイルからなります。

¥DB2e¥PalmOS¥Samples¥Demo¥Src¥Demo.mpc
プロジェクト・ファイル

¥DB2e¥PalmOS¥Samples¥Demo¥Demo.prc
実行可能ファイル

¥DB2e¥PalmOS¥Samples¥Demo¥Src¥Demo.c
C ソース・コード

¥DB2e¥PalmOS¥Samples¥Demo¥Src¥DemoRsc.h
資源生成組み込みファイル

¥DB2e¥PalmOS¥Samples¥Demo¥Src¥Demo.rsrc
資源ファイル

¥DB2e¥PalmOS¥Samples¥Demo¥Src¥Resource.frk¥Demo.rsrc
資源ファイル

¥DB2e¥PalmOS¥Samples¥palm-pref.r
Palm OS アプリケーション用の Pref #0 資源設定値ファイル

Windows CE の場合:

Windows CE 用の Visiting Nurse デモのサンプル・コードは、1 つの作業スペース・ファイル (Visiting Nurse.dsw) および 2 つの完全なプロジェクト・ファイルから構成され、Windows CE プラットフォーム上で DB2 Everywhere 用のアプリケーションを作成する方法を示しています。 2 つのプロジェクト・ファイルを以下に示します。

DemoInit アプリケーション

DB2 Everywhere Synchronization Server を使用しないで、定義済みのレコードと表のセットを使用して、DB2 Everywhere データベースを定義しデータを入力するために使用します。 DB2 Everywhere Synchronization Server を使用してデータを入手することもできます。 DemoInit アプリケーションは、以下の 8 つのファイルからなります。

¥DB2e¥WinCE¥Samples¥DemoInit¥Src¥DemoInit.dsp
プロジェクト・ファイル

¥DB2e¥WinCE¥Samples¥DemoInit¥Src¥DemoInit.c
C ソース・コード

¥DB2e¥WinCE¥Samples¥DemoInit¥Src¥DemoInitRes.h
資源生成ヘッダー・ファイル

¥DB2e¥WinCE¥Samples¥DemoInit¥Src¥DemoInit.rc
資源ファイル

¥DB2e¥WinCE¥X86em¥Samples¥DemoInit.exe
Windows CE エミュレーター用の実行可能ファイル

¥DB2e¥WinCE¥MIPS¥Samples¥DemoInit.exe
Windows CE MIPS プロセッサー用の実行可能ファイル

¥DB2e¥WinCE¥SH3¥Samples¥DemoInit.exe
Windows CE SH3 プロセッサー用の実行可能ファイル

¥DB2e¥WinCE¥ARM¥Samples¥DemoInit.exe
Windows CE ARM プロセッサー用の実行可能ファイル

デモ・アプリケーション

Visiting Nurse アプリケーション。このアプリケーションには、DemoInit アプリケーションから入手できるか、または DB2 Everywhere Synchronization Server を使用してサーバーと同期化するこ

とによって入手できるデータのセットが必要です。デモ・アプリケーションは、以下の 8 つのファイルからなります。

¥DB2e¥WinCE¥Samples¥Demo¥Src¥Demo.dsp
プロジェクト・ファイル

¥DB2e¥WinCE¥Samples¥Demo¥Src¥Demo.c
C ソース・コード

¥DB2e¥WinCE¥Samples¥Demo¥Src¥DemoRes.h
資源生成ヘッダー・ファイル

¥DB2e¥WinCE¥Samples¥Demo¥Src¥Demo.rc
資源ファイル

¥DB2e¥WinCE¥x86em¥Samples¥Demo.exe
Windows CE エミュレーター用の実行可能ファイル

¥DB2e¥WinCE¥MIPS¥Samples¥Demo.exe
Windows CE MIPS プロセッサー用の実行可能ファイル

¥DB2e¥WinCE¥SH3¥Samples¥Demo.exe
Windows CE SH3 プロセッサー用の実行可能ファイル

¥DB2e¥WinCE¥ARM¥Samples¥Demo.exe
Windows CE ARM プロセッサー用の実行可能ファイル

Visiting Nurse デモのセットアップ

Palm OS の場合:

エミュレーターまたは PDA 装置上で Visiting Nurse デモを使用する前に、デモ・アプリケーションと DemoInit アプリケーションを導入し、DB2 Everywhere データベースを定義済みのレコードと表のセットを使用して初期設定する必要があります。以下のステップを実行してください。

1. DB2eCmp.prc および DB2eRunTime.prc 共用ライブラリー・ファイル、DemoInit アプリケーション (DemoInit.prc)、およびデモ・アプリケーション (Demo.prc) をエミュレーターまたは PDA 装置にロードする。
2. **DemoInit** アイコンをタップする。

3. **スケジュール**、**個人情報**、**医療記録**、および **連絡先** ボタンをそれぞれタップして、各レコードをスケジュール、個人情報、医療記録、および連絡先の各表に作成し挿入する。DemoInit アプリケーションは、各レコードを各表に挿入します。
4. **終了** をタップして終了する。

Windows CE の場合:

エミュレーターまたは PDA 装置上で Visiting Nurse デモを使用する前に、デモ・アプリケーションと DemoInit アプリケーションを導入し、DB2 Everywhere データベースを定義済みのレコードと表のセットを使用して初期設定する必要があります。以下のステップを実行してください。

1. DB2e.dll 共用ライブラリー・ファイル、DemoInit アプリケーション (DemoInit.exe)、およびデモ・アプリケーション (Demo.exe) をエミュレーターまたは PDA 装置にロードする。
2. **DemoInit** アイコンをタップする。
3. **スケジュール**、**個人情報**、**医療記録**、および **連絡先** ボタンをそれぞれタップして、各レコードをスケジュール、個人情報、医療記録、および連絡先の各表に作成し挿入する。DemoInit アプリケーションは、各レコードを各表に挿入します。
4. **終了** をタップして終了する。

Visiting Nurse デモの表示

以上で、Visiting Nurse デモ・アプリケーションを表示する準備ができました。 33ページの図11 は、Palm OS PDA 装置上で **デモ** アイコンがどのように表示されるかを示しています。

図 11. 「アプリケーション」ウィンドウ

1. 「アプリケーション」ウィンドウから、**デモ** アイコンをタップして、Visiting Nurse アプリケーションを開始する。「スケジュール」ウィンドウがオープンし、当日に訪問する必要がある患者のリストを表示します。

図 12. 「スケジュール」ウィンドウ

2. リストから患者の名前を選択し、**情報** ボタンをタップして、患者に関する一般情報を表示する。

図 13. 「患者情報」ウィンドウ

3. 新しい医療記録を入力する。
   a. **記録** ボタンをタップする。「医療記録リスト」ウィンドウがオープンし、その患者に関して以前に作成されたすべてのレコードのリストを表示します。

図 14. 「医療記録リスト」ウィンドウ

b. **追加** ボタンをタップする。「医療記録」ウィンドウがオープンします。

図 15. 「医療記録」ウィンドウ

c. 患者に関する重要な統計データを入力し、**保管** ボタンをタップして医療記録を保管する。医療記録が、現在の日付とタイム・スタンプとともに保管されます。

4. 患者の緊急連絡先リストを表示する。

   a. **連絡先** ボタンをタップする。「緊急連絡先リスト」ウィンドウがオープンし、患者の緊急連絡先のリストを表示します。

図 16. 「緊急連絡先リスト」ウィンドウ

b. リストから連絡先の名前を選択し、**情報** ボタンをタップして、連絡先に関する情報を表示する。

付録A. サポートされる DB2 CLI 関数

この付録では、DB2 Everywhere によってサポートされる 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。 最初の節では、各関数の目的について簡単に説明するとともに、DB2 Everywhere によってサポートされる DB2 CLI 関数と標準の DB2 CLI 関数の相違点について簡単に要約します。後続の節では、各関数について詳細に説明します。 最後の節には、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す表があります。

DB2 CLI 関数の要約

表4 では、DB2 Everywhere 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 Everywhere 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 4. カテゴリー別の DB2 CLI 関数リスト

関数名	目的	相違点の要約
	データ・ソースへの接続	
SQLAllocConnect (43)	接続ハンドルを入手する。	接続ハンドルはダミー・ハンドルであり、使用可能な情報を持っていない。
SQLAllocEnv (44)	環境ハンドルを入手する。	環境ハンドルはダミー・ハンドルであり、使用可能な情報を持っていない。
SQLAllocHandle (44)	ハンドルを入手する。	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ない。 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ない。

表 4. カテゴリー別の DB2 CLI 関数リスト (続き)

関数名	目的	相違点の要約
SQLConnect (59)	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。	関数呼び出しのすべてのパラメーターは無視される。
	SQL 要求の準備	
SQLAllocStmt (47)	ステートメント・ハンドルを割り当てる。	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ない。
SQLBindParameter (52)	SQL ステートメントのパラメーター用の記憶域を割り当てる。	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ドをサポートしない。 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。
SQLPrepare (87)	後で実行するために SQL ステートメントを準備する。	なし。
	要求の実行依頼	
SQLExecDirect (66)	ステートメントを実行する。	リターン・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 (68)	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。
	結果の取り出しおよび結果に関する情報	
SQLBindCol (48)	結果列用の記憶域を割り当て、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。

表 4. カテゴリー別の DB2 CLI 関数リスト (続き)

関数名	目的	相違点の要約
SQLDescribeCol (60)	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。
SQLError (65)	追加のエラーまたは状況情報を戻す。	なし。
SQLFetch (69)	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。リターン・コード SQL_STILL_EXECUTING をサポートしない。
SQLGetData (79)	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。 リターン・コード SQL_STILL_EXECUTING をサポートしない。
SQLGetDiagRec (83)	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・ハンドルに関連し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。
SQLNumResultCols (86)	結果セット内の列数を戻す。	なし。
SQLRowCount (90)	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。	なし。
	ステートメントの終了	
SQLFreeHandle (73)	ハンドル資源を解放する。	なし。
SQLFreeStmt (76)	ステートメント処理を終了し、保留中の結果を廃棄し、任意選択で、ステートメント・ハンドルに関連したすべての資源を解放する。	SQL_DROP および SQL_RESET_PARAMS オプションのみをサポートする。
	接続の終了	
SQLDisconnect (64)	接続をクローズする。	なし。
SQLFreeConnect (72)	接続ハンドルを解放する。	なし。
SQLFreeEnv (73)	環境ハンドルを解放する。	なし。

DB2 CLI 関数の説明

この付録の残りの部分では、各関数について詳細に説明します。各説明には、以下の項があります。

目的 この項では、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 この項には、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 この項では、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。重要事項として、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、依然として有効である点があげられます。

使用法 この項では、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。起こり得るエラー状態については、ここではなく、診断の項の方でとりあげます。

リターン・コード

この項では、関数について発生し得るすべてのリターン・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 この項には、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

この項では、アプリケーションに影響を与える可能性がある、DB2 Everywhere CLI と ODBC の相違点または制限を示します。

参照 この項では、DB2 Everywhere がサポートする、DB2 CLI 関連関数をリストアップします。

リターン・コード、診断、例、CLI 環境のセットアップ、およびサンプル・アプリケーションへのアクセスに関する情報など、DB2 CLI に関する詳細な説明については、*IBM DB2 Universal Database call level interface Guide and Reference* を参照してください。

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() によって置き換えられました。詳細については、44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 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。

新しい関数への移行

たとえば、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);
```

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() によって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。

新しい関数への移行

たとえば、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);
```

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 5. 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 1 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 で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検査は行いません。また、上書きの前に、ハンドルの直前の内容を検査することもしません。

DB2 Everywhere 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、アプリーションでデータ・

ソースに接続した後 SQLAllocHandle() を呼び出し、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 Everywhere 資源が解放されるため、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 Everywhere 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。

リターン・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とき、 SQLAllocHandle() が SQL_ERROR を戻してきたならば SQLAllocHandle() は *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、出力引き数がヌル・ポインターでなければ *HandleType* の値によって決まります。この後で、アプリケーションでは、*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 6. SQLAllocHandle の SQLSTATE

SQLSTATE	記述	説明
01000	警告。	通知メッセージ。(関数は SQL_SUCCESS_WITH_INFO を戻す。)

表 6. SQLAllocHandle の SQLSTATE (続き)

SQLSTATE	記述	説明
08003	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* 引き数によって指定された接続がオープンされていなかった。 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。	特定の SQLSTATE が存在しなかったためのエラーが発生した。 SQLGetDiagRec() によって **MessageText* バッファーに戻されたエラー・メッセージが、エラーとその原因について示している。
HY001	メモリー割り振りエラー。	DB2 CLI で、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かった。
HY013	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かった。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かったものとおもわれる。
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かった。 SQL_HANDLE_ENV SQL_HANDLE_DBC SQL_HANDLE_STMT

制約事項

なし。

参照

- 66ページの『SQLExecDirect - ステートメントの直接実行』
- 73ページの『SQLFreeHandle - ハンドル資源の解放』

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() によって置き換えられました。詳細については、44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。

新しい関数への移行

たとえば、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);
```

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 7. 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_TINYINT SQL_C_DEFAULT を指定すると、データがその省略時の C データ・タイプに転送される。
SQLPOINTER	*TargetValuePtr*	入出力 (据え置き)	列データの取り出し時に DB2 CLI が列データを格納するバッファーへのポインター。 *TargetValuePtr* がヌルである場合、列はバインドされない。
SQLINTEGER	*BufferLength*	入力	列データの格納に使用可能な *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。

表 7. SQLBindCol の引き数 (続き)

データ・タイプ	引き数	用途	説明
SQLINTEGER *	*StrLen_or_IndPtr*	入出力 (据え置き)	DB2 CLI が *TargetValuePtr* バッファーに戻すために使用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、使用法の項を参照。

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記憶場所は、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。たとえば、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・タイプと長さの記憶場所を割り当て、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ドを変更したり、以前にアンバインドした列をバインドしたりすることができます。 新しいバインドは、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記憶域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記憶域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みなします。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、アプリケーションに戻すために使用可能な *TargetValuePtr* の実際のサイズに設定されます。

リターン・コード

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

診断

表 8. SQLBindCol の SQLSTATE

SQLSTATE	記述	説明
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が失敗した。

表 8. SQLBindCol の SQLSTATE (続き)

SQLSTATE	記述	説明
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。
HY003	プログラム・タイプが範囲外。	*TargetType* が無効なデータ・タイプまたは SQL_C_DEFAULT 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かった。
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能である。	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

なし。

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 9. 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* バッファーに対応する長さ値または SQL_NTS、あるいは SQL_NULL_DATA が入っていなければならない。 DB2 Everywhere は SQLPutData() をサポートしないので、*ParameterValuePtr* バッファーにパラメーター値を格納してはならない。

表 9. SQLBindParameter の引き数 (続き)

データ・タイプ	引き数	用途	説明
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_TINYINT SQL_C_DEFAULT を指定すると、データがその省略時の 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_VARCHAR
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・ストリングまたは単一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。

表 9. *SQLBindParameter* の引き数 (続き)

データ・タイプ	引き数	用途	説明
SQLPOINTER	*ParameterValuePtr*	入力 (据え置き) および/または出力 (据え置き)	入力の場合 (SQL_PARAM_INPUT または SQL_PARAM_INPUT_OUTPUT に設定された *InputOutputType*): 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。
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みなされる。
SQLINTEGER *	*StrLen_or_IndPtr*	入力 (据え置き) および/または出力 (据え置き)	*ParameterValuePtr* に格納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、この記憶場所に SQL_NULL_DATA を入れる必要がある。 *ValueType* が SQL_C_CHAR の場合、この記憶場所に、*ParameterValuePtr* に格納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。 *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、アプリケーションでは常に *ParameterValuePtr* にヌル終了ストリングを提供するとみなされる。これにより、このパラメーター・マーカーがヌル値を持たないことも暗黙指定する。

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記憶域をパラメーター・マーカーにバインドするためには、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、記憶場所は有効であり、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、あるいはこれらの記憶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したい場合があります。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ド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、記憶場所は有効であり、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、あるいはこれらの記憶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 Everywhere は SQL_PARAM_INPUT のみをサポートします。 DB2 Everywhere 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格納してはなりません。

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。

リターン・コード

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

診断

表 10. SQLBindParameter の SQLSTATE

SQLSTATE	記述	説明
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。(たとえば、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が失敗した。
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。

表 10. SQLBindParameter の SQLSTATE (続き)

SQLSTATE	記述	説明
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無効なデータ・タイプまたは SQL_C_DEFAULT である。
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・ポインターであった。また、*InputOutputType* が SQL_PARAM_OUTPUT でない。
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かった。
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 より小さかった。
HY093	無効なパラメーター番号。	引き数 *ValueType* に指定した値が、1 より小さかったか、サーバーによってサポートされる最大数より大きかった。
HY094	無効な位取り値。	*ParameterType* に指定した値が SQL_DECIMAL または SQL_NUMERIC で、*DecimalDigits* に指定した値が 0 より小さかったか、引き数 *ParamDef* (精度) の値より大きかった。
HY104	無効な精度値。	*ParameterType* に指定した値が SQL_DECIMAL または SQL_NUMERIC であり、*ParamDef* に指定した値が 1 より小さかった。
HY105	無効なパラメーター・タイプ。	*InputOutputType* が SQL_PARAM_INPUT でない。
HYC00	ドライバーが使用不能である。	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。

制約事項

なし。

参照

- 66ページの『SQLExecDirect - ステートメントの直接実行』
- 68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 1.1	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 11. 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・ソース: データベースの名前または別名
SQLSMALLINT	*NameLength1*	入力	*ServerName* 引き数の内容の長さ
SQLCHAR *	*UserName*	入力	許可名 (ユーザー識別子)
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)
SQLSMALLINT	*NameLength3*	入力	*Authentication* 引き数の内容の長さ

使用法

DB2 Everywhere の場合、サーバー名、ユーザー名、およびパスワードは無視されます。

リターン・コード

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

診断

表 12. SQLConnect の SQLSTATE

SQLSTATE	記述	説明
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かった。
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拒否した。	データ・ソース (サーバー) が接続の確立を拒否した。
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。

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。

参照

- 44ページの『SQLAllocHandle - ハンドルの割り当て』
- 64ページの『SQLDisconnect - データ・ソースからの切断』

SQLDescribeCol - 列の属性セットを戻す

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、バインド列関数 (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ド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 13. 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* 引き数の値を戻すために使用可能なバイト数。 *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)。

表 13. SQLDescribeCol の引き数 (続き)

データ・タイプ	引き数	用途	説明
SQLSMALLINT *	*NullablePtr*	出力	この列に NULL が許可されるかどうかを示す。 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おらず、何も戻されないとみなします。

リターン・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 14. SQLDescribeCol の SQLSTATE

SQLSTATE	記述	説明
01004	データが切り捨てられる。	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)。
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)。

表 14. SQLDescribeCol の SQLSTATE (続き)

SQLSTATE	記述	説明
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かった。
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が失敗した。
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。
HY002	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、結果セットの中の列数より大きかった。
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	関数の順序エラー。	この関数が、*StatementHandle* に関する `SQLPrepare()` または `SQLExecDirect()` の呼び出しの前に呼び出された。
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能である。	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 Everywhere 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_VARCHAR

参照

- 66ページの『SQLExecDirect - ステートメントの直接実行』
- 86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 15. 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・ハンドルを解放し終る前に 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けれども、追加のエラー情報または導入システムに固有の情報があることを意味します。たとえば、切断に続く終結処理で問題が検出されたか、またはアプリケーションとは別個に発生したイベント (通信障害など) のために、現行接続されていません。

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。

リターン・コード

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

診断

表 16. SQLDisconnect の SQLSTATE

SQLSTATE	記述	説明
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。

参照

- 44ページの『SQLAllocHandle - ハンドルの割り当て』
- 59ページの『SQLConnect - データ・ソースへの接続』
- 73ページの『SQLFreeHandle - ハンドル資源の解放』

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()` によって置き換えられました。詳細については、83ページの『SQLGetDiagRec - 診断レコードの複数フィールド設定値の入手』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。

新しい関数への移行

たとえば、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);
```

新しい関数の使用により以下のように書き直してください。

```
SQLGetDiagRec(SQL_HANDLE_STMT, hstmt, 1, szSqlState, pfNativeError,
    szErrorMsg, cbErrorMsgMax, pcbErrorMsg);
```

SQLExecDirect - ステートメントの直接実行

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 17. 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。

リターン・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 18. SQLExecDirect の SQLSTATE

SQLSTATE	記述	説明
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。
42xxx	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様々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。たとえば、**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・ポインターであった。
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かった。
HY014	ハンドル数の限界に到達済み。	DB2 CLI が内部資源のために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。	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。

参照

- 48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 19. 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、 ? 文字によって表し、SQLExecute() の呼び出し時に、アプリケーション提供の値を置き換えたいステートメント内の位置を示すために使用します。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記憶域をパラメーター・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。

アプリケーションは、SQLExecute() 呼び出しからの結果を処理した後に、新しい (または同じ) パラメーター値を使用してステートメントを再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。

リターン・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 は、SQLExecDirect() に関するすべての SQLSTATE (67ページの表18 を参照) を含みます。ただし、**HY**009 と **HY**090 を除き、表20 の SQLSTATE が加わります。

表 20. SQLExecute の SQLSTATE

SQLSTATE	記述	説明
HY010	関数の順序エラー。	指定した *StatementHandle* が準備済み状態ではなかった。最初に SQLPrepare() を呼び出さないで、SQLExecute() を呼び出した。

制約事項

なし。

参照

- 66ページの『SQLExecDirect - ステートメントの直接実行』
- 87ページの『SQLPrepare - ステートメントの準備』
- 『SQLFetch - 次の行の取り出し』
- 52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』
- 48ページの『SQLBindCol - アプリケーション変数への列のバインド』

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記憶域にバインドすることができます。

SQLFetch() を呼び出すと、列のバインド時に変換が指示されていた場合にはデータ変換を伴う適切なデータ転送が行われます。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 21. 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。

列をアプリケーションの記憶域にバインドすることができます。アプリケーションの記憶域を列にバインドするには、SQLBindCol() を使用します。取り出し時に、データはサーバーからアプリケーションに転送されます。戻すために使用可能なデータの長さも設定されます。

バインド記憶域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 (診断の項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資源を廃棄します。

DB2 Everywhere 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 Everywhere 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、長さバッファーを修正しません。

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。

リターン・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 22. SQLFetch の SQLSTATE

SQLSTATE	記述	説明
01004	データが切り捨てられた。	1 つまたは複数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。)
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。
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。
HY010	関数の順序エラー。	この関数が、*StatementHandle* に関する SQLPrepare() または SQLExecDirect() の呼び出しの前に呼び出された。
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かった。

制約事項

なし。

参照

- 48ページの『SQLBindCol - アプリケーション変数への列のバインド』
- 66ページの『SQLExecDirect - ステートメントの直接実行』
- 79ページの『SQLGetData - 列からのデータの入手』

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() によって置き換えられました。詳細については、73ページの『SQLFreeHandle - ハンドル資源の解放』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、DB2 CLI プログラムでは SQLFreeHandle() を使用してください。

新しい関数への移行

たとえば、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);
```

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() によって置き換えられました。詳細については、『SQLFreeHandle - ハンドル資源の解放』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。

新しい関数への移行

たとえば、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);
```

SQLFreeHandle - ハンドル資源の解放

目的

仕様:	DB2 CLI 5.0	ODBC 3.0	ISO CLI

SQLFreeHandle() は、特定の環境、接続、ステートメント、または記述子の各ハンドルに関連する資源を解放します。

この関数は資源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 23. 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。

使用法

SQLFreeHandle() は、環境、接続、ステートメント、および記述子用の各ハンドルを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検査しません。

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、この環境と活動接続は有効のままとなります。

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 SQLFreeHandle() への呼び出しによって割り当てられたすべての資源が解放されます。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。アプリケーションでステートメント・ハンドルを解放すると、DB2 CLI は、そのハンドルに関連した、すべての自動生成記述子を解放します。 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すべてのステートメントと記述子を自動的に除去します。アプリケーションでステートメント・ハンドルを解放すると、DB2 CLI は、そのハンドルに関連した、すべての自動的生成記述子を解放します。

リターン・コード

- SQL_SUCCESS
- SQL_ERROR
- SQL_INVALID_HANDLE

SQLFreeHandle() が SQL_ERROR を戻した場合、ハンドルは依然として有効です。

診断

表 24. SQLFreeHandle の SQLSTATE

SQLSTATE	記述	説明
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。	特定の SQLSTATE がないエラーが発生した。SQLGetDiagRec() によって **MessageText* バッファーに戻されたエラー・メッセージが、エラーとその原因について示す。
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かった。

表 24. SQLFreeHandle の SQLSTATE (続き)

SQLSTATE	記述	説明
HY010	関数の順序エラー。	*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。SQL_HANDLE_ENV の *HandleType* を指定して SQLFreeHandle() を呼び出す前に、接続ごとに SQL_HANDLE_DBC の *HandleType* 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、接続に関して SQLDisconnect() を呼び出す前に、この関数を呼び出した。 *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資源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* 引き数が SQL_HANDLE_STMT であり、関数呼び出しは処理できなかった。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かったためとおもわれる。
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、自動的に割り当てられた記述子または導入先の記述子用のハンドルに設定されている。

制約事項

なし。

参照

- 44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する。
- アプリケーション変数から列をアンバインドする。
- 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 資源を解放する。

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 25. 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 資源が解放され、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。

照会に関連付けられたステートメント・ハンドルが、異なる数のパラメーターまたはパラメーター・タイプにバインドされている場合、そのハンドルを再使用して別のステートメントを実行するには、パラメーターをリセットする必要があります。

照会に関連付けられたステートメント・ハンドルが、異なる数の列バインドまたは異なる列バインドのタイプにバインドされている場合、そのハンドルを再使用して別のステートメントを実行するには、列をアンバインドする必要があります。

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。

リターン・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・ハンドルがないからです。

診断

表 26. SQLFreeStmt の SQLSTATE

SQLSTATE	記述	説明
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が失敗した。
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。
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 ではなかった。

制約事項

なし。

参照

- 44ページの『SQLAllocHandle - ハンドルの割り当て』
- 48ページの『SQLBindCol - アプリケーション変数への列のバインド』

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。

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 27. 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 から始まる。

表 27. SQLGetData の引き数 (続き)

データ・タイプ	引き数	用途	説明
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_TINYINT SQL_C_DEFAULT の指定により、データはその省略時の 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格納先のバッファーへのポインター。
SQLINTEGER	*BufferLength*	入力	*TargetValuePtr* によって指し示すバッファーの最大サイズ
SQLINTEGER *	*StrLen_or_IndPtr*	出力	DB2 CLI が *TargetValuePtr* バッファーに戻すために使用できるバイト数を示す値へのポインター。データをばらばらに取り出している場合、この引き数に、まだ残っているバイト数が入る。 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する。
2. SQLGetData() によって指定した列に関するデータを転送する。
3. SQLGetData() によって、必要な列に関してステップ 2 を繰り返す。
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する。
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。 行全体に関して取り出さなかったデータを廃棄したい場合、アプリケーションでは SQLFetch() を呼び出して、次の行へ進む必要があります。後者の場合、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記憶域に列データを入れる前に必要なデータ変換 (もし、あれば) 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、常にヌルで終了します。

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 (診断の項を参照してください)。

リターン・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() を呼び出してはなりません。

診断

表 28. SQLGetData の SQLSTATE

SQLSTATE	記述	説明
01004	データが切り捨てられる。	指定した列 (*ColumnNumber*) 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。 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た。
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。	リターン値が、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が失敗した。
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。
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かった。
HY003	プログラム・タイプが範囲外。	*TargetType* が無効なデータ・タイプまたは SQL_C_DEFAULT であった。
HY010	関数の順序エラー。	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かった。
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、省略時タイプが SQL_C_CHAR、SQL_C_BINARY、または SQL_C_DBCHAR のいずれかである。

表 28. SQLGetData の SQLSTATE (続き)

SQLSTATE	記述	説明
HYC00	ドライバーが使用不可能である。	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。

参照

- 48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フィールド設定値の入手

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。

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 29. 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・タイプ識別子。 SQL_HANDLE_STMT でなければならない。

表 29. SQLGetDiagRec の引き数 (続き)

データ・タイプ	引き数	用途	説明
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* に戻すために使用可能な合計バイト数 (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へのポインター。戻すために使用可能なバイト数が *BufferLength* より大きい場合、**MessageText* 内のエラー・メッセージ・テキストは *BufferLength* からヌル終了文字の長さを引いた文字数に切り捨てられる。

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。

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() を除く別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すべて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 Everywhere ではステートメント・ハンドルのみがサポートされています。

リターン・コード

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

診断

SQLGetDiagRec() は自分自身に関するエラー値を通知しません。
SQLGetDiagRec() は自分自身の実行の結果を報告するために、以下のリターン値を使用します。

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でした。診断レコードは生成されませんでした。切り捨てが発生したかどうかを判別するには、アプリケーションで、*StringLengthPtr* に書き込まれた使用可能な実際のバイト数と *BufferLength* を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でした。

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

ード数より大でした。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。

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 30. 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。

リターン・コード

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

診断

表 31. SQLNumResultCols の SQLSTATE

SQLSTATE	記述	説明
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が失敗した。
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。
HY010	関数の順序エラー。	この関数が、*StatementHandle* に関する SQLPrepare() または SQLExecDirect() の呼び出しの前に呼び出された。
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かった。

制約事項

なし。

参照

- 48ページの『SQLBindCol - アプリケーション変数への列のバインド』
- 60ページの『SQLDescribeCol - 列の属性セットを戻す』
- 66ページの『SQLExecDirect - ステートメントの直接実行』
- 79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 32. 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
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されるか、もしステートメント・テキストがヌルで終了しているとすれば、SQL_NTS に設定される必要があります。

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、 ? 文字によって表し、SQLExecute() の呼び出し時に、アプリケーション提供の値を置き換えたいステートメント内の位置を示すために使用します。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、68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・マーカーに関連する規則については、*DB2 SQL 解説書* の PREPARE の項を参照してください。

アプリケーションでは、SQLExecute() 呼び出しからの結果を処理した後に、新しい (または同じ) パラメーター値を使用してステートメントを再実行することができます。

リターン・コード

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

診断

表 33. SQLPrepare の SQLSTATE

SQLSTATE	記述	説明
42xxx [a]	構文エラー。	**42**xxx SQLSTATES は、ステートメントに関する様々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。たとえば、**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・ポインターであった。
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かった。
HY014	ハンドル数の限界に到達済み。	DB2 CLI が内部資源のために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。	引き数 *TextLength* が 1 より小さいが、SQL_NTS と等しくなかった。

制約事項

なし。

参照

- 52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』
- 60ページの『SQLDescribeCol - 列の属性セットを戻す』
- 66ページの『SQLExecDirect - ステートメントの直接実行』

- 68ページの『SQLExecute - ステートメントの実行』
- 86ページの『SQLNumResultCols - 結果の列数の入手』

SQLRowCount - 行カウントの入手

目的

仕様:	DB2 CLI 1.1	ODBC 1.0	ISO CLI

SQLRowCount() は、表または表に基づくビューに対して実行された UPDATE、INSERT、または DELETE ステートメントによって影響された表内の行数を戻します。

SQLExecute() または SQLExecDirect() は、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 34. 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	影響された行数が格納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最後に実行されたステートメントが UPDATE、INSERT、または DELETE ステートメントでなかった場合、あるいは、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他の表の行 (たとえば、連鎖削除) は、このカウントに含まれません。

リターン・コード

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR

- SQL_INVALID_HANDLE

診断

表 35. SQLRowCount の SQLSTATE

SQLSTATE	記述	説明
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が失敗しました。
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ません。
HY010	関数の順序エラー。	この関数が、*StatementHandle* に関する SQLExecute() または SQLExecDirect() の呼び出しの前に呼び出されました。
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ませんでした。

制約事項

なし。

参照

- 66ページの『SQLExecDirect - ステートメントの直接実行』
- 68ページの『SQLExecute - ステートメントの実行』
- 86ページの『SQLNumResultCols - 結果の列数の入手』

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 Everywhere 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・タイプを指示します。 これらの関数は記号名 (SQL_CHAR および SQL_C_CHAR など) を使用して、関連するデータ・タイプを指定します。

たとえば、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の適切な 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36 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36 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:

D この変換はサポートされ、SQL データ・タイプの省略時変換です。

X DB2 Everywhere はこの変換をサポートします。

ブランク
DB2 Everywhere はこの変換をサポートしません。

データ・タイプの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については、*IBM SQL 解説書* を参照してください。

表 36. サポートされるデータ変換

SQL データ・タイプ	SQL_C_CHAR	SQL_C_LONG	SQL_C_SHORT	SQL_C_TINYINT	SQL_C_FLOAT	SQL_C_DOUBLE	SQL_C_TYPE_DATE	SQL_C_TYPE_TIME	SQL_C_BINARY	SQL_C_BIT
BLOB	X								D	
CHAR	D	X	X	X			X	X	X	X
DATE	X						D			
DECIMAL	D	X	X	X						X
INTEGER	X	D	X	X	X	X				X
SMALLINT	X	X	D	X	X	X				X
TIME	X							D		
VARCHAR	D	X	X	X			X	X	X	X

付録B. サポートされる SQL ステートメント

この章では、DB2 Everywhere 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。以下の実行可能な SQL ステートメントは、QBE 内でコマンド行プロセッサー (CLP) を使用して PDA 装置から対話式に出すことができます。また、アプリケーション・プログラム内で DB2 Everywhere データベース内のデータにアクセスするために使用することもできます。表37 は、DB2 Everywhere 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 37. サポートされる SQL ステートメント

SQL ステートメント	機能	ページ
CREATE TABLE	表を定義する。	93
DELETE	表から 1 行または複数行を削除する。	97
DROP	データベース内の表を削除する。	101
INSERT	表に 1 行を挿入する。	101
SELECT	1 つまたは複数の表から照会された結果表を指定する。	104
UPDATE	表の 1 行または複数行の中の 1 つまたは複数の列の値を更新する。	111

116ページの『診断』 は、DB2 Everywhere 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 2048 文字を超えてはなりません。

カタログには、DB2 Everywhere によって管理される DB2 Everywhere システム表 DB2eSYSTABLES および DB2eSYSCOLUMNS が含まれています。

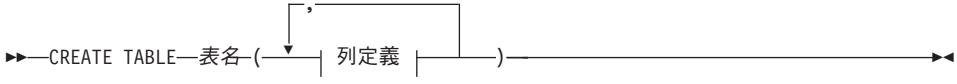
CREATE TABLE

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、表の基本キーなど、表に関するその他の属性も含めることもできます。

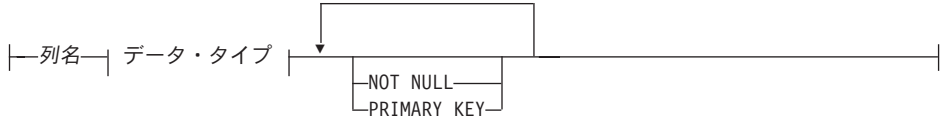
呼び出し

このステートメントをアプリケーション・プログラム内で用いる場合は、DB2 CLI 関数を使用します。このステートメントは QBE 内で CLP を介して出すこともできます。

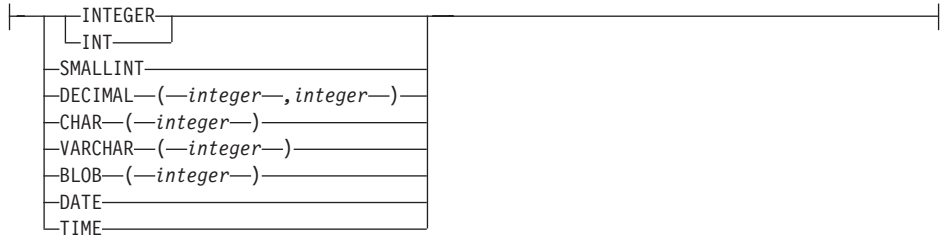
構文



列定義:



データ・タイプ:



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 PDA 装置として固有なものでなければなりません。

表名は、カタログに格納される前に大文字に変換されます。このような変換を防ぐには、区切り識別子 (二重引用符) を使用してください。表名にブランクまたは特殊文字が含まれている場合にも区切り識別子を使用することができます。

列定義

列の属性を定義します。

列のバイト・カウントの合計は 65536 を超えてはなりません。

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、カタログに格納される前に大文字に変換されます。このような変換を防ぐには、区切り識別子 (二重引用符) を使用してください。列名にブランクまたは特殊文字が含まれている場合にも区切り識別子を使用することもできます。

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 -2147483647 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*(precision-integer, scale-integer)*

10 進数の場合。最初の整数は精度、すなわち、総桁数です。この範囲は 1 から 15 までです。 2 番目の整数は数値の位取り、すなわち、小数点の右側に来る桁数です。この範囲は 0 からその数値の精度までです。

CHAR*(integer)*

長さが *integer* 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*(integer)*

最大長が *integer* 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*(integer)*

バイト単位で指定した最大長の 2 進ラージ・オブジェクト・ストリングの場合。

最大長の範囲は 1 バイトから 32767 バイトまでです。

integer は最大長です。

DATE
日付の場合。入力値の形式は以下のいずれかです。
MM/DD/YYYY、YYYY-MM-DD、または MM.DD.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。入力値の形式は以下のいずれかです。HH:MM AM (または PM)、HH:MM:SS、または HH.MM.SS。 後の 2 つの形式の 3 番目の構成要素 (SS、秒) は任意選択です。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。

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、列はヌル値を含むことができます。

PRIMARY KEY
基本キー列は NOT NULL で、その値は固有でなければなりません。DB2 Everywhere では、基本キーは単一列のみから構成しなければなりません。

規則

- 列のバイト・カウントの合計は 65536 を超えてはなりません。詳細については、『注意事項』を参照してください。

注意事項

- PDA 装置上に新しい表を作成する場合、PDA 装置 をサーバーと同期化することによって、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。 同期化の前に、エンタープライズ・データベース上に表を作成する必要があります。
- SQLSTATE 42710 (指定されたオブジェクトがすでに存在している) により、作成しようとしている表が DB2 Everywhere のカタログにはないが、PDA 装置にすでに存在している状態を示す場合があります。このような表は、手操作で作成またはインポートした可能性があります。この場合、次の DROP TABLE 操作によっても SQLSTATE 42704 (未定義のオブジェクト)

が戻されます。 この名前を使用して表を作成するには、既存の表を PDA 装置から手操作で除去する必要があります。

- ***Palm OS データのバイト・カウント:*** 以下のリストでは、データ・タイプ別にヌル不可能列とヌル可能列のバイト・カウントを示しています。

データ・タイプ	ヌル不可能列	ヌル可能列
INTEGER	4	5
SMALLINT	2	3
DECIMAL(n, m)	floor(n/2)+1	floor(n/2)+2
CHAR(n)	n+1	n+2
VARCHAR(n)	n+3	n+3
LOB タイプ	n+4	n+4
DATE	4	5
TIME	4	5

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。 基本キーは列 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)
```

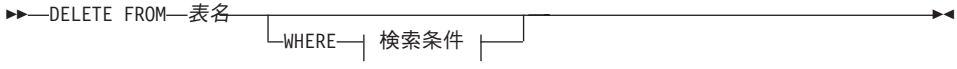
DELETE

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。

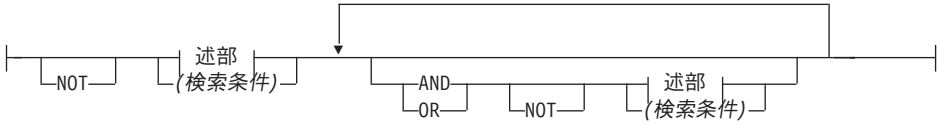
呼び出し

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 関数を使用して使用することもできますし、QBE 内で CLP を介して出すこともできます。

構文

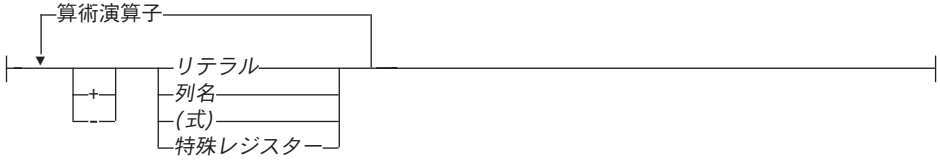


検索条件:



述部:

式:



算術演算子:

関係演算子:

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、または不明である条件を指定します。

検索条件 の結果は、指定したそれぞれの述部の結果に、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、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、および TIME データ・タイプに関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、または TIME 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。現在の日付と時刻を生成するために、特殊レジスター CURRENT DATE および CURRENT TIME を使用することができます。

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 つまたは複数の同じ文字を持つ

NOT LIKE
同じ文字を 1 つも持たない

IS NULL
ヌル値を含んでいる

IS NOT NULL
ヌル値を含んでいない

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

- 複数行の DELETE ステートメントの実行中にエラーが発生した場合、エラーの発生前に削除された行は削除されたままになります。

例

従業員番号 (EMPNO) 003002 を EMPLOYEE 表から削除します。

```
DELETE FROM EMPLOYEE
  WHERE EMPNO = '003002'
```

DROP

DROP ステートメントは表を削除します。

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、QBE 内で 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)。

規則

なし。

注意事項

- SQLSTATE 42704 (オブジェクトが未定義) は、除去しようとしている表が DB2 Everywhere カタログにはないが、PDA 装置にすでに存在している状態を示す場合があります。 このような表は、手操作で作成またはインポートした可能性があります。 この場合、次の CREATE TABLE 操作によって SQLSTATE 42710 (指定されたオブジェクトがすでに存在している) も戻されます。この表を除去するには、手操作で PDA 装置から除去する必要があります。

例

表 EMPLOYEE を除去します。

```
DROP TABLE EMPLOYEE
```

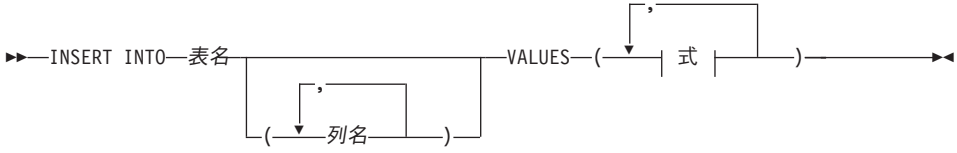
INSERT

INSERT ステートメントは、提供された値を使用して、1 つの表に単一の行を挿入します。

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、QBE 内で CLP を介して出すこともできます。

構文



式:

算術演算子:

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

挿入すべき値の 1 行を指定します。

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

式としてリテラルまたは特殊レジスターを指定することができます。

BLOB(n)、DATE、および TIME データ・タイプに関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、または TIME の値にすることができます。

特殊レジスター

現在の日付と時刻を生成するために、特殊レジスター CURRENT DATE および CURRENT TIME を使用することができます。

規則

- ***省略時値:*** 列リストの中に存在しない列にヌル値が挿入されます。ヌル値を許されない列は、列リストに入れる必要があります。
- ***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 SQL 解説書* に示されている割り当て規則に基づいて列に割り当てられます。

注意事項

なし。

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
- First name (FIRSTNAME) is Jim
- ラストネーム (LASTNAME) は Gray

```
INSERT INTO EMPLOYEE (EMPNO, FIRSTNAME, LASTNAME)
  VALUES ('003002', 'Jim', 'Gray')
```

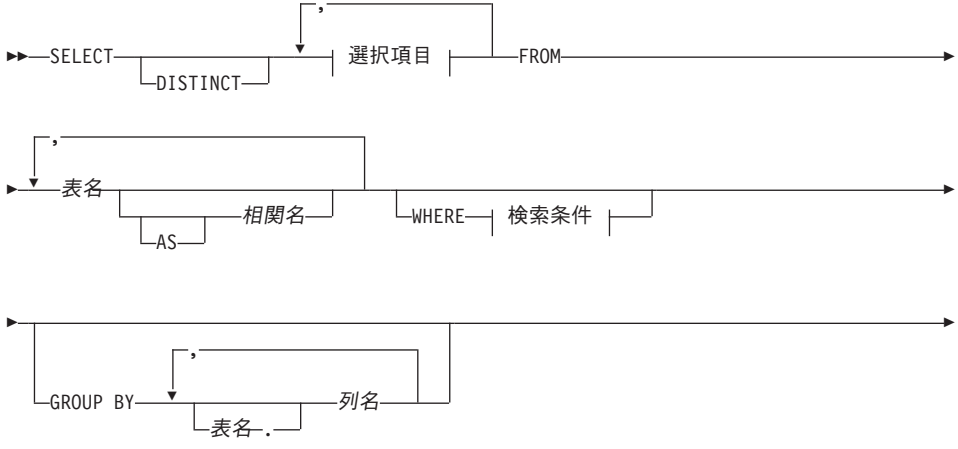
SELECT

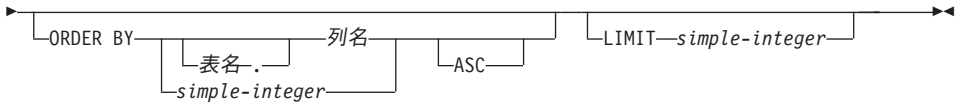
SELECT ステートメントは照会の 1 形式です。

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、QBE 内で CLP を介して出すこともできます。

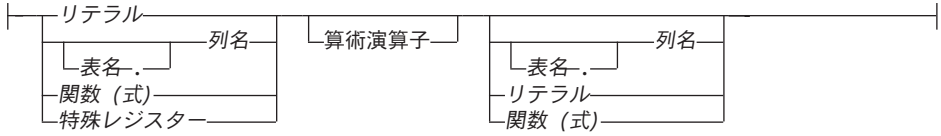
構文





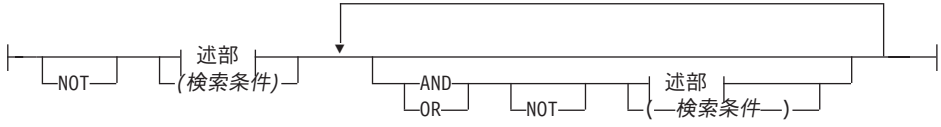
選択項目:

選択項目の式:

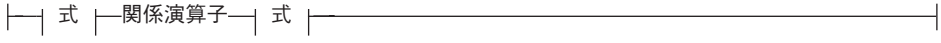


算術演算子:

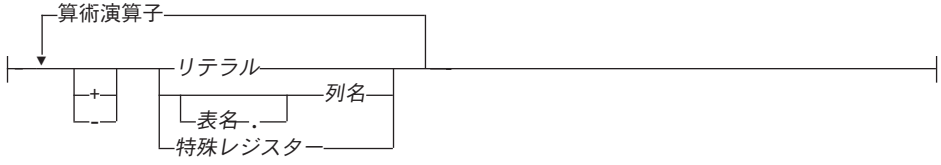
検索条件:



述部:



述部の式:



関係演算子:

説明

DISTINCT

最終結果表の重複行の各セットのうち 1 つを残して、すべての行を取り除きます。

2 つの行が互いに重複するのは、最初の行のそれぞれの値が 2 番目の行の対応する値に等しい場合だけです。 (重複行の判別に際して、2 つのヌル値は等しいものと見なされます。)

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。 有効な関数は COUNT、AVG、SUM、MIN、および MAX です。

BLOB(n)、DATE、および TIME データ・タイプに関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、または TIME 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識別子 *表名.列名* の区切り記号。

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。ヌル値を取り除くことにより、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。 ヌル値を取り除くことにより、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、*列名* 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。 ヌル値を取り除くことにより、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 1 組の値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。ヌル値を取り除くことにより、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 1 組の値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。ヌル値を取り除くことにより、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。

特殊レジスター
現在の日付と時刻を生成するために、特殊レジスター CURRENT DATE および CURRENT TIME 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。

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 *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、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、および TIME 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、または TIME 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識別子 *表名.列名* の区切り文字。

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。現在の日付と時刻を生成するために、特殊レジスター CURRENT DATE および CURRENT TIME を使用することができます。

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 つまたは複数の同じ文字を持つ

NOT LIKE
同じ文字を 1 つも持たない

IS NULL
ヌル値を含んでいる

IS NOT NULL
ヌル値を含んでいない

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。

simple-integer
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
列の値を降順で使用します。

LIMIT *simple-integer*
選択すべき行数を制限します。1 以上で、結果表の行数以下でなければなりません。

規則

なし。

注意事項

- SELECT ステートメントを使用すると、照会が長時間実行される可能性があります。 Palm OS PDA 装置上で長時間実行されている照会を停止するには、装置上の任意のボタン (たとえば、**up** または **down** ボタン) を押すと、SQLSTATE 57014 が DB2 Everywhere によって生成されます。その後、アプリケーションで割り込みを処理する方法を決定することができます。

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
```

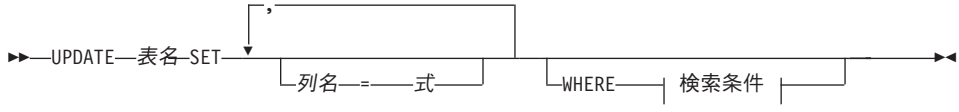
UPDATE

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、QBE 内で CLP を介して出すこともできます。

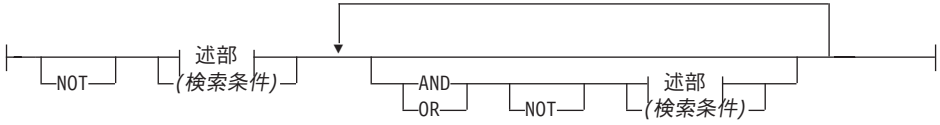
構文



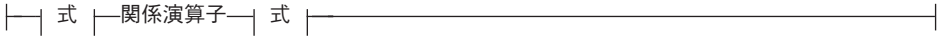
式:

算術演算子:

検索条件:



述部:



関係演算子:

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、および TIME 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、または TIME の値にすることができます。

特殊レジスター

現在の日付と時刻を生成するために、特殊レジスター CURRENT DATE および CURRENT TIME 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、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 *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。

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 つまたは複数の同じ文字を持つ

NOT LIKE
同じ文字を 1 つも持たない

IS NULL
ヌル値を含んでいる

IS NOT NULL
ヌル値を含んでいない

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 SQL 解説書* に示されている割り当て規則に基づいて列に割り当てられます。

注意事項

- 更新値がいずれかの制約に従っていない場合、または UPDATE ステートメントの実行中にその他のエラーが発生した場合には、行は更新されません。複数行が更新される順序は未定義です。
- 複数行の UPDATE ステートメントの実行中にエラーが発生した場合、エラーの発生前に更新された行は更新されたままになります。

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'
```

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。 各演算に関係するオペランドのデータ・タイプには、表38 に示すような互換性が必要です。

データ・タイプの列の文字の意味:

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 38. データ・タイプの互換性

SQL データ・タイプ	INT	VARCHAR	BLOB	DECIMAL	CHAR	SMALLINT	DATE	TIME
INT	X			X		X		
VARCHAR		X			X		X	X
BLOB			X					
DECIMAL	X			X		X		
CHAR		X			X		X	X
SMALLINT	X			X		X		
DATE		X			X		X	
TIME		X			X			X

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個別パラメーター・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、92ページの表36 を参照してください。

パラメーター・マーカー

DB2 Everywhere 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39 では、これらの使用法をリストアップしています。

表 39. タイプ無しパラメーター・マーカーの使用法

タイプ無しパラメーター・マーカーを置く場所	データ・タイプ
式	
選択リスト内のみ	エラー
算術演算子の両方のオペランド	エラー
算術演算子の片方のオペランド	もう一方のオペランドのデータ・タイプ
UPDATE ステートメントの SET 文節の右側の値として	列のデータ・タイプ
INSERT ステートメント内の VALUES 文節の値としてのみ	列のデータ・タイプ
述部	
関係演算子の両方のオペランド	エラー
関係演算子の片方のオペランド	もう一方のオペランドのデータ・タイプ
関数	
総計関数のオペランド	エラー

診断

表40 は、DB2 Everywhere 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 によって報告される SQLSTATE は、39ページの『付録A. サポートされる DB2 CLI 関数』の各 DB2 CLI 関数の説明の個所に記載してあります。

表 40. SQLSTATE

SQLSTATE	記述	説明
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ました。
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中に行が検出されませんでした。
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ません。
08002	接続がすでに存在している。	接続がすでに存在しています。
22001	値に切り捨てが必要。	値に、システムのキャスト関数または調整関数による切り捨てが必要です。

表 40. SQLSTATE (続き)

SQLSTATE	記述	説明
22002	ヌル標識が提供されない。	NULL 値は、記憶域が提供されていないため、割り当てることができません。
22003	数値が範囲外。	数値がターゲット列の範囲外です。
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ありません。
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す。
22012	0 による除算。	0 による除算が試みられました。
23502	ヌル値が許されない。	NOT NULL 列に対する NULL 値の割り当ては許されません。
23505	値が固有でない。	この操作は、重複キーを生成することになるため、無効です。
23515	複数の基本キー文節が指定された。	複数の基本キー文節が指定されました。
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 は無効です。
42601	構文エラー。	SQL ステートメント内の構文エラーが検出されました。
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識別子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ます。パラメーター・マーカーの有効な使用法については、116ページの表39 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ます。
42622	名前が長すぎる。	識別子の名前が長すぎます。
42702	あいまいな列名参照。	複数の列が参照されている可能性があります。
42703	未定義の列名。	列名が参照されている表にありません。
42704	未定義のオブジェクト。	表が存在していません。
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ます。
42711	重複した列名。	同じ列名が複数回指定されています。
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ありません。
42803	SELECT リスト内の列参照が GROUP BY 文節に指定されていない。	列名および総計関数が選択リスト内に含まれていますが、GROUP BY 文節がありません。

表 40. SQLSTATE (続き)

SQLSTATE	記述	説明
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・タイプに互換性がありません。
42820	リテラル値が範囲外。	指定された数値が許容範囲外です。
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ありません。
42822	無効な ORDER BY 項目。	ORDER BY 項目が選択リストにありません。
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ません。
42884	不明な関数名。	不明な関数が参照されました。
42962	LOB 列はキーとして使用できない。	LOB 列は基本キーとして使用できません。
42997	サポートされない機能。	機能が現行のリリースではサポートされていません。
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ます。
57011	メモリー不足。	システムで動的メモリーを割り当てることができません。
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ました。
58004	内部システム・エラー (継続)。	重大でないシステム・エラーが発生しました。
58005	内部システム・エラー (停止)。	重大なシステム・エラーが発生しました。

特記事項

本書において、日本では発表されていないIBM製品（機械およびプログラム）、プログラミングまたはサービスについて言及または説明する場合があります。しかし、このことは、弊社がこのようなIBM製品、プログラミングまたはサービスを、日本で発表する意図があることを必ずしも示すものではありません。本書で、 IBMライセンス・プログラムまたは他のIBM製品に言及している部分があっても、このことは当該プログラムまたは製品のみが使用可能であることを意味するものではありません。これらのプログラムまたは製品に代えて、IBMの知的所有権を侵害することのない機能的に同等な他社のプログラム、製品またはサービスを使用することができます。ただし、IBMによって明示的に指定されたものを除き、これらのプログラムまたは製品に関連する稼働の評価および検証はお客様の責任で行っていただきます。

IBMおよび他社は、本書で説明する主題に関する特許権（特許出願を含む）商標権、または著作権を所有している場合があります。本書は、これらの特許権、商標権、および著作権について、本書で明示されている場合を除き、実施権、使用権等を許諾することを意味するものではありません。実施権、使用権等の許諾については、下記の宛先に、書面にてご照会ください。

〒106-0032 東京都港区六本木3丁目2-31
AP事業所
IBM World Trade Asia Corporation
Intellectual Property Law & Licensing

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 Corporation
J74/G4
555 Bailey Avenue
P.O. Box 49023
San Jose, CA 95161-9023
U.S.A.

本プログラムに関する上記の情報は、適切な条件の下で使用することができますが、有償の場合もあります。

商標

本書で使用されている下記の用語は、IBM Corporation の商標です。

AIX
AS/400
DATABASE 2
DB2
DB2 Universal Database
IBM
MVS/ESA
OS/2
OS/390

Lotus Notes は Lotus Development Corporation の登録商標です。

Microsoft、Windows、および Windows NT は Microsoft Corporation の登録商標です。

UNIX は X/Open Company Limited の登録商標です。

その他の会社名、製品名、およびサービス名は、他社の商標またはサービス・マークである場合があります。

用語および省略語

B

2 進ラージ・オブジェクト (BLOB) (binary large object (BLOB)). 連続したバイトで、そのサイズの範囲は 0 バイトから 2 ギガバイトまで。このストリングには関連するコード・ページも文字セットもない。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 に格納される。

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。

BLOB. 2 進ラージ・オブジェクト (Binary large object)。

C

クライアント (client). データベース・サーバーと通信したり、データベース・サーバーにアクセスするプログラム (または、このようなプログラムが実行されているワークステーション)。

D

データベース管理システム (DBMS) (database management system (DBMS)). データベース・マネージャー (database manager) の同義語。

データベース・マネージャー (database manager). 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、回復、並行制御、プライバシー、機密保護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。

DB2 コール・レベル・インターフェース (CLI) (DB2 Call Level Interface (DB2 CLI)). DB2 ファミリーのデータベース・サーバーに対する IBM の呼び出し可能 SQL インターフェース。リレーショナル・データベース・アクセス用の C および C++ アプリケーションのプログラミング・インターフェースである。このインターフェースは、関数呼び出しを使用して、動的 SQL ステートメントを関数の引き数として渡す。これは組み込み動的 SQL の代替になるが、組み込み SQL と異なり、ホスト変数またはプリコンパイラーを必要としない。

DB2 CLI は Microsoft オープン・データベース・コネクティビティー (ODBC) 仕様および X/Open コール・レベル・インターフェース仕様に基づいている。

DB2 CLI. DB2 コール・レベル・インターフェース (DB2 Call Level Interface)。

DBMS. データベース管理システム (Database management system)。

区切り識別子 (delimited identifier). 二重引用符 (″) で囲まれた連続した文字。この連続した文字は、英字とその後に続く 0 個または 1 個の文字からなり、各文字は英字、数字、または下線でなければならない。

特殊タイプ (distinct type). 既存のタイプ (ソース・タイプ) として内部表現されているが、セマンティクスとしては、別個で互換性のないタイプと考えられるユーザー定義データ・タイプ。ユーザー定義タイプ (UDT) (user-defined type (UDT)) も参照。

H

携帯用装置 (handheld device). 携帯可能なコンピューティング装置。携帯用装置としては、手のひらサイズの PC およびパーソナル・ディジタル・アシスタント (PDA) がある。

J

結合 (join). 関係演算の 1 つ。突き合わせ列値を基準にして 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。

K

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。

L

ラージ・オブジェクト (LOB) (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) または DBCLOB (2 バイト文字) の 3 つがある。

LOB. ラージ・オブジェクト (Large object)。

M

モバイル (mobile). 様々なロケーションへ頻繁に移動し、様々なタイプのネットワーク接続 (たとえば、ダイヤル呼び出し、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携帯用装置上で行うコンピューティングに関する用語。*リモート (remote)* と対比。

N

ヌル可能 (nullable). 列、関数パラメーター、または結果の値として、値を持ち得ないことが許される状態。たとえば、個人のミドル・イニシャルのフィールドは値を必要としない。

ヌル値 (null value). 値が指定されていないパラメーター。

O

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。たとえば、表、ビュー、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。

ODBC. オープン・データベース・コネクティビィティー (Open Database Connectivity)。

ODBC ドライバー (ODBC driver). ODBC 関数呼び出しを実行し、データ・ソースと対話するドライバー。

オープン・データベース・コネクティビティー (ODBC) (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に直接リンクする必要はない。

通常識別子 (ordinary identifier). SQL の場合、1 つの英字の後に 0 個または 1 個以上の文字が続いたものであり、名前を作成するために使用される。2 文字目以降はそれぞれ、英字 (a-z および A-Z)、記号、数字、または下線文字でなければならない。

P

PDA. パーソナル・ディジタル・アシスタント (Personal digital assistant)。

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記憶装置に保持されるデータに関する用語。

パーソナル・ディジタル・アシスタント (PDA) (personal digital assistant (PDA)). 個人の編成作業 (たとえば、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の各機能を備えた携帯用装置。

パーベイシブ・コンピューティング (PVC) (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。

プラグイン (plug-in). 特定のアプリケーション内の関数を修正 (追加または変更) する自己完結型ソフトウェア構成要素。 ユーザーは、元のアプリケーションの基礎を変更しないまま、アプリケーションにプラグインを容易に追加することができる。

述部 (predicate). 比較演算を表すか、あるいはそれを暗黙に指定する検索条件の要素。

基本キー (primary key). 表定義の一部をなす固有のキー。 基本キーは、参照制約定義の省略時の親キー。

PVC. パーベイシブ・コンピューティング (Pervasive computing)。

Q

QBE. 例示照会 (Query By Example)。

照会 (query). 特定の条件に基づいた情報をデータベースから要求すること。たとえば、顧客表の中から残高が $1000 を超えるすべての顧客のリストを要求すること。

例示照会 (Query By Example). DB2 Everywhere 表に格納されているデータをユーザーが動的に表示したり変更できるようにするアプリケーション。

R

RDBMS. リレーショナル・データベース管理システム (Relational database management system)。

リモート (remote). 中央側 (たとえば、オフィス・ロケーション) から離れたロケーションで、通常はネットワーク接続を介して行われるコンピューティングに関する用語。リモート・コンピューティング装置は、据え付けで移動できないものも、移動できるものもある。*モバイル (mobile)* と対比。

リモート・データベース (remote database). 使用中のワークステーション以外のワークステーションに物理的に存在するデータベース。

S

SQL. 構造化照会言語 (Structured Query Language)。

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。

T

タップ (tap). 携帯用装置と対話するためにスタイラスを使用すること。

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。

V

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。

W

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。

参考文献

ここには、本書で引用したすべての資料と、DB2 Everywhere ユーザーに追加情報を提供する資料がリストアップしてあります。

IBM DB2 Everywhere 関連資料

DB2 Everywhere ライブラリーとして以下の資料があります。

- *管理およびアプリケーション・プログラミングの手引き*、SC88-8509

  この資料では、DB2 Everywhere 製品用のアプリケーションに関するセットアップ、使用、ならびに開発の方法について記述してあります。

- *DB2 Everywhere Synchronization Server Administration Guide*

IBM DB2 ユニバーサル・データベース関連資料

IBM DB2 ユニバーサル・データベースに関する資料として次のものがあります。

- *IBM DB2 ユニバーサル・データベース CLI ガイドおよび解説書*、SB88-7228
- *IBM DB2 ユニバーサル・データベース API 解説書*、SB88-7236
- *IBM DB2 ユニバーサル・データベース SQL 解説書*、SB88-7234

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]

[ハ行]

［ヤ行］

［ラ行］

［ワ行］

A

B

C

D

F

I

N

P

Q

S

T

U

V

W

プログラム番号: 5648-C61

Printed in Japan

SC88-8509-00

日本アイ・ビー・エム株式会社
〒106-8711　東京都港区六本木3-2-12